THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY FOR WIRELESS SENSOR NETWORKS

Master Thesis

Utku GÜLEN

İSTANBUL, 2014

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

Graduate School of Natural and Applied Sciences
Computer Engineering

IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY FOR WIRELESS SENSOR NETWORKS

Master Thesis

Utku GÜLEN

Supervisor: Asst. Prof. Dr. Selçuk BAKTIR

İSTANBUL, 2014

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY
Graduate School of Natural and Applied Sciences
Computer Engineering

Title of the Master's Thesis	:	Implementing Elliptic Curve Cryptography for Wireless Sensor Networks
Name/Last Name of the Student	:	Utku GÜLEN
Date of Thesis Defense	:	1 September 2014

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. Dr. F. Tunç BOZBURA
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Arts.

Asst. Prof. Dr. Tarkan AYDIN
Program Coordinator

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Arts.

Examining Commitee Members	Signature
Asst. Prof. Dr. Selçuk BAKTIR (Supervisor)	: _____
Assoc. Prof. Mehmet Alper TUNGA	: _____
Asst. Prof. Dr. Tarkan AYDIN	: _____

ÖNSÖZ

Eğitimime ve bu çalışmanın hazırlanmasına vermiş olduğu emeği için danışman hocam Yrd. Doç. Dr. Selçuk BAKTIR'a, değerli zamanlarını bana ayırıp tez jürime katılan Yrd. Doç. Dr. Tarkan AYDIN ve Doç. Dr. M. Alper TUNGA'ya, bölümümdeki ve derslerini aldığım sayın hocalarıma, bilgilerini paylaşıp motivasyonumu olumlu yönde etkileyen değerli çalışma arkadaşlarıma ve aileme sonsuz teşekkürlerimi sunarım.

İSTANBUL 2014 Utku GÜLEN

ABSTRACT

IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY FOR WIRELESS SENSOR NETWORKS

Utku Gülen

COMPUTER ENGINEERING
Thesis Supervisor: Asst. Prof. Dr. Selçuk BAKTIR

1 September 2014, 39 Pages

In this work, elliptic curve cryptography (ECC) is implemented in the frequency domain on the constrained MSP430 microcontroller widely used in wireless sensor networks (WSN). 169-bit ECC implementation performs a scalar point multiplication in only 1.55 and 0.77 s for multiplication of random and fixed points, respectively. The timings are similar to or faster than existing implementations on the same platform. Furthermore, an alternative implementation of ECC, without utilizing hardware multiplier support is achieved for low-power applications on WSNs. Surprisingly, the implementations without hardware multiplier support were only 21.9 percent and 22.1 percent slower than with hardware multiplier support, for random and fixed point multiplication, respectively. This is the first ever practical software implementation of ECC in the frequency domain on a constrained low-power microcontroller without hardware multiplier support.

Keywords: Elliptic Curve Cryptography, Wireless Sensor Networks, Discrete Fourier Transform, MSP430, Microcontroller,

ÖZET

KABLOSUZ SENSÖR AĞLARINDA ELİPTİK EĞRİYLE ŞİFRELEMENİN UYGULANMASI

Utku Gülen

Bilgisayar Mühendisliği
Tez Danışmanı: Yrd Doç. Dr. Selçuk BAKTIR

1 Eylül 2014, 39 Sayfa

Bu çalışmada, sınırlı işlem kapasitesine sahip MSP430 mikrodenetleyicileri için frekans bölgesinde Eliptik Eğriyle şifreleme (ECC) uygulaması yapılmıştır. MSP430, Kablosuz Sensör Ağları'nda (WSN) çokça kullanılan bir mikrodenetleyicidir. Yaptığımız çalışmada, 169 bit ECC için, rastlantısal ve sabit nokta çarpımı sırasıyla 1.55s ve 0.77s sürmektedir. İşlem zamanlamaları aynı platformu kullanan diğer çalışmalara göre hemen hemen aynı veya daha hızlıdır. Ayrıca bu çalışmada ECC'yi, düşük güç tüketimi amaçlayan WSN uygulamaları için, MSP430 içinde bulunan donanımsal çapma birimini kullanmadan da gerçekledik. ECC'de kullanılan rastlantısal ve sabit nokta çarpma işlemleri, donanımsal çarpma birimi kullanmadan sırasıyla yüzde 21.9 ve yüzde 22.1 daha yavaş çalışmaktadır. Bu çalışmadaki ECC yazılımı, sınırlı performansı olan mikrodenetleyicilerde frekans bölgesi işlemleri kullanılarak gerçeklenmiş ilk uygulamadır.

Anahtar Kelimeler: Eliptik Eğriyle Şifreleme, Kablosuz Sensör Ağları, Ayrık Fourier Dönüşümü, MSP430, Mikrodenetleyici

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

ECC	:	Elliptic Curve Cryptography
ECDL	:	Elliptic Curve Discrete Logarithm
ECDSA	:	Elliptic Curve Digital Signature Algorithm
DFT	:	Discrete Fourier Transform
MSB	:	Most Significant Bit
NAF	:	Non-Adjacent Form
NIST	:	National Institute of Technology
NTT	:	Number Theoretic Transform
OEF	:	Optimal Extension Field
PKC	:	Public Key Cryptography
RISC	:	Reduced Instruction Set Computing
Rivest, Shamir, Adleman	:	RSA
SHA	:	Secure Hash Algorithm
WSN	:	Wireless Sensor Network

SYMBOLS

Elliptic Curve	:	E
Elliptic Curve Points	:	P,Q,R
Discriminant	:	Δ
Finite Field	:	GF(p)
Integers	:	$\mathbb{Z}$
Polynomials in time domain	:	a_i, b_i, c_i
Polynomials in frequency domain	:	A_i, B_i, C_i
Real Numbers	:	$\mathbb{R}$

1. INTRODUCTION

Wireless sensor networks (WSN) have many applications [Akyildiz et al. (2002), Akyildiz and Can Vuran (2010)], including battlefield reconnaissance, emergency rescue operations, surveillance and environmental monitoring, e.g. forest fires, air pollution, humidity, etc. In WSN nodes, for economic reasons and power constraints, typically cheap devices with a tiny microcontroller are used. These tiny devices are usually spread around in the field and vulnerable against potential attacks. In applications where these devices communicate sensitive data to each other, providing security and privacy is essential. Data confidentiality can easily be achieved by using symmetric-key cryptographic algorithms, however distribution of the symmetric key remains a problem, and can be overcome most effectively by utilizing public-key cryptography [Diffie and Hellman (1976)]. RSA [Rivest et al. (1978)] and elliptic curve cryptography (ECC) [Koblitz (1987), Miller (1986)] are the two most popular public-key cryptographic algorithms. RSA requires using at least a 1024 bit long key which necessitates computations over 1024 bit operands. For the same level of security, ECC requires a 160 bit key and computations over 160 bit numbers. WSN nodes usually run on battery and hence power efficiency is an important criterion for cryptographic algorithms to be practically run on these devices. Furthermore, these are tiny devices with usually minimal storage available. ECC is computationally less costly and requires less storage due to its shorter key size. Therefore, it is considered the public-key cryptographic algorithm of choice for WSNs [Walters et al. (2007), Zhou et al. (2008)].

The underlying finite field arithmetic is a determining factor in the efficiency of an ECC implementation. In ECC implementations, projective coordinates are typically preferred to avoid costly inversion operations, therefore multiplication remains as the most time-consuming arithmetic operation. Any speed-up in the multiplication operation would directly contribute to a speedup in an ECC implementation. A 160 bit multiplication, e.g. for implementing ECC, is realized by performing around 100 word addition and 100 word multiplication operations, using the classical schoolbook method, on a 16 bit microcontroller. On a constrained microcontroller, a word multiplication usually takes several times longer to execute than a word addition. This usually holds true even when

the constrained microcontroller has a built-in hardware multiplier circuitry and a related multiply instruction. For instance, on the MSP430 microcontroller, a 16 bit multiplication operation takes 14 clock cycles using its hardware multiplier, whereas a 16 bit addition operation takes only 1 clock cycle. And if an on-board hardware multiplier is not available, or it is available but not used, e.g. for power efficiency, a word multiplication is achieved through a sequence of shift and add instructions which would take up to 100 clock cycles.

In this work, we implemented ECC by performing the required finite field multiplication operation in the frequency domain, using a discrete Fourier transform (DFT) based algorithm. Multiplication in the frequency domain is potentially more efficient for implementations on constrained microcontrollers since they typically require a small number of word multiplications, in addition to a large number of simpler operations such as addition. Frequency domain arithmetic was shown to be efficient for constrained hardware implementations of ECC [Baktir et al. (2007)]. A hardware architecture for large integer multiplication in the frequency domain was proposed in [Kalach and David (2005)]. However, the authors presented only analytical results, claiming their proposed hardware multiplier architecture is more efficient than multiplication with the classical method for the operand size of 4096 bit or longer. No implementation results were provided for timing performance or circuit area. In [Baktir and Sunar (2008), Baktir (2008)], analytical results for the implementation of frequency domain inversion for ECC were provided, however no implementation results were given. Frequency domain arithmetic was also proposed for lattice-based cryptography in [Göttert et al. (2012), Pöppelmann and Güneysu (2012), Aysu et al. (2013), Güneysu et al. (2012)]. In [Chen et al. (2012)] and [Yao et al. (2010)], hardware architectures were proposed for modular multiplication in the frequency domain for RSA bitlengths, and actual implementation results were given. Finally, a frequency domain hash function was proposed in [Cheung et al. (2009)]. A method for frequency domain multiplication for ECC was proposed in [Baktir and Sunar (2006a)], however only theoretical complexity figures were given and no actual implementation results. While point multiplication with random points is more general, point multiplication with a fixed point is much faster and commonly used for digital signature generation with the ECDSA.

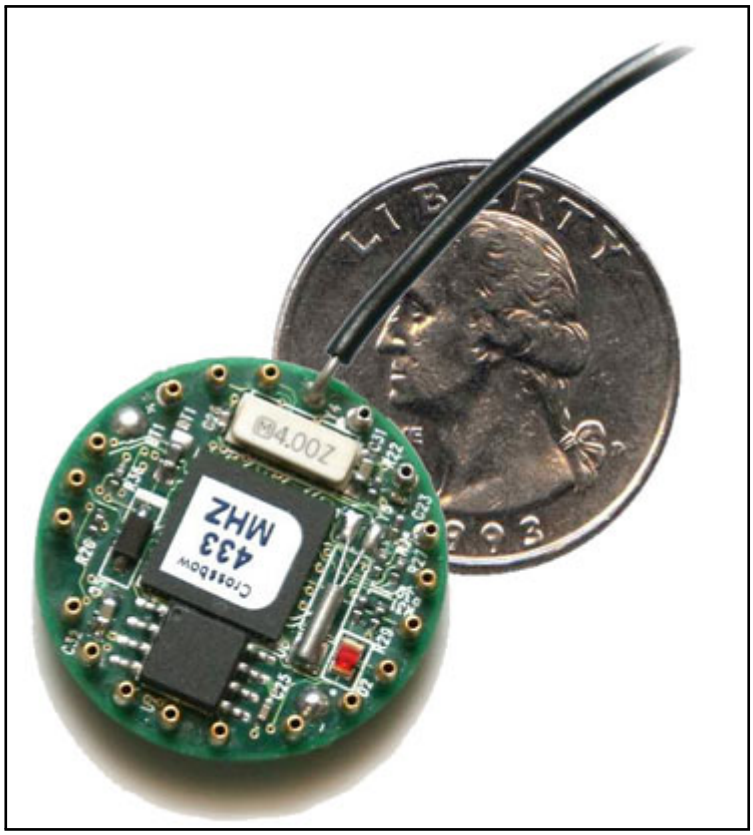
With this work, we present efficient software implementations of ECC, for point multiplication with both random and fixed points, on a constrained low-power microcontroller

without hardware multiplier support by using frequency domain multiplication. To our knowledge, this is the first work that presents a practical software implementation of ECC in the frequency domain on a constrained microcontroller without hardware multiplier support.

2. WIRELESS SENSOR NETWORK APPLICATIONS

Wireless sensor networks (WSN) are widespread and enabling technologies, applicable almost every part of people's lives. A WSN consists of nodes which are tiny, cheap and low power constrained and communicate with each other via radio transceivers [see figure 2.1]. WSN nodes usually have a microcontroller to process data. Designing a WSN for an application requires to fulfill several factors such as hardware limitations, error tolerance, cost of the devices [Akyildiz and Can Vuran (2010)]. In terms of hardware limitations, power efficiency is the main concern in a WSN. In a wide range of WSN applications, the architecture of the network topology and protocols which are used can vary according to specifications. While WSN applications expand increasingly, new protocols and algorithms are still being developed to provide higher scalability and advanced topologies.

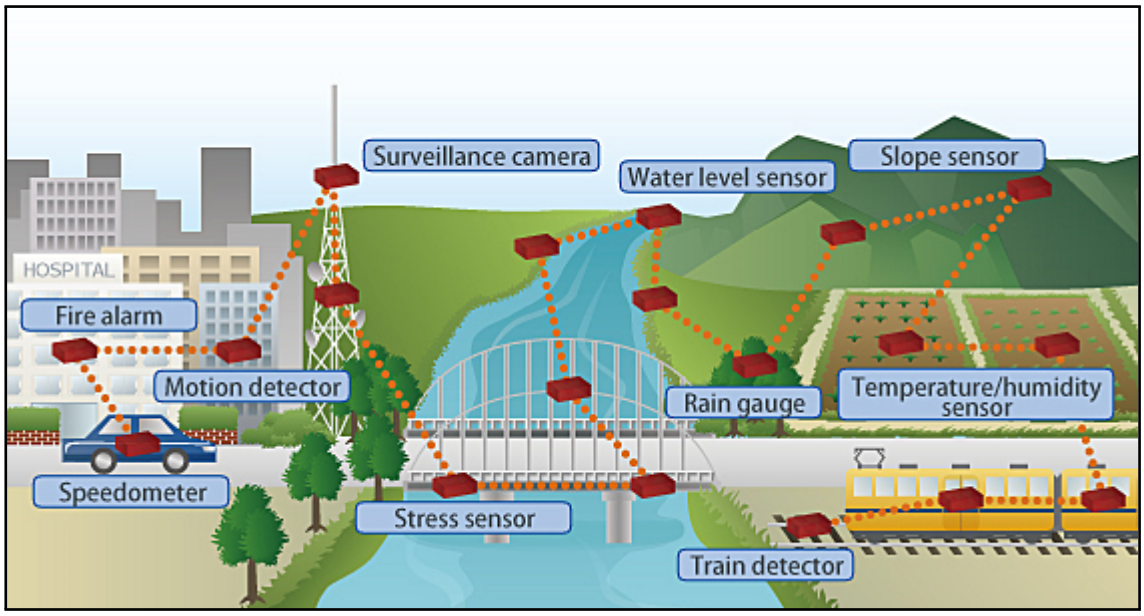
Figure 2.1: A Sensor node



Reference: www.comsys.rwth-aachen.de/teaching/ss-12/wireless-sensor-networks-lab/

Diverse sensor nodes monitor different environmental conditions such as traffic, stress, temperature, humidity, etc. depending on the application. Thus wireless sensor networks can be deployed on land, to prevent forest fire , underground, to measure seismic activities, underwater, to help fishery. WSN can be used on animals to track their movements i.e. $ZebraNet$ is a system helps biologists to monitor zebras migration in a year [Zhang et al. (2004)]. Using the wireless sensor nodes in a large scale environment complicates to refresh them, hence sensor nodes which are equipped with solar power are very advantageous if the application is power efficient. Beyond monitoring nature, in a city, wireless sensor networks benefit people life through traffic, home and office applications [see figure 2.2].

Figure 2.2: A wireless sensor network in a city



Reference:www.fujitsu.com/global/solutions/business-technology/intelligent-society/sensor-network/future/

Another promising area for WSN is health applications. Patient monitoring, alerting systems for infants, perception sensors for handicapped people, tracking implanted biomedical devices and medicine administration in hospitals are some applications. Advances in WSN technology in health applications may improve health care system remarkably and help save thousands of lives in the near future hopefully.

Military applications using sensor nodes are one of the earliest usages of WSNs. Battlefield surveillance, reconnaissance, enemy and assault detection are already used in WSN applications. WSN applications are also suitable for the battlefield area, since nodes are low cost and tiny devices which are dispensable. With the help of WSN, the protection of military personnel from hazardous situations and development of unmanned systems and vehicles for military actions are aimed.

In industry, WSNs offer many solutions in process control, inventory management, machine monitoring, chemical compounds measuring, structural health monitoring, etc. Their low cost, ease of deployment and accuracy makes WSN preferable for the applications in industry.

2.1 SECURING WSN

Since a wide variety of applications on sensitive areas such as military, health and environmental prevention, a WSN should be protected against possible threats. Although providing security in a WSN means additional computations over constrained resources, implementing cryptosystems for sensor nodes is a necessity. Efficient cryptosystems for security protocols on WSN should consider processing, storage and power limitations to be useful. Our work presents such a cryptosystem, applicable and efficient, to perform in numerous applications of WSNs.

3. ELLIPTIC CURVE CRYPTOGRAPHY

Security in a system intends the following objectives in a communication:

a. **Confidentiality**: Keeping the message from unauthorized third parties,

b. **Integrity**: Detecting whether the message has been altered or not,

c. **Authentication**: Validating the source of a message and identity of the sender,

d. **Non-repudiation**: Preventing the denial of message sender.

Cryptosystems are categorized into two: Private key (symmetric) and public key (asymmetric) cryptography. Elliptic curve cryptography (ECC) is a relatively recent public key cryptosystem. ECC was proposed by Neil Koblitz [Koblitz (1987)] and Victor Miller [Miller (1986)] in 1985. Elliptic curve protocols were accredited by standards organizations in late 90's and come have been used commonly in security protocols. Considerably shorter key size for ECC (at least 160 bit) makes it a preferable option among the other cryptosystems. Cryptosystems are based on one way functions or mathematical problems hard to solve e.g. factorization of large integers, discrete logarithm problem. ECC based on discrete logarithm problem also allows it to be applicable for other discrete logarithm protocols easily such as Diffie-Hellman key exchange. Choosing the cryptosystem for a specific application must be done considering a few criteria. Functionality, security and performance are important factors in security applications. ECC fulfills functionality as being a public key cryptosystem thus it's convenient to use in existing protocols. Security in ECC depends on the discrete logarithm problem and 160 bit key size for it provides 80 bit level of security. For instance, RSA is another popular public key cryptosystem and requires 1024 bit keys to reach the same level of security. ECC benefits from its smaller key size while the performance in an implementation highly dependent on ECC group operations (point addition and doubling operations over curves) while performing it.

3.1 CRYPTOGRAPHIC PROTOCOLS BASED ON ECC

Elliptic curve cryptography is based on discrete logarithm problem to provide security for the protocols. Elliptic curves are defined over finite fields and since the finite field is a multiplicative cyclic group, all elements of the field can be generated from the primitive elements. This allows to create elliptic curve discrete logarithm problem (ECDL) for ECC. ECDL can be defined if P is a point on elliptic curve E of order of $n-1$ over a finite field of $GF(q)$, P can be obtained by primitive element R, in k steps as shown in equation 3.1:

$$R.k \equiv P \ \ mod \ \ n \tag{3.1}$$

The ECDL problem is basically the problem of obtaining k back while R, P and n are known. It creates a one-way function due to easy forward computation P but hard reverse calculation, obtaining k. Naively, ECDL problem can be solved by trying all the possibilities for k. If the order of the field is $(n-1)$, the number of possibilities for k is $n-1$. It's assumed that it takes $n/2$ steps to guess k on average. If k is chosen large enough i.e. 160-bit, it would take a very long computation by time. However there exist more effective attacks to the discrete logarithm problem than the naive method. Shank's baby step-giant step method, Pollards-Rho method, Pohlig-Hellman algorithm and index calculus method are well known faster attacks to the discrete logarithm problem. Nevertheless choosing the elliptic curve parameters carefully (as recommended by NIST or IEEE) and obtaining key size of at least 160-bit, provides security for ECC protocols currently.

3.1.1 ECC Key Exchange

Secure communication over an insecure channel needs a shared private key between the users to encrypt their messages. Key exchange protocols allow sharing private keys with another user in a reliable way and yield a shared private key between the entities. Thus users who have the same private key can encrypt their messages and use an insecure channel to communicate. The most famous key exchange protocol, Diffie-Hellman key exchange algorithm, was proposed in 1976 [Diffie and Hellman (1976)]. For ECC there

are several protocols used for key exchange. Elliptic curve Diffie-Hellman (ECDH) key exhange, ECMQV are the best known, standardized protocols using widely.

3.1.2 ECC Signature Algorithms

Digital signature algorithms cover functionality of a cryptosystem in aspect of integrity, authentication and non-repudiation. Insecure channels i.e. web services require such security services. With the RSA algorithm presented by Ronald Rivest, Adi Shamir and Len Adleman, digital signatures can be produced [Rivest et al. (1978)]. ElGamal signature scheme, Rabin signature scheme, Schnorr signature algorithm and digital signature algorithm (DSA) are famous signature protocols. Elliptic curve digital signature algorithm (ECDSA) derived from DSA and widely used for applications.

3.1.3 ECC Encryption

Although public key cryptography encryption performs slower compared to private key cryptography, it is utilized to encrypt crucially important and relatively smaller data e.g. bank account password, personal information, etc. Using public key encryption over insecure channels while sharing session keys, password etc., private key encryption may use in same application while dealing larger data. Thus both performance and security can be provided.

3.2 FINITE FIELDS

In ECC, messages to be encrypted, or digitally signed, are represented as elements of a finite field, and cryptographic operations on these messages are realized through a sequence of finite field arithmetic operations. The finite field representation that is used influences the performance of an ECC implementation. Finite fields also known as Galois fields (GF) are cyclic abelian groups which have limited number of elements. Arithmetic operations defined over a finite field such as multiplication/divison and addition/subtraction have identity elements also called neutral elements. Arithmetic operations satisfy the

law of distribution over the finite field [Hankerson et al. (2003)]. Size of a finite field determined by total number of elements over the finite field and also called the order of the finite field. For a finite field $GF(q)$ of order q, the order should be power of a prime number i.e. $q = p^k$ where p is a prime number and $k \in \mathbb{Z}$. In ECC, implementations usually use three kinds of finite fields: prime fields, binary fields and extension fields. Since elliptic curve cryptography is performed over finite fields, the field used effects arithmetic operations considerably.

If q is the order of a finite field and equals $q = p^k$ where p is a prime and $k = 1$, the finite field called a prime field and represented as $GF(p)$, where p is called the modulus of the field and all finite field operations over the prime field need to be reduced with p.

If the order of a finite field is formed as a power of two e.g. $q = 2^m$, the field $GF(2^m)$ is called a binary field. Here, m is usually selected to be larger than 160 bits in length to provide secure ECC keys. The elements of a binary field are binary polynomials which can be of degree $m - 1$ at most. Finite field arithmetic over binary fields are achieved via bitwise operations such as bitwise exclusive or, bitwise and, logical right and left shifts. Multiplication of the binary polynomials over binary field are performed with reduction by an irreducible polynomial $f(x)$.

Extension fields $GF(p^m)$ are combination of binary and prime fields. As noticed, selecting $p = 2$ or $m = 1$ transforms the extension field to binary field or prime field. If p and m are selected considering hardware specification of the implementation, i.e. p smaller than a word size of the processor, finite field arithmetic in the field would perform optimally. Thus an optimal extension field (OEF) defined as a finite field which specially chosen for application itself and it's hardware. Reduction modulo p as prime fields and irreducible polynomial $f(x)$ as binomial are used in OEF arithmetic. $GF(p)$ is the base field for an OEF $GF(p^m)$. Base fields are subgroups of OEFs and each element of $GF(p^m)$ which are polynomials, consist of coefficients over base field $GF(p)$.

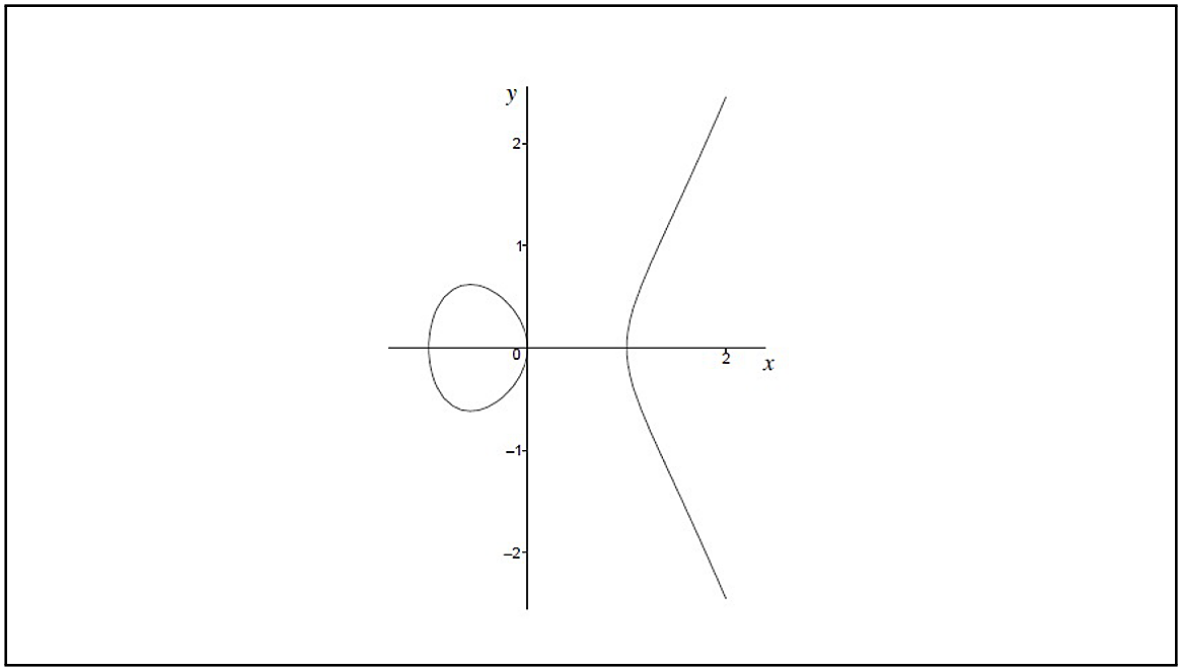
3.3 ELLIPTIC CURVE GROUP OPERATIONS

An elliptic curve over a field $GF(q)$ is defined as a set of all pairs (x,y), that satisfy the following equation :

$$y^2 = x^3 + a.x + b \tag{3.2}$$

where x,$y \in GF(q)$ and the discriminant of the equation is different from zero. Equation given in (3.2) also called $Weierstrass$ equation[Hankerson et al. (2003)]. Figure 3.1 shows an elliptic curve over real numbers $\mathbb{R}$.

Figure 3.1: $\mathbf{y^2 = x^3 - x}$

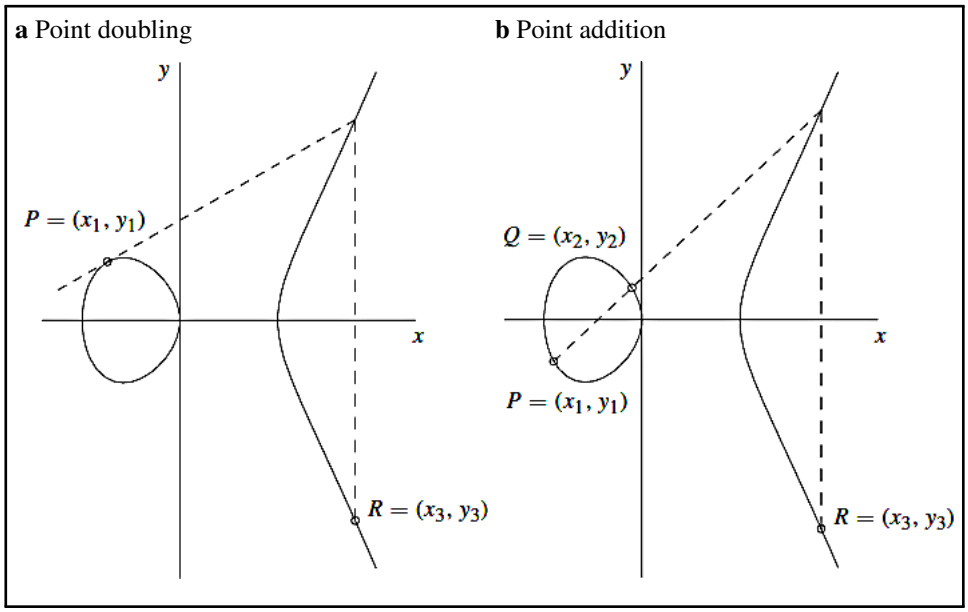


Reference: [Hankerson et al (2003)]

In elliptic curve cryptography, curves are defined over finite fields thus points are elements of $GF(q)$. Point multiplication on an elliptic curve builds the cryptosystem. To achieve point multiplication, group operations, point addition and point doubling have to be performed consecutively . If P =(x_1,y_1) and Q = (x_2,y_2) $\in GF(q)$ point doubling is

simply P+P = $2P$ = R and point addition is R= P+Q where $R \in GF(q)$ according to the group law. Geometric representation of group operations presented in figures 3.2a and 3.2b.

Figure 3.2: Group operations over an elliptic curve



Reference: [Hankerson et al (2003)]

Group law over an elliptic curve **E**: y^2=$x^3 + ax + b$ can be defined with following expressions[Hankerson et al. (2003)]:

i. Point at infinity Q is the identity for group operation. e.g P+Q = P = Q+P

ii. Negative of a point $P(x_1,y_1)$ is $-P(x_1,-y_1)$ and (P)+$(-P)$ = Q.

iii. Doubling the point $P(x_1,y_1) \Rightarrow 2P(x_3,y_3)$ can be achieved from following equations:

$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \ , \ y_3 = (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1. \tag{3.3}$$

iv. Point addition of $P(x_1,y_1)$ and $Q(x_2,y_2) \Rightarrow R(x_3,y_3)$ can be achieved from the following equations:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \ , \ \ y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1. \tag{3.4}$$

3.3.1 Point Representation on Projective Coordinates

Point operations for $Weierstrass$ equation $y^2 = x^3 + a.x + b$ were presented in section 3.3 previously. In 3.3, point addition and doubling formulas given straightforwardly for affine coordinates. In group operations there are division (or inversion) operations and divisions are the most computationally expensive operations compared to multiplication or addition/subtraction. Projective coordinate representation of the points can be used beneficially to avoid complex division/inversion operations. Affine coordinates representation of a point over the finite field $GF(q)$ defined as :

$$\text{Point } \ P(x,y) \ \text{ on } \ E : y^2 = x^3 + ax + b \tag{3.5a}$$

$$Negative \ of \ \ P(x,y) : P'(x,-y) \tag{3.5b}$$

$$Point \ at \ infinity \ \ Q : (0,1) \tag{3.5c}$$

and projective coordinates have another coordinate Z for point representation:

$$(x,y) \rightarrow (X/Z, Y/Z) \ \ , where \ \ Z \neq 0 \tag{3.6a}$$

$$\text{Point } \ P(X,Y,Z) \ \text{ on } \ E : Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{3.6b}$$

$$Negative \ of \ \ P(X,Y,Z) : P'(X,-Y,Z) \tag{3.6c}$$

$$Point \ at \ infinity \ \ Q : (0,1,0) \tag{3.6d}$$

The third coordinate Z holds the inverse values in point operations, thus inverse and direct value calculations occur separately. As a result, the product of point operations obtained also in projective coordinates and can be transform back to affine coordinates easily with

only two division by Z. Since, more division operations are eliminated while computing in projective coordinates, using projective coordinates effect performance significantly.

4. OUR ECC IMPLEMENTATION

In this work, we implement ECC over $GF(p^m)$ using the Optimal Extension Field (OEF) representation [Bailey and Paar (1998, 2001)]. A finite field of the form $GF(p^m)$ is considered secure for implementing ECC, when the field extension degree m chosen to be a prime [Hankerson et al. (2003)]. In the implementation of ECC over $GF(p^m)$, a large number of finite field arithmetic operations, such as multiplication, inversion, addition and subtraction, are performed in $GF(p^m)$. Inversion, the slowest arithmetic operation in ECC, can be avoided if projective coordinates are used. Hence, if we do not count inversion, multiplication remains the most costly arithmetic operation in ECC and any speedup in the multiplication operation would directly improve the speed of an ECC implementation. Our ECC implementation performs finite field operations in the frequency domain to speed up modular multiplication.

4.1 FREQUENCY DOMAIN REPRESENTATION AND FINITE FIELD ARITHMETIC

Elements of $GF(p^m)$ are represented by polynomials of degree $m-1$, with coefficients in $GF(p)$. E.g., $A \in GF(p^m)$ is represented as

$$A = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \tag{4.1}$$

where $a_i \in GF(p)$. Addition/subtraction of A and B, both elements of $GF(p^m)$, is performed in only linear time by pairwise addition/subtraction of polynomial coefficients, i.e equation 4.2.

$$C = A \pm B = \sum_{i=0}^{m-1} (a_i \pm b_i) x^i \bmod p. \tag{4.2}$$

Multiplication of A and B, both elements of $GF(p^m)$, is performed by first computing the polynomial product

$$C' = A \cdot B = \sum_{i=0}^{2m-2} c'_i x^i \tag{4.3}$$

and then the modular reduction

$$C = C' \bmod P(x) \tag{4.4}$$

Using OEFs, the modular reduction $C = C' \bmod P(x)$ has only linear, i.e. $O(m)$, complexity. However, the polynomial multiplication $C' = A \cdot B = \sum_{i=0}^{2m-2} c'_i x^i$ has quadratic, i.e. $O(m^2)$, complexity using the classical schoolbook method.

We perform finite field arithmetic operations, such as addition, subtraction and multiplication, all in the frequency domain. To facilitate finite field arithmetic in the frequency domain, one needs to first convert finite field elements to their frequency domain representations. Using the Number Theoretic Transform (NTT) [Pollard (1971)], the frequency domain representation of an element in $GF(p^m)$ can be computed as following equations:

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \tag{4.5}$$

in $GF(p^m)$, is represented with the following sequence after appending $d - m$ zeros to the end, where $d \geq 2m - 1$ is length of the NTT.

$$(a) = (a_0, a_1, a_2, \ldots, a_{m-1}, 0, 0, \ldots, 0) \tag{4.6}$$

The frequency domain representation (A) of the time domain sequence (a) is obtained with the NTT computation:

$$A_j = \sum_{i=0}^{d-1} a_i r^{ij}, \ 0 \leq j \leq d-1 \tag{4.7}$$

The time domain representation (a) can be obtained similarly, from the frequency domain representation (A), by with the inverse NTT computation:

$$a_i = \frac{1}{d} \cdot \sum_{j=0}^{d-1} A_j r^{-ij} \ \ , \ 0 \leq i \leq d-1 \tag{4.8}$$

In an NTT computation of length d, a d^{th} primitive root of unity, denoted as r, is needed. We use $r = -2 \in GF(p)$ in this work.

4.2 FINITE FIELD MULTIPLICATION IN THE FREQUENCY DOMAIN

We perform Montgomery multiplication in $GF(p^m)$, in the frequency domain, using the DFT modular multiplication algorithm [Baktir and Sunar (2006b)]. The discrete Fourier transform (DFT) [Tolimieri et al. (1989), Burrus and Parks (1985)] operation over a finite field, as applied in the DFT modular multiplication algorithm, is the same operation as the NTT. The inputs to the DFT modular multiplication algorithm are two elements in $GF(p^m)$, represented in the frequency domain, and the output is the frequency domain representation of their Montgomery product [Montgomery (1985)].

Algorithm 4.1 DFT Modular Multiplication

Input: $(A) \equiv a(x) \in GF(p^m), (B) \equiv b(x) \in GF(p^m), (P') \equiv P(x)/P(0) \bmod p$.

Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod P(x)$

1: **for** $i = 0$ to $d - 1$ **do**
2: $\quad C_i \leftarrow A_i \cdot B_i \bmod p$
3: **for** $j = 0$ to $m - 2$ **do**
4: $\quad S \leftarrow 0$
5: $\quad$ **for** $i = 0$ to $d - 1$ **do**
6: $\quad\quad S \leftarrow S + C_i \bmod p$
7: $\quad S \leftarrow -S/d \bmod p$
8: $\quad$ **for** $i = 0$ to $d - 1$ **do**
9: $\quad\quad C_i \leftarrow (C_i + P'_i \cdot S) \cdot X_i^{-1} \bmod p$
10: Return (C)

Reference: [Baktir and Sunar (2006b)]

We give Algorithm 4.2 which is an optimized form of the DFT Modular Multiplication algorithm [Baktir and Sunar (2006b)]. Algorithm 4.2 facilitates the Fast Fourier Transform (FFT) to speed up Algorithm 4.2. All the parameters here are in their frequency domain representations and $GF(p^m)$ multiplication is realized in the frequency domain. The parameters used in the algorithm are the input operands $a(x), b(x) \in GF(p^m)$ and the Montgomery product is:

$$c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod P(x) \in GF(p^m) \tag{4.9}$$

The time domain sequences for these parameters are $(a), (b)$ and (c), and the corresponding frequency domain sequences are $(A), (B)$ and (C). In the algorithm, a Mersenne prime, of the form $p = 2^n - 1$, is used as the field characteristic and the field extension degree is selected as $m = n$. In this setting, for obtaining the frequency domain sequences (A) and (B), the NTT with length $d = 2m$ and the $r = -2 \in GF(p)$ is used. This specialized NTT computation, modulo a Mersenne prime, is called the *Mersenne transform* [Rader (1972)]. Note that in the optimized DFT modular multiplication al-

gorithm, only a linear number of $GF(p)$ multiplications are performed, in addition to a quadratic number of simple addition/subtraction and bitwise rotation operations.

Algorithm 4.2 DFT Modular Multiplication algorithm optimized for $p = 2^n - 1$, m odd, $m = n$ and $P(x) = x^m - 2$

Input: $(A) \equiv a(x) \in GF(p^m)$, $(B) \equiv b(x) \in GF(p^m)$

Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod P(x) \in GF(p^m)$

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

Reference: [Baktir et al. (2007)]

4.3 IMPLEMENTING ECC ON MSP430

The read/write instructions from/to the memory has a significant impact on the efficiency of arithmetic operations on MSP430. MSP430 has a RISC architecture with only 27 instructions and 7 addressing modes, and the adressing mode that is used determines the number of clock cycles for an instruction execution. Instruction execution takes less clock cycles using the register addressing mode, however there are only 12 general purpose registers are available. We used these 12 registers for storing our operands as much as possible to make our implementations faster. And to eliminate extra clock cycles, we stored frequently used constants in these registers.

For implementing the ECC, we use the finite field $GF(p^m)$ with the Mersenne prime field characteristic $p = 2^{13} - 1$ and the extension degree $m = 13$. We implement ECC in the frequency domain, and use the NTT length $d = 26$ and the d^{th} primitive root of unity $r = -2 \in GF(2^{13} - 1)$. Note that, for $r = -2$ and $p = 2^{13} - 1$, multiplications in $GF(p)$ with powers of r, as required in Algorithm 4.1, turn into multiplications with powers of -2 (see Algorithm 4.2), and can be achieved with only a simple bitwise rotation, in addition to a negation if r has an odd power. In our ECC implementations, we use the projective coordinates, therefore inversions are avoided and the only arithmetic operations required are addition, subtraction and multiplication. In the frequency domain, one can achieve addition and subtraction very easily, by pairwise addition/subtraction of frequency domain coefficients, which are elements of $GF(2^{13} - 1)$. We performed multiplication in $GF((2^{13} - 1)^{13})$ using the DFT modular multiplication algorithm. All operations needed for implementing the DFT modular multiplication algorithm are modular addition, subtraction, multiplication and rotation operations in $GF(2^{13} - 1)$. Hence efficient implementation of $GF(2^{13} - 1)$ arithmetic is critical. On the MSP430 microcontroller, we implemented modular multiplication in $GF(2^{13} - 1)$ both with and without hardware multiplier support.

4.3.1 Addition and Subtraction in $GF(p)$

In our ECC implementations, $GF(2^{13} - 1)$ addition/subtraction is the most commonly used operation. Note that modular subtraction in $GF(2^{13} - 1)$ is the similar to addition, where an additional "xor" instruction is applied to flip the bits of the sum, and costs 1 extra clock cycle than addition. We allocate two registers to store constant values for masking and checking the Most Significant Bit (MSB) of operands during these operations. We realize modular addition and subtraction in $GF(2^{13} - 1)$ in 4 and 5 clock cycles, respectively, using the following Assembly codes.

Assembly Code 1: Modular addition

```
ADD R15 , R14
BIT R11 , R14 ; (R11=0x2000)
ADC R14
AND R13 , R14 ; (R13=0x1FFF)
```

Reference: Utku Gülen, Selçuk Baktır (2014)

Assembly Code 2: Modular Subtraction

```
XOR R13 , R14 ; (R13=0x1FFF)
ADD R15 , R14
BIT R11 , R14
ADC R14
AND R13 , R14
```

Reference: Utku Gülen, Selçuk Baktır (2014)

4.3.2 Modular Multiplication in $GF(p)$

It is crucial to perform modular multiplication in $GF(2^{13}-1)$ efficiently, since it directly influences the efficiency of multiplication in $GF((2^{13}-1)^{13})$. We consider two approaches for modular multiplication to satisfy possible constraints for power efficiency or timing performance. There is a 16-bit hardware multiplier available on most MSP430 microcontrollers. This hardware multiplier can be used for achieving modular multiplication in less clock cycles, with the drawback of higher power consumption.

There is no multiplication instruction in the MSP430 instruction set, in spite of the available 16-bit hardware multiplier. The cost of 16-bit multiplication using the hardware multiplier is just the cost of the write/read operations to/from the special function registers for the hardware multiplier. Utilizing the hardware multiplier, a 16x16-bit multiplication can be achieved in 14 clock cycles. Our implementation of modular multiplication in $GF(2^{13}-1)$ using the hardware multiplier takes 26 clock cycles. We perform modular

reduction after reading the 26-bit product from the hardware multiplier, as described with the following Assembly routine.

Assembly Code 3: Modular multiplication in $GF(2^{13}-1)$ using hardware multiplier support

```
MOV R14 , &MPY ; LOAD OPERANDS TO REGISTERS
MOV R15 , &OP2
MOV &RESLO , R14 ; GET BYTES OF PRODUCT
MOV &RESHI , R15
MOV R14 , R10 ; PERFORM MODULAR REDUCTION
RLC R10
RLC R15
RLC R10
RLC R15
RLC R10
RLC R15
AND R13 , R14 ; (R13 = 0X1FFF)
ADD R15 , R14
BIT R11 , R14 ; (R11 = 0X2000)
ADC R14
AND R13 , R14
```

Reference: Utku Gülen, Selçuk Baktır (2014)

We achieved multiplication in $GF(2^{13}-1)$ both with and without hardware multiplier support. In our first approach, we made used hardware multiplier support. And in our second approach, instead of using hardware multiplier support, we achieved the required 13-bit multiplication operation by performing a sequence of additions. In this approach, all the bits of one operand are scanned through. If the scanned bit is 1, the other operand is added to the partial product, and then the partial product is shifted. We postponed the modular reduction operation until after the 26-bit integer product is computed. Our implementation of multiplication in $GF(2^{13}-1)$ without hardware multiplier support takes 90 clock cycles on average, depending on the Hamming weight of the multiplier. We present the second approach with the following assembly routine.

Assembly Code 4: $GF(2^{13}-1)$ **multiplication without hardware multiplier support**

```
       CLR R4
       CLR R5
       MOV @R12+ , R6
       MOV @R13+ , R7
       RLA R6
       RLA R6
       RLA R6
       RLA R6
       JNC DONEAB1
       ADD R7 , R4
DONEAB1: RLA R4
       RLA R6
       JNC DONE1AB1
       ADD R7 , R4
DONE1AB1: RLA R4
       RLA R6
       JNC DONE2AB1
       ADD R7 , R4
DONE2AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE3AB1
       ADD R7 , R4
       ADC R5
DONE3AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE4AB1
       ADD R7 , R4
       ADC R5
DONE4AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE5AB1
       ADD R7 , R4
       ADC R5
DONE5AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE6AB1
       ADD R7 , R4
       ADC R5
DONE6AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE7AB1
       ADD R7 , R4
       ADC R5
DONE7AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE8AB1
       ADD R7 , R4
       ADC R5
DONE8AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE9AB1
       ADD R7 , R4
       ADC R5
DONE9AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE10AB1
       ADD R7 , R4
       ADC R5
DONE10AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE11AB1
       ADD R7 , R4
       ADC R5
DONE11AB1: RLA R4
       RLC R5
       RLA R6
       JNC DONE12AB1
       ADD R7 , R4
       ADC R5
```

```
DONE12AB1: MOV R4 , R6
       RLC R6
       RLC R5
       RLC R6
       RLC R5
       RLC R6
       RLC R5
       AND R15 , R4
       ADD R5 , R4
       BIT R8 , R4
       ADC R4
       AND R15 , R4
       MOV R4 , 2(R14)
```

Reference: Utku Gülen, Selçuk Baktır (2014)

4.3.3 Bitwise Rotations in $GF(p)$

Since a large number of bitwise rotations in $GF(2^{13} - 1)$ are performed in the DFT modular multiplication algorithm, we optimized this operation as much as possible in our implementations. We used the 1-bit **arithmetic shift** and **shift with carry** instructions in the MSP430 instruction set, which execute both in a single clock cycle, as often as possible. We also used the **set bit**, **test bit** and **swap byte** instructions to realize the rotation operations in the minimal number of clock cycles. For rotations by different numbers of bits, we pursued various strategies to reduce the number of required clock cycles.

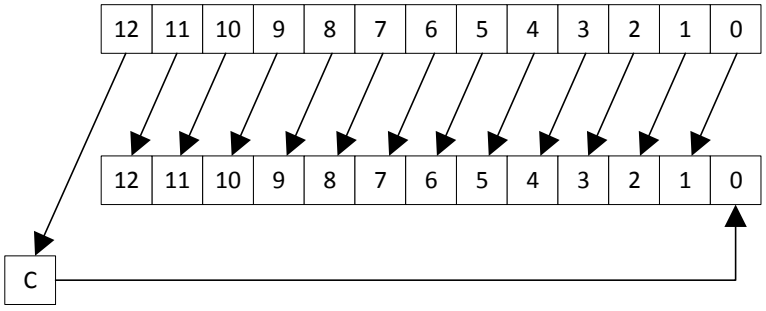
We carry out 1-bit left-rotation by checking the MSB of the operand and then shifting it to the right through carry. A masking operation is required to complete this operation, therefore a 1-bit left-rotation, as shown in Figure 4.1 and given in the following Assembly code, is accomplished in a total number of 3 clock cycles.

Assembly Code 5: 1-bit left-rotation

```
BIT R14 , R10 ;(R14 = 0X1000)
RLC R10
AND R13 , R10 ;(R13 = 0X1FFF)
```

Reference: Utku Gülen, Selçuk Baktır (2014)

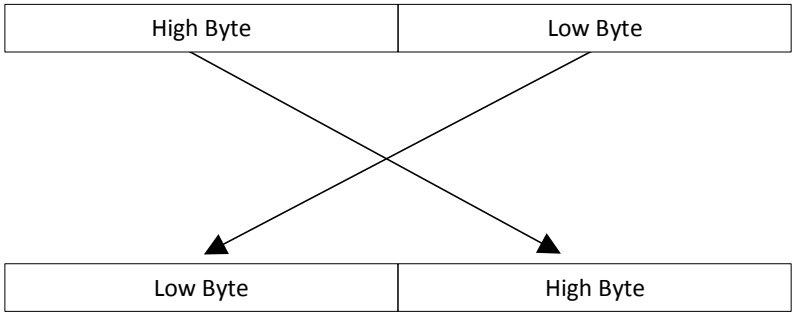
Figure 4.1: 1-bit left-rotation



Reference: Utku Gülen, Selçuk Baktır (2014)

We perform $2, 3$ and 4-bit left-rotations by repeated 1-bit left-rotations.For rotations by more than 4 bits, we utilize the **swap byte** instruction. We use the **swap byte** instruction to exchange the low and high bytes of a 13-bit operand, as shown in Figure 4.2, and thus simplify the rotation.

Figure 4.2: Swap byte instruction



Reference: Utku Gülen, Selçuk Baktır (2014)

We optimize the $5, 6, 7, 8, 9$ and 10-bit left-rotation operations through the use of the **swap byte** instruction and **mask/store** operations. We handle rotations by different numbers of bits slightly differently, i.e. by using different orders of codes, although we use a similar strategy to achieve the best cycle time in each case. Exemplarily, the Assembly code for the 6-bit left-rotation operation, which takes 9 clock cycles, is given as follows.

Assembly Code 6: 6-bit left-rotation

```
MOV R15 , R10 ;STORE OPERAND IN R10
AND R7 , R10 ;(R7 = 0X007F)
SWPB R10
RRA R10
RRA R10
RLA R15
SWPB R15
AND R6 , R15 ;(R6 = 0X003F)
BIS R10 , R15
```

Reference: Utku Gülen, Selçuk Baktır (2014)

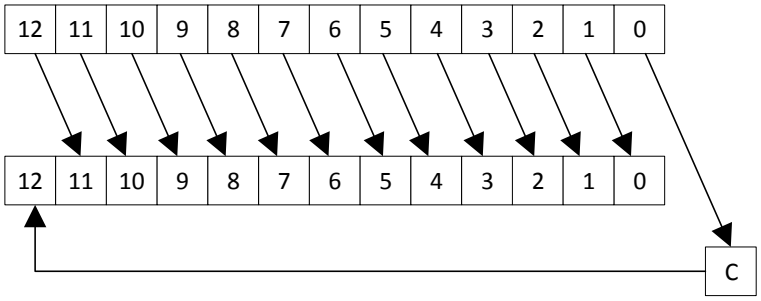
We achieve the 12-bit left-rotation operation through a 1-bit right-rotation, as given in Figure 4.3 and implemented with the following Assembly code. The required 1-bit right-rotation operation is done by shifting the Least Significant Bit (LSB) to the carry flag and setting the 13^{th} bit of the operand depending on the value of the carry. We achieve a 11-bit left-rotation by performing two 12-bit left-rotations.

Assembly Code 7: 12-bit left-rotation

```
    RRA R10
    JNC DONE
    BIS R14 , R10; (R14 = 0X1000)
DONE:
```

Reference: Utku Gülen, Selçuk Baktır (2014)

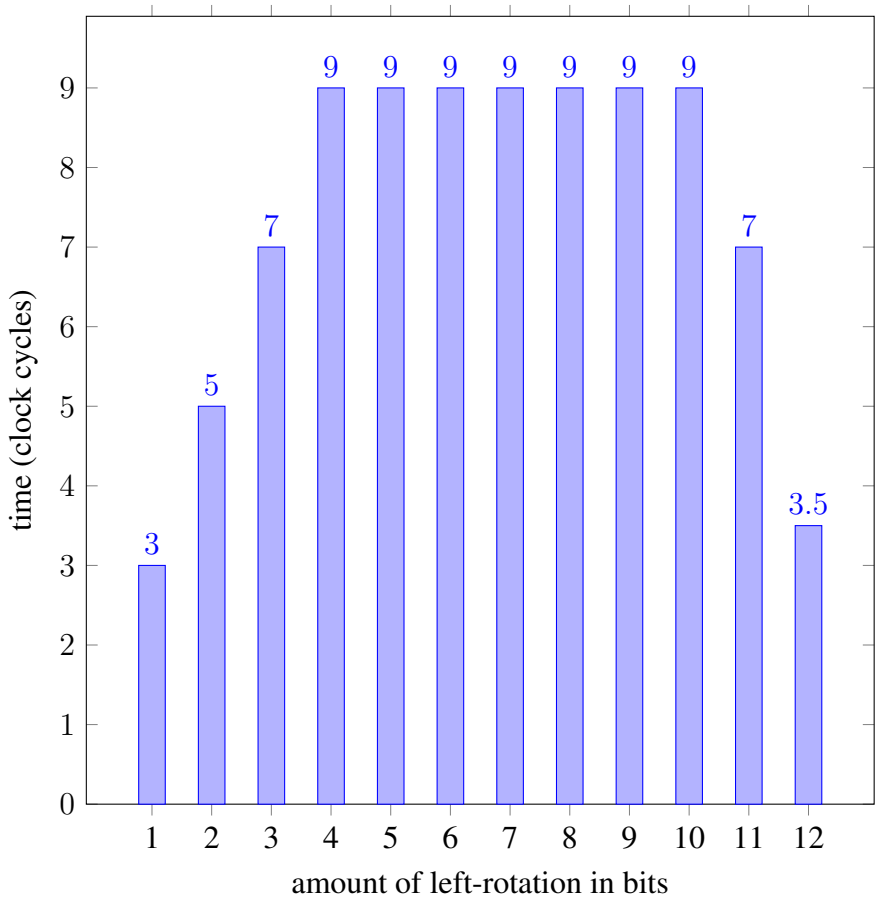
Figure 4.3: 12-bit left-rotation



Reference: Utku Gülen, Selçuk Baktır (2014)

In the DFT modular multiplication algorithm, the rotation operations, by differing numbers of bits, are performed an equal number of times. Our implementation of the rotation operation takes 7.4 clock cycles on average, for rotations by different numbers of bits. In figure 4.4, we give the required number of clock cycles individually for the rotation operation by different numbers of bits.

Figure 4.4: Execution times for bitwise left-rotation operations on MSP430



Reference: Utku Gülen, Selçuk Baktır (2014)

5. IMPLEMENTATION RESULTS AND COMPARISON

We implemented the ECC scalar point multiplication operation, for both random and fixed points, on MSP430F149, Texas Instrument's low-power 1 series 16 bit microcontroller. We used IAR Embedded Workbench as our development tool and obtained accurate clock cycle counts using it's debugger. For the elliptic curve point addition and doubling operations, we used Edwards curve formulas in projective coordinates [Bernstein and Lange (2007)]. Edwards curves [Edwards (n.d.)] are a new form of elliptic curves proposed for ECC. In this work, we used an Edwards curve, defined by the equation

$$x^2 + y^2 = c^2(1 + dx^2y^2) \tag{5.1}$$

with $c = 1$, d random and $dc^4 \neq 1$, over the prime field $GF((2^{13} - 1)^{13})$.

5.1 IMPLEMENTATION RESULTS FOR ECC SCALAR POINT MULTIPLICATION WITH A RANDOM POINT

For the ECC scalar point multiplication operation with a random point, we utilized the NAF method with a window size of 4. In Table 5.1, we present our performance result and compare it with related work.

Table 5.1: Timings for ECC scalar point multiplication with a random point on MSP430

Platform	**Field**	**Method**	**Time** (s)
MSP430 @8MHz (this work)	$GF((2^{13} - 1)^{13})$	4NAF	1.55
MSP430 @8MHz [Wang et al. (2006)]	F_{p160}	4NAF	3.51
MSP430 @8MHz [Araz and Qi (2006)]	$GF(2^{163})$	not specified	32.5

Reference: Utku Gülen, Selçuk Baktır (2014)

As seen in Table 5.1, our timing figures for 169-bit ECC scalar multiplication with a random point is only 1.55 s on MSP430 running at 8MHz. Wang et al.'s implementation of the smaller sized 160-bit point multiplication, on the same microcontroller, has a timing performance of 3.13 s [Wang et al. (2006)]. Finally, again on the same platform, an implementation by Araz et al. for the smaller sized 163-bit point multiplication takes 32.5 s [Araz and Qi (2006)].

5.2 IMPLEMENTATION RESULTS FOR ECC SCALAR POINT MULTIPLICATION WITH A FIXED POINT

For the ECC scalar point multiplication operation with a fixed point, we utilized the Comb method with a window size of 4. In Table 5.2, we present our performance result and compare it with related work.

Table 5.2: Timings for point multiplication with a fixed point on MSP430

Platform	**Field**	**Method**	**Time** (s)
MSP430 @8MHz (this work)	$GF((2^{13}-1)^{13})$	4Comb	0.77
MSP430 @8MHz [Liu and Ning (2008)]	F_{p160}	sliding window w/4	1.58
MSP430 @8MHz [Wang et al. (2006)]	F_{p160}	sliding window w/4	3.13

Reference: Utku Gülen, Selçuk Baktır (2014)

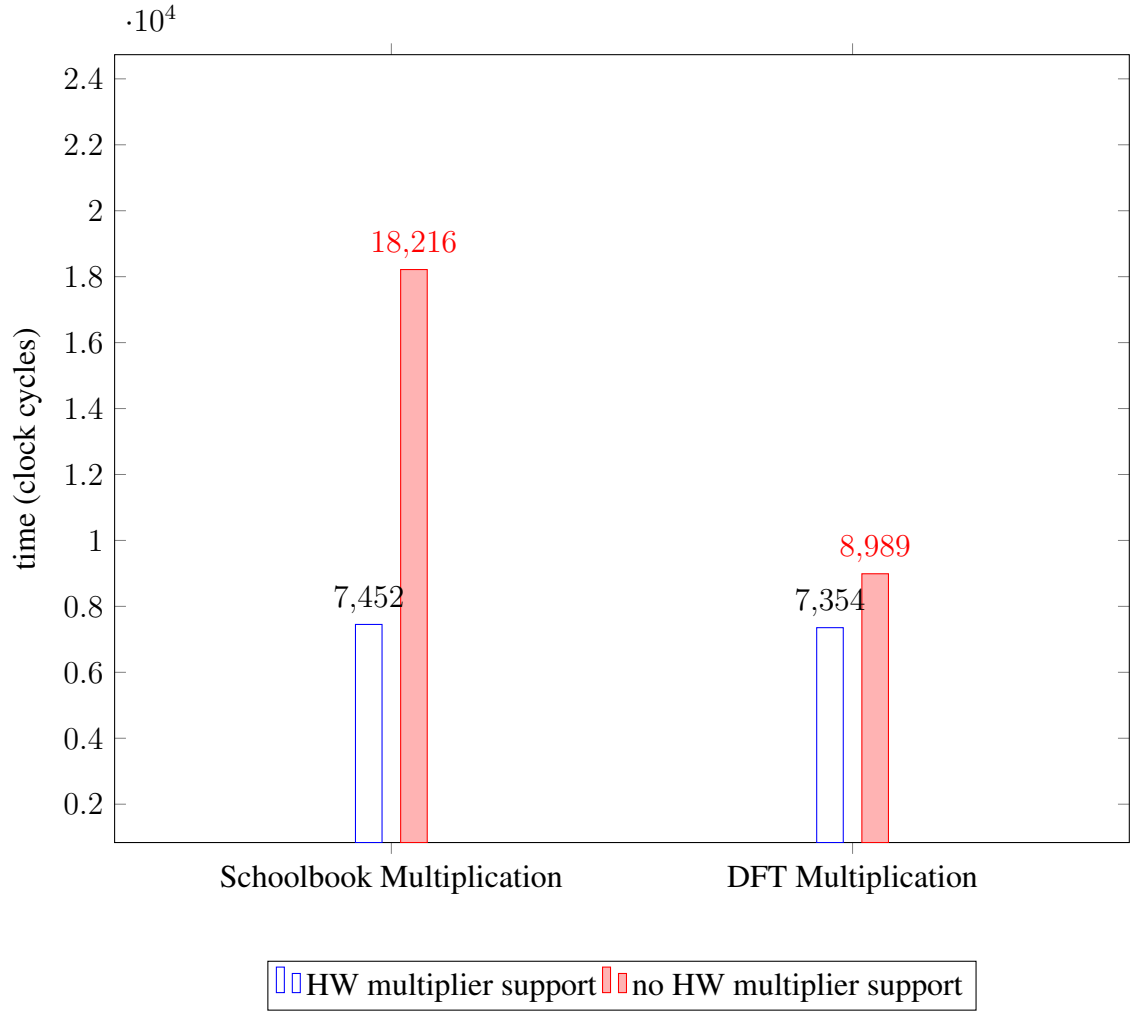
As seen in Table 5.2, our timing figures for 169-bit ECC scalar multiplication with a fixed point is only 0.77 s on MSP430 running at 8MHz. Liu et al.'s implementation of the smaller sized 160-bit point multiplication, on the same microcontroller, has a timing performance of around 1.58 s [Liu and Ning (2008)] (this is the timing for ECDSA, the timing for scalar multiplication is not given explicitly in the paper). Finally, again on the same platform, an implementation by Wang et al. for the smaller sized 160 bit point multiplication takes 3.13 s [Wang et al. (2006)].

5.3 IMPLEMENTATION RESULTS FOR ECC WITHOUT HARDWARE MULTIPLIER SUPPORT

Large integer multiplication requires executing a significantly smaller number of word multiplications in the frequency domain than with the classical schoolbook method in the time domain. A word multiplication operation is realized in multiple clock cylex on the MSP430 microcontroller, while other operations such as addition can be achieved in only a single clock cycle. Hence, on a constrained microcontroller such as MSP430, frequency domain multiplication can be more efficient than time domain multiplication for large operand sizes, e.g. 160 bits or longer, as required for ECC. This advantage of frequency domain multiplication would become even more obvious for low-power implementations on constrained microcontrollers without hardware multiplier support. For instance, if one prefers to realize a low-power ECC implementation, as would be required for many WSN applications, they may pick a simpler MSP430 microcontroller without an onboard hardware multiplier. Even when the microcontroller at hand has an onboard hardware multiplier, in order to satisfy power constraints, it may be wise not to use it for certain applications. In such scenarios, a word multiplication operation would be implemented as a sequence of additions and take an even larger number of clock cycles. On the MSP430 microcontroller, a 16 bit word multiplication takes 14 clock cycles with the hardware multiplier and up to 100 clock cycles without it.

In Figure 5.1, we present performance results for multiplication in $GF((2^{13}-1)^{13})$ using the DFT modular multiplication algorithm and the schoolbook method, both with and without hardware multiplier support. As seen in figure 5.1, frequency domain multiplication is only around 1.3% faster than the schoolbook method. However, more importantly, the performance of frequency domain multiplication slows down by only 18% if no hardware multiplier is available. Whereas, the lack of a hardware multiplier unit slows down the schoolbook multiplication by more than 59%. Note that frequency domain multiplication performs more than 2.02 times faster than the schoolbook method on an MSP430 microcontroller without hardware multiplier support, and thus would be more preferable for constrained low power implementations.

Figure 5.1: Execution times for Multiplication in $\mathrm{GF}((2^{13}-1)^{13})$, using the schoolbook and DFT modular multiplication methods, on MSP430

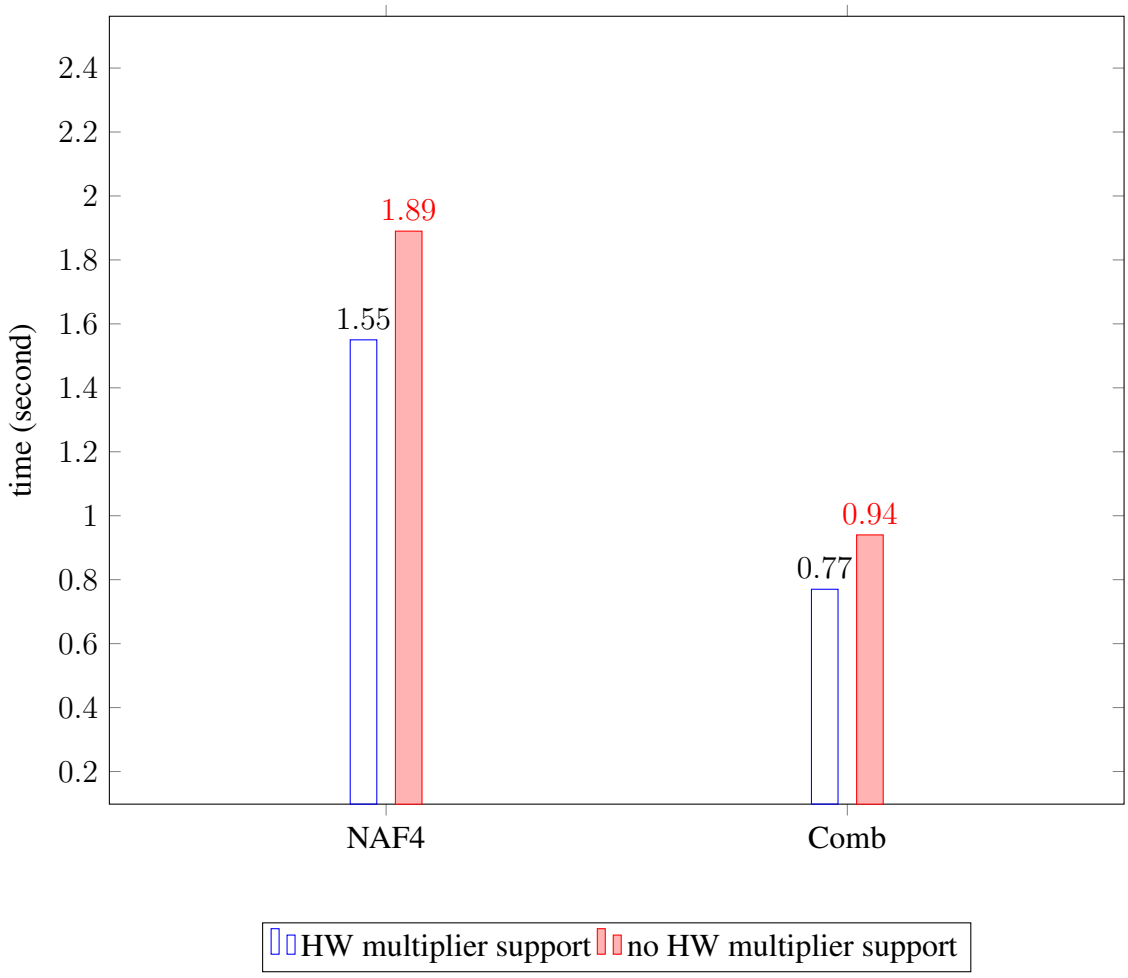


Reference: Utku Gülen, Selçuk Baktır (2014)

While others' implementations of ECC on the MSP430 microcontroller, listed in Tables 5.1 and 5.2, all utilize the onboard hardware multiplier, we propose our performance results also without hardware multiplier support. We implemented the ECC scalar point multiplication operation on MSP430 both with and without hardware multiplier support. Our timings for both cases are satisfactory with similar or better results compared to previous implementations on the same microcontroller. It is particularly interesting to see that, with the utilization of frequency domain arithmetic, the performance of ECC does not deteriorate significantly if an onboard hardware multiplier is not available on the mi-

crocontroller (see Figure 5.2). Our performance figures for point multiplication, without using hardware multiplier support, are 1.89 s and 0.94 s for point multiplication with random and fixed points, respectively. Surprisingly, our performance figures even without hardware multiplier support is comparable to or better than others' results with hardware multiplier support. Our implementations without hardware multiplier support are only 21.9% and 22.1% slower than with hardware multiplier support, for random and fixed point multiplication, respectively. Hence, our proposed method would be well-suited for constrained low-power implementations of ECC on constrained microcontrollers without an on-board hardware multiplier.

Figure 5.2: Performance comparisons for point multiplication using NAF4 and Comb methods for random and fixed points, respectively (with and without hardware multiplier support)



Reference: Utku Gülen, Selçuk Baktır (2014)

6. CONCLUSION

We implemented ECC in the frequency domain on the constrained MSP430 microcontroller which is used widely in constrained environments such as WSNs. We presented our performance figures for MSP430 both with and without hardware multiplier support. Our timings for both cases are satisfactory with similar or better results compared to previous implementations on the same platform. It was particularly interesting to see that, with the utilization of frequency domain arithmetic, the performance of ECC does not deteriorate significantly on a microcontroller without hardware multiplier support. This is due to fact that frequency domain multiplication requires only a small number of word multiplications. Hence, for power efficient implementations of ECC on constrained devices, such as WSN nodes, using frequency domain arithmetic would be desirable. With this work, we present the first ever practical software implementation of ECC in the frequency domain on a constrained microcontroller with no hardware multiplier support, with promising results for low-power applications of ECC in WSNs.

Our Main Contributions:

a. We realized in the frequency domain an implementation of the elliptic curve scalar point multiplication operation with a fixed point, as used in ECDSA [*FIPS PUB 186-4: Digital Signature Standard (DSS)* (2013)], as well as with a random point, on the constrained MSP430 microcontroller. This work presents the first ever such implementation in the frequency domain and provides similar or better performance compared to previous implementations on the same platform.

b. We presented the first ever low-power implementations of ECC scalar point multiplication in the frequency domain, for both random and fixed points, on the constrained MSP430 microcontroller *without hardware multiplier support*. In spite of not utilizing hardware multiplier support, our implementations exhibit similar or better performance compared to previous implementations on the same platform with hardware multiplier support.

REFERENCES

Books

Akyildiz, I. and Can Vuran, M.: 2010, *Wireless Sensor Networks*. cited By (since 1996)181.

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

Hankerson, D., Menezes, A. J. and Vanstone, S.: 2003, *Guide to Elliptic Curve Cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

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

Periodicals

Akyildiz, I., Su, W., Sankarasubramaniam, Y. and Cayirci, E.: 2002, Wireless sensor networks: a survey, *Computer Networks* **38**(4), 393 – 422.

Araz, O. and Qi, H.: 2006, Load-balanced key establishment methodologies in wireless sensor networks, *International Journal of Security and Networks* **1**(3), 158–166.

Aysu, A., Patterson, C. and Schaumont, P.: 2013, Low-cost and area-efficient fpga implementations of lattice-based cryptography., *HOST*, IEEE, pp. 81–86.

Bailey, D. V. and Paar, C.: 1998, Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, *in* H. Krawczyk (ed.), *Advances in Cryptology — CRYPTO '98*, Vol. LNCS 1462, Springer-Verlag, Berlin, Germany, pp. 472–485.

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

Baktir, S., Kumar, S., Paar, C. and Sunar, B.: 2007, A state-of-the-art elliptic curve cryptographic processor operating in the frequency domain, *Mobile Networks and Applications* **12**(4), 259–270.

Baktir, S. and Sunar, B.: 2006a, Achieving efficient polynomial multiplication in fermat fields using the fast fourier transform, *Proceedings of the 44th Annual Southeast Regional Conference*, ACM-SE 44, ACM, pp. 549–554.

Baktir, S. and Sunar, B.: 2006b, Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography, *Proceedings of the 21th International Symposium on Computer and Information*, Vol. 4263 of *Lecture Notes in Computer Science (LNCS)*, Springer, Heidelberg, pp. 991–1001.

Baktir, S. and Sunar, B.: 2008, Optimal extension field inversion in the frequency domain, *in* J. Gathen, J. L. Imana and c. K. Koç (eds), *Arithmetic of Finite Fields*, Vol. 5130 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 47–61.

Bernstein, D. J. and Lange, T.: 2007, Faster addition and doubling on elliptic curves, *Advances in cryptology–ASIACRYPT 2007*, Springer, pp. 29–50.

Chen, D., Yao, G., Koç, C. and Cheung, R.: 2012, Low complexity and hardware-friendly spectral modular multiplication, *Field-Programmable Technology (FPT), 2012 International Conference on*, pp. 368–375.

Cheung, R. C. C., Çetin Kaya Koç and Villasenor, J. D.: 2009, A high-performance hardware architecture for spectral hash algorithm, *ASAP*, pp. 215–218.

Diffie, W. and Hellman, M.: 1976, New directions in cryptography, *Information Theory, IEEE Transactions on* **22**(6), 644–654.

Edwards, H. M.: n.d., A normal form for elliptic curves, *Bulletin of the American Mathematical Society*, pp. 393–422.

FIPS PUB 186-4: Digital Signature Standard (DSS): 2013, *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION* .

Göttert, N., Feller, T., Schneider, M., Buchmann, J. and Huss, S.: 2012, On the design of hardware building blocks for modern lattice-based encryption schemes, *in* E. Prouff and P. Schaumont (eds), *Cryptographic Hardware and Embedded Systems (CHES 2012)*, Vol. 7428 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 512–529.

Güneysu, T., Lyubashevsky, V. and Pöppelmann, T.: 2012, Practical lattice-based cryptography: A signature scheme for embedded systems, *in* E. Prouff and P. Schaumont (eds), *Cryptographic Hardware and Embedded Systems (CHES 2012)*, Vol. 7428 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 530–547.

Kalach, K. and David, J. P.: 2005, Hardware implementation of large number multiplication by FFT with modular arithmetic., *Proceedings of the 3rd International IEEE-NEWCAS Conference*, IEEE, pp. 267– 270.

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

Liu, A. and Ning, P.: 2008, Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks, *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, IPSN '08, IEEE Computer Society, Washington, DC, USA, pp. 245–256.

Miller, V.: 1986, Uses of Elliptic Curves in Cryptography, *in* H. C. Williams (ed.), *Advances in Cryptology — CRYPTO '85*, Vol. LNCS 218, Springer-Verlag, Berlin, Germany, pp. 417–426.

Montgomery, P. L.: 1985, Modular multiplication without trial division, *Mathematics of Computation* **44**(170), 519–521.

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

Pöppelmann, T. and Güneysu, T.: 2012, Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware, *in* A. Hevia and G. Neven (eds), *LATINCRYPT 2012*, Vol. 7533 of *Lecture Notes in Computer Science*, pp. 139–158.

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

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

Walters, J. P., Liang, Z., Shi, W. and Chaudhary, V.: 2007, Wireless sensor network security: A survey, *Security in distributed, grid, mobile, and pervasive computing* **1**, 367.

Wang, H., Sheng, B. and Li, Q.: 2006, Elliptic curve cryptography-based access control in sensor networks, *International Journal of Security and Networks* **1**(3), 127–137.

Yao, G., Cheung, R. C. C., Koç, C. and Man, K. F.: 2010, Reconfigurable number theoretic transform architectures for cryptographic applications, *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 308–311.

Zhang, P., Sadler, C. M., Lyon, S. A. and Martonosi, M.: 2004, Hardware design experiences in zebranet, *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, ACM, New York, NY, USA, pp. 227–238.

Zhou, Y., Fang, Y. and Zhang, Y.: 2008, Securing wireless sensor networks: A survey., *IEEE Communications Surveys and Tutorials* **10**(1-4), 6–28.

Others

Baktir, S.: 2008, *Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography*, PhD thesis, Electrical and Computer Engineering Department, Worcester Polytechnic Institute, Worcester, MA, USA.