TÜBİTAK

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 94 – 108
© TÜBİTAK
doi:10.3906/elk-2009-95

FFT enabled ECC for WSN nodes without hardware multiplier support

Utku GÜLEN, Selçuk BAKTIR*
Computer Engineering, Faculty of Engineering and Natural Sciences, Bahçeşehir University, İstanbul, Turkey

Received: 15.09.2021 • **Accepted/Published Online:** 07.12.2021 • **Final Version:** 19.01.2022

Abstract: ECC is a popular cryptographic algorithm for key distribution in wireless sensor networks where power efficiency is desirable. A power efficient implementation of ECC without using hardware multiplier support was proposed earlier for wireless sensor nodes. The proposed implementation utilized the number theoretic transform to carry operands to the frequency domain, and conducted Montgomery multiplication, in addition to other finite field operations, in that domain. With this work, we perform in the frequency domain only polynomial multiplication and use the fast Fourier transform to carry operands between the time and frequency domains. Our ECC implementation over $GF((2^{13}-1)^{13})$ on the MSP430 microcontroller implements multiplications without using a hardware multiplier. It achieves scalar multiplication with fixed and random points in only 0.89 s and 1.74 s, respectively. Our implementation achieves ECC point multiplication of fixed and random points 10% and 13% faster, and consuming 12% and 15% less energy, in comparison to the existing work.

Key words: Elliptic curve cryptography, public-key cryptography, Internet of things, wireless sensor networks, Edwards curve, fast Fourier transform

1. Introduction

Wireless sensor networks (WSNs) find applications in various areas, e.g., healthcare monitoring [1], environmental monitoring [2, 3], smart grids [4] and defence applications [5]. Securing sensitive data transmitted by WSN nodes is essential. On the other hand, a WSN node typically contains only a constrained low-power microcontroller, with limited compute power and memory size, which makes it difficult to efficiently implement complex cryptographic algorithms [6–8]. While symmetric key cryptography is simpler and can be implemented efficiently on WSN nodes, public key cryptography (PKC) is significantly more complex and yet still needed for the distribution of the symmetric keys [9].

Elliptic curve cryptography (ECC) [10, 11] is a commonly used public-key cryptosystem and considered a viable remedy for distributing the secret keys in WSNs [12–16]. The efficiency of ECC depends on the speed of the performed arithmetic. While projective coordinates are typically used to avoid expensive inversion, multiplication still needs to be performed. A word multiplication instruction typically takes much longer to execute than a word addition instruction on constrained microcontrollers. On some microcontrollers, for power efficiency and cost reasons, a multiplication circuitry does not even exist and word multiplications are implemented with shift and add instructions. For instance, the MSP430F1232, MSP430F2274 and MSP430G2955 versions of the MSP430 microcontroller are some of the several available microcontrollers which do not have a hardware multiplier [17–19]. Note that, among these microcontrollers, the MSP430G2955 microcontroller is used in WiSense

*Correspondence: selcuk.baktir@eng.bau.edu.tr sensor nodes, namely the sensor nodes WSN1120L, WSN1120CL, WSN1101ANL and WSN1101ACL [20–23]. Moreover, Texas Instruments' (TI) development tool for wireless sensor applications, named as the TI eZ430-RF2500 wireless module, is also equipped with MSP430F2274 [24]. Using a simple power efficient microcontroller is particularly important for wireless sensor network nodes which are spread around in the field and harvest their energy from the environment [25]. For energy-harvesting wireless sensor nodes, it is a concern whether the sensor node is able to perform a power-hungry cryptographic algorithm within the limitations of the harnessed power obtained through solar energy, mechanical vibration, electromagnetic radiation, etc. In [16], a competent ECC implementation on the constrained MSP430 microcontroller is proposed over the optimal extension field (OEF) $GF((2^{13}-1)^{13})$ [26]. The implementation uses the number theoretic transform and Edwards curve point arithmetic. For power efficiency, no hardware multiplier is used and arithmetic operations are carried out in the frequency domain. In their implementation, the number theoretic transform is utilized to initially carry elliptic curve point coordinates to the frequency domain. All arithmetic operations required in ECC point multiplication are then conducted in the frequency domain. Montgomery multiplication is used for performing multiplication in the frequency domain [27–29].

In our study, we improve upon the work in [16] by performing only the squaring and multiplication operations in the frequency domain and all other arithmetic in the usual time domain. While the previous work conducted in the frequency domain an adaptation of Montgomery multiplication, we carry out only polynomial multiplication. We conduct reduction in the time domain and utilize the fast Fourier transform (FFT) [30] to accelerate conversions between the two domains.

Main contribution:

We utilize the FFT over a finite field to implement ECC on a constrained microcontroller without using hardware multiplier support for the first time in the literature. Over $GF((2^{13}-1)^{13})$, we achieve ECC point multiplication of random points in 1.74 s which is 13% faster than the existing work in [16]. Furthermore, we achieve ECC point multiplication of fixed points in 0.89 s which is 10% faster than the existing work. Our proposed implementation with the FFT achieves ECC random and fixed point multiplication consuming 29.81 mWs and 15.27 mWs which are 15% and 12% less than the energy consumed by the existing implementation. We show that, in terms of both timing performance and energy consumption, FFT based multiplication would result in better performance for ECC than frequency domain Montgomery multiplication. With our proof-of-concept implementation, we show that on an extremely constrained platform that does not use a hardware multiplier, ECC can be performed efficiently when the FFT is used. Power savings gained through our proposed implementation would be significant for battery powered WSN nodes whose lifetime is limited by their stored energy, and more particularly for energy harvesting WSN nodes which harness energy from the environment and may have more strict power constraints.

2. Background

ECC is performed over a finite field. Hence, picking an efficient finite field representation and using efficient arithmetic algorithms over the selected finite field significantly effects the performance of ECC. We implement ECC over an optimal extension field (OEF). The OEF representation is an efficient finite field representation that is proposed for implementing ECC on constrained devices [26]. The OEF representation constructs the finite field $GF(p^m)$ by choosing p as a pseudo-Mersenne prime, such that $GF(p)$ elements fit in a single processor register, and by using an irreducible binomial of the form $x^m - w$ where w is a small integer. The special forms of p and w facilitate efficient coefficient arithmetic and modular reduction. By fitting $GF(p)$ elements in a single register word, only a single instruction cycle is spent to execute microcontroller instructions over $GF(p)$ elements. When the extension degree m is selected as a prime number, ECC over $GF(p^m)$ is considered secure [31]. In this work, we implement ECC over $GF(p^m)$ where $m = 13$ is prime and $p = 2^{13} - 1$ is a Mersenne prime. Selecting p as a Mersenne prime allows for very efficient reduction modulo p which is an operation commonly performed in $GF(p^m)$ arithmetic. The finite field $GF((2^{13} - 1)^{13})$ is used in [28, 32] for constrained hardware implementations of ECC and proved efficient.

On an elliptic curve defined over $GF(p^m)$, the coordinates of a curve point are $GF(p^m)$ elements and represented as degree $m-1$ polynomials whose coefficients are in $GF(p)$ [33, 34]. In ECC a large number of divisions, multiplications, subtractions and additions are carried out. The subtraction/addition of $a(x)$ with $b(x)$ in $GF(p^m)$ is achieved easily through pairwise modular word additions/subtractions of their polynomial coefficients, as given below:

$$a(x) \pm b(x) = \sum_{j=0}^{m-1} (a_j \pm b_j) x^j \bmod p \ .$$

Whereas, multiplication of $GF(p^m)$ elements is significantly more complex and necessitates a quadratic number of coefficient multiplications modulo p and a final modular reduction with the field polynomial, given as follows:

$$r'(x) = a(x) \cdot b(x) = \sum_{j=0}^{2m-2} r'_j x^j \ ,$$

$$r(x) = r'(x) \bmod p(x) \ ,$$

where $p(x)$ can be selected as $x^m - 2$ to make modular reduction simple. In this work, we use $p(x) = x^m - 2$ to construct the finite field $GF((2^{13} - 1)^{13})$. Polynomial multiplication requires computing a quadratic number of expensive modular coefficient multiplications. The convolution theorem states that time domain polynomial multiplication produces the same result as frequency domain pairwise coefficient multiplications. Hence, the number of performed coefficient multiplications is reduced dramatically if frequency domain is used for polynomial multiplication. The discrete Fourier transform (DFT), or its optimized form the fast Fourier transform (FFT), can be used for carrying $GF(p^m)$ elements into frequency domain. In this work, we implement ECC by utilizing the FFT [35] to speed up multiplication and squaring operations over $GF(p^m)$. Our selection of the finite field $GF(p^m)$ with $p = 2^{13} - 1$ allows for efficient forward and inverse FFT computations, as described in Sections 2.2.1 and 2.2.3.

2.1. Discrete Fourier transform based Montgomery multiplication

Algorithm 1 was proposed for achieving $GF(p^m)$ multiplication using discrete Fourier transform based Montgomery multiplication [27, 36, 37]. The algorithm was utilized in ECC implementations for constrained wireless sensor nodes [16, 28]. The algorithm takes as inputs $(\bar{A})$ and $(\bar{B})$, which are the frequency domain series for $\bar{a}(x) = a(x)x^{m-1}, \bar{b}(x) = b(x)x^{m-1} \in GF(p^m)$, and produces their Montgomery product. Note that $\bar{a}(x)$ and $\bar{b}(x)$ are the Montgomery forms of $a(x)$ and $b(x)$. The output of the algorithm is denoted with $\bar{R}$ and represents the Montgomery product of $\bar{a}(x)$ and $\bar{b}(x)$, i.e. $\bar{a}(x)\bar{b}(x)x^{-(m-1)} \bmod p(x)$. Note that

$\bar{R} = \bar{a}(x)\bar{b}(x)x^{-(m-1)} \bmod p(x)$ is equal to $a(x)b(x)x^{m-1} \bmod p(x)$ which is the Montgomery form of the product of $a(x)$ and $b(x)$ in $GF(p^m)$. Using $p = 2^m - 1$, a Mersenne prime, results in more efficient DFT computations [38]. In Algorithm 1, a linear number of word products are computed. Whereas, the number of performed bitwise rotations, subtractions and additions are quadratic. Since multiplication is more complex compared to other arithmetic operations on a constrained microcontroller, Algorithm 1 is desirable.

Algorithm 1: Discrete Fourier transform based Montgomery multiplication over $GF(p^m)$, with $p = 2^m - 1$ and $p(x) = x^m - 2$ [16]

```
Input: $(\bar{A})$ and $(\bar{B})$ are the frequency domain series for $\bar{a}(x) = a(x)x^{m-1}$ and $\bar{b}(x) = b(x)x^{m-1}$ in $GF(p^m)$
Output: $\bar{R} = \bar{a}(x)\bar{b}(x)x^{-(m-1)} \bmod x^m - 2$
for $j \leftarrow 0$ to $2m-1$ do
    $\bar{R}_j \leftarrow \bar{A}_j\bar{B}_j \bmod p$
end
for $i \leftarrow 0$ to $m-2$ do
    $T \leftarrow \bar{R}_0$
    for $k \leftarrow 1$ to $2m-1$ do
        $T \leftarrow T + \bar{R}_k \bmod p$
    end
    $T \leftarrow -T/2m \bmod p$
    $T_e \leftarrow T/2 \bmod p$
    $T_o \leftarrow T + T_e \bmod p$
    for $j \leftarrow 0$ to $m-1$ do
        $\bar{R}_{2j} \leftarrow (\bar{R}_{2j} + T_e)/2^{2j} \bmod p$
        $\bar{R}_{2j+1} \leftarrow -(\bar{R}_{2j+1} + T_o)/2^{2j+1} \bmod p$
    end
end
Return $\bar{R}$
```

2.2. Fast Fourier transform based multiplication

In a recent work, ECC is implemented using a different DFT based approach to realize $GF(p^m)$ multiplications and squarings [39]. In [39], the FFT [30, 35, 36] is used to transform $GF(p^m)$ elements into the frequency domain. Once the frequency domain representations for $GF(p^m)$ elements are obtained, their polynomial multiplication is computed simply by pairwise multiplying their frequency domain coefficients. Utilizing the inverse fast Fourier transform (IFFT) algorithm, the resulting product is carried to the time domain. In Algorithm 2, we present this approach which is composed of the three stages: FFT, pairwise multiplication (PM) and IFFT. With this work, similar to [39], we use FFT based multiplication to implement ECC. However, unlike in [39], we implement FFT based ECC without using hardware multiplier support to show that ECC can be achieved practically on an extremely constrained microcontroller without even a hardware multiplier. Furthermore, we obtain the energy consumption profile of our ECC implementation and show that it is more energy efficient than the existing implementation in [16] which also does not use hardware multiplier support.

Algorithm 2: Fast Fourier transform based multiplication.

```
   Input: $a(x)$ and $b(x)$ in $GF(p^m)$
   Output: $r(x) = a(x)b(x) \bmod p(x)$
1  $(A) \longleftarrow FFT(a(x))$
2  $(B) \longleftarrow FFT(b(x))$
3  $(R) \longleftarrow PM((A),(B))$
4  $r(x) \longleftarrow IFFT((R))$
5  Return $r(x)$
```

2.2.1. Conversion of operands into the frequency domain:

The finite field elements $a(x), b(x) \in GF((2^{13}-1)^{13})$ are carried to the frequency domain with the DFT as

$$A_i = \sum_{j=0}^{25} a_j e^{ji} \bmod p, \; 0 \le i \le 25 \tag{1}$$

and

$$B_i = \sum_{j=0}^{25} b_j e^{ji} \bmod p, \; 0 \le i \le 25 \; , \tag{2}$$

where e is a 26^{th} primitive root of unity. Algorithm 3 performs the above DFT computations over $GF((2^{13}-1)^{13})$ efficiently by using the FFT. It is designed such that the seven general purpose registers on MSP430 are used heavily to store intermediary results so that the least number of time-consuming memory read/write instructions are executed. The algorithm uses the binomial $x^{13}-2$ as field generating polynomial and $e=-2$ as the 26^{th} primitive root of unity. For the selected finite field $GF((2^{13}-1)^{13})$, e is chosen as -2. This allows us to perform multiplications of $GF(p)$ elements with positive powers of e, as it heavily takes place in the FFT computation (in lines 7 and 19 of Algorithm 3), with a simple bitwise left rotation, in addition to a simple negation if the power of e is odd. Note that for $p = 2^{13}-1, e = -2$, $a \in GF(p)$ and k a positive integer, the computation $a \times e^k = a \times (-1)^k \times 2^k$ modulo p is equivalent to the simple bitwise left rotation of a by $(k \bmod 13)$ bits followed by a simple negation if k is odd.

2.2.2. Pairwise coefficient multiplication in the frequency domain:

Frequency domain multiplication of $GF((2^{13}-1)^{13})$ elements is carried out through pairwise coefficient multiplications. For (A) and (B), the 26-coefficient frequency domain sequences for $a(x)$ and $b(x)$ in $GF((2^{13}-1)^{13})$, $r'(x) = a(x)b(x) \bmod 2^{13}-1$ is computed in the frequency domain as

$$R'_j = A_j B_j \quad \bmod 2^{13}-1 \; , \qquad 0 \le j \le 25 \; . \tag{3}$$

The above 26 coefficient multiplications are the only $GF(2^{13}-1)$ multiplications performed for computing (R') which is dramatically faster than performing 169 coefficient multiplications as needed in schoolbook multiplication. However, (R') needs to be carried back to time domain to complete the $GF((2^{13}-1)^{13})$ multiplication and find $r(x) = r'(x) \bmod p(x)$. Modular reduction with $p(x)$ becomes very simple when $p(x)$ is selected as $x^{13}-2$. However, one still needs convert (R') to the time domain polynomial $r'(x)$.

Algorithm 3: FFT computation over $GF((2^{13}-1)^{13})$ on MSP430.

```
    Input: $a(x) = a_0 + a_1x + a_2x^2 \cdots + a_{12}x^{12} \in GF(p^{13})$ where $p = 2^{13}-1$. $R_0, R_1 ... R_6$ denote used
           registers. $E_0, E_1 ... E_{12}$ and $O_0, O_1 ... O_{12}$ denote used temporary variables.
    Output: $(A) = (A_0, A_1, A_2 ... A_{25})$, frequency domain coefficients of $a(x)$ .
 1  for $j \leftarrow 0$ to $6$ do
 2      $R_j \longleftarrow a_{2j}$
 3  end
 4  $E_0 \longleftarrow R_0 + R_1 + ... + R_6 \bmod p$
 5  for $j \leftarrow 1$ to $12$ do
 6      for $i \leftarrow 1$ to $6$ do
 7          $R_i \longleftarrow R_i 2^{2i} \bmod p$
 8      end
 9      $E_j \longleftarrow R_0 + R_1 + ... + R_6 \bmod p$
10  end
11  for $j = 0$ to $5$ do
12      $R_j \longleftarrow a_{2j+1}$
13  end
14  $O_0 \longleftarrow R_0 + R_1 + ... + R_5 \bmod p$
15  $A_0 \longleftarrow E_0 + O_0 \bmod p$
16  $A_{13} \longleftarrow E_0 - O_0 \bmod p$
17  for $j = 1$ to $12$ do
18      for $i = 0$ to $5$ do
19          $R_i \leftarrow R_i 2^{2i+1} \bmod p$
20      end
21      $O_j \leftarrow R_0 + R_1 + ... + R_5 \bmod p$
22      $A_j \leftarrow E_j + (-1)^j O_j \bmod p$
23      $A_{j+13} \longleftarrow E_j - (-1)^j O_j \bmod p$
24  end
25  Return $(A_0, A_1, A_2 ... A_{25})$
```

2.2.3. Conversion of the polynomial product back to the time domain:

In order to finalize the finite field multiplication operation in $GF((2^{13}-1)^{13})$, the 26-element sequence (R') for $r'(x) = a(x)b(x)$ needs to be carried into time domain to realize modular reduction, i.e. $r(x) = r'(x) \bmod x^{13}-2$, efficiently. The conversion can be done using the inverse DFT as follows:

$$r'_i = \frac{1}{26}\sum_{j=0}^{25} R'_j e^{-ji} \bmod p, \; 0 \le i \le 25 \; . \tag{4}$$

Algorithm 4 performs the above inverse DFT computation efficiently to convert (R') in the frequency domain to $r(x) = a(x)b(x) \bmod x^{13}-2$ in the time domain. The algorithm optimizes (4) by utilizing the inverse FFT and by interleaving it with the reduction operation modulo the field generating polynomial $x^{13}-2$. For the selected finite field $GF((2^{13}-1)^{13})$, the 26^{th} primitive root of unity e for the inverse FFT computation is chosen as -2. This allows us to perform multiplications of $GF(p)$ elements with negative powers of e, as it heavily takes place in the inverse FFT computation (in lines $7, 17, 27$ and 38 of Algorithm 4), with a simple bitwise right rotation, in addition to a simple negation if the power of e is odd. Note that for $p = 2^{13}-1, e = -2$, $a \in GF(p)$ and a positive integer k, the computation $a \times e^{-k} = a \times (-1)^k \times 2^{-k}$ modulo p is equivalent to the simple bitwise right rotation of a by $(k \bmod 13)$ bits followed by a simple negation if k is odd.

2.3. ECC over $GF((2^{13}-1)^{13})$ using Edwards curves

Edwards curves, described by $x^2 + y^2 = c^2(1 + dx^2y^2)$, are proposed for ECC [40]. Given below is the Edwards curve point addition formula for the operation $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$:

$$x_3 = \frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)} \quad \text{and} \quad y_3 = \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)} \; .$$

Algorithm 4: Inverse FFT over $GF((2^{13}-1)^{13})$ on MSP430.

```
Input: $(R') = (R'_0, R'_1, R'_2 ... R'_{25})$, the frequency domain coefficients for $r'(x) = a(x)b(x)$, where $a(x)$ and
       $b(x)$ are in $GF(p^{13})$ and $p = 2^{13} - 1$. $R_0, R_1 ... R_6$ denote used registers. $E_0, E_1 ... E_{12}$ and
       $O_0, O_1 ... O_{12}$ denote used temporary variables.
Output: $r(x) = r'(x) \bmod p(x)$ where $p(x) = x^{13} - 2$ .
1  for $j \leftarrow 0$ to 6 do
2      $R_j \longleftarrow R'_{2j}$
3  end
4  $E_0 \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
5  for $j \leftarrow 1$ to 12 do
6      for $i \leftarrow 1$ to 6 do
7          $R_i \longleftarrow R_i/2^{2i} \bmod p$
8      end
9      $E_j \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
10 end
11 for $j \leftarrow 7$ to 12 do
12     $R_{j-7} \longleftarrow R'_{2j+1}$
13 end
14 $E_0 \leftarrow E_0 + R_0 + R_1 + ... + R_5 \bmod p$
15 for $j \leftarrow 1$ to 12 do
16     for $i \leftarrow 7$ to 12 do
17         $R_{i-7} \longleftarrow R_{i-7}/2^{2i} \bmod p$
18     end
19     $E_j \leftarrow E_j + R_0 + R_1 + ... + R_5 \bmod p$
20 end
21 for $j \leftarrow 0$ to 6 do
22     $R_j \longleftarrow R'_{2j+1}$
23 end
24 $O_0 \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
25 for $j \leftarrow 1$ to 12 do
26     for $i \leftarrow 0$ to 6 do
27         $R_i \longleftarrow R_i/2^{2i+1} \bmod p$
28     end
29     $O_j \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
30 end
31 for $j \leftarrow 7$ to 12 do
32     $R_{j-7} \longleftarrow R'_{2j+1}$
33 end
34 $O_0 \leftarrow O_0 + R_0 + R_1 + ... + R_5 \bmod p$
35 $r_0 \longleftarrow (3E_0 - O_0)7876 \bmod p$
36 for $j \leftarrow 1$ to 12 do
37     for $i \leftarrow 7$ to 12 do
38         $R_{i-7} \longleftarrow R_{i-7}/2^{2i+1} \bmod p$
39     end
40     $O_j \leftarrow O_j + R_0 + R_1 + ... + R_5 \bmod p$
41     $r_j \longleftarrow (3E_j + O_j(-1)^{j-1})7876 \bmod p$
42 end
43 Return $r_0 + r_1 x + r_2 x^2 + ....r_{12} x^{12}$
```

The ECC point doubling operation $P_2(x_2, y_2) = 2P_1(x_1, y_1)$ on the Edwards curve is computed as

$$x_2 = \frac{2x_1y_1c}{x_1^2 + y_1^2} \quad \text{and} \quad y_2 = \frac{(y_1^2 - x_1^2)c}{2c^2 - (x_1^2 + y_1^2)} .$$

For $x^2 + y^2 = c^2(1 + dx^2y^2)$, where $dc^4 \neq 1$, $c = 1$, in [41] efficient formulae (Algorithms 5 and 6) are given for Edwards curve point doubling/addition in projective coordinates. For our ECC implementation, we use the improved forms of Algorithms 5 and 6, given as Algorithms 7 and 8, respectively.

Algorithm 5: Edwards curve point doubling formula for projective coordinates.

```
Input: $P_1(x_1, y_1, z_1)$
Output: $P_2(x_2, y_2, z_2) = 2 \times P_1$
1  $t_1 \leftarrow x_1$, $t_2 \leftarrow y_1$, $t_3 \leftarrow z_1$
2  $t_4 \leftarrow t_1 + t_2$
3  $t_1 \leftarrow t_1^2$
4  $t_2 \leftarrow t_2^2$
5  $t_3 \leftarrow t_3^2$
6  $t_4 \leftarrow t_4^2$
7  $t_3 \leftarrow 2t_3$
8  $t_5 \leftarrow t_1 + t_2$
9  $t_2 \leftarrow t_1 - t_2$
10 $t_4 \leftarrow t_4 - t_5$
11 $t_3 \leftarrow t_5 - t_3$
12 $t_1 \leftarrow t_3 t_4$
13 $t_3 \leftarrow t_3 t_5$
14 $t_2 \leftarrow t_2 s t_5$
15 $x_2 \leftarrow t_1$, $y_2 \leftarrow t_2$, $z_2 \leftarrow t_3$
```

Algorithm 6: ECC point addition in projective coordinates over Edwards curves.

```
Input: $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$
Output: $P_3(x_3, y_3, z_3) = P_1 + P_2$
1  $t_1 \leftarrow x_1$ , $t_2 \leftarrow y_1$ , $t_3 \leftarrow z_1$ , $t_4 \leftarrow x_2$ , $t_5 \leftarrow y_2$ , $t_6 \leftarrow z_2$
2  $t_3 \leftarrow t_3 t_6$
3  $t_7 \leftarrow t_1 + t_2$
4  $t_8 \leftarrow t_4 + t_5$
5  $t_1 \leftarrow t_1 t_4$
6  $t_2 \leftarrow t_2 t_5$
7  $t_7 \leftarrow t_7 t_8$
8  $t_7 \leftarrow t_7 - t_1$
9  $t_7 \leftarrow t_7 - t_2$
10 $t_7 \leftarrow t_7 t_3$
11 $t_8 \leftarrow t_1 t_2$
12 $t_8 \leftarrow t_8 d$
13 $t_2 \leftarrow t_2 - t_1$
14 $t_2 \leftarrow t_2 t_3$
15 $t_3 \leftarrow t_3^2$
16 $t_1 \leftarrow t_3 - t_8$
17 $t_3 \leftarrow t_3 + t_8$
18 $t_2 \leftarrow t_2 t_3$
19 $t_3 \leftarrow t_3 t_1$
20 $t_1 \leftarrow t_1 t_7$
21 $x_3 \leftarrow t_1$ , $y_3 \leftarrow t_2$ , $z_3 \leftarrow t_3$
```

Algorithm 7: Edwards curve point doubling formula using the FFT.

```
Input: $P = (x_1, y_1, z_1)$
Output: $2P = (x_2, y_2, z_2)$
1  $(T_1) \longleftarrow FFT(x_1) + FFT(y_1)$     // Store $(X_1)$ and $(Y_1)$
2  $t_1 \longleftarrow t_1^2$                     // Use $(T_1)$
3  $x_1 \longleftarrow x_1^2$                     // Use $(X_1)$
4  $y_1 \longleftarrow y_1^2$                     // Use $(Y_1)$
5  $z_1 \longleftarrow 2{z_1}^2$
6  $t_2 \longleftarrow x_1 + y_1$
7  $y_1 \longleftarrow x_1 - y_1$
8  $t_1 \longleftarrow t_1 - t_2$
9  $z_1 \longleftarrow t_2 - z_1$
10 $x_2 \longleftarrow z_1 t_1$                    // Store $(Z_1)$
11 $z_2 \longleftarrow z_1 t_2$                    // Use $(Z_1)$, Store $(T_2)$
12 $y_2 \longleftarrow y_1 t_2$                    // Use $(T_2)$
```

3. Implementation and performance results

In this work, without using hardware multiplier support, we implement ECC on Edwards curves, projective coordinates and the FFT. For efficient multiplication in $GF((2^{13}-1)^{13})$, we facilitate FFT based finite field multiplication. We develop our code using the IAR Embedded Workbench[1] development tool and use its debugger for timing measurements. We use the C and the assembly languages.

We pick the 1-series basic MSP430 microcontroller, MSP430F1611, to realize our ECC implementation [42]. MSP430F1611 is a low-power and low-cost microcontroller which is the control unit of widely used sensor nodes such as TelosB/Tmote Sky, Shimmer3, MTM-CM3300-MSP and MTM-CM5000-MSP. It has 48 kB flash memory and 10 kB RAM and runs at 8 MHz maximum clock frequency. It draws a current of less than 1 μA in idle mode. Its low cost and low power consumption makes it a preferred microcontroller for constrained ubiquitous sensor nodes.

An efficient implementation of ECC on Edwards curves in projective coordinates over $GF((2^{13}-1)^{13})$ is presented in [16] where Algorithm 1 (DFT Montgomery multiplication) is utilized for performing modular

[1]https://www.iar.com/iar-embedded-workbench/ [accessed 22.12.2020]

Algorithm 8: Edwards curve point addition formula using the FFT.

```
Input: $P_1 = (x_1, y_1, z_1)$ , $P_2 = (x_2, y_2, z_2)$
Output: $P_1 + P_2 = (x_3, y_3, z_3)$
1  $z_1 \longleftarrow z_1 z_2$
2  $(T_1) \longleftarrow FFT(x_1) + FFT(y_1)$      // Store $(X_1), (Y_1)$ and $(T_1)$
3  $(T_2) \longleftarrow FFT(x_2) + FFT(y_2)$      // Store $(X_2), (Y_2)$ and $(T_2)$
4  $t_1 \longleftarrow t_1 t_2$                    // Use $(T_1)$ and $(T_2)$
5  $x_1 \longleftarrow x_1 x_2$                    // Use $(X_1)$ and $(X_2)$
6  $y_1 \longleftarrow y_1 y_2$                    // Use $(Y_1)$ and $(Y_2)$
7  $t_1 \longleftarrow t_1 - x_1$
8  $t_1 \longleftarrow t_1 - y_1$
9  $t_1 \longleftarrow t_1 z_1$                    // Store $(Z_1)$
10 $t_2 \longleftarrow d x_1 y_1$                  // Store $(X_1)$ and $(Y_1)$
11 $y_1 \longleftarrow y_1 - x_1$                  // Use $(X_1)$ and $(Y_1)$, store $(Y_1)$
12 $y_1 \longleftarrow y_1 z_1$                    // Use $(Y_1)$ and $(Z_1)$
13 $z_1 \longleftarrow z_1^2$                      // Use $(Z_1)$
14 $x_1 \longleftarrow z_1 - t_2$
15 $z_1 \longleftarrow z_1 + t_2$
16 $y_3 \longleftarrow y_1 z_1$                    // Store $(Z_1)$
17 $z_3 \longleftarrow z_1 x_1$                    // Use $(Z_1)$, store $(X_1)$
18 $x_3 \longleftarrow x_1 t_1$                    // Use $(X_1)$
```

squaring and multiplication operations. In that work, ECC is implemented without using a hardware multiplier to show that, when frequency domain arithmetic is utilized, it is feasible to run ECC on extremely constrained microcontrollers without utilizing a hardware multiplier.

With this work, as an alternative to DFT Montgomery multiplication, we utilize an FFT based multiplication algorithm (Algorithm 2) for our ECC implementation. Similar to the previous work, in this work we implement ECC on Edwards curves in projective coordinates over $GF((2^{13}-1)^{13})$ on the constrained MSP430 microcontroller, namely on MSP430F1611@8 MHz, without using the hardware multiplier. For the forward and inverse FFT computations in Algorithm 2, we use Algorithm 3 and Algorithm 4, respectively.

When Algorithm 2 is used for $GF((2^{13}-1)^{13})$ multiplication, only 26 word multiplications are performed for the pairwise multiplications on line 3 of the algorithm, as explained in Equation (3), in addition to the 13 constant word multiplications with 7876 on lines 35 and 41 of Algorithm 4. We compute these 13-bit multiplications in $GF(2^{13}-1)$ to generate the 26-bit result with a series of 12 shift and 12 add operations. The 26-bit intermediary result is then reduced modulo $GF(2^{13}-1)$. Our $GF(2^{13}-1)$ multiplication operation is implemented to run in constant time as a side-channel attack countermeasure.

Squaring in $GF((2^{13}-1)^{13})$ is simpler than $GF((2^{13}-1)^{13})$ multiplication when Algorithm 2 is utilized. This is due to the fact that in squaring only one FFT computation (Algorithm 3) is performed for the single operand whereas for multiplication two FFT computations are performed.

We give in Table 1 the timings for our FFT based multiplication and squaring operations in $GF((2^{13}-1)^{13})$ as well as the timings in [16] for DFT Montgomery multiplication. While DFT based Montgomery multiplication in $GF((2^{13}-1)^{13})$ takes 1.18 ms, our FFT based multiplication implementation takes 1.3 ms which is 10.2% slower. However, our FFT based squaring implementation takes only 1.06 ms while the squaring operation using DFT based Montgomery multiplication takes the same time as multiplication, i.e. 1.18 ms. Hence, FFT based squaring is shown to be 11.3% faster than squaring using DFT Montgomery multiplication.

We implement ECC over $GF((2^{13}-1)^{13})$ using Edwards curves and projective coordinates. We use the ECC point doubling and addition formulae given with Algorithms 7 and 8, respectively. These are our adaptations and the improved versions of Algorithms 5 and 6 for frequency domain arithmetic. We utilize FFT based finite field multiplication and squaring. We take advantage of FFT based arithmetic to eliminate redundant computations. For instance, if the FFT of an operand is already computed earlier, we just store it the first time it is computed and use the precomputed value in later computations. Furthermore, if the FFTs of two operands are already available and if we are interested in finding the FFT of the sum, we find the sum in the frequency domain, eliminating the need for additional inverse and forward FFT computations. The relationship between the algorithms used in this work are given with the dependency graph in Figure. In Table 1 are our timings for ECC point arithmetic using FFT based multiplication/squaring and comparisons against the timings of [16] which uses DFT based Montgomery multiplication under the same setting.

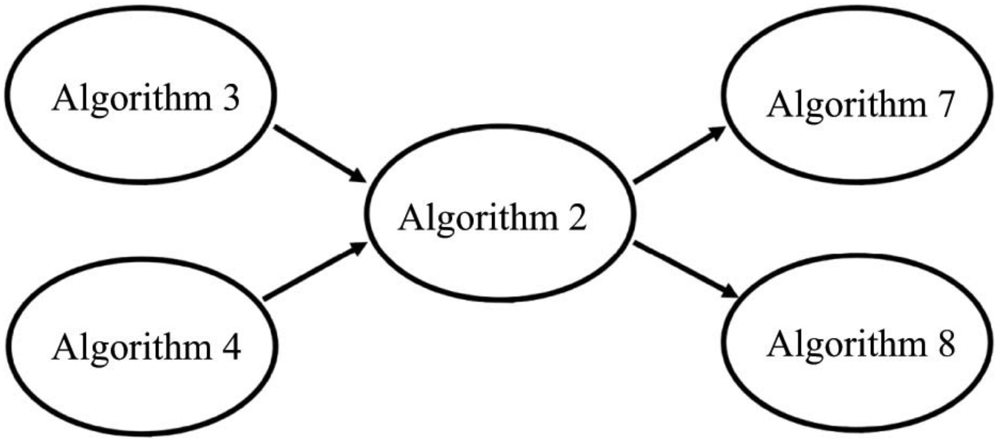

Figure. Dependency graph for the algorithms used in the proposed ECC implementation.

We implement ECC scalar point multiplication, the main operation for encryption/decryption in ECC, using the NAF4 method for multiplication of random points and the Comb method for multiplication of fixed points. We achieve ECC scalar point multiplication in 1.74 s for random points and 0.88 s for fixed points. In [16], the same operation, in the same setting but by using DFT Montgomery multiplication, was achieved in 1.97 s and 0.98 s for random and fixed points. As given with Table 1, this work achieves 11% and 10% better timings for random and fixed point multiplication operations, respectively.

A summary of our timing results for ECC point multiplication and the underlying elliptic curve point operations and arithmetic operations, as well as the timing results of [16], are given in Table 1. Our ECC implementation utilizes FFT based multiplication (Algorithms 2, 3 and 4) for squaring and multiplication, whereas [16] uses DFT based Montgomery multiplication (Algorithm 1). Note that while DFT based Montgomery multiplication is faster than FFT based multiplication, FFT based squaring is even faster. In our ECC random point multiplication implementation over $GF((2^{13}-1)^{13})$ with NAF4, significantly more point doublings than point additions are performed. On average around 37 point additions and 170 point doublings are performed [31]. Hence, the performance of ECC point doubling is the determining factor in the performance of ECC random point multiplication. In our ECC point doubling implementation (Algortihm 5), four squarings

Table 1. Timings for ECC on MSP430F1611@8MHz over $GF((2^{13}-1)^{13})$ using Edwards curves and projective coordinates, with DFT based Montgomery multiplication vs. FFT based multiplication.

Operations	**Timings**
ECC with DFT Montgomery multiplication [16]	
DFT based Montgomery multiplication	1.18 ms
Edwards curve point doubling	8.52 ms
Edwards curve point addition	14.49 ms
Edwards curve random point multiplication with NAF4	1.97 s
Edwards curve fixed point multiplication with Comb4	0.98 s
ECC with FFT based multiplication/squaring (this work)	
FFT based multiplication/squaring (Algorithm 2)	1.3 ms / 1.06 ms
Improved Edwards curve point doubling (Algorithm 7)	7.55 ms
Improved Edwards curve point addition (Algorithm 8)	12.95 ms
Edwards curve random point multiplication with NAF4	1.74 s
Edwards curve fixed point multiplication with Comb4	0.88 s

and three multiplications in $GF((2^{13}-1)^{13})$ are performed. Since squaring with FFT based multiplication (Algorithms 2, 3 and 4) is faster than squaring with DFT based Montgomery multiplication (Algorithm 1), ECC random point multiplication is faster when FFT based multiplication is used.

Similarly, in our ECC fixed point multiplication implementation over $GF((2^{13}-1)^{13})$, around 42 point doublings and 39 point additions are performed [31]. Hence, the performance of ECC point doubling is the determining factor in the performance of ECC random point multiplication. Since more squarings than multiplications are performed in ECC point doubling (Algortihm 5) and squaring with FFT based multiplication (Algorithms 2, 3 and 4) is faster than squaring with DFT based Montgomery multiplication (Algorithm 1), our ECC fixed point multiplication implementation with FFT based multiplication is faster than the previous work which uses DFT Montgomery multiplication.

We investigate the energy efficiency of our implementation and compare it with the previous implementations. In order to obtain energy measurements, we run our codes on the experimenter board MSP-EXP430FG4618 which has an MSP430FG4618 microcontroller onboard [43]. Since our ECC implementation is for the basic 1-series MSP430F1611, we can run our implementation on the experimenter board without changing our code. Using the flash emulation tool (MSP-FET) [44] and the Power Log [2] feature of the IAR Embedded Workbench development tool, we are able to obtain energy measurements.

The average power consumption for our ECC random point multiplication implementation in this work is 17.3 mW without using the hardware multiplier. The work in [39] uses the hardware multiplier and achieves the same operation with a power consumption figure of 17.66 mW. The average power consumption for our ECC fixed point multiplication implementation in this work is 17.5 mW without using the hardware multiplier. The work in [39] uses the hardware multiplier and achieves the same operation with a power consumption figure of 17.88 mW. For ECC random and fixed point multiplication operations, we achieve around 2% improved power efficiency. The previous work in [39], which utilizes the hardware multiplier unit of the MSP430 microcontroller, has faster timings on the experimental board, i.e. 1.35 s and 0.64 s for random

[2]https://www.iar.com/iar-embedded-workbench/power-debugging/ [accessed 22.12.2020]

and fixed point multiplication, respectively. Since these execution times are less than the execution times in the proposed implementation, the total energy consumptions are also lower. The total energy consumptions for the implementations in [39] are 23.84 mWs and 11.44 mWs for random and fixed point multiplication, respectively. While these total energy consumption figures are better than those of the proposed implementation, the average power consumption figures for the proposed implementation are better. Note that we run our ECC implementations on the MSP430FG4618 microcontroller without using its hardware multiplier and obtain our energy/power measurements on it. While this helps us gain power/energy efficiency in terms of dynamic power usage, the microcontroller still uses static power due to its onboard hardware multiplier. We could only use this microcontroller for power/energy measurements because it is the microcontroller contained in the experimental board with the FET emulator which we use to obtain timing and power/energy measurements. We believe that better energy/power efficiency could be achieved on another version of MSP430, such as MSP430F2274 or MSP430G2955, which does not have a hardware multiplier. We would like to note that low-cost, low-power MPS430 microcontroller versions do not have a hardware multiplier unit. For instance, from the low-power 1-series MSP430 versions, only the microcontrollers with the device names MSP430x14x and MSP430x16x have a hardware multiplier unit. Whereas, other low-power 1-series MSP430 versions, such as MSP430F1122, MSP430F1232, MSP430F135 and MSP430F155, do not have an onboard hardware multiplier [42]. Among other series of MSP430 microcontrollers, there are also models without a hardware multiplier unit. One such example is MSP430F2274 which is equipped in the Texas Instrument ez430-RF2500 wireless module that is designed to be deployed in wireless sensor network applications [24]. Unlike the ECC implementation in [39], our ECC implementation, which does not require a hardware multiplier, has the additional advantage of being able to run efficiently also on these extremely constrained microcontrollers without a hardware multiplier. The main motivation for using a processor without a hardware multiplier would be to increase the battery lifetime or for applications where sensor nodes harvest their own energy and need to operate under extremely low power constraints.

We also compare our work against the previous work in [16] in terms of power efficiency. Note that both works implement ECC without using the hardware multiplier. This work uses the FFT (Algorithm 2), whereas the work in [16] uses DFT Montgomery multiplication (Algorithm 1) for finite field multiplication. Our work achieves ECC scalar point multiplication with a power consumption figure of 17.3 mW for random points and 17.5 mW for fixed points. In [16], the same operation, in the same setting but by using DFT Montgomery multiplication, was achieved with a power consumption figure of 18.2 mW for both random and fixed points. Hence, this work achieves 5% and 4% better power efficiency for random and fixed point multiplication, respectively. Furthermore, our work achieves ECC scalar point multiplication with an energy consumption of 29.81 mWs for random points and 15.27 mWs for fixed points. In [16], the same operation, in the same setting but by using DFT Montgomery multiplication, was achieved with the energy consumption of 34.97 mWs and 17.36 mWs for random and fixed points. Hence, this work achieves 15% and 12% better energy efficiency for random and fixed point multiplication, respectively. A summary of all the energy/power measurements for our implementations of ECC point multiplication and the underlying elliptic curve point operations and arithmetic operations, as well as the energy/power measurements for the ECC implementation in [16], are given in Table 2.

4. Conclusion

We implemented ECC on the MSP430 microcontroller without using hardware multiplier support and using FFT based finite field arithmetic. We showed that ECC can be run efficiently on extremely constrained devices

Table 2. Power measurements taken on MSP-EXP430FG4618@8 MHz for hardware multiplierless ECC implementations over $GF((2^{13}-1)^{13})$ using DFT based Montgomery multiplication vs. FFT based multiplication.

Operations	**Total energy**	**Average power**
ECC with DFT Montgomery multiplication [16]		
DFT based Montgomery multiplication	0.021 mWs	18.2 mW
Edwards curve point doubling	0.15 mWs	18.2 mW
Edwards curve point addition	0.25 mWs	18.2 mW
Edwards curve random point multiplication with NAF4	34.97 mWs	18.2 mW
Edwards curve fixed point multiplication with Comb4	17.36 mWs	18.2 mW
ECC with FFT based multiplication/squaring (this work)		
FFT based multiplication/squaring (Algorithm 2)	0.022/0.018 mWs	17.1/17.2 mW
Improved Edwards curve point doubling (Algorithm 7)	0.13 mWs	17.2 mW
Improved Edwards curve point addition (Algorithm 8)	0.22 mWs	17.2 mW
Edwards Curve random point multiplication with NAF4	29.81 mWs	17.3 mW
Edwards Curve fixed point multiplication with Comb4	15.27 mWs	17.5 mW

when FFT based squaring and multiplication operations are used. Since FFT based squaring and multiplication require dramatically fewer word multiplications, we discarded the hardware multiplier supported by MSP430 microcontrollers. Instead of utilizing the hardware multiplier, we realized a fixed-time word multiplication subroutine with addition and shift operations. Thus our ECC implementation is also suitable for power-critical applications. We realized ECC point multiplication in 0.89 s and 1.74 s for fixed and random points, which are 10% and 13% faster, respectively, in comparison with the previous work in [16]. Moreover, the total energy consumption of our ECC implementation for fixed and random point multiplication is 12% and 15% less than the previous implementation. In our proof-of-concept implementation, we realized ECC without using the hardware multiplier on the MSP430FG4618 microcontroller and obtained energy/power measurements on it. We achieved power/energy savings by not using the available onboard hardware multiplier and thus by eliminating dynamic power consumption due to the use of the hardware multiplier. We used this microcontroller because it is the microcontroller contained in the FET emulator which we use to obtain our timing and energy/power measurements. Bu using the MSP430 versions MSP430F2274, MSP430G2955, MSP430F1122, MSP430F1232, MSP430F135 or MSP430F155, which do not contain an onboard hardware multiplier, static power/energy consumption due to the hardware multiplier can also be eliminated, and thus better power/energy efficiency could be achieved by using our proposed algorithms and implementations. We identify power/energy efficient hardware-multiplier free implementations of ECC on hardware-multiplierless microcontrollers such as MSP430F2274 or MSP430G2955, which are utilized in WiSense nodes [20–23] and the TI eZ430-RF2500 wireless module [24], as directions for future research.

Acknowledgment

This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant no. 215E208. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions which helped improve the quality of this paper.

References

[1] Baronti P, Pillai P, Chook VWC, Chessa S, Gotta A et al. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. Computer Communications 2007; 30 (7): 1655–1695. doi: 10.1016/j.comcom.2006.12.020

[2] Pottie GJ, Kaiser WJ. Wireless integrated network sensors. Communications of the ACM 2000; 43 (5): 51–58. doi: 10.1145/332833.332838

[3] Chong CY, Kumar SP. Sensor networks: evolution, opportunities, and challenges. Proceedings of the IEEE 2003; 91 (8): 1247–1256. doi: 10.1109/JPROC.2003.814918

[4] de Souza RWR, Moreira LR, Rodrigues JJ, Moreira RR, de Albuquerque VHC. Deploying wireless sensor networks–based smart grid for smart meters monitoring and control. International Journal of Communication Systems 2018; 31 (10): e3557. doi: 10.1002/dac.3557

[5] Yick J, Mukherjee B, Ghosal D. Wireless sensor network survey. Computer Networks 2008; 52 (12): 2292–2330. doi: 10.1016/j.comnet.2008.04.002

[6] Feng D, Jiang C, Lim G, Cimini LJ, Feng G et al. A survey of energy-efficient wireless communications. IEEE Communications Surveys & Tutorials 2013; 15 (1): 167–178. doi: 10.1109/SURV.2012.020212.00049

[7] Li M, Lou W, Ren K. Data security and privacy in wireless body area networks. IEEE Wireless Communications 2010; 17 (1): 51–58. doi: 10.1109/MWC.2010.5416350

[8] Chen X, Makki K, Yen K, Pissinou N. Sensor network security: a survey. IEEE Communications Surveys & Tutorials 2009; 11 (2): 52–73. doi: 10.1109/SURV.2009.090205

[9] Diffie W, Hellman ME. New directions in cryptography. IEEE Transactions on Information Theory 1976; 22 (6): 644–654. doi: 10.1109/TIT.1976.1055638

[10] Koblitz N. Elliptic curve cryptosystems. Mathematics of Computation 1987; 48: 203–209. doi: 10.1090/S0025-5718-1987-0866109-5

[11] Miller VS. Use of elliptic curves in cryptography. In:Conference on the Theory and Application of Cryptographic Techniques (CRYPTO); Santa Barbara, California, USA; 1985. pp. 417–426.

[12] Liu Z, Großschädl J, Li L, Xu Q. Energy-efficient elliptic curve cryptography for MSP430-based wireless sensor nodes. In: Australasian Conference on Information Security and Privacy; Melbourne, Australia; 2016. pp. 94–112.

[13] Gülen U, Baktır S. Elliptic curve cryptography on constrained microcontrollers using frequency domain arithmetic. In: International Conference on Computational Science and Its Applications (ISCIS); Guimaraes, Portugal; 2014. pp. 493–506.

[14] Gouvêa CPL, Oliveira LB, López J. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. Journal of Cryptographic Engineering 2012; 2 (1): 19–29. doi: 10.1007/s13389-012-0029-z

[15] Szczechowiak P, Oliveira LB, Scott M, Collier M, Dahab R. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In: Wireless Sensor Networks (EWSN); Bologna, Italy; 2008. pp. 305–320.

[16] Gulen U, Baktir S. Elliptic-curve cryptography for wireless sensor network nodes without hardware multiplier support. Security and Communication Networks 2016; 9 (18): 4992–5002. doi: 10.1002/sec.1670

[17] MSP430x11x2, MSP430x12x2 Mixed Signal Microcontroller Datasheet, Texas Instrument Incorporated, 2004.

[18] MSP430F22x2, MSP430F22x4 Mixed Signal Microcontroller Datasheet, Texas Instrument Incorporated, 2012.

[19] MSP430G2x55 Mixed Signal Microcontroller Datasheet, Texas Instrument Incorporated, 2013.

[20] WiSense WSN1120L Datasheet, WiSense Technologies Private Limited, 2019.

[21] WiSense WSN1120CL Datasheet, WiSense Technologies Private Limited, 2019.

[22] WiSense WSN1101ANL Datasheet, WiSense Technologies Private Limited, 2019.

[23] WiSense WSN1101ACL Datasheet, WiSense Technologies Private Limited, 2019.

[24] eZ430-RF2500 Development Tool User's Guide, Texas Instrument Incorporated, 2015.

[25] Adu-Manu K S, Adam N, Tapparello C, Ayatollahi H, Heinzelman W. Energy-Harvesting Wireless Sensor Networks (EH-WSNs): A Review. ACM Trans. Sen. Netw. 2018; 14 (2): 10. doi: 10.1145/3183338

[26] Bailey DV, Paar C. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. Journal of Cryptology 2001; 14 (3): 153–176. doi: 10.1007/s001450010012

[27] Baktır S, Sunar B. Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography. In: International Symposium on Computer and Information Sciences (ISCIS); Istanbul, Turkey; 2006. pp. 991–1001.

[28] Baktır S, Kumar S, Paar C, Sunar B. A state-of-the-art elliptic curve cryptographic processor operating in the frequency domain. Mobile Networks and Applications 2007; 12 (4): 259–270. doi: 10.1007/s11036-007-0022-4

[29] Montgomery PL. Modular multiplication without trial division. Mathematics of Computation 1985; 44 (170): 519–521. doi: 10.1090/S0025-5718-1985-0777282-X

[30] Cooley JW, Tukey JW. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 1965; 19 (90): 297–301. doi: 10.2307/2003354

[31] Hankerson D, Menezes AJ, Vanstone S. Guide to Elliptic Curve Cryptography. Secaucus, NJ, USA: Springer-Verlag New York, 2004.

[32] Mentens N, Batina L, Baktır S. An elliptic curve cryptographic processor using Edwards curves and the number theoretic transform. In: International Conference on Cryptography and Information Security in the Balkans (BalkanCryptSec); Istanbul, Turkey; 2014. pp. 94–102.

[33] McEliece RJ. Finite Fields for Computer Scientists and Engineers. Norwel, MA, USA: Kluwer Academic Publishers, 2003.

[34] Lidl R, Niederreiter H. Introduction to Finite Fields and their Applications. Cambridge, UK: Cambridge University Press, 1994.

[35] Pollard JM. The fast Fourier transform in a finite field. Mathematics of Computation 1971; 25: 365–374. doi: 10.1090/S0025-5718-1971-0301966-0

[36] Baktır S. Frequency domain finite field arithmetic for elliptic curve cryptography. PhD, Worcester Polytechnic Institute, Massachusetts, USA, 2008.

[37] Baktır S, Sunar B. Optimal extension field inversion in the frequency domain. In: International Workshop on the Arithmetic of Finite Fields (WAIFI); Siena, Italy; 2008. pp. 47–61.

[38] Rader CM. Discrete convolutions via Mersenne transforms. IEEE Transactions on Computers 1972; C-21(12): 1269–1273. doi: 10.1109/T-C.1972.223497

[39] Gulen U, Baktir S. Elliptic curve cryptography for wireless sensor networks using the number theoretic transform. Sensors 2020; 20 (5): 1507. doi: 10.3390/s20051507

[40] Edwards HM. A normal form for elliptic curves. Bulletin of the American Mathematical Society 2007; 44 (3): 393–422. doi: 10.1090/S0273-0979-07-01153-6

[41] Bernstein DJ, Lange T. Faster addition and doubling on elliptic curves. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT); Kuching, Sarawak, Malaysia; 2007. pp. 29-50.

[42] MSP430x1xx Family User's Guide, Texas Instruments Incorporated, 2006.

[43] MSP430FG4618/F2013 Experimenter's Board User's Guide, Texas Instruments Incorporated, 2018.

[44] MSP Debuggers User's Guide, Texas Instruments Incorporated, 2020.