sensors

MDPI

Article

Elliptic Curve Cryptography for Wireless Sensor Networks Using the Number Theoretic Transform

Utku Gulen and Selcuk Baktir *

Computer Engineering Department, Faculty of Engineering and Natural Sciences, Bahcesehir University, 34353 Istanbul, Turkey; utku.gulen@eng.bau.edu.tr

* Correspondence: selcuk.baktir@eng.bau.edu.tr

Received: 15 January 2020; Accepted: 23 February 2020; Published: 9 March 2020

Abstract: We implement elliptic curve cryptography on the MSP430 which is a commonly used microcontroller in wireless sensor network nodes. We use the number theoretic transform to perform finite field multiplication and squaring as required in elliptic curve scalar point multiplication. We take advantage of the fast Fourier transform for the first time in the literature to speed up the number theoretic transform for an efficient realization of elliptic curve cryptography. Our implementation achieves elliptic curve scalar point multiplication in only 0.65 s and 1.31 s for multiplication of fixed and random points, respectively, and has similar or better timing performance compared to previous works in the literature.

Keywords: elliptic curve cryptography; fast Fourier transform; number theoretic transform; wireless sensor network; finite field multiplication

1. Introduction

Wireless sensor network (WSN) technology is a widespread and enabling technology that has been rapidly penetrating our daily lives. It has environmental applications such as temperature, humidity, pressure and fire monitoring [1,2], health applications such as patient monitoring [3], military applications such as enemy detection and reconnaissance [4], and applications to smart cities such as in smart grids [5]. Securing WSN applications is an important task since sensitive information they communicate should be kept confidential from malicious third parties. A sensor node, which is a single unit of a WSN, is a tiny, cheap and constrained embedded system that is usually equipped with a simple microcontroller. Cryptographic solutions are needed for applications running on constrained microcontrollers on sensor nodes [6–13]. However, due to the complex nature of cryptographic algorithms and the constrained nature of WSN nodes, e.g., their CPU power and memory size limitations, it is a challenge to implement cryptographic algorithms efficiently on WSN nodes [14–18].

Among different types of cryptosystems, symmetric key cryptography comes forward as a good choice to be used for WSNs due to its simplicity and efficiency. However, for many WSN applications, the distribution of the private key between sensor nodes remains as a problem that needs to be addressed. Public key cryptography (PKC) [19] provides a solution to the key distribution problem, yet it is considered computationally expensive for constrained WSN nodes. On the other hand, previous works prove PKC to be applicable on constrained WSN nodes for solving the key distribution problem [20–24]. Elliptic curve cryptography (ECC) [25,26] is a popular option for PKC. It requires a 160-bit or longer key to be considered secure, while the same level of security can be achieved with much longer key sizes with other PKC algorithms, e.g., a 1024-bit key is needed to achieve the same level of security using the RSA cryptosystem [27]. In this work, we realize an efficient implementation of ECC for solving the key distribution problem in WSNs. We present a novel implementation of

ECC over an *optimal extension field* [28,29] by using Edwards curves [30] and the number theoretic transform [31].

The underlying finite field has a significant influence on the performance of an ECC implementation. An optimal extension field [28,29] is a finite field $GF(p^m)$ where p is a pseudo-Mersenne prime of the form $p = 2^k - c$, k is the processor word size and $\log_2|c| < \left\lfloor \frac{k}{2} \right\rfloor$. Since the coefficients of a finite field element fit in a single processor word in an optimal extension field, no multi-precision arithmetic is needed and elliptic curve point operations can be achieved efficiently. Furthermore, an irreducible field generating polynomial of the form $P(x) = x^m - w$, where w is a small integer, is used in an optimal extension field, which allows the result of a finite field multiplication operation to be reduced efficiently with only linear complexity.

The number theoretic transform (NTT), also known as the discrete Fourier transform over a finite field, has long been known for its applications in signal processing and communications [32–37]. Recently, the use of the NTT has been explored to speed up multiplication of large operands as they appear in RSA [38], fully homomorphic encryption [39–42] and post-quantum cryptography [43–50] algorithms. However, the application of the NTT for ECC has been considered impractical and widely neglected due to much shorter operands used in ECC arithmetic. There are only a few existing ECC implementations in the literature that use the NTT and they use only partial NTT computations to achieve finite field multiplication. Efficient low-area hardware implementations of ECC are given in [51,52] where finite field arithmetic is achieved in the frequency domain and partial NTT computations are performed for the modular reduction operation after a finite field multiplication. Using the same approach, in [20] an efficient implementation of ECC is presented for constrained microcontrollers. NTT-based multiplication is in general considered efficient only for large operands and believed to be not feasible for constrained microcontrollers. However, it also has the unique advantage of requiring only a linear number of word multiplications which we take advantage of in this work. Our target platform, i.e., MSP430, is a constrained microcontroller with a 16-bit RISC architecture and used widely in WSN nodes. While there is an on-board hardware multiplier on the MSP430, a word multiplication operation using the hardware multiplier still takes 14 clock cycles. Whereas, a word addition on the same microcontroller takes only a single clock cycle. Hence, by reducing the number of performed word multiplications and exchanging them with simpler operations such as addition, the speed of finite field multiplication can be significantly improved.

With this work, we use the NTT to implement finite field multiplication. We show that NTT-based finite field multiplication is feasible for small operand sizes and can be taken advantage of to speed up ECC on WSN nodes. We introduce novel implementations of the forward and inverse NTT computations over a finite field which exploit the *Fast Fourier Transform* (FFT) [53,54]. Edwards curves, introduced in [30], are a new form for elliptic curves which provide efficient formulae for elliptic curve point arithmetic. In our ECC implementation, we use the optimal extension field $GF((2^{13}-1)^{13})$ and Edwards curves with our improved formulae for point arithmetic that take advantage of NTT-based finite field multiplication.

Our Main Contribution: We present a novel realization of ECC which uses Edwards curves for point arithmetic and the NTT for the underlying finite field multiplication and squaring operations. To the best of our knowledge, our work presents the first realization of ECC using the *Fast Fourier Transform* (FFT) [53,54] to speed up NTT computations. Our implementation achieves similar or faster timings for ECC scalar point multiplication compared to existing implementations in the literature and proves that NTT-based arithmetic is feasible for ECC implementations on constrained devices such as WSN nodes.

The paper continues as follows. In Section 2, we explain ECC using Edwards curves and also give a detailed explanation of finite field multiplication in $GF((2^{13}-1)^{13})$ using the NTT. In Section 3, we give the details of our optimized implementation of ECC point multiplication which uses our improved

Edwards curves formulae for point arithmetic and NTT-based finite field multiplication/squaring over $GF((2^{13}-1)^{13})$. In Section 4, we present our implementation results and comparisons with the existing work in the literature. Finally, Section 5 includes our conclusion.

2. Background

2.1. Finite Field Multiplication Using the NTT

In elliptic curve cryptography, a large number of multiplication and squaring operations are performed in a finite field. Elements of the finite field $GF(p^m)$ are typically represented in the time domain as polynomials of degree $m-1$ with coefficients in $GF(p)$ [55,56]. For instance, $a(x) \in GF(p^m)$ is represented as $a(x) = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$, where $a_i \in GF(p)$ for $0 \leq i \leq m-1$. Multiplication of two $GF(p^m)$ elements, e.g., $a(x)$ and $b(x)$, is achieved typically by computing the polynomial product

$$c'(x) = a(x) \cdot b(x) \bmod p$$

followed by the modular reduction

$$c(x) = c'(x) \bmod P(x)\,,$$

where $P(x)$ is the irreducible field generating polynomial. Please note that if the field generating polynomial can be selected as the binomial $x^m - 2$, the cost of the modular reduction computation becomes negligible. Due to the convolution theorem, the classical polynomial multiplication operation in the time domain, e.g., the computation of $c'(x) = a(x) \cdot b(x) \bmod p$, which has quadratic complexity, is equivalent to the simple pairwise multiplication of the corresponding frequency domain sequence coefficients which has only linear complexity [53]. Thus, the complexity of polynomial multiplication can be reduced by performing this computation in the frequency domain.

The coefficients of a $GF(p^m)$ element form a time domain sequence. To perform the polynomial multiplication of two $GF(p^m)$ elements in the frequency domain, the time domain sequences for the two $GF(p^m)$ elements should be transformed into their corresponding frequency domain sequences. This conversion is achieved by using the NTT [31]. After the polynomial multiplication operation is completed in the frequency domain, the result can be converted back to the time domain by using the inverse NTT computation. Algorithm 1 gives an overview of how polynomial multiplication can be achieved in the frequency domain.

Algorithm 1: Polynomial Multiplication in the Frequency Domain Using the NTT

Input: (a) and (b), the time domain sequences for $a(x), b(x) \in GF(p^m)$
Output: (c'), the time domain sequence for $c'(x) = a(x) \cdot b(x) \bmod p$

1 $(A) \longleftarrow NTT((a))$ //Compute the NTT of (a)
2 $(B) \longleftarrow NTT((b))$ //Compute the NTT of (b)
3 $(C') \longleftarrow PCM((A),(B))$ //Pairwise Coefficient Multiplication
4 $(c') \longleftarrow INTT((C'))$ //Compute the inverse NTT of (C')
5 **Return** (c')

A finite field element $a(x) \in GF(p^m)$ can be converted to its d-element frequency domain sequence representation, where $d \geq m$, in two steps as explained below:

1. Represent $a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$ as the following time domain sequence

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

by appending $d-m$ zeros at the end.

2. Obtain the frequency domain sequence representation $(A) = (A_0, A_1, A_2, ..., A_{d-1})$ for $a(x)$ by performing the following NTT computation over (a):

$$A_j = \sum_{i=0}^{d-1} a_i r^{ij} \bmod p,\ 0 \leq j \leq d-1\,, \tag{2}$$

where r is a d^{th} primitive root of unity.

Please note that in order to obtain the time domain sequence (a) back from the frequency domain sequence (A), the *inverse* NTT computation can be used as follows:

$$a_i = \frac{1}{d} \cdot \sum_{j=0}^{d-1} A_j r^{-ij} \bmod p\ \ ,\ 0 \leq i \leq d-1\,. \tag{3}$$

In Algorithm 1, since $c'(x) = a(x) \cdot b(x) \bmod p$ may have up to $2m-1$ coefficients, representing it in the frequency domain with a sequence of length shorter than $2m-1$ may result in its value being corrupted. Therefore, for Algorithm 1 to always generate the correct result, one should have $d \geq 2m-1$ as the NTT length. We now describe with an example the execution of Algorithm 1 for computing $c'(x) = a(x) \cdot b(x) \bmod p$ where $a(x), b(x) \in GF(p^{13})$. As described in (1), the polynomials $a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{12} x^{12}$ and $b(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{12} x^{12}$ are first converted into their corresponding 26-element time domain sequence representations as

$$(a) = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

and

$$(b) = (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\,.$$

Secondly, the NTT is applied to (a) and (b), as described in (2), to obtain the following frequency domain sequences:

$$(A) = (A_0, A_1, A_2, \ldots, A_{23}, A_{24}, A_{25})\,,$$

$$(B) = (B_0, B_1, B_2, \ldots, B_{23}, B_{24}, B_{25})\,.$$

Thirdly, the coefficients of (A) and (B) are pairwise multiplied in the frequency domain, i.e., by computing $C'_i = A_i B_i \bmod p$ for $0 \leq i \leq 25$, and thus the following sequence is obtained in the frequency domain:

$$(C') = (C'_0, C'_1, C'_2, C'_3, C'_4, C'_5, \ldots, C'_{21}, C'_{22}, C'_{23}, C'_{24}, C'_{25})\,,$$

which corresponds to $c'(x) = a(x) \cdot b(x) \bmod p$ in the time domain. Finally, the inverse NTT is applied to (C'), as described in (3), to obtain the following time domain sequence for $c'(x) = a(x) \cdot b(x) \bmod p$:

$$(c') = (c'_0, c'_1, c'_2, c'_3, c'_4, c'_5, \ldots, c'_{21}, c'_{22}, c'_{23}, c'_{24}, 0)\,. \tag{4}$$

Please note that since $c'(x)$ is a polynomial of degree 24, c'_{25} is zero and the first 25 coefficients of (c') give us the polynomial $c'(x) = a(x) \cdot b(x) \bmod p$, given as follows:

$$c'(x) = c'_0 + c'_1 x + c'_2 x^2 + \cdots + c'_{22} x^{22} + c'_{23} x^{23} + c'_{24} x^{24}\,. \tag{5}$$

As a final step in $GF(p^{13})$ multiplication, the polynomial $c'(x)$ needs to be reduced modulo the field generating polynomial by computing $c(x) = c'(x) \bmod P(x)$, which has only linear complexity.

2.2. Elliptic Curve Cryptography Using Edwards Curves

The main operation in ECC is scalar point multiplication, i.e., computing $s \cdot P$ for an integer s and a point P on the elliptic curve. ECC scalar point multiplication involves performing several ECC point addition and doubling operations. To achieve ECC point multiplication, the binary method [57] can be used, where the bits of the scalar s are scanned one bit at a time starting with the most significant bit, and for each scanned bit, a point doubling operation is performed, in addition to a point addition operation if the scanned bit is 1. However, the binary method is both inefficient and vulnerable against simple power analysis [58]. As an alternative to the binary method, and in order to help mitigate its drawbacks, the NAF4 and Comb methods can be used for ECC scalar point multiplication of random and fixed points, respectively. NAF4 and Comb require computing a significantly reduced number point additions and doublings compared to the binary method [59].

Edwards curves, proposed in [30], are a new form for elliptic curves and defined by the following equation:

$$x^2 + y^2 = c^2(1 + dx^2y^2) \, .$$

The ECC point addition of the two distinct points P_1 and P_2 on an Edwards curve is computed as

$$P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2) \, ,$$

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

The ECC point doubling operation on the point $P_1(x_1, y_1)$ on an Edwards curve is computed as

$$P_2(x_2, y_2) = 2 \cdot P_1(x_1, y_1) \, ,$$

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

The above ECC point operations can be achieved in *projective coordinates* [59–61] to avoid costly inversions. For the Edwards curve $x^2 + y^2 = c^2(1 + dx^2y^2)$, with $c = 1$, d random and $d \cdot c^4 \neq 1$, the formulae for ECC point doubling and addition in projective coordinates over prime fields are given in Algorithms 2 and 3, respectively [62].

Algorithm 2: Elliptic curve point doubling in projective coordinates over prime fields using Edwards curves [62]

Input: $P_1(X_1 : Y_1 : Z_1)$
Output: $P_2(X_2 : Y_2 : Z_2) = 2 \cdot P_1$

1 $T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow Z_1$
2 $T_4 \longleftarrow T_1 + T_2$
3 $T_1 \longleftarrow T_1^2$
4 $T_2 \longleftarrow T_2^2$
5 $T_3 \longleftarrow T_3^2$
6 $T_4 \longleftarrow T_4^2$
7 $T_3 \longleftarrow 2 \cdot T_3$
8 $T_5 \longleftarrow T_1 + T_2$
9 $T_2 \longleftarrow T_1 - T_2$
10 $T_4 \longleftarrow T_4 - T_5$
11 $T_3 \longleftarrow T_5 - T_3$
12 $T_1 \longleftarrow T_3 \cdot T_4$
13 $T_3 \longleftarrow T_3 \cdot T_5$
14 $T_2 \longleftarrow T_2 \cdot T_5$
15 $X_2 \leftarrow T_1, Y_2 \leftarrow T_2, Z_2 \leftarrow T_3$
16 **Return** $(X_2 : Y_2 : Z_2)$

Algorithm 3: Elliptic curve point addition in projective coordinates over prime fields using Edwards curves [62]

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 $T_3 \longleftarrow T_3 \cdot T_6$
3 $T_7 \longleftarrow T_1 + T_2$
4 $T_8 \longleftarrow T_4 + T_5$
5 $T_1 \longleftarrow T_1 \cdot T_4$
6 $T_2 \longleftarrow T_2 \cdot T_5$
7 $T_7 \longleftarrow T_7 \cdot T_8$
8 $T_7 \longleftarrow T_7 - T_1$
9 $T_7 \longleftarrow T_7 - T_2$
10 $T_7 \longleftarrow T_7 \cdot T_3$
11 $T_8 \longleftarrow T_1 \cdot T_2$
12 $T_8 \longleftarrow d \cdot T_8$
13 $T_2 \longleftarrow T_2 - T_1$
14 $T_2 \longleftarrow T_2 \cdot T_3$
15 $T_3 \longleftarrow (T_3)^2$
16 $T_1 \longleftarrow T_3 - T_8$
17 $T_3 \longleftarrow T_3 + T_8$
18 $T_2 \longleftarrow T_2 \cdot T_3$
19 $T_3 \longleftarrow T_3 \cdot T_1$
20 $T_1 \longleftarrow T_1 \cdot T_7$
21 $X_3 \leftarrow T_1$, $Y_3 \leftarrow T_2$, $Z_3 \leftarrow T_3$
22 **Return** $(X_3 : Y_3 : Z_3)$

3. Our ECC Implementation Using the NTT and Edwards Curves

We implement ECC over an *optimal extension field* [29,63], namely $GF(p^m)$ with the Mersenne prime field characteristic $p = 2^{13} - 1$ and the prime field extension degree $m = 13$. Please note that ECC over a prime extension field of the form $GF(p^m)$ is considered secure when the finite field is sufficiently large and its extension degree m is a prime number [59]. We select the field characteristic p such that polynomial coefficients fit in a single processor word, in our case a 16-bit word, eliminating the need for performing multiprecision arithmetic. We use the binomial $x^{13} - 2$ as the field generating polynomial which facilitates efficient modular reduction. For finite field multiplication and squaring, we use the NTT and use the approach described in Algorithm 1. For NTT computations, we use the NTT length of $d = 26$ and the 26^{th} primitive root of unity as $r = -2$. We use the FFT [53,54,64] to speed up NTT computations. For ECC point doubling and addition, we use our improved versions of Algorithms 2 and 3, respectively. Finally, we use the NAF and Comb methods, with a 4-bit window, to perform ECC scalar point multiplication with random and fixed points, respectively [59].

3.1. Finite Field Multiplication and Squaring in $GF((2^{13} - 1)^{13})$ with the NTT

As explained in Algorithm 1, polynomial multiplication, which is the main operation in $GF((2^{13} - 1)^{13})$ multiplication, can be achieved using the NTT in three stages: (1) Forward NTT Computation, (2) Pairwise Coefficient Multiplication, (3) Inverse NTT Computation. We apply the FFT [53,54,64] to speed up our NTT and inverse NTT computations.

Forward NTT for Converting $GF((2^{13} - 1)^{13})$ Elements to the Frequency Domain:

As described in (2), the frequency domain sequence representation $(A) = (A_0, A_1, A_2, \ldots, A_{25})$ of $a(x) \in GF((2^{13} - 1)^{13})$ is obtained by computing the NTT of the corresponding 26-element time domain sequence $(a) = (a_0, a_1, a_2, \ldots, a_{12}, 0, 0, \ldots, 0)$ as

$$A_j = \sum_{i=0}^{25} a_i r^{ij} \bmod p, \ 0 \leq j \leq 25\,, \tag{6}$$

where $p = 2^{13} - 1$. The above NTT computation can be optimized by applying the FFT as

$$A_j = \sum_{i=0}^{12} a_{2i}(r^2)^{ij} + r^j \sum_{i=0}^{12} a_{2i+1}(r^2)^{ij} \mod p \tag{7}$$

and

$$A_{j+13} = \sum_{i=0}^{12} a_{2i}(r^2)^{ij} - r^j \sum_{i=0}^{12} a_{2i+1}(r^2)^{ij} \mod p\,, \tag{8}$$

for $0 \leq j \leq 12$ [64]. Please note that the first summations in (7) and (8) are the same NTT computation. Likewise, the second summations in (7) and (8) are also the same NTT computation. Both NTT computations are of length 13. Hence, using the FFT, the computation in (6), which is a 26-element NTT computation, is reduced to the computation of roughly two 13-element NTT computations. Since $a_i = 0$ for $13 \leq i \leq 25$, we compute the summations in (7) and (8) only for i running from 0 to 6 in the first NTT computation, and from 0 to 5 in the second. Our optimized algorithm for computing the forward NTT of $a(x) \in GF((2^{13}-1)^{13})$ on the MSP430 is given in Algorithm 4.

Algorithm 4: Forward NTT Computation on the MSP430 Using the FFT

Input: $(a) = (a_0, a_1, a_2, \ldots, a_{10}, a_{11}, a_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, the time domain sequence for $a(x) \in GF(p^{13})$ where $p = 2^{13}-1$. $R_0, R_1 ... R_6$ are microcontroller registers. $E_0, E_1 ... E_6$ and $O_0, O_1 ... O_6$ are variables.

Output: $(A) = (A_0, A_1, A_2 ... A_{25})$, the frequency domain sequence for $a(x)$.

1 **for** $i \leftarrow 0$ **to** 6 **do**
2 $\quad R_i \longleftarrow a_{2i}$
3 **end**
4 $E_0 \longleftarrow R_0 + R_1 + ... + R_6 \mod p$
5 **for** $i \leftarrow 1$ **to** 12 **do**
6 $\quad$ **for** $j \leftarrow 1$ **to** 6 **do**
7 $\quad\quad R_j \longleftarrow R_j \cdot 2^{2j} \mod p$
8 $\quad$ **end**
9 $\quad E_i \longleftarrow R_0 + R_1 + ... + R_6 \mod p$
10 **end**
11 **for** $i = 0$ **to** 5 **do**
12 $\quad R_i \longleftarrow a_{2i+1}$
13 **end**
14 $O_0 \longleftarrow R_0 + R_1 + ... + R_5 \mod p$
15 $A_0 \longleftarrow E_0 + O_0 \mod p$
16 $A_{13} \longleftarrow E_0 - O_0 \mod p$
17 **for** $i = 1$ **to** 12 **do**
18 $\quad$ **for** $j = 0$ **to** 5 **do**
19 $\quad\quad R_j \leftarrow R_j \cdot 2^{2j+1} \mod p$
20 $\quad$ **end**
21 $\quad O_i \leftarrow R_0 + R_1 + ... + R_5 \mod p$
22 $\quad A_i \leftarrow E_i + (-1)^i \cdot O_i \mod p$
23 $\quad A_{i+13} \longleftarrow E_i - (-1)^i \cdot O_i \mod p$
24 **end**
25 **Return** $(A_0, A_1, A_2 ... A_{25})$

We implement Algorithm 4 with an assembly routine and optimize it by using microcontroller registers as much as possible to minimize the number of memory read/write operations. For the additions in lines 4, 9, 14 and 21, there is no need to do modular reduction after every addition. We reduce the number of modular reductions by accumulating the sums and deferring modular reduction as much as possible.

Multiplication of a $GF(2^{13}-1)$ element with a power of 2, e.g., in lines 7 and 19 of Algorithm 4, corresponds to a bitwise left rotation of the $GF(2^{13}-1)$ element. For instance, for $R \in GF(2^{13}-1)$, $2^j R \mod 2^{13}-1$ can be computed by rotating the bits of R by $j \mod 13$ bits to the left. We realize multiplications of $GF(2^{13}-1)$ elements with powers of 2 with an optimized assembly routine.

Please note that for multiplying two distinct $GF((2^{13}-1)^{13})$ elements with Algorithm 1, Algorithm 4 needs to be executed twice, i.e., once for each input operand to obtain its frequency domain sequence representation. On the other hand, for squaring a $GF((2^{13}-1)^{13})$ element, Algorithm 4 needs to be executed only once for the single input operand. Hence, squaring using Algorithm 1 is faster than multiplication.

Pairwise Coefficient Multiplication of $GF((2^{13}-1)^{13})$ Elements in the Frequency Domain:

Polynomial multiplication of two $GF((2^{13}-1)^{13})$ elements can be achieved in the frequency domain with only linear complexity by multiplying pairwise their frequency domain sequence coefficients. Let $a(x), b(x) \in GF((2^{13}-1)^{13})$, and let $(A) = (A_0, A_1, \cdots, A_{25})$ and $(B) =$

$(B_0, B_1, \cdots, B_{25})$ be their 26-element frequency domain sequence representations obtained using Algorithm 4. The following 26 pairwise coefficient multiplications generate the 26-element frequency domain sequence representation (C′) of the product $c'(x) = a(x) \cdot b(x) \bmod p$:

$$C'_i = A_i B_i \mod p\,, \quad 0 \leq i \leq 25\,. \tag{9}$$

The frequency domain sequence (C') can be converted back to the time domain, by applying the inverse NTT, to give us the coefficients of the polynomial product $c'(x) = a(x) \cdot b(x) \bmod p$.

The multiplications in (9) are the only $GF(p)$ multiplications required for computing the polynomial product $c'(x) = a(x) \cdot b(x) \bmod p$ in the NTT-based multiplication approach. Please note that only 26 coefficient multiplications are performed here, which is significantly less than the 169 coefficient multiplications required in the classical schoolbook method for multiplication.

Inverse NTT for Converting the Frequency Domain Product to a Time Domain $GF((2^{13}-1)^{13})$ Element:

As described in (3), the time domain sequence representation $(c') = (c'_0, c'_1, c'_2, \ldots, c'_{25})$ of $c'(x) = a(x) \cdot b(x) \bmod p$ can be obtained by computing the inverse NTT of the corresponding 26-element frequency domain sequence $(C') = (C'_0, C'_1, C'_2, \ldots, C'_{23}, C'_{24}, C'_{25})$ as follows

$$c'_j = \frac{1}{26} \sum_{i=0}^{25} C'_i r^{-ij} \mod p\,, \quad 0 \leq j \leq 25\,. \tag{10}$$

The above inverse NTT computation can be optimized by applying the inverse FFT as

$$c'_j = \sum_{i=0}^{12} C'_{2i} (r^2)^{-ij} + r^{-j} \sum_{i=0}^{12} C'_{2i+1} (r^2)^{-ij} \mod p \tag{11}$$

and

$$c'_{j+13} = \sum_{i=0}^{12} C'_{2i} (r^2)^{-ij} - r^{-j} \sum_{i=0}^{12} C'_{2i+1} (r^2)^{-ij} \mod p\,, \tag{12}$$

for $0 \leq j \leq 12$ [64]. Please note that the first summations in (11) and (12) are the same inverse NTT computation. Likewise, the second summations in (11) and (12) are the same inverse NTT computation. Furthermore, both inverse NTT computations are of length 13. Hence, using the inverse FFT, the computation of (10), which is a 26-element inverse NTT computation, is reduced to the computation of roughly two 13-element inverse NTT computations. Since $c'_{25} = 0$, we compute the second summations in (11) and (12) only for i running from 0 to 11. Our inverse FFT algorithm for computing the inverse NTT of (C'), the frequency domain sequence corresponding to $c'(x) = a(x) \cdot b(x) \bmod p$, and for obtaining $c(x) = c'(x) \bmod P(x)$, where $P(x) = x^{13} - 2$, is given in Algorithm 5.

Please note that, unlike in the inverse NTT computation in Algorithm 1, in Algorithm 5 (lines 35 − 42) we embed the modular reduction of $c'(x) = a(x) \cdot b(x) \bmod p$ by the field generating polynomial $P(x) = x^{13} - 2$. Hence, for $a(x), b(x) \in GF((2^{13}-1)^{13})$, while the output of Algorithm 1 is a polynomial of degree 24 (with 25 coefficients in $GF(2^{13}-1)$), the output of Algorithm 5 is an element of $GF((2^{13}-1)^{13})$ with 13 coefficients.

Similar to our implementation of Algorithm 4, we implement Algorithm 5 with an assembly routine and optimize it by using microcontroller registers exhaustively to minimize the number of memory read/write operations. We reduce the number of performed modular reductions in lines 4, 9, 14, 19, 24, 29, 34 and 40 of Algorithm 5 by accumulating the sums and deferring the modular reduction computation as much as possible.

Division of a $GF(2^{13}-1)$ element by a power of 2, e.g., in lines 7, 17, 27 and 38 of Algorithm 5, can be achieved with a bitwise right rotation. For instance, for $R \in GF(2^{13}-1)$, $R/2^j \bmod 2^{13}-1$ can

be computed by rotating the bits of R by j mod 13 bits to the right. We realize this bitwise rotation operation with an optimized assembly routine.

Algorithm 5: Inverse NTT Computation on the MSP430 Using the FFT

Input: $(C) = (C_0, C_1, C_2 ... C_{25})$, the frequency domain sequence representation of $c(x) = a(x) \cdot b(x) \bmod p$, where $a(x), b(x) \in GF(p^{13})$ and $p = 2^{13} - 1$. $R_0, R_1 ... R_6$ are microcontroller registers. $E_0, E_1 ... E_6$ and $O_0, O_1 ... O_6$ are variables.

Output: $c(x) \bmod P(x) \in GF(p^{13})$, where $P(x) = x^{13} - 2$ is the field generating polynomial.

1 **for** $i \leftarrow 0$ **to** 6 **do**
2 $\quad R_i \longleftarrow c_{2i}$
3 **end**
4 $E_0 \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
5 **for** $i \leftarrow 1$ **to** 12 **do**
6 $\quad$ **for** $j \leftarrow 1$ **to** 6 **do**
7 $\quad\quad R_j \longleftarrow R_j / 2^{2j} \bmod p$
8 $\quad$ **end**
9 $\quad E_i \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
10 **end**
11 **for** $i \leftarrow 7$ **to** 12 **do**
12 $\quad R_{i-7} \longleftarrow C_{2i+1}$
13 **end**
14 $E_0 \leftarrow E_0 + R_0 + R_1 + ... + R_5 \bmod p$
15 **for** $i \leftarrow 1$ **to** 12 **do**
16 $\quad$ **for** $j \leftarrow 7$ **to** 12 **do**
17 $\quad\quad R_{j-7} \longleftarrow R_{j-7} / 2^{2j} \bmod p$
18 $\quad$ **end**
19 $\quad E_i \leftarrow E_i + R_0 + R_1 + ... + R_5 \bmod p$
20 **end**
21 **for** $i \leftarrow 0$ **to** 6 **do**
22 $\quad R_i \longleftarrow c_{2i+1}$
23 **end**
24 $O_0 \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
25 **for** $i \leftarrow 1$ **to** 12 **do**
26 $\quad$ **for** $j \leftarrow 0$ **to** 6 **do**
27 $\quad\quad R_j \longleftarrow R_j / 2^{2j+1} \bmod p$
28 $\quad$ **end**
29 $\quad O_i \leftarrow R_0 + R_1 + ... + R_6 \bmod p$
30 **end**
31 **for** $i \leftarrow 7$ **to** 12 **do**
32 $\quad R_{i-7} \longleftarrow c_{2i+1}$
33 **end**
34 $O_0 \leftarrow O_0 + R_0 + R_1 + ... + R_5 \bmod p$
35 $c_0 \longleftarrow (3 \cdot E_0 - O_0) \cdot 7876 \bmod p$
36 **for** $i \leftarrow 1$ **to** 12 **do**
37 $\quad$ **for** $j \leftarrow 7$ **to** 12 **do**
38 $\quad\quad R_{j-7} \longleftarrow R_{j-7} / 2^{2j+1} \bmod p$
39 $\quad$ **end**
40 $\quad O_i \leftarrow O_i + R_0 + R_1 + ... + R_5 \bmod p$
41 $\quad c_i \longleftarrow (3 \cdot E_i + O_i \cdot (-1)^{i-1}) \cdot 7876 \bmod p$
42 **end**
43 **Return** $c_0 + c_1 x + c_2 x^2 + ... c_{12} x^{12}$

3.2. ECC Point Arithmetic with NTT Based Multiplication and Squaring

For ECC operations, we use the Edwards curve $x^2 + y^2 = c^2(1 + dx^2y^2)$, with $c = 1$, d random and $d \cdot c^4 \neq 1$, over the 169-bit prime extension field $GF((2^{13} - 1)^{13})$, and use our optimized versions of the elliptic curve point addition and doubling formulae given in Algorithms 2 and 3. We improve Algorithms 2 and 3 by taking advantage of NTT-based multiplication and squaring operations. Our improved algorithms are given in Algorithms 6 and 7.

Algorithm 6: Elliptic curve point doubling in projective coordinates over prime fields using Edwards curves and NTT-based multiplication/squaring

Input: $P = (X_1 : Y_1 : Z_1)$, R_1 and R_2 are temporary registers.

Output: $2P = (X_2 : Y_2 : Z_2)$

1 $R_1 \longleftarrow NTT(X_1) + NTT(Y_1)$ $\quad //NTTs\ stored$
2 $R_1 \longleftarrow \mathbf{R_1}^2$
3 $X_1 \longleftarrow \mathbf{X_1}^2$
4 $Y_1 \longleftarrow \mathbf{Y_1}^2$
5 $Z_1 \longleftarrow 2{Z_1}^2$
6 $R_2 \longleftarrow X_1 + Y_1$
7 $Y_1 \longleftarrow X_1 - Y_1$
8 $R_1 \longleftarrow R_1 - R_2$
9 $Z_1 \longleftarrow R_2 - Z_1$
10 $X_2 \longleftarrow Z_1 \cdot R_1$ $\quad //NTT\ of\ Z_1\ stored$
11 $Z_2 \longleftarrow \mathbf{Z_1} \cdot R_2$ $\quad //NTT\ of\ R_2\ stored$
12 $Y_2 \longleftarrow Y_1 \cdot \mathbf{R_2}$
13 **Return** $(X_2 : Y_2 : Z_2)$

Algorithm 6 is a reordered and optimized version of Algorithm 2. It takes advantage of NTT-based finite field multiplication and squaring computations. In line 1 of the algorithm, the NTTs of X_1 and

Y_1 are computed, and then added in the frequency domain to find the NTT of $R_1 = X_1 + Y_1$. The computed NTTs of X_1, Y_1 and R_1 are stored. The stored frequency domain representations of X_1, Y_1 and R_1 are used in lines $2-4$ (marked bold) for the three finite field squarings. Please note that for these three finite field squarings, a total number of only two forward NTT computations are performed, i.e., $NTT(X_1)$ and $NTT(Y_1)$ in line 1, instead of three as required in Algorithm 1. Furthermore, in line 10, the computed NTT of Z_1 is stored and reused in line 11 (marked bold). Similarly, in line 11, the computed NTT of R_2 is stored and reused in line 12 (marked bold). Please note that each time the stored result of an NTT computation is reused, a forward NTT computation is saved in Algorithm 1.

Algorithm 7: Elliptic curve point addition in projective coordinates over prime fields using Edwards curves and NTT-based multiplication/squaring

Input: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$, R_1 and R_2 are temporary registers.
Output: $P + Q = (X_3 : Y_3 : Z_3)$

1 $Z_1 \longleftarrow Z_1 \cdot Z_2$
2 $R_1 \longleftarrow NTT(X_1) + NTT(Y_1)$ *//NTTs stored*
3 $R_2 \longleftarrow NTT(X_2) + NTT(Y_2)$ *//NTTs stored*
4 $R_1 \longleftarrow \mathbf{R_1} \cdot \mathbf{R_2}$
5 $X_1 \longleftarrow \mathbf{X_1} \cdot \mathbf{X_2}$
6 $Y_1 \longleftarrow \mathbf{Y_1} \cdot \mathbf{Y_2}$
7 $R_1 \longleftarrow R_1 - X_1$
8 $R_1 \longleftarrow R_1 - Y_1$
9 $R_1 \longleftarrow R_1 \cdot Z_1$ *//NTT of Z_1 stored*
10 $R_2 \longleftarrow d \cdot X_1 \cdot Y_1$ *//NTTs of X_1 and Y_1 stored*
11 $Y_1 \longleftarrow \mathbf{Y_1} - \mathbf{X_1}$ *//NTT of Y_1 stored*
12 $Y_1 \longleftarrow \mathbf{Y_1} \cdot \mathbf{Z_1}$
13 $Z_1 \longleftarrow \mathbf{Z_1}^2$
14 $X_1 \longleftarrow Z_1 - R_2$
15 $Z_1 \longleftarrow Z_1 + R_2$
16 $Y_3 \longleftarrow Y_1 \cdot Z_1$ *//NTT of Z_1 stored*
17 $Z_3 \longleftarrow \mathbf{Z_1} \cdot X_1$ *//NTT of X_1 stored*
18 $X_3 \longleftarrow \mathbf{X_1} \cdot R_1$
19 **Return** $(X_3 : Y_3 : Z_3)$

Algorithm 7 is a reordered and optimized version of Algorithm 3. It takes advantage of NTT-based finite field multiplication and squaring computations. In lines $2-3$ of the algorithm, the NTTs of X_1, X_2, Y_1 and Y_2 are computed and stored. Only two addition operations are performed in the frequency domain on the stored NTTs to readily obtain the NTTs of $R_1 = X_1 + Y_1$ and $R_2 = X_2 + Y_2$. The NTTs of R_1 and R_2 are also stored. The stored NTTs of R_1, R_2, X_1, X_2, Y_1 and Y_2 are readily used in lines $4-6$ (denoted with bold color) for the three finite field multiplication computations. Thus, for three finite field multiplications, a total number of only four forward NTT computations are performed, instead of six as required in Algorithm 1. Furthermore, in lines $11-13$ of the algorithm, the stored NTTs of Y_1, X_1 and Z_1 are reused (marked bold). Similarly, in line 16, the NTT of Z_1 is computed and stored. The stored NTT of Z_1 is reused in line 17 (marked bold). Likewise, in line 17, the NTT of X_1 is computed and stored, and reused in line 18 (marked bold).

4. Implementation Results

We use Texas instrument's MSP430 microcontroller, which is commonly used in wireless sensor nodes, and select version MSP430F1611 [65]. Our target device, MSP430*F*1611, is a 1-series low power microcontroller which runs at 8 MHz clock frequency, and has a 48 kB flash memory in addition to a 10 kB RAM. We develop our code in the C language but also use the assembly language for computationally intensive and/or commonly performed operations. We use the IAR Workbench IDE as our development environment [66]. We obtain timings by using the IAR Workbench IDE's clock cycle counter in debug mode. The detailed timing figures for our ECC implementations are given in Table 1.

In Table 2 and Figure 1, we present our timings for ECC random point multiplication on the MSP430F1611 as well as the timings of the related work in the literature on the same microcontroller. Liu et al.'s work, which uses a 159-bit Montgomery curve, presents the fastest timing for random point multiplication on the MSP430 microcontroller [67]. They use the Montgomery ladder method and achieve random point multiplication in $3,460,000$ clock cycles which is equivalent to 0.48 s at 8 MHz clock frequency. Gouvêa et al.'s work, which uses the 160-bit curve secp160r1 that has a slightly

smaller elliptic curve group order than ours, achieves ECC random point multiplication in 0.58 s [68]. Our previous ECC implementation over $GF((2^{13}-1)^{13})$ on the MSP430F149, a similar microcontroller to the MSP430F1611, achieves random point multiplication in 1.55 s [20]. Please note that our ECC random point multiplication implementation in this work, which exploits the NTT-based finite field multiplication/squaring and the FFT, is more than 18% faster than our previous implementation on the same elliptic curve. Wang et al.'s implementation of elliptic curve random point multiplication over a 160-bit elliptic curve has a timing value of 3.51 s which is significantly worse than our timing result [69]. In a later work, the same authors improve their timing to 1.60 s; however, their new implementation is still 22% slower than our work [24]. Please note that the timing figure for our ECC implementation is for a 169-bit elliptic curve with a higher security level, whereas the others' works use the smaller ordered 159-bit and 160-bit elliptic curves.

Table 1. Our timings for $GF((2^{13}-1)^{13})$ arithmetic and ECC operations on MSP430F1611 @ 8 MHz.

Operation		Timing
Forward NTT	(Algorithm 4)	0.21 ms
Inverse NTT	(Algorithm 5)	0.44 ms
NTT Squaring		0.78 ms
NTT Multiplication		1.02 ms
ECC Point Doubling	(Algorithm 6)	5.63 ms
ECC Point Addition	(Algorithm 7)	9.64 ms
NAF4 ECC Random Point Multiplication		1.31 s
Comb4 ECC Fixed Point Multiplication		0.65 s

Table 2. Timings for ECC random point multiplication.

Microcontroller	Field	Method	Timing
MSP430F1611 @ 8 MHz [67]	F_{P159}	Montgomery ladder	0.48 s
MSP430F1611 @ 8 MHz [68]	F_{P160}	4NAF	0.58 s
MSP430F1611 @ 8 MHz **(This work)**	$F_{(2^{13}-1)^{13}}$	4NAF	1.31 s
MSP430F149 @ 8 MHz [20]	$F_{(2^{13}-1)^{13}}$	4NAF	1.55 s
MSP430F1611 @ 8 MHz [24]	F_{P160}	-	1.60 s
MSP430F1611 @ 8 MHz [69]	F_{P160}	-	3.51 s

In Table 3 and Figure 2, we present our timings for ECC fixed point multiplication on the MSP430F1611 as well as the timings of the related work in the literature on the same microcontroller. Liu et al.'s work, which uses a 159-bit twisted Edwards curve, presents the fastest timing for fixed point multiplication on the MSP430 microcontroller [67]. They use the Comb method and twisted Edwards curves to achieve fixed point multiplication in $1,920,000$ clock cycles which is equivalent to 0.24 s at 8 MHz clock frequency. Gouvêa et al.'s work, which uses the 160-bit elliptic curve secp160r1 and the 4NAF method, achieves ECC fixed point multiplication in 0.52 s [68]. Liu et al.'s timing for 160-bit ECDSA signature generation (considered to have around the same timing value as elliptic curve fixed point multiplication) is 1.58 s, which is twice slower than our implementation that uses a larger 169-bit elliptic curve. Wang et al.'s work on the same microcontroller achieves elliptic curve fixed point multiplication in 1.44 s over a 160-bit elliptic curve. Wenger et al.'s implementation of elliptic curve fixed point multiplication on a 160-bit elliptic curve takes $8,779,931$ clock cycles which is equivalent to 1.09 s at 8 MHz clock frequency [70]. Szczechowiak et al.'s work achieves elliptic curve fixed point multiplication in 0.72 s using a 160-bit elliptic curve over a prime field [22] and in 1.04 s using a 163-bit elliptic curve over a binary field [22]. Our timing for elliptic curve fixed point multiplication over a larger ordered 169-bit elliptic curve is slightly better than their results. Please note that the timing figure for our ECC implementation is for a 169-bit elliptic curve with a higher security level, whereas the others' works use the the smaller ordered 159-bit, 160-bit and 163-bit elliptic curves.

Table 3. Timings for ECC fixed point multiplication.

Microcontroller	Field	Method	Timing
MSP430F1611 @ 8 MHz [67]	F_{P159}	Comb	0.24 s
MSP430F1611 @ 8 MHz [68]	F_{P160}	4NAF	0.52 s
MSP430F1611 @ 8 MHz **(This work)**	$F_{(2^{13}-1)^{13}}$	Comb	0.65 s
MSP430F1611 @ 8 MHz [22]	F_{P160}	Comb	0.72 s
MSP430F1611 @ 8 MHz [22]	$F_{2^{163}}$	Comb	1.04 s
MSP430F1611 @ 8 MHz [70]	F_{P160}	-	1.09 s
MSP430F1611 @ 8 MHz [24]	F_{P160}	Sliding Window	1.44 s
MSP430F1611 @ 8 MHz [71]	F_{P160}	Sliding window	1.58 s

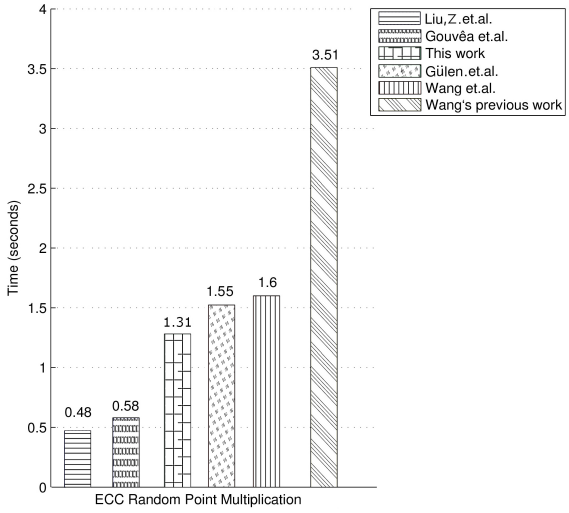

Figure 1. Comparison for random point multiplication timings.

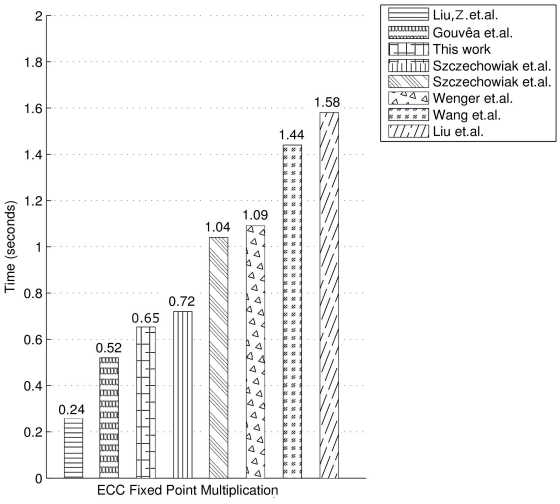

Figure 2. Comparison for fixed point multiplication timings.

5. Conclusions

We implemented ECC on the MSP430 microcontroller, which is a widely used microcontroller in WSNs, by using Edwards curves for point arithmetic and the number theoretic transform for the underlying finite field multiplication and squaring operations. In our work, we realized a

novel implementation of the fast Fourier transform over $GF((2^{13}-1)^{13})$ to speed up the number theoretic transform on the MSP430 microcontroller. Furthermore, for the point addition and doubling operations on Edwards curves, we introduced optimized formulae where some arithmetic operations are eliminated by taking advantage of the number theoretic transform. Our ECC implementation resulted in comparable or better timing values than the existing work in the literature on the same microcontroller. Please note that the techniques introduced in this paper can be applied to ECC implementations over other elliptic curves with more efficient formulae for point arithmetic. We identify the application of the introduced techniques to ECC implementations on other elliptic curves, such as Montgomery curves or twisted Edwards curves, as directions for future research.

Author Contributions: Conceptualization, S.B.; Methodology, S.B. and U.G.; Software, U.G.; Validation, U.G.; Investigation, S.B. and U.G.; Writing—original draft preparation, S.B. and U.G.; Writing—review and editing, S.B. and U.G.; Supervision, S.B. All authors have read and agreed to the published version of the manuscript

Funding: This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No. 215E208.

Acknowledgments: The authors would like to thank the anonymous reviewers whose valuable comments and suggestions helped improve the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pottie, G.J.; Kaiser, W.J. Wireless integrated network sensors. *Commun. ACM* **2000**, *43*, 51–58. [CrossRef]
2. Chong, C.Y.; Kumar, S.P. Sensor networks: Evolution, opportunities, and challenges. *Proc. IEEE* **2003**, *91*, 1247–1256. [CrossRef]
3. Baronti, P.; Pillai, P.; Chook, V.W.; Chessa, S.; Gotta, A.; Hu, Y.F. Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. *Comput. Commun.* **2007**, *30*, 1655–1695. [CrossRef]
4. Yick, J.; Mukherjee, B.; Ghosal, D. Wireless sensor network survey. *Comput. Netw.* **2008**, *52*, 2292–2330. [CrossRef]
5. De Souza, R.W.R.; Moreira, L.R.; Rodrigues, J.J.P.C.; Moreira, R.R.; de Albuquerque, V.H.C. Deploying wireless sensor networks–based smart grid for smart meters monitoring and control. *Int. J. Commun. Syst.* **2018**, *31*, e3557. [CrossRef]
6. Perrig, A.; Szewczyk, R.; Tygar, J.D.; Wen, V.; Culler, D.E. SPINS: Security protocols for sensor networks. *Wirel. Netw.* **2002**, *8*, 521–534. [CrossRef]
7. Wang, Y.; Attebury, G.; Ramamurthy, B. A survey of security issues in wireless sensor networks. *IEEE Commun. Surv. Tutorials.* **2006**, *8*, 2–23. [CrossRef]
8. Chen, X.; Makki, K.; Yen, K.; Pissinou, N. Sensor network security: A survey. *IEEE Commun. Surv. Tutor.* **2009**, *11*, 52–73. [CrossRef]
9. Ozdemir, S.; Xiao, Y. Secure data aggregation in wireless sensor networks: A comprehensive overview. *Comput. Netw.* **2009**, *53*, 2022–2037. [CrossRef]
10. Roman, R.; Alcaraz, C.; Lopez, J. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Netw. Appl.* **2007**, *12*, 231–244. [CrossRef]
11. Li, M.; Lou, W.; Ren, K. Data security and privacy in wireless body area networks. *IEEE Wirel. Commun.* **2010**, *17*, 51–58. [CrossRef]
12. Yu, Y.; Li, K.; Zhou, W.; Li, P. Trust mechanisms in wireless sensor networks: Attack analysis and countermeasures. *J. Netw. Comput. Appl.* **2012**, *35*, 867–880. [CrossRef]
13. He, D.; Kumar, N.; Chilamkurti, N. A secure temporal-credential-based mutual authentication and key agreement scheme with pseudo identity for wireless sensor networks. *Inf. Sci.* **2015**, *321*, 263–277. [CrossRef]
14. Akyildiz, I.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless sensor networks: A survey. *Comput. Netw.* **2002**, *38*, 393–422. [CrossRef]
15. Buratti, C.; Conti, A.; Dardari, D.; Verdone, R. An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors* **2009**, *9*, 6869–6896. [CrossRef]

16. Chandrakasan, A.; Amirtharajah, R.; Cho, S.; Goodman, J.; Konduri, G.; Kulik, J.; Rabiner, W.; Wang, A. Design considerations for distributed microsensor systems. In Proceedings of the IEEE 1999 Custom Integrated Circuits Conference, San Diego, CA, USA, 19 May 1999; pp. 279–286.
17. Zhou, Y.; Fang, Y.; Zhang, Y. Securing wireless sensor networks: A survey. *IEEE Commun. Surv. Tutor.* **2008**, *10*, 6–28. [CrossRef]
18. Feng, D.; Jiang, C.; Lim, G.; Cimini, L.J.; Feng, G.; Li, G.Y. A survey of energy-efficient wireless communications. *IEEE Commun. Surv. Tutor.* **2013**, *15*, 167–178. [CrossRef]
19. Diffie, W.; Hellman, M. New directions in cryptography. *Inf. Theory IEEE Trans.* **1976**, *22*, 644–654. [CrossRef]
20. Gülen, U.; Baktır, S. Elliptic Curve Cryptography on Constrained Microcontrollers Using Frequency Domain Arithmetic. In *International Conference on Computational Science and Its Applications*; Springer: New York, NY, USA, 2014; pp. 493–506.
21. Gulen, U.; Baktir, S. Elliptic-curve cryptography for wireless sensor network nodes without hardware multiplier support. *Secur. Commun. Netw.* **2016**, *9*, 4992–5002. [CrossRef]
22. Szczechowiak, P.; Oliveira, L.B.; Scott, M.; Collier, M.; Dahab, R. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Wireless Sensor Networks*; Springer: New York, NY, USA, 2008; pp. 305–320.
23. Gouvêa, C.P.L.; López, J. Software implementation of pairing-based cryptography on sensor networks using the MSP430 microcontroller. In *Progress in Cryptology-INDOCRYPT 2009*; Springer: New York, NY, USA, 2009; pp. 248–262.
24. Wang, H.; Li, Q. Efficient implementation of public key cryptosystems on mote sensors (short paper). In *Information and Communications Security*; Springer: New York, NY, USA, 2006; pp. 519–528.
25. Koblitz, N. Elliptic Curve Cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
26. Miller, V. Uses of Elliptic Curves in Cryptography. In *Advances in Cryptology—CRYPTO '85*; Williams, H.C., Ed.; Springer: Berlin, Germany, 1986; Volume LNCS 218, pp. 417–426.
27. Rivest, R.L.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
28. Bailey, D.V.; Paar, C. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In *Advances in Cryptology—CRYPTO '98*; Krawczyk, H., Ed.; Springer: Berlin, Germany, 1998; Volume LNCS 1462, pp. 472–485.
29. Bailey, D.V.; Paar, C. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *J. Cryptol.* **2001**, *14*, 153–176. [CrossRef]
30. Edwards, H.M. A normal form for elliptic curves. *Bull. Am. Math. Soc.* **2007**, *44*, 393–422. [CrossRef]
31. Pollard, J.M. The Fast Fourier Transform in a Finite Field. *Math. Comput.* **1971**, *25*, 365–374. [CrossRef]
32. Lawrence, B. Application of the fast Fourier number theoretic transform to radar. In Proceedings of the 1991 IEEE National Radar Conference, Los Angeles, CA, USA, 12–13 March 1991; pp. 137–141. [CrossRef]
33. Nussbaumer, H.J. Digital Filtering Using Complex Mersenne Transforms. *IBM J. Res. Dev.* **1976**, *20*, 498–504. [CrossRef]
34. Shakaff, A.Y.M.; Pajayakrit, A.; Holt, A.G.J. Practical implementations of block-mode image filters using the Fermat number transform on a microprocessor-based system. *IEE Proc. G-Electron. Circuits Syst.* **1988**, *135*, 141–154. [CrossRef]
35. Xu, S.; Dai, L.; Lee, S.C. Autocorrelation analysis of speech signals using Fermat number transform (FNT). *IEEE Trans. Signal Proc.* **1992**, *40*, 1910–1914. [CrossRef]
36. Madre, G.; Baghious, E.H.; Azou, S.; Burel, G. Fast pitch modelling for CS-ACELP coder using fermat number transforms. In Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology, Darmstadt, Germany, 17 December 2003; pp. 765–768. [CrossRef]
37. Toivonen, T.; Heikkilä, J. Video filtering with Fermat number theoretic transforms using residue number system. *IEEE Trans. Circuits Syst. Video Technol.* **2006**, *16*, 92–101. [CrossRef]
38. Wang, W.; Huang, X. A novel fast modular multiplier architecture for 8192-bit RSA cryposystem. In Proceedings of the 2013 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 10–12 September 2013; pp. 1–5.
39. Cao, X.; Moore, C.; O'Neill, M.; O'Sullivan, E.; Hanley, N. Optimised Multiplication Architectures for Accelerating Fully Homomorphic Encryption. *IEEE Trans. Comput.* **2016**, *65*, 2794–2806. [CrossRef]

40. Cao, X.; Moore, C.; O'Neill, M.; Hanley, N.; O'Sullivan, E. High-Speed Fully Homomorphic Encryption Over the Integers. In *Financial Cryptography and Data Security*; Böhme, R., Brenner, M., Moore, T., Smith, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 169–180.
41. Van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology–EUROCRYPT 2010*; Gilbert, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–43.
42. Feng, X.; Li, S. Accelerating an FHE Integer Multiplier Using Negative Wrapped Convolution and Ping-Pong FFT. *IEEE Trans. Circuits Syst. II Express Briefs* **2019**, *66*, 121–125. [CrossRef]
43. Pöppelmann, T.; Güneysu, T. Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware. In *Progress in Cryptology–LATINCRYPT 2012*; Hevia, A., Neven, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 139–158.
44. Lyubashevsky, V.; Peikert, C.; Regev, O. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* **2013**, *60*, 43:1–43:35. [CrossRef]
45. Güneysu, T.; Lyubashevsky, V.; Pöppelmann, T. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In *Cryptographic Hardware and Embedded Systems–CHES 2012*; Prouff, E., Schaumont, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 530–547.
46. Aguilar-Melchor, C.; Barrier, J.; Guelton, S.; Guinet, A.; Killijian, M.O.; Lepoint, T. NFLlib: NTT-Based Fast Lattice Library. In *Topics in Cryptology-CT-RSA 2016*; Sako, K., Ed.; Springer: Cham, Switzerland, 2016; pp. 341–356.
47. Liu, Z.; Pöppelmann, T.; Oder, T.; Seo, H.; Roy, S.S.; Güneysu, T.; Groädl, J.; Kim, H.; Verbauwhede, I. High-Performance Ideal Lattice-Based Cryptography on 8-Bit AVR Microcontrollers. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 117:1–117:24. [CrossRef]
48. Sinha Roy, S.; Verbauwhede, I. Ring-LWE Public Key Encryption Processor. In *Lattice-Based Public-Key Cryptography in Hardware*; Springer: Singapore, 2020; pp. 65–81.
49. Seo, H.; Liu, Z.; Park, T.; Kwon, H.; Lee, S.; Kim, H. Secure Number Theoretic Transform and A Speed Record for Ring-LWE Encryption on Embedded Processors. In *Information Security and Cryptology–ICISC 2017*; Kim, H., Kim, D.C., Eds.; Springer: Cham, Switzerland, 2018; pp. 175–188.
50. Feng, X.; Li, S.; Xu, S. RLWE-Oriented High-Speed Polynomial Multiplier Utilizing Multi-lane Stockham NTT Algorithm. *IEEE Trans. Circuits Syst. II Express Briefs* **2019**, 1. [CrossRef]
51. Baktir, S.; Kumar, S.; Paar, C.; Sunar, B. A State-of-the-art Elliptic Curve Cryptographic Processor Operating in the Frequency Domain. *Mobile Netw. Appl.* **2007**, *12*, 259–270. [CrossRef]
52. 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*; Springer: New York, NY, USA, 2014; pp. 94–102.
53. Burrus, C.S.; Parks, T.W. *DFT/FFT and Convolution Algorithms.*; John Wiley & Sons: London, UK, 1985.
54. Tolimieri, R.; An, M.; Lu, C. *Algorithms for Discrete Fourier Transform and Convolution.*; Springer: New York, NY, USA, 1989.
55. McEliece, R.J. *Finite Fields for Computer Scientists and Engineers*; Springer: New York, NY, USA, 2012; Volume 23,
56. Lidl, R.; Niederreiter, H. *Introduction to Finite Fields and Their Applications*; Cambridge University Press: Cambridge, UK, 1994.
57. Menezes, A.J.; Van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC press: Boca Raton, FL, USA,1996; Chapter 14; pp. 610–613.
58. Kocher, P.C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Proceedings of the Advances in Cryptology—CRYPTO '96. LNCS, Santa Barbara, CA, USA, 18–22 August 1996.
59. Hankerson, D.; Menezes, A.J.; Vanstone, S. *Guide to Elliptic Curve Cryptography*; Springer: New York, NY, USA, 2003.
60. Enge, A. *Elliptic Curves and Their Applications to Cryptography: An Introduction*; Springer: New York, NY, USA, 2012.
61. Blake, I.; Seroussi, G.; Smart, N. *Elliptic Curves in Cryptography*; London Mathematical Society Lecture Notes Series 265; Cambridge University Press: Cambridge, UK, 1999.

62. Bernstein, D.J.; Lange, T. Faster addition and doubling on elliptic curves. In *Advances in Cryptology–ASIACRYPT 2007*; Springer: New York, NY, USA, 2007; pp. 29–50.
63. Bailey, D.V.; Paar, C. Optimal extension fields for fast arithmetic in public-key algorithms. In *Annual International Cryptology Conference*; Springer: New York, NY, USA, 1998; pp. 472–485.
64. Baktır, S. Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography. Ph.D. Thesis, Worcester Polytechnic Institute, Worcester, MA, USA, 2008.
65. MSP430x1xx Family User's Guide. Available online: http://www.ti.com/lit/ug/slau049f/slau049f.pdf (accessed on 8 March 2020).
66. IAR Embedded Workbench IDE. Available online: https://www.brown.edu/Departments/Engineering/Courses/En164/EWARM_UserGuide.ENU.pdf (accessed on 8 March 2020).
67. 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*; Springer: New York, NY, USA, 2016; pp. 94–112.
68. Gouvêa, C.P.; Oliveira, L.B.; López, J. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. *J. Cryptogr. Eng.* **2012**, *2*, 19–29. [CrossRef]
69. Wang, H.; Sheng, B.; Li, Q. Elliptic curve cryptography-based access control in sensor networks. *Int. J. Secur. Netw.* **2006**, *1*, 127–137. [CrossRef]
70. Wenger, E.; Werner, M. Evaluating 16-bit processors for elliptic curve cryptography. In *Smart Card Research and Advanced Applications*; Springer: New York, NY, USA, 2011; pp. 166–181.
71. Liu, A.; Ning, P. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IEEE Computer Society, IPSN '08, Washington, DC, USA, 22–24 April 2008; pp. 245–256.