

# Efficient Implementation of Elliptic Curve Cryptography on DSP for Underwater Sensor Networks

Hai Yan, Zhijie Jerry Shi, and Yunsi Fei\*

Department of Computer Science and Engineering,

\* Department of Electrical and Computer Engineering,  
University of Connecticut, Storrs, CT 06269

**Abstract**—As emerging sensor networks are normally deployed in the field and thus vulnerable to many types of attacks, it is critical to implement cryptographic algorithms in sensor nodes to provide security services. Public-key algorithms, such as RSA and Elliptic Curve Cryptography (ECC), have been widely used for security purposes like digital signature and authentication. In resource-constrained sensor networks, ECC is more suitable than other public-key algorithms based on large integer fields because it provides similar security strength with shorter keys and thus is more computation-efficient. However, even ECC is slow on many sensor nodes. In underwater sensor networks (UWSNs), sensor nodes communicate with each other through acoustic channels. Because more complicated algorithms are needed to encode and decode acoustic signals transmitted through the noisy underwater channels, many acoustic modems include a DSP to meet the performance requirement of underwater communications. In this paper, we study the implementation of ECC on DSPs. We optimize the SECG elliptic curves `secp160r1` and `secp224r1` on a TMS320C6416 DSP board from Texas Instruments. In our implementation, it takes 0.81 ms to compute a random scalar point multiplication for `secp160r1`, an order of magnitude faster than the communication algorithm decoding a data block. Therefore, we believe that it is feasible to adopt ECC in UWSNs.

## 1. Introduction

Sensor networks have been envisioned as powerful solutions for many applications, such as monitoring, surveillance, measurement, control and health care, etc. [31], [39], [36], [25]. Recently, deploying sensor networks in aquatic environments has received growing interests [1], [48], [13], [8], [20], [30], [35]. Underwater sensor networks (UWSNs) consist of many sensor nodes deployed in water, where they sense the activities of interest and transmit data back to control center when needed.

Acknowledgments: This work was supported by the National Science Foundation under grants CCF-0541102, CNS-0821597, and CNS-0644188.

Because of unique characteristics of underwater environments, UWSNs differ significantly from land-based sensor networks. One major difference is the communication method. Since electromagnetic waves cannot propagate over a long distance in water, UWSNs must rely on acoustic channels to communicate. Acoustic channels have large propagation delays because the speed of sound in water is about  $1.5 \times 10^3$  m/s, five orders of magnitude lower than the speed of radio in air ( $3 \times 10^8$  m/s). The bandwidth of underwater acoustic channels is also limited [22], [43], [5], [6]. Moreover, underwater acoustic channels are affected by many factors such as path loss, noise, multipath, and Doppler spread, which all cause high error probability in acoustic channels. Because of these difficulties, reliable and high-data-rate acoustic communications are more computation-intensive than in-air radio communications [37].

Recently multicarrier modulation in the form of orthogonal frequency division multiplexing (OFDM) has been adopted in underwater environments successfully to provide higher data rates [42], [21], [27], [29], [28], [26]. The complexity of OFDM algorithms for underwater environments requires a lot of computational power. DSPs, designed for digital signal processing, support commonly-used signal processing operations, such as convolution, dot product, and FFT, efficiently at low cost. Compared to customized circuit, DSP-based systems are also easier to upgrade when advanced algorithms are available. Therefore, DSPs have already been adopted in existing acoustic modems like the Micro Modem [16]. Many prototypes of OFDM acoustic modem are also based on DSP, e.g., our design in [50]. It can be expected that DSPs will be indispensable in many underwater communication modules.

Since underwater sensor nodes are deployed in public environments and communicate through wireless channels, they are vulnerable to various attacks which could eavesdrop the communications, alter transmitted data, or

attach unauthorized nodes to the network. To protect sensor networks from these attacks, it is desirable to provide security services at the sensor nodes, in addition to the conventional tasks of sensing, computation, and communications.

Public-key cryptosystems, such as RSA and Elliptic Curve Cryptography (ECC), have been widely used for security purposes such as digital signature and authentication. However, public-key algorithms are more computation-intensive than other types of crypto algorithms like symmetric-key algorithms and hash functions. Especially, they are slow on resource-constrained sensor nodes. Since public-key algorithms have many unique advantages, a lot of work has been done on the optimization of public-key algorithms on low-end processors, e.g., [18], trying to make public-key algorithms feasible on land-based sensor nodes.

Among public-key algorithms, ECC algorithms are considered as better candidates to be adopted in sensor networks because they require shorter key sizes than other large integer-based algorithms to achieve the same security strength and thus they have better computational efficiency. Considering the presence of DSP in underwater sensor nodes for communication signal processing, we propose to utilize DSP for security services as well and investigate performance of ECC on DSP. Although the DSP implementation of ECC is not the fastest nor the most power-efficient, it provides a cost-effective solution for many DSP-based systems without changing existing designs.

In this paper, we implemented and optimized the secp160r1 elliptic curve standardized by SECG [7] on a TMS320C6416 DSP board from Texas Instruments [45]. To precisely control the computational resources on C6416, we use assembly code for all the performance-critical operations including finite field addition, subtraction, multiplication, and modular reduction. Our experiments show that it takes 0.81ms to perform a random point multiplication, while it takes  $4.7\mu\text{s}$  and  $4.5\mu\text{s}$  to perform point addition and doubling, respectively. Our optimized implementation is an order of magnitude faster than the C implementation. The results indicate that it is feasible to adopt ECC algorithms in UWSNs if sensor nodes are equipped with a DSP.

The rest of the paper is organized as follows. Section 2 gives the related work in ECC implementation. The ECC algorithms are briefly reviewed in Section 3. We describe our ECC implementation in Section 4 and present the performance evaluations in Section 5. Finally, Section 6 concludes the paper.

## 2. Related Work

There exists some implementation work of public-key cryptosystems on various low-end processors and land-based sensor nodes. Gura et al. [18] implemented both ECC and RSA on 8-bit CPUs. They used two target CPUs for the implementation: ATmega128 at 8 MHz and Chipcon CC1010 at 14.7456 MHz. On ATmega128, they achieved the performance of 0.81s for 160-bit ECC point multiplication. The elliptic curves used in their implementation were NIST/SECG curves over  $GF(p)$ . Wang et al. [47] implemented ECC on Berkeley Motes, MICAz and TelosB. MICAz motes use ATmega128 and TelosB motes use MSP430, a 16-bit processor running at 8 MHz. Their implementation, having been adopted in many research groups, takes 1.35s and 1.60s to perform a random point multiplication on MICAz and TelosB platforms, respectively.

On DSPs, fast public-key cryptography implementations have also been studied [2], [4], [14]. These studies focused on the implementation of RSA, leaving ECC poorly addressed.

Recently, researchers have explored the techniques to accelerate cryptography algorithms using Graphic Processing Units (GPU) [12], [19], [32], [38], [44]. In [44], authors achieved the throughput of 1412 point multiplications per second for NIST 224-bit elliptic curve.

Our work aims to examine the feasibility and performance of ECC algorithms on DSPs, so as to possibly embed public-key security services in the DSP-based underwater sensor nodes.

## 3. Elliptic Curve Cryptography

In this section, we give some background on ECC algorithms. The elliptic curves used in cryptography can be defined on either a prime field  $GF(p)$  or a finite field of characteristic two  $GF(2^m)$ , which is also called a binary field. Elliptic curves on binary field are more recognized in hardware implementations since the binary field operations can be implemented more efficiently with hardware. Due to the lack of support for binary field operations, especially binary field multiplications, in most commercially available processors, elliptic curves defined on prime field are more widely adopted in software implementations. In this paper, we will focus on ECC defined on prime field.

### 3.1. Elliptic Curves on Finite Field $GF(p)$

When defined on a prime field  $GF(p)$ , an elliptic curve can be represented as

$$Y^2 = X^3 + a \cdot X + b \quad (1)$$

where  $a$  and  $b$  are constants in  $GF(p)$  and  $p$  is a prime greater than three. All the points on an elliptic curve together with a special point  $O$  that is defined as the identity element form the set  $E(GF(p))$ . For any point  $P = (x, y) \in E$ , we have

$$P + O = O + P = P, P + (-P) = O \quad (2)$$

where  $-P = (x, -y)$ . The addition of two points  $P_1 = (x_1, y_1)$  and  $P_2 = (x_2, y_2)$ , defined as  $P_3 = (x_3, y_3) = P_1 + P_2$ , can be computed as follows.

If  $P_1 \neq P_2$ ,

$$x_3 = \lambda^2 - x_1 - x_2, y_3 = (x_1 - x_3)\lambda - x_3 - y_1, \quad (3)$$

where  $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ .

If  $P_1 = P_2$ ,

$$x_3 = \lambda^2 - 2x_1, y_3 = (x_1 - x_3)\lambda - x_3 - y_1, \quad (4)$$

where  $\lambda = \frac{3x_1^2 + a}{2y_1}$ .

The scalar point multiplication  $Q = k \cdot P$ , where  $k$  is an integer and  $P$  is a point, is defined as additions of  $k$  copies of  $P$ .

$$Q = k \cdot P = \underbrace{P + \dots + P}_k. \quad (5)$$

The security strength of ECC lies in the fact that given the curve, the points  $P$  and  $Q = k \cdot P$ , it is hard to recover  $k$ . This is also called the Elliptic Curve Discrete Logarithm Problem (ECDLP). Currently, there is no known sub-exponential algorithm that solves ECDLP.

Several security protocols have their elliptic curve analogues. For example, the Elliptic Curve Digital Signature Algorithm (ECDSA) proposed in [46] is the elliptic curve version of DSA. The most time-consuming operation in ECDSA is scalar point multiplication, which is performed once when signing a message and twice when verifying a signature.

### 3.2. Scalar Point Multiplication Algorithms

Many methods have been proposed for exponentiation in a multiplicative group [17], [23], [33]. They can also be extended to scalar point multiplication on elliptic curves. In this section, we describe several algorithms that can perform the scalar point multiplication.

An  $m$ -bit large integer  $k$  can be represented in binary format as  $k = (k_{m-1}, \dots, k_i, \dots, k_0)$ , where  $k_i \in \{0, 1\}$ . Based on the binary format of  $k$ , the point multiplication  $k \cdot P$  can be computed using Algorithm 1.

In Algorithm 1, the binary method scans the scalar one bit a time, from the most significant bit. It requires  $m$  point doublings and  $w_k - 1$  point additions, where  $w_k$

---

#### Algorithm 1 Binary method for scalar point multiplication

---

**Require:** An integer  $k = (k_{m-1}, \dots, k_0)$  in binary format, where  $k_i \in \{0, 1\}$ , and  $k_{m-1} = 1$ . A point  $P = (x, y)$  on elliptic curve  $E(GF(p))$ .

**Ensure:** The point  $Q = k \cdot P$  on the same curve.

```

1:  $Q \leftarrow P$ 
2: for  $i = m - 2$  to 0 do
3:    $Q \leftarrow 2Q$  ▷ Point doubling
4:   if  $k_i = 1$  then
5:      $Q \leftarrow Q + P$  ▷ Point addition
6:   end if
7: end for
8: return  $Q$ 

```

---

is the number of 1's in the binary representation of integer  $k$ .

Several generalizations of the binary method, such as the  $n$ -ary method, sliding-window method, etc. [24], [3], process a block of bits a time, instead of just one bit. Pre-computations are required in these methods. Compared to the binary method, these methods improve the performance by leveraging pre-computed values and reducing the number of point additions. The number of point doublings, however, remains the same as in the basic binary method.

## 4. Implementation

We implemented the scalar point multiplication for elliptic curves secp160r1 and secp224r1 standardized by SECG [7]. Since the optimizations for both elliptic curves are similar, we will take the secp160r1 as the example to show how ECC is implemented.

The target platform we chose for our implementation is the TMS320C6416 fixed-point DSP from Texas Instruments. The C6416 DSP operates at 1 GHz clock rate and provides 1024 KB on-chip L2 cache which can also be configured as SRAM. It has two register files A and B with 32 32-bit general-purpose registers on each side. So the total number of registers is 64. The C6416 DSP core adopts VelocityTI.2, the advanced Very-Long-Instruction-Word (VLIW) technology. There are eight independent functional units, including six ALUs and two multipliers. Each of the ALUs supports single 32-bit, dual 16-bit, or quad 8-bit arithmetic per clock cycle. Two multipliers support two 16x16-bit multiplications (32-bit results) per clock cycle. Hence, the C6416T could execute up to eight instructions in one clock cycle.

Our optimizations utilize two architectural features available on C6416: 1) large number of registers in the

register files and 2) eight functional units that can be occupied in parallel. To fully utilize the functional units, the use of two register files needs to be balanced. Software pipelining is also employed to improve the performance of time-consuming kernel loops. We try to start the iterations of a loop as early as possible and execute multiple iterations in parallel. In order to precisely control the computational resources on C6416, all the performance-critical operations, including modular addition, subtraction, multiplication and reduction, are implemented and hand-tuned in assembly code.

We next elaborate on the detailed implementation of each type of operation. We first discuss basic integer operations used in point addition and doublings: addition and subtraction, multiplication, and modular reduction. Then we discuss projective coordinates, which are used in our implementation to handle complexity of integer inversion. At the end, we discuss how scalar point multiplication is performed with point addition and doubling.

#### 4.1. Addition and Subtraction

Prime field addition and subtraction of two integers  $a$  and  $b$  can be computed by first applying normal integer addition and subtraction, then reducing the results to  $GF(p)$ . Given that both  $a$  and  $b$  are in  $GF(p)$ , the result of addition  $c = a + b$  can be reduced to  $GF(p)$  by subtracting  $p$  from  $c$  if  $c \geq p$ . Similarly, the result of subtraction  $c = a - b$  can be reduced to  $GF(p)$  by adding  $p$  to  $c$  if  $c < 0$ . In our implementation, a large integer is stored in 32-bit words. For example, a 160-bit integer  $k = (k_{159}, \dots, k_0)$  will be stored in five 32-bit words  $(w_4, w_3, w_2, w_1, w_0)$  with word  $w_0$  holding the least significant 32 bits  $(k_{31}, \dots, k_0)$  of  $k$ . In this format, addition and subtraction of large integers will be computed by applying single-word addition and subtraction and propagating the carry to higher order words accordingly. Since C6416 does not provide carry flag nor addition-with-carry instructions, we use a register pair to store the intermediate result of single-word addition or subtraction. Eventually, the carry will be propagated to higher order words.

#### 4.2. Multiplication

The multiplication of two  $n$ -word integers will generate a  $2n$ -word result. For multiplication in prime field  $GF(p)$ , the result needs to be reduced regarding to the prime  $p$ . The most straightforward algorithm for multiplication is the paper-and-pencil method. The paper-and-pencil method multiplying two integers  $a$  and  $b$  can be performed in row-wise or column-wise fashion. In the row-wise fashion, operand  $a$  is multiplied by one word

from operand  $b$  once a time to produce an intermediate product of  $n + 1$  words, and the intermediate product is left shifted several words according to the position of the word from  $b$ . Finally, the intermediate products are accumulated to get the result of the multiplication. While in a column-wise fashion, one column of the final result is computed each time. Although the same  $n^2$  single word multiplications are required in both methods, row-wise multiplication requires fewer memory access but more working registers than column-wise multiplication.

In [18], a hybrid multiplication is proposed to balance the overhead of memory access and the number of working registers. It works by combining the row-wise and column-wise multiplications hierarchically. The hybrid multiplication divides each  $n$ -word operand into groups of  $d$  words. It performs column-wise multiplications for all the groups at the higher level and row-wise multiplications between any two groups at the lower level. When  $d = n$ , the hybrid multiplication becomes row-wise multiplication. When  $d = 1$ , it is the same as column-wise multiplication.

We implemented both row-wise and hybrid multiplications. For the multiplication of two  $n$ -word integers,  $n^2$  32x32-bit integer multiplications are required for both methods. However, only 16x16-bit unsigned integer multiplication instructions are supported on C6416, which generate 32-bit results to destination registers. In our implementation, a 32x32-bit multiplication (with 64-bit result) is computed by four 16x16-bit multiplication instructions. As a result, totally  $4n^2$  16x16-bit multiplications are required to compute the multiplication of two  $n$ -word integers. Algorithm 2 shows the multi-precision multiplication using 16-bit multiplier. Here, the *mpyu* and *mpyu* instructions perform unsigned multiplications on lower half words (16-bit) and higher half words, respectively. The instruction *mpyhl*( $a, b$ ) performs unsigned multiplication on the higher half word of  $a$  and the lower half word of  $b$ , and *mpylh*( $a, b$ ) is equivalent to *mpyhl*( $b, a$ ). It should be noted that the squaring of an integer is faster than the general multiplication of two integers, since only  $\lceil \frac{n(n+1)}{2} \rceil$  single-word multiplications rather than  $n^2$  are required.

For row-wise multiplication, because a large number of general-purpose registers are available on C6416, we loaded the operand  $a$  into a register file only once at the beginning and kept it for all the following computations. In order to take full advantage of the computational resources on C6416, we rearranged data to two separate sides for register file  $A$  and  $B$ , and tried to balance the operations on both sides and maximize the number of instructions that can be executed simultaneously. The data dependency graph of the inner loop in the multiplication

---

**Algorithm 2** Multi-precision multiplication using 16-bit multiplier
 

---

**Require:**  $a = (a_{n-1}, \dots, a_0)$  and  $b = (b_{n-1}, \dots, b_0)$  are two  $n$ -word integers.

**Ensure:**  $z = (z_{2n-1}, \dots, z_0)$  is a  $2n$ -word integer, s.t.  $z = a \cdot b$ .

- 1: **for**  $j = 0$  to  $n - 1$  **do**
  - 2:    $(c_1, tmp2) \leftarrow mpyhl(a_0, b_j) + mpylh(a_0, b_j)$
  - 3:    $(c_2, z_j) \leftarrow y_0 + (tmp2 \ll 16) + mpyu(a_0, b_j)$
  - 4:    $c_2 \leftarrow c_2 + mpyhu(a_0, b_j)$
  - 5:    $tmp1 \leftarrow tmp2$
  - 6:   **for**  $i = 1$  to  $n - 1$  **do**
  - 7:      $(c_1, tmp2) \leftarrow mpyhl(a_i, b_j) + mpylh(a_i, b_j) + c_1$
  - 8:      $mid \leftarrow or(tmp2 \ll 16, tmp1 \gg 16)$
  - 9:      $(c_2, y_{i-1}) \leftarrow y_i + mid + c_2 + mpyu(a_i, b_j)$
  - 10:     $c_2 \leftarrow c_2 + mpyhu(a_i, b_j)$
  - 11:     $tmp1 \leftarrow tmp2$
  - 12:    **end for**
  - 13:     $y_{n-1} \leftarrow c_2$
  - 14: **end for**
  - 15:  $(z_{2n-1}, z_{2n-2}, \dots, z_n) \leftarrow (y_{n-1}, \dots, y_0)$
  - 16: **return**  $z$
- 

algorithm is shown in Figure 1, where data is put in nodes, and both the instruction and the corresponding functional unit are labeled on each edge in the figure. Extra memory access instructions, load and store, are eliminated by holding the multiplicand  $a$  and the intermediate product  $y$  in registers. Since the value  $y_i$  is used to compute  $y_{i-1}$ , there is no loop carry on  $y$ . The figure shows that two instructions  $mpyu$  and  $mpylh$  use the same functional unit  $.M1$  on side A, hence it requires at least two cycles to execute them. With the data flow dependency, it takes four more cycles to compute the value  $mid$  (on B side) and extra two cycles to generate  $y_{i-1}$  (on A side). As a result, the minimal iteration interval for the inner loop is 8 cycles in this case.

Our experiments show that it takes  $0.15\mu s$  and  $0.32\mu s$  to perform the row-wise and hybrid multiplication with group size  $d = 2$  for 160-bit integers, respectively. The row-wise multiplication is about two times faster than hybrid method by taking advantage of a large number of registers to reduce the memory access. The squaring takes  $0.13\mu s$ , about 15% faster than the general multiplication.

### 4.3. Reduction

Since modular reduction has to be applied after every large integer multiplication, it also plays a critical role in the overall performance of ECC implementation. The most straightforward method for modular reduction is

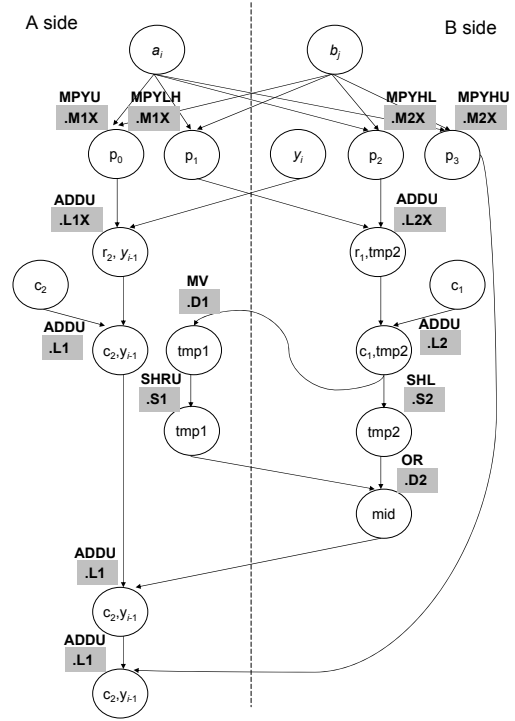


Fig. 1. Dependency graph of multi-precision integer multiplication

long-division in which a large integer  $a$  is divided by the prime  $p$  and the remainder  $r$  is returned as  $r = a \bmod p$ . The division, however, is computationally expensive and should be avoided when possible. Montgomery proposed an algorithm [34] to remove the division for the modular reduction. Instead of computing  $a \bmod p$ , Montgomery reduction computes  $aR^{-1} \bmod p$ . When  $R$  is properly chosen, the complexity of the computation will be greatly reduced. In addition to the Montgomery reduction, optimizations are also available for moduli with special format, such as pseudo-Mersenne primes. A pseudo-Mersenne prime has the format  $p = 2^m - \omega$ , where  $\omega$  is the sum of a few powers of two and  $\omega \ll 2^m$ . Based on the congruence  $2^m \equiv \omega$ , reduction of a  $2m$ -bit integer  $C'$  can be computed using Algorithm 3 [7].

In Algorithm 3, the large integer  $C'$  is split into two  $m$ -bit halves  $C'_1$  and  $C'_0$ . Since  $\omega$  has sparse items, the multiplication by  $\omega$  is commonly implemented with only additions and shiftings. Moreover, because the  $\omega$  is normally a very small value, the *while* loop in the algorithm will be executed only for very limited times.

We optimized Algorithm 3 for the prime  $p = 2^{160} - 2^{31} - 1$  specifically, which is proposed in SECG secp160r1 elliptic curve. The optimized algorithm is shown in Algorithm 4. Since we have  $(t_5, t_4, t_3, t_1, t_0) < 4p$  at step 6 in Algorithm 4, the *while* loop will iterate at most three

---

**Algorithm 3** Modular reduction for  $p = 2^m - \omega$ 

---

**Require:**  $C' = (C'_1, C'_0)$  is a  $2m$ -bit integer, where  $C'_1$  and  $C'_0$  are  $m$ -bit integers.

**Ensure:**  $C = C' \bmod p$ .

- 1: **while**  $C'_1 \neq 0$  **do**
  - 2:      $(C'_1, C'_0) \leftarrow C'_1 * \omega + C'_0$
  - 3: **end while**
  - 4: **if**  $C'_0 \geq p$  **then**
  - 5:      $C'_0 \leftarrow C'_0 - p$
  - 6: **end if**
  - 7:  $C \leftarrow C'_0$
- 

times. Hence, the modular reduction has been reduced to four additions and no more than three subtractions of 160-bit integers plus a few shifting operations. In our implementation, Algorithm 3 takes  $0.76\mu\text{s}$ , while Algorithm 4 takes  $0.14\mu\text{s}$  which is more than 5 times faster.

---

**Algorithm 4** Optimized modular reduction for  $p = 2^{160} - 2^{31} - 1$ 

---

**Require:** A double-sized integer  $C' = (C'_9, \dots, C'_1, C'_0)$ , where  $C'_k$  is a 32-bit word. The prime  $p = 2^{160} - 2^{31} - 1$ .

**Ensure:**  $C = C' \bmod p$ .

- 1:  $(t_5, t_4, t_3, t_2, t_1, t_0) \leftarrow (0, C'_4, C'_3, C'_2, C'_1, C'_0)$
  - 2:  $(t'_4, t'_3, t'_2, t'_1, t'_0) \leftarrow (C'_9, C'_8, C'_7, C'_6, C'_5)$
  - 3:  $(s_4, s_3, s_2, s_1, s_0) \leftarrow (C'_9, C'_8, C'_7, C'_6, C'_5) \lll 31$
  - 4:  $(t_5, t_4, t_3, t_2, t_1, t_0) \leftarrow (t_5, t_4, t_3, t_2, t_1, t_0) + (t'_4, t'_3, t'_2, t'_1, t'_0) + (s_4, s_3, s_2, s_1, s_0)$
  - 5:  $(t_5, t_4, t_3, t_2, t_1, t_0) \leftarrow (t_5, t_4, t_3, t_2, t_1, t_0) + (0, 0, 0, 0, 0, C'_9 \ggg 1) + (0, 0, 0, 0, C'_9 \ggg 2, (C'_9 \ggg 1) \lll 31)$
  - 6: **while**  $(t_5, t_4, t_3, t_2, t_1, t_0) \geq p$  **do**
  - 7:      $(t_5, t_4, t_3, t_2, t_1, t_0) \leftarrow (t_5, t_4, t_3, t_2, t_1, t_0) - p$
  - 8: **end while**
  - 9: **return**  $C \leftarrow (t_4, t_3, t_2, t_1, t_0)$
- 

#### 4.4. Projective Coordinates

In affine coordinates, a point on elliptic curve is represented as  $P(x, y)$ , where  $x$  and  $y$  are the  $x$ -coordinate and  $y$ -coordinate of point  $P$ , respectively. When using affine coordinates, the prime field inversion is required in both point addition and doubling as shown in Section 3.1. The prime field inversion of an integer  $a$  in  $GF(p)$ , denoted by  $a^{-1} \bmod p$ , is an integer  $b$  in  $GF(p)$ , such that  $a \cdot b \equiv 1 \bmod p$ . It is known that the inversion operations are normally very expensive in terms of computation. In our implementation, we adopt the algorithm proposed in

[40] to compute the prime field inversion. It takes 0.689ms to perform one prime field inversion which is two orders of magnitude slower than the multiplication.

Adopting projective coordinates can reduce the number of computationally expensive inversion operations [9], [41], [10]. Later it is shown that the mixed coordinate system combining the modified Jacobian and affine coordinates offers the best performance [11]. In the mixed coordinate system, point addition is performed on two points in different coordinates: one point  $P_1$  in modified Jacobian coordinates  $(X_1, Y_1, Z_1, aZ_1^4)$  and the other point  $P_2$  in affine coordinates  $(x_2, y_2)$ . The mixed point addition is computed as shown in Formula 6, where the result is presented by point  $P_3 = (X_3, Y_3, Z_3, aZ_3^4)$  in modified Jacobian coordinates.

$$\begin{aligned} X_3 &= -H^3 - 2X_1H^2 + r^2, \\ Y_3 &= -Y_1H^3 + r(X_1H^2 - X_3), \\ Z_3 &= Z_1H, \\ aZ_3^4 &= aZ_3^4 \end{aligned} \quad (6)$$

with  $H = x_2Z_1^2 - X_1$  and  $r = y_2Z_1^3 - Y_1$ .

The point doubling of a point  $P_1$  in modified Jacobian coordinates is shown in Formula 7.

$$\begin{aligned} X_3 &= T, \\ Y_3 &= M(S - T) - U, \\ Z_3 &= 2Y_1Z_1, \\ aZ_3^4 &= 2U(aZ_1^4) \end{aligned} \quad (7)$$

with  $S = 4X_1Y_1^2, U = 8Y_1^4, M = 3X_1^2 + (aZ_1^4)$ , and  $T = -2S + M^2$ .

Formulas 6 and 7 show that general point addition requires 9 multiplications and 5 squarings while point doubling requires 4 multiplications and 4 squarings. As shown in Section 3.1, when using affine coordinates, the point addition requires 1 inversion and 3 multiplications while point doubling requires 1 inversion and 4 multiplications. As a result, when the speed of multiplication is more than 6 times faster than the speed of inversion, the projective coordinates can get better performance. Otherwise, affine coordinates will perform better. In our implementation, we chose projective coordinates to represent the points on the curve as the multiplication is much faster than the inversion.

#### 4.5. Non-Adjacent Format

For better performance, an integer can also be represented in Non-Adjacent Format (NAF) [3], where a bit can be either 0, 1 or  $-1$ , and two non-zero bits are not adjacent to each other. NAF has the lowest expected number of non-zero digits in the representation of an integer,

thus reducing the number of additions/subtractions in point multiplication. For example, if we represent  $-1$  with  $\bar{1}$ , the integer  $k = (01111111)_2$  is represented in NAF as  $k = (1000000\bar{1})_{NAF}$ . The number of non-zero bits is reduced from 7 to 2, which leads to fewer addition/subtraction operations.

When representing the scalar in NAF, a point subtraction need to be performed for each  $-1$  digit. The point subtraction  $Q - P$  is computed by point addition  $Q + (-P)$ , where the negation of a point  $P = (x, y)$  is  $-P = (x, -y)$ . Hence, the cost of point addition and subtraction is almost the same. In our implementation, we adopt the binary method for point multiplications shown in Algorithm 1 with NAFs for the scalars. For a fixed point, the scalar multiplication can be further optimized using window methods in which a number points are pre-computed, depending on the window size.

## 5. Performance Evaluation

We next present the experimental performance results of our ECC implementation. We first show the performance of arithmetic in prime field  $GF(p)$ . Then we present the performance of ECC point operations including point addition, point doubling, and scalar point multiplication. We also compare the performance with previously reported results. In our experiments, we use the 32-bit timer in C6416 DSP to measure the performance. The timer shares the same clock signal with DSP cores but counts at 125MHz (the main clock rate over 8).

To measure the performance of the finite field arithmetic, we randomly generate multi-precision integers and perform arithmetic operations on them, including addition (ADD), subtraction (SUB), multiplication (MPY), squaring (SQR), and inversion (INV). The results are shown in Table 1, where the numbers are the execution time in  $\mu s$ .

Table 1. Execution time of arithmetic operations in  $GF(p)$  on C6416 DSP (in  $\mu s$ )

Operation	secp160r1	secp224r1
ADD	0.059	0.082
SUB	0.059	0.082
MPY	0.30	0.51
SQR	0.28	0.47
INV	68.6	111.8

Since the multiplication is critical to the performance of ECC, we carefully optimize the multiplication function. Table 2 compares the different implementations performing multiplications in secp160r1 and secp224r1. With  $-o3$  and  $-pm$  optimization options, C compiler achieves 2.64x and 2.38x speedups for multiplications in secp160r1 and

secp224r1, respectively. Hand-tuning assembly code, we further improve the performance to  $0.30\mu s$  and  $0.51\mu s$  for multiplications in secp160r1 and secp224r1, respectively. Compared with the compiler-optimized code, hand-tuned code has a speedup of about 14x.

Table 2. Execution time of field multiplication on C6416 DSP (in  $\mu s$ )

Code	secp160r1	secp224r1
C w/o compiler optimization	11.71	16.94
C w/ compiler optimization	4.43	7.12
Hand-tuned assembly	0.30	0.51

Table 3 compares the performance of multiplication with previously reported results. Our DSP-based implementation is much faster than other processors compared in the table because of the higher clock rate and rich hardware resources such as larger multipliers and larger issue width. Even at the same clock rate of 8 MHz, DSP still outperforms those processors, performing a field multiplication in  $37.5 \mu s$ .

Table 3. Execution time of field multiplication in secp160r1

Platform	Time
C6416 @ 1GHz	$0.30\mu s$
ATmega128 @ 8MHz [18]	0.39ms
CC1010 @ 14.75MHz [18]	2.53ms
MICAZ @ 8MHz [47]	0.47ms

To measure the performance of scalar point multiplication, we randomly generate a multi-precision integer and multiply it with a random point on the curve. We repeat the experiment 100 times and take the average execution time as the result. The scalar point multiplication consists of two main operations: point addition and point doubling. Table 4 compares the execution time of these two operations for elliptic curve secp160r1.

Table 4. Execution time of point addition and doubling for SECP160r1 on the DSP

Platform	Point Addition	Point Doubling
C6416 @ 1GHz	$4.7\mu s$	$4.5\mu s$
MICAZ @ 8MHz	6.2ms	5.8ms
TelosB @ 4MHz	7.3ms	7.0ms

The results for scalar point multiplication are shown in Table 5 and Table 6. The performance for two elliptic curves, secp160r1 and secp224r1, are presented. Table 5 shows the execution time of scalar point multiplication on C6416 DSP with different codes. Table 6 compares the performance with previously reported results. It can be observed from the tables that the point multiplication takes longer time for larger key size. In our implementation, it takes 0.81ms to perform point multiplication for

secp160r1, which is about two times faster than 1.69ms for secp224r1. The implementation of NIST224 curve on Nvidia 8800GTS GPU [44] is more than two times faster than our implementation for the same key size. Their implementation takes advantage of higher shader clock of GPU and parallel threads on multi-core architecture. The Intel Core2 implementation also has better performance than ours due to higher clock rates, larger multipliers, and lower latency of multiplication instructions.

Table 5. Performance of point multiplication on C6416 DSP

Settings	secp160r1	secp224r1
C w/o compiler optimization	18.66ms	39.06ms
C w/ compiler optimization	7.13ms	16.28ms
Hand-tuned assembly	0.81ms	1.69ms

Table 6. Performance of point multiplications on different platforms

Platform	secp160r1	secp224r1
C6416 @ 1GHz	0.81ms	1.69ms
ATmega128 @ 8MHz [18]	0.81s	n/a
CC1010 @ 14.75MHz [18]	4.58s	n/a
MICAz @ 8MHz [47]	1.35s	n/a
TelosB @ 4MHz [47]	1.60s	n/a
Nvidia 8800GTS GPU @ 1.2GHz [44]	n/a	0.71ms
Intel Core2 @ 2.13GHz [15]	0.38ms	0.54ms

Although the performance of ECC on the DSP is not as good as that on general-purpose processors (GPPs) or that on high-end GPUs, which have more resources, it is only two to three times slower. Currently no such GPP or GPU has been employed in any sensor nodes due to their costs and power consumption.

When selecting an ECC algorithm for underwater sensor networks, we are more likely to choose secp160r1 with shorter key sizes. Thus generating a signature will take less than 1 ms. At the same time, decoding a single OFDM data block of 712 bits takes about 38.29 ms on the same DSP [49]. So we believe that it is practical to leverage existing resources in underwater sensor nodes for security services and adopt ECC in UWSNs.

Table 7 shows the code size of our ECC implementation. In the table, *bint*, *mul*, and *ecc* represent the large integer module, prime field arithmetic module, and elliptic curve module, respectively.

Table 7. Code size of ECC implementation

Module	Size (Byte)
bint	4544
mul	4032
ecc	7712
Total	16288

## 6. Conclusions

In this paper, we present an efficient ECC implementation on a TMS320C6416 DSP board. We take advantage of hardware features of DSP to accelerate the ECC operations. Our experiments show that the random point multiplication for curve secp160r1 can be performed in 0.81 *ms*. Compared to the decoding time of an OFDM block (about 40 *ms*), we believe that it is feasible to embed ECC algorithms in underwater sensor nodes that are already equipped with a DSP.

## References

- [1] Ian F. Akyildiz, Dario Pompili, and Tommaso Melodia. Challenges for efficient communication in underwater acoustic sensor networks. *ACM SIGBED Review*, Vol. 1(No. 1), July 2004.
- [2] Paul Barrett. Implementing the rivest, shamir, and adleman public-key encryption algorithm on a standard digital signal processor. *Advances in Cryptology-CRYPTO*, pages 311–323, 1987.
- [3] I.F. Blake, G. Seroussi, and N.P. Smart. *Elliptic Curves in cryptography*. Cambridge University Press, 1999.
- [4] E. F. Brickell. A survey of hardware implementations of rsa. *Advances in Cryptology-CRYPTO*, pages 368–370, 1990.
- [5] V. Capellano. Performance improvement of a 50km acoustic transmission through adaptive equalization and spatial diversity. *Proc. of Oceans*, 1997. Nova Scotia, Canada.
- [6] V. Capellano, G. Loubet, and G. Jourdain. Adaptive multichannel equalizer for underwater communications. *Proc. of Oceans*, 1996. Ft. Lauderdale, FL, USA.
- [7] Certicom Research. *SEC 2: recommended elliptic curve domain parameters*, 2000.
- [8] M. Chitre, S. Shahabudeen, and M. Stojanovic. Underwater acoustic communication and networks: Recent advances and future challenges. *The Spring 2008 MTS Journal*, Spring 2008.
- [9] D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Math*, 1986(7):385–434, 1986.
- [10] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation. In *ICICS*, 1997.
- [11] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *ASIACRYPT*, pages 51–65, 1986.
- [12] D.L. Cook, J. Ioannidis, A.D. Keromytis, and J. Luck. Cryptographics: secret key cryptography using graphics cards. *CT-RSA, LNCS*, 3376, 2005.
- [13] J.-H. Cui, J. Kong, M. Gerla, and S. Zhou. Challenges: Building scalable mobile underwater wireless sensor networks for aquatic applications. *IEEE Network, Special Issue on Wireless Sensor Networking*, 20(3):12–18, May 2006.
- [14] S. R. Dusse and B. S. Kaliski Jr. A cryptographic library for the motorola dsp56000. *Advances in Cryptology-Eurocrypt*, pages 230–244, 1991.
- [15] ECRYPT. ebats: Ecrypt benchmarking of asymmetric systems. Technical report, ECRYPT, 2007. <http://www.ecrypt.eu.org/ebats>.
- [16] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball. The WHOI Micro-Modem: An acoustic communications and navigation system for multiple platforms. *Proceeding of OCEANS*, 2005.
- [17] D.M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.
- [18] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *CHES*, 2004.

- [19] O. Harrison and J. Waldron. Aes encryption implementation and analysis on commodity graphics processing unit. *CHES*, pages 209–226, 2007.
- [20] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li. Research challenges and applications for underwater sensor networking. *IEEE Wireless Communications and Networking Conference*, April 2006.
- [21] T. Kang and R.A. Iltis. Matching pursuits channel estimation for an underwater acoustic OFDM modem. In *Proc. of Intl. Conf. on ASSP*, March 2008.
- [22] A. Kaya and S. Yauchi. An acoustic communication system for subsea robot. *Proc. of OCEANS*, 3:765–770, September 1989.
- [23] C.K. Koc. High-speed rsa implementation. Technical report, RSA Laboratories, 1994. TR201.
- [24] K. Koyama and Y. Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. *Advances in Cryptography*, 740:345–357, 1992.
- [25] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. Modelling data-centric routing in wireless sensor networks. In *IEEE INFOCOM 2002*, New York, USA, June 2002.
- [26] B. Li, S. Zhou, J. Huang, and P. Willett. Scalable OFDM design for underwater acoustic communications. *Proc. of Intl. Conf. on ASSP*, March 2008.
- [27] B. Li, S. Zhou, M. Stojanovic, and L. Freitag. Pilot-tone based ZP-OFDM demodulation for an underwater acoustic channel. *Proc. of MTS/IEEE OCEANS conference*, pages 18–21, September 2006.
- [28] B. Li, S. Zhou, M. Stojanovic, L. Freitag, J. Huang, and P. Willett. MIMO-OFDM over an underwater acoustic channel. *Proc. of MTS/IEEE OCEANS conference*, September 2007.
- [29] B. Li, S. Zhou, M. Stojanovic, L. Freitag, and P. Willett. Non-uniform Doppler compensation for zero-padded OFDM over fast-varying underwater acoustic channels. *Proc. of MTS/IEEE OCEANS conference*, pages 18–21, June 2007.
- [30] L. Liu, S. Zhou, and J.-H. Cui. Prospects and problems of wireless communication for underwater sensor networks. *Wireless Communications & Mobile Computing*, 8:977–994, 2008.
- [31] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA'02*, Atlanta, Georgia, USA, September 2002.
- [32] S.A. Mannavski. Cuda compatible gpu as an efficient hardware accelerator for aes cryptography. In *ICSPC*, pages 65–68, 2007.
- [33] A. Menezes and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [34] P. Montgomery. Modular multiplication without trial division. *Mathematics of Communication*, 170(44):519–521, 1985.
- [35] J. Partan, J. Kurose, and B. N. Levine. A survey of practical issues in underwater networks. *Proc. of ACM International Workshop on UnderWater Networks (WUWNet)*, pages 17–24, September 2006.
- [36] G. Pottie and W.J.Kaiser. Wireless intergrated network sensors. *Communications of the ACM*, 43(5):551–8, May 2000.
- [37] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002.
- [38] U. Rosenberg. Using graphic processing unit in block cipher calculations. Master’s thesis, University of Tartu, 2007.
- [39] Loren Schwiebert, Sandeep K. S. Gupta, and Jennifer Weinmann. Research challenges in wireless networks of biomedical sensors. In *ACM SIGMOBILE'01*, Rome, Italy, July 2001.
- [40] Sheueling Chang Shantz. From euclid’s gcd to montgomery multiplications to the great divide. Technical report, Sun Microsystems, 2001. SMLI TR-2001-95.
- [41] J. H. Silverman. *The arithmetic of Elliptic Curves*. Springer-Verlag, 1986.
- [42] M. Stojanovic. Low complexity OFDM detector for underwater channels. *Proc. of MTS/IEEE OCEANS conference*, pages 18–21, September 2006.
- [43] M. Suzuki, K. Nemoto, T. Tsuchiya, and T. Nakanishi. Digital acoustic telemetry of color video information. *Proc. of OCEANS*, pages 693–696, September 1989.
- [44] Rober Szerwinski and Tim Guneyasu. Exploiting the power of gpus for asymmetric cryptography. In *CHES*, pages 79–99, 2008.
- [45] Texas Instruments. *TMS320C64x/C64x+ DSP CPU and instruction set reference guide*, 2008.
- [46] S. Vanstone. Responses to nist’s proposal. *Communications of ACM*, pages 50–52, July 1992.
- [47] Haodong Wang and Qun Li. Efficient implementation of public key cryptosystems on mote sensors (short paper). In *ICICS*, pages 519–528, 2006.
- [48] P. Xie, J.-H. Cui, and L. Lao. VBF: Vector-based forwarding protocol for underwater sensor networks. *Proc. of IFIP Networking*, pages 15–19, May 2006.
- [49] H. Yan and Z. Shi. Efficient implementation of OFDM algorithms on DSPs. Technical report, University of Connecticut, 2009. CSE-SALUC-0901.
- [50] Hai Yan, Shengli Zhou, Zhijie Jerry Shi, and Baosheng Li. A dsp implementation of ofdm acoustic modem. In *Proceedings of the second workshop on Underwater networks*, pages 89–92, 2007.