Chapter

A Survey of Fast Scalar Multiplication on Elliptic Curve Cryptography for Lightweight Embedded Devices

Youssou Faye, Hervé Guyennet and Ibrahima Niang

Abstract

Elliptic curve cryptography (ECC) is one of the most famous asymmetric cryptographic schemes which offers the same level of security with much shorter keys than the other widely used asymmetric cryptographic algorithm, Rivest, Shamir, and Adleman (RSA). In ECC, the main and most heavily used operation is the scalar multiplication kP, where the scalar value k is a private integer and must be secured. Various methods for fast scalar multiplication are based on the binary/ternary representation of the scalar. In this chapter, we present various methods to make fast scalar multiplication on ECC over prime field for lightweight embedded devices like wireless sensor network (WSN) and Internet of Things (IoT).

Keywords: elliptic curve cryptography, fast scalar multiplication, wireless sensor network, IoT

1. Introduction

Nowadays WSNs become a part of the Internet; the integration of WSNs into the Internet of Things (IoT) must involve new security issues. Symmetric cryptography can be the best solution in a constrained platform and embedded devices such as sensor. For a large number of nodes, the asymmetric key cryptography is the widely used algorithm because of its scalability. Elliptic curve cryptography (ECC) is one of the most famous asymmetric cryptographic schemes, which offers the same level of security with much shorter keys than the other widely used asymmetric cryptographic algorithm, Rivest, Shamir, and Adleman (RSA) [1]. Scalar multiplication is denoted by kP (where P is a point on an elliptic curve and k represents a scalar). The scalar multiplication is the recurrent and most heavily used operation in ECC because it is used for key generation, encryption/decryption of data, and signing/verification of digital signatures. The mathematics of an elliptic curve implies three arithmetic levels: scalar arithmetic, point arithmetic, and field arithmetic [2]. To make fast computation of scalar multiplication, which is the major computation involved in ECC, many works are devoted to the point arithmetic and scalar arithmetic. Point operations mean point addition and doubling, tripling, or quadrupling (or similar operation). In the framework of this chapter, we will concisely examine various researchers on the scalar arithmetic level.

IntechOpen

The discussion on this chapter proceeds as follows: In Section 2, we start with the background on ECC over prime fields. Section 3 gives works on fast scalar multiplication on scalar arithmetic, followed by Section 4 which describes works on parallelization of scalar multiplication on scalar arithmetic. The conclusion and perspectives are given in the last section.

2. Overview on ECC over finite prime fields

2.1 Preliminaries

In this section, we give a brief overview on ECC over finite prime fields. By definition, an elliptic curve E over finite field F (of order n) denoted by $E(F)$ can be described by the Weierstrass Eq. [3]:

$$\boldsymbol{E}: \boldsymbol{y^2} = \boldsymbol{x^3} + \boldsymbol{ax} + \boldsymbol{b} \tag{1}$$

where a and $b \in F_p$ and Fp is a prime field. Most important finite fields used to date to implement cryptosystem have been binary, prime, and extension fields. In this chapter, we work in the context of prime field F_p, where p > 3 and $p = q^r$, with r = 1 and q a prime number called the characteristic of Fp.

Before it can be used for cryptography, the necessary condition is the discriminant of polynomial:

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{x^3} + \boldsymbol{ax} + \boldsymbol{b}, \boldsymbol{\Delta} = 4a^3 + 27b^2 \neq 0 \tag{2}$$

The set of pairs (x, y) solves (1), where $x,y \in Fp$ and the point at infinity (denoted ∞) forms an abelian group. The scalar multiplication directly depends on two basic operations over points on an elliptic curve: point doubling (2P) and point addition (P + Q) where P and Q are two different points on the elliptic curve. If $P = (x_p,y_p)$ and $Q = (x_q,y_q)$, two points ($\neq \infty$) on the elliptic curve over F_p denoted by $E(F_p)$, then point addition $P + Q = (x_{pq},y_{pq})$ or point doubling $2P = P + Q = (x_{pq},y_{pq})$ if P = Q can be calculated as

$$\begin{cases} x_{pq} = \lambda^2 - x_p - x_q \\ y_{pq} = \lambda(x_p - x_{p+q}) - y_p \end{cases} \tag{3}$$

$$\begin{cases} \lambda = \dfrac{y_q - y_p}{x_q - x_p} \ if\ P \neq Q \\ \lambda = \dfrac{3x_p^2 + a}{2y_p} \ \ if\ P = Q \end{cases} \tag{4}$$

The negative of point $P = (xp,yp)$ is point -P (xp,-yp), where P and -P are two points on the elliptic curve (**Figure 1**).

2.2 Encryption/decryption with ECC

The security of ECC relies on the difficulty of solving the elliptic curve discrete log problem (ECDLP). Let E be an elliptic curve over finite field F and P ∈ E(F), given a multiple Q of P, the elliptic curve discrete log problem is to find d ∈ F such that dP = Q. For example, if $P = (2, 2)$ and $Q = (0, 6)$, then 3P = Q, so d=3 is a solution to the discrete logarithm problem. Three operations are very much required to formulate a valid cryptosystem in ECC: key generation, encryption, and decryption.

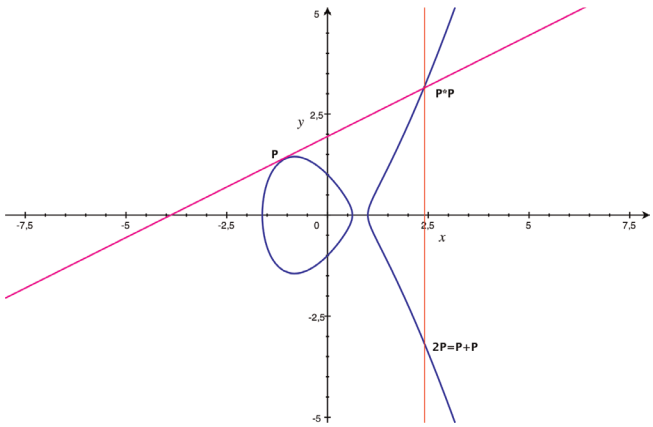

Figure 1.
Point addition in ECC.

2.2.1 Key generation with ECC

Public and private keys are associated with public parameters *(p, E, P, n)* where P is the generator point, with order n. n is always equal to the order of the elliptic curve group E and *nP*= . The private key d is randomly selected in the interval [1, n − 1], and the corresponding public key is *Q=dP*. The ECDLP consists in determining d from public parameters *(p, E, P, n)*.

Algorithm 1. Keys generation

Input: p, E, P, n // public parameters generated
Output: Public key (Q) and private key d generated
begin
1. Select randomly d in interval [1, n-1]
2. Compute Q=dP
3. Return(Q, d)
end

2.2.2 Encryption with ECC

For encryption, the message m is mapped to a valid point *M* on the curve and is encrypted by point addition with kQ where *k* is a random positive integer chosen by the sender and *Q = dP* represents the public key of receiver. The random *k* makes sure that even for a same message, the cipher text generated is different each time. The sender then sends the pair of cipher point C_2=*M*+*kQ* and C_1=*kP* to the receiver. The receiver, upon receiving the cipher point pair C_1 and C_2, computes dC_1=*d(k) P=k(dP)= kQ* by its own private key d and subtracts the result from the second point: *m*=C_2 -*kQ*.

Algorithm 2. Encryption.

Input: (p, E, P, n), Q and m //public parameters,public key and plaintext message m
Output: (C_1,C_2) // encrypted text
begin
1. Mapping message m to a point M ∈ E(Fp)
2. Select k ∈[1, n-1]
3. Compute C_1=kP
4. Compute C_2=M+kQ
5. Return(C_1, C_2)
end

2.2.3 Decryption with ECC

Algorithm 3. Decryption.

Input: (p, E, P, n), d, C_1, C_2 //public parameters, private key encrypted message
Output: m// plaintext message
Begin
1. Compute M =C_2 - dC_1,
2. Reverse mapping to retrieve m from M
3. Return (m)
end

The process of mapping a plaintext message m to point M on the curve is important in ECC. There are several mapping schemes that are used to map a plaintext message to a point on the elliptic curve [3–7]. A good mapping scheme must follow several guidelines:

- Mapped points should be on the elliptic curve. If G is the mapping function, $G(m) \longrightarrow (x,y) \in Ep(a,b)$.
- Mapping should always be invertible so that the receiver after decryption can reverse map the points to original plain text: $m = G^{-1}(x,y)$.
- Message mapping in ECC also plays a significant role as it decides how vulnerable the encrypted message is to attacks.
- A good message mapping scheme must reduce the use of unnecessary bandwidth.
- A good mapping scheme should not take much time to map the message to points on the map.

There are several mapping schemes using different approaches: maps each character in the plaintext to a point on the elliptic, maps each sequence of characters in the plaintext to a point on the elliptic, maps the full plaintext to a point on the elliptic, etc. For example, as in [8], if we use 192 bit key length, the National Institute of Standards and Technology (NIST) recommended elliptic curve with the following parameters:

$\boldsymbol{a}$=−3.
$\boldsymbol{b}$ =245,515,554,600,894,381,774,029,391,519,745,178,476 9,108,058,161,191,238,065.
Prime $\boldsymbol{p}$ = 6,277,101,735,386,680,763,835,789,423,176,059,013,767,194,773, 182,842,284,081.
Point $\boldsymbol{P}$ ={60,204,628,237,568,865,675,821,348,058,752,611,191,669,876,636, 884,684,818,174,050,332,293,622,031,404,857,552,280,219,410,364,023,488, 927,386,650,641}.
$\boldsymbol{d}$ = 28,186,466,892,849,679,686,038,856,807,396,267,537,577,176,687, 436,853,369.
$\boldsymbol{Q}$ ={2,803,000,786,541,617,331,377,384,897,435,095,499,124,748,881,890,727, 495,642, 4,269,718,021,105,944,287,201,929,298,168,253,040,958,383,009, 157,463,900,739}.

A plaintext message "**National Institute of Technology"** is taken as input.

1. Encryption process

    - Convert the text to ASCII values.
    - Partition the ASCII value as group size of 11 ASCII values.
    - Its equivalent ASCII values are {78,97,116, 105, 111, 110, 97, 108, 32, 73, 110},{115, 116, 105, 116, 117, 116, 101, 32, 111, 102, 32}, and{84, 101, 99,104,110, 111, 108, 111, 103,121, 44}.
    - Each group is converted into big integers using FromDigits function (in Mathematica) with base 65,536. The values for "National Institute" corresponding to the two first groups are as follows: {113,999,290,923, 567,984,853,125,612,857,907,836,245,105,850,253,422} and {168,075, 275,215,227,115,988,112,137,860,778,550,742,826, 363,519,008}.
    - A sends National Institute to B and computes scalar multiplication $kP = C_1$ = {95,058,406,573,787,743,380,879,387,493 754,072,690,640,209,963, 862,157,133, 5,437,547,807,282,051,947,615, 392,556,992,837,333,921,930, 872,121,480,709,807}.
    - A computes point addition $M+kQ = C_2$ ={5,357,129,649,847,875,387,947, 498,550,298,509,562,929,834,704 857,479,081,282,775,001,499,802, 163,650,458,076,998,673,808,830,204,345,207,458,648,302,309}, {6,179,418,438,352,156,963, 426,038,838,668,574,778,107,168,582, 785,759,775,636,5,950,440,184,023,478,909,084,289,343,254, 612,149,604,486,787,772,222,099,923}.

2. Decryption process

    - B receives C1 =kP and C2=M+kQ values.
    - Using the private key d, B performs scalar multiplication dC1.
    - Convert the subtraction operation to addition format: −dC1 = −kQ = 3,141,192 528,502,843,791,482,798,499,504,492,303,369,782,687,173, 663,895,377, − 2,544,834 938,121,667,890,493,126,265,872,103,594, 828,330,153,127,462,384,491}.
    - B performs point addition operation with -kQ: M = {113,999,290,923,567,984,853 125,612,857,907,836,245,105,850, 253,422, 16,807,527,521,522,711,598,811,213,786,077 8,550,742,826,363, 519,008}, {122,768,389,944,749,391,054,808,248,629,988,098,406, 227,392,397,356, 46,769,769,584,977,140,992,804,375,150,062, 379,259,053,557,678,135}.
    - Convert each bloc into ASCII values using IntegerDigits function (in Mathematica) with base 65,536, and retrieve ASCII values.

3. Fast scalar multiplication on scalar arithmetic

On a scalar arithmetic level, the double-and-add (DA) technique is the traditional binary algorithm, which is used and based on point operation, namely,

doubling of a point and addition of points. Well-known algorithms, such as nonadjacent form (NAF), window NAF, and sliding window [3, 9, 10], can reduce effectively the number of point operations. Some other algorithms, such as double-base chains, have been developed to compute faster scalar multiplication by using binary and ternary representation [11–15]. Algorithms, based on the aforementioned algorithms, optimize faster scalar multiplication [16–18]. Optimization is done by some approaches, which also use the binary representation of the scalar k [19–21]. For other solutions, optimization is based on selecting a set of elliptic curves for cryptography (Weierstrass curve, twisted Edwards curve) on which scalar multiplication is faster than the recent implementation record on the corresponding NIST curve.

3.1 Double-and-add algorithm

The double-and-add technique is the traditional binary algorithm, which is based on point operations, namely, doubling of a point and addition of points. The double-and-add algorithm is an additive representation of the algorithms used for exponentiation. As shown on Algorithm 4, the scalar is represented in binary on l bits: $\sum_{i=0}^{l-1} k_i 2^i$, where $k_i \in \{0, 1\}$. The binary method i=0 scans the bits of scalar $[k]P = \underbrace{(P + P + P + ... + P)}_{k\ times}$ either from left to right or right to left. A doubling operation is done for each scanned bit k_i of k, followed by a point addition if the scanned bit is non-zero $(k_i \neq 0)$.

Algorithm 4. Double-and-add LSB/MSB.

```
Input: k=(k_{l-1},....., k_1,k_0)_2, P ∈ E (Fp)
Output: Q=[k]P
Begin
  Q ← ∞
  for i ← 0 to l − 1 do
    if k_i = 1 then                    // begin scanning bits from right-to-left.
      |Q ← Q + P
        end                            // an addition operation is performed
    P ← 2P
        end                            // a doubling operation is performed
    return (Q)
end
```

For a given scalar k, the number of point doubling operation is *(l-1)*, and those of point addition operation is equal to the number of non-zero bits (denoted by hamming weight h) -1. The cost of multiplication depends on the length of the binary representation of k and the number of Harming weight (the number of 1's) of scalar in this representation. The average Harming weight on all scalar k of length l bits is approximately *l/2*. Thus, in an average, binary Algorithm 4 requires *(l-1) doublings and (l-1)/2 additions*.

For example, k = 379 = $(101111011)_2$, *l=9*, and the number of non-zero bits *h* is equal to 7. So computation 379P requires 8 doublings and 5 additions.

The *double-and-add* method can be generalized by using fixed or variable size windows. The scalar *k* is divided into *m* blocks of *w* bit(s) (*w* an integer of variable size), for each block corresponds to a number V_i.

As in DA where bits are scanned one by one, and if the scanned bit is equal to 0, $Q=2^1Q$ (point doubling) is performed; if not (scanned bit equal to 1 > 0), $Q=2^1Q$ (point doubling) and $Q=Q + 1P$ (point addition) are performed. In window

algorithm where blocks are scanned one by one, and if the value of the block is equal to 0, we perform $Q=2^wQ$; if not (values of blocks w bits performed =Vi), we performed $Q=2^wQ$ and $Q=Q+V_iP$ as shown in Algorithm 5.

For example, k = 379 = $(101111011)_2$ partitioned into blocks $\underbrace{1011}_{w=4}\ \underbrace{110}_{w=3}\ \underbrace{11}_{w=2}$, so, V_i is, respectively, equal to 3, 6, and 11 corresponding, respectively, to precomputed points 3P, 6P, and 11P. Thus, for this example, the scalar multiplication from block *(m-1)* to block 0 can be done as follows: [11]P→2^3. [11]P(3 repeated doublings) + [6]P(addition) → 2^2, and [94]P(2 repeated doublings) + [3]P (addition) → [379]P. Thus, five point doubling operations and two point addition operations are calculated.

Algorithm 5. Windows algorithm.

Input: $k = \left(\underbrace{k_{l-1}k_{l-2}k_{l-3}k_{l-4}}_{block\ (m-1)}, \ldots\ldots, \underbrace{k_5k_4k_3}_{block\ 1}, \underbrace{k_2k_1k_0}_{block\ 0}\right)_2$, P ∈ *E (Fp)*
Output: Q=*[k]P*
Begin
  $Q \leftarrow \infty$
  for $i \leftarrow m-1$ *to* $l-1$ **do**
    $Q = 2^wQ$ // begin scanning block by block.
    if $V_i > 0$ **then** // compute repeated point doublings w times
    $Q \leftarrow Q + V_iP$ // compute addition with precomputed point V_iP
    end
  end
  return (Q)
end

However, this algorithm involves precomputed points whose number depends on the size of the blocks. If the blocks have a fixed-size w bits, the number of precomputed points is $(2^w - 2)$ where −2 represents the blocks for V_i= 0 or 1. If the blocks have variable size, as, for example, with three blocks of *w1* bits, *w2* bits, and *w3* bits, the number of precomputed points p is $(2^w - 2)$. It should be noted that using the window method reduces the computation time and increases the memory storage and calculation time of precomputed points. If the size of the blocks increases, the number of precomputed points increases exponentially, and the number of performed operations decreases. Thus, the selection of the window size implies the computation time. A compromise is needed between the size of the blocks and the computation time related to precomputed points. According to NIST recommendations, the best window length is *w=4*. To reduce the number of precomputed points, the sliding window method of variable size with maximum digits equal to w can be used. For this method, the values Vi of blocks are odd; consecutive zeroes are taken into account. Therefore, a window starts and ends with a non-zero number.

For example, scalar k = 379 = $(101111011)_2$ is partitioned into blocks $\underbrace{1}_{w=4}\ \underbrace{1111}_{w=3}\ \underbrace{11}_{w=2}$, so, V_i values are, respectively, 1, 15, and 3 corresponding, respectively, to precomputed points 1P, 15P, and 3P. The scalar multiplication from block *(m-1)* to block 0 can be performed as follows: P → [2]P(1 point doubling) → 24[62] P(4 repeated point doublings) + [15]P(point additions)→ 2.[47]P(1 point doubling) → 22[94]P(2 repeated point doublings)+ [3]P(point addition) → [379]P. The result is eight point doublings and two additions.

Optimization can be done by finding a representation with a minimum zero bits in order to reduce the number of addition operations: this is the objective of the solutions described in the next section.

3.2 Nonadjacent form

Like addition, point subtraction on an elliptic curve is also effective especially when it comes to computing easily the opposite of a point on which we only change a coordinate: *P (x, y)* to *-P(x,-y)*. We can use a signed representation of bits of the integer k. One of the particularly interesting representations is the nonadjacent form which uses {−1, 0, 1}: $\sum_{i=0}^{l-1} k_i 2^i$, where *ki* ∈ {−1, 0, 1}. To compute scalar multiplication *[k]P* by NAF, digits on NAF representation of scalar k are scanned from most significant digit to last significant digit. For each digit, a point doubling operation is performed, and point addition is computed when the digit is equal to 1 or a point subtraction when the digit is equal to −1. The advantage of this representation is that it possesses the following properties:

1. *k* has a unique NAF denoted NAF (k).

2. NAF(k) has the fewest non-zero digits of any signed digit representation of k.

3. The length of NAF(k) is at most one more than the length of binary k.

For example, for k = 255_2 = $(11111111)_2$ where the density of non-zero digits is maximum, the computation of 255P implies seven point additions. But if we transform it into 256P -P which is equal to (10000000–1)P, only one addition is needed. Thus, the NAF(k) = $(10000000\overline{1})_2$where $\overline{1}$ representes −1. The NAF(k) can be generated by dividing successively k by 2. If k is odd, the rest r ∈ {−1, 1} is chosen so that the quotient *(k-r)/2* is even. Thus, the next digit of NAF representation will be equal to 0.

Algorithm 6. Computing NAF for scalar *k*.

Input: k = the scalar k (integer)
Output: NAF(k),
Begin

```
        i ← 0
    while(k ≥ 1)do
      if(k_i odd)then
   | k_i ← 2 − (kmod4);
   | k = k_i;
        end
        else
  |k_i ← 0
        end
      k_i=k/2;
      i ← i + 1;
        end
return (k_{i−1}, k_i, .....k_1, k_0)
end
```

Based on DA algorithm from left to right, Algorithm 6 computes scalar multiplication by using NAF(k).

Thus, the average density of non-zero digits (−1 or 1) for all NAF (*k*) with length *(l-1)* digits is approximately *(l-1)/3*. The average computation of Algorithm 7 is *(l-1)* point doublings and *(l-1)/3* point additions. However, it requires a scalar conversion time from k to NAF(k) (see Algorithm 6). The NAF method can be generally used for a set of digits $\left(C_{2^w} = \left\{-2^{w-1}, ...2^{w-1}\right\}\right)$ to represent the scalar k. That's equivalent to split it into fixed − size windows w. For example,

Algorithm 7. NAF method.

```
Input: NAF(k), P ∈ E (Fp)
Output: Q= [k]P
Begin
    Q ← ∞                                   //scan from most significant digit to less significant
  for(i ← l − 1 to 0)do
    Q ← 2Q                                  // compute point doubling
  if(k_i = 1)then                           // compute point addition
  |Q ← Q + P
      end
  if (k_i = −1) then                        //compute point substraction
    |Q ← Q − P
      end
      end
    return (Q)
end
```

$(C_{2^3} = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. We can define NAFw(k) as follows : $NAF(k)_w = \sum_{i=0}^{l} k_i 2^i P$, with $|k_i| < 2^{w-1}$.

For example, if the scalar k = 379 = $(101111011)_2$, so $NAF_2(k)$, $NAF_3(k)$, and $NAF_4(k)$ can be computed (with $-1 = \overline{1}$):

1. $NAF_2(k) = (1\ 0\ \overline{1}\ 0\ 0\ 0\ 0\ \overline{1}\ 0\ \overline{1})$

2. $NAF_3(k) = (3\ 0\ 0\ 0\ 0\ \overline{1}\ 0\ 0\ 3)$

3. $NAF_4(k) = (3\ 0\ 0\ 0\ 0\ 0\ 0\ \overline{5})$

Algorithm 8 presents DA method using NAF of scalar k on fixed-size windows.

Algorithm 8. NAF method with fixed size windows.

```
Input: NAF(k), P ∈ E (Fp), precomputed points [j]P for j ={1, 3,..., (2^(w−1)-1)}
Output: Q= [k]P
Begin
    Q ← ∞                                   //begin scanning from most significant digit to last
  for(i ← l − 1 to 0)do                     significant digit.
    Q ← 2Q                                  // compute point doubling
  if (k_i ≠ 0) then
    if (k_i > 0) then
      |Q ← Q + [k_i]P;                      // compute point addition
    end
    else
      |Q ← Q − [k_i]P;
    end                                     //compute point substraction
      end
      end
    return (Q)
end
```

The average density of non-zero digits for all NAF (k) with length l digits is approximately $l/(w+1)$. Thus, Algorithm 8 performs on average $(l-1)$ point doublings and $l/(1+w)$ point additions. However, this method generates precomputed points $[j]P$ for j=1, 3, . . ., $2^{w-1}-1$. Despite the cost of precomputed points $\left(1 \text{ point doubling} + \left(2^{w-2}\text{ -1}\right) \text{ point additions}\right)$, the usage of $NAF_w(k)$ with windows remains more interesting than the one without window.

A last generalization of this method is to use $NAF_w(k)$ with variable window size (or sliding) lengths with a maximum number of digits. These windows begin and end with a non-zero. If we take the example of $NAF_2(k)$ =(1 0 $\overline{1}$ 0 0 0 0 $\overline{1}$ 0 $\overline{1}$) with sliding windows having a maximum length of three digits, these windows begin and end with non-zero digits. We thus obtain

$$NAF_2(k) = \underline{1\,0\,\overline{1}}\ 0\ 0\ 0\ 0\ \underline{\overline{1}\,0\,\overline{1}}$$

The precomputed points are [3]P and [5]P; the scalar multiplication is as follows: [3]P → [6]P(point doublings) → [12]P(point doublings) → [24]P(point doublings) → [48]P(point doublings) → [96]P(point doublings) → [192]P(point doublings) → [384]P(point doublings) → [379]P(point subtraction of -[5]P). Thus, we perform 8 point operations, against 12 in the case where the windows are fixed.

3.3 Mutual opposite form (MOF) algorithm

More recent mechanisms like the mutual opposite form (MOF) [22] and the complementary recoding algorithm [23] used signed representation digits $\{-1, 0, 1\}$.

In MOF, the representation of the scalar k is obtained by subtracting each k_{i-1} bit from that of k_i. The most significant bit is 1 and the least significant digit is -1. Its output is comparable to that of NAF.

For example, if the scalar k = 379 = $(101111011)_2$, then MOF (k) = 1 $\overline{1}$ 1 0 0 0 $\overline{1}$ 1 0 $\overline{1}$ can be calculated. The conversion is simpler than that of NAF because it only requires subtraction operations. In addition MOF can scan bits or digits from left to right or vice versa, which is more flexible.

3.4 One's complementary recoding algorithm (CR1)

In one's complementary recoding method, the representation of the scalar k is obtained through its complement $\overline{k}$: $\sum_{i=0}^{l-1} k_i 2^i = 2^l - \overline{k} - 1$. The $\overline{k}$ complement is obtained by inverting each bit of the k scalar. For example, if the scalar $k = 379 = (101111011)_2$, then it can be computed : $k = 29 - \overline{k} - 1 = (1000000000–010000100 - 1)_2 = \left(10\overline{1}0000\overline{1}00–1\right)_2$. Thus, we can see that the density of the non-zero bits is reduced from 7 to 4. However, if the number of 1 in the original k scalar is greater than $l/2$, the method is not more interesting because the goal is to have the least 1 in the final representation.

3.5 Double-base number system

In the methods discussed above, the scalar is represented in a single base; the double-base numbering system (DBNS) offers a representation in two bases [11]. The scalar k is represented as a sum of combined powers of 2 and 3: k=$\sum_{i=1}^{l} k_i 2^{a_i} 3^{b_i}$, where $k_i \in \{-1, 1\}$ and a_i, $b_i \geq 0$. The direct usage of this system can induce a high computational cost: $\sum_{bi}$triples, $\sum_{ai}$doublings. Significant improvement can reduce

costs by reusing all intermediate calculations. We keep the initial representation of k with the additional constraint that the exponents form two decreasing sequences: $a_{max} \geq a_1 \geq a_2 \geq$ ..... a_{the} and $b_{max} \geq b_1 \geq b_2 \geq$ .....a_{the}. This formulation makes it possible to calculate only a_{max} doublings, b_{max} triplements, and *(lt - 1)* additions. For example, $752 = 2^3 \times 3^4 + 2^2 \times 3^3 - 2^2$.

The scalar multiplication is as follows: 2^2 (3^3 ($2 \times 3P + P$) -P). Thus, the cost of scalar multiplication is *4 triplements + 3 doublings + 2 additions*. This approach has been generalized using a slightly larger number space requiring pre-calculated points [24]. In this case the values of k_i are prime numbers other than 3: {±1, ±5, ±7, ±11}.

3.6 Comparison

If memory storage is available, the precomputed points can be used to decrease the computation time. The window method or block can be used differently on signed representations such as NAF, MOF, complement coding, or unsigned representations such as double-and-add. If we are interested in sliding window representation, the number of precomputed points varies according to the methods. Take the example of variable windows size (sliding) having a maximum number of five digits.

For the double-and-add method, we will have all the odd combinations of the maximum of 5 bits, that is, which begin and end with a 1. We will thus have at most 15 precomputed points: [3]P, [5]P, [7]P, [9]P, [11]P, [13]P, [15]P, [17]P, [19]P, [21]P, [23]P, [25]P, [27]P, [29]P, and [31]P.

For wNAF method, the blocks are processed through variable windows size (or sliding) having a maximum number of five digits. These windows begin and end with a non-zero digit. As a result, the value V_i of each block of the scalar k is odd and is less than 2^w. There are no two consecutive non-zero digits, so the number of zeros is at least equal to the number of zero digits in the −1 block. The maximum number of precomputed points required is $(2^{w-2}) - 1$. If the maximum length of the window is 5 bits, the largest corresponding precomputed point is $\underbrace{10101}_{w=5}$ = 21P, and possible combinations for precomputed points are the following:

w=3bits	w=4bits	w=5bits	w=5bits
1 0 1=5P	1 0 0 1= 9P	1 0 1 0 1= 21P	$\bar{1}$ 0 $\bar{1}$ 0 $\bar{1}$=-21P
1 0 $\bar{1}$=3P	1 0 0$\bar{1}$ = 7P	1 0 1 0 $\bar{1}$= 19P	$\bar{1}$ 0 $\bar{1}$ 0 1=-19P
$\bar{1}$ 0 1=-3P	$\bar{1}$ 0 0 1= -7P	1 0 0 0 1= 17P	$\bar{1}$ 0 0 0 $\bar{1}$=-17P
$\bar{1}$ 0 $\bar{1}$=-5P	$\bar{1}$ 0 0 $\bar{1}$= -9P	1 0 0 0$\bar{1}$ = 15P	$\bar{1}$ 0 0 0 1=-16P
		1 0 $\bar{1}$ 0 1= 13P	$\bar{1}$ 0 1 0 $\bar{1}$=-13P
		1 0 $\bar{1}$ 0 $\bar{1}$= 11P	$\bar{1}$ 0 1 0 1=-13P

Note that the negative points are the symmetrical positive points, they are neither stored nor computed, and they are obtained almost free. For windows with a maximum size of 5 bits, the number of precomputed points is 10.

MOF uses a signed representation just like NAF, but there can be two consecutive non-zero digits. For windows with a maximum of 5 bit length, the derivation of the computed points is done by subtracting each bit k_{i-1} from the block with that of k_i.

For example, for some values (16–31) of 5-bit blocks, we have:

10000. .10000 $1\bar{1}000 \rightarrow P$	10001. .10001 $1\bar{1}001\bar{1} \rightarrow 9P$	10010. .10010 $1\bar{1}01\bar{1}0 \rightarrow 9P$	10011. .10011 $1\bar{1}010\bar{1} \rightarrow 5P$
10100. .10100 $1\bar{1}\bar{1}00 \rightarrow 5P$	10101. .10101 $1\bar{1}\bar{1}\bar{1}\bar{1} \rightarrow 11P$	10110. .10110 $1\bar{1}10\bar{1}0 \rightarrow 11P$	10111. .10111 $1\bar{1}100\bar{1} \rightarrow 3P$
11000. .11000 $10\bar{1}000 \rightarrow 3P$	11001. .11001 $10\bar{1}\bar{1}\bar{1}0 \rightarrow 13P$	11010. .11010 $10\bar{1}01\bar{1} \rightarrow 13P$	11011. .11011 $10\bar{1}10\bar{1} \rightarrow 7P$
11100. .11100 $100\bar{1}00 \rightarrow 7P$	11101. .11101 $100\bar{1}1\bar{1} \rightarrow 15P$	11110. .11110 $1000\bar{1}0 \rightarrow 15P$	11111. .11111 $10000\bar{1} \rightarrow P$

The remaining combinations give the same negative values. Thus, the number of precomputed points is 7: [3]P, [5]P, [7]P, [9]P, [11]P, [13]P, and [15]P.

Complement recoding uses the same representation of MOF, but for the derivation of precomputed points, it takes all combinations of up to 5 bits, beginning and ending with a non-zero number, i.e., $(2^{w-1} - 1) = 15$ precomputed points: [3]P, [5]P, [7]P, [9]P, [11]P, [13]P, [15]P, [17]P, [19]P, [21]P, [23]P, [25]P, [27]P, [29]P, and [31]P. **Table 1** presents a comparison between different methods.

3.7 Scalar reduction method

We have developed a scalar reduction (SR) algorithm; its main advantage is that it can be easily applied to almost all existing fast scalar multiplication methods described in previous sections. This scalar reduction scheme is an improvement based on the negative of a point. Through this, it makes a specific reduction of the scalar in a selected interval. Using negation is a well-known trick in cryptanalysis as well as in cryptography for computation of scalar multiplication with addition-subtraction chains [25, 26]. This scheme replaces point kP by an equivalent representation of another point tP in the scalar multiplication operation where k and t are scalars and $k > t$. This technique is applied in the interval $[\lfloor n/2 \rfloor + 1, n-1]$, where $\lfloor n/2 \rfloor$ is the integer part function of n/2. As the negative of a point is obtained almost free, we have used it to make fast computation. Given point $P=(x_p, y_p)$ in affine coordinates, the negative of point $kP=(x_{kp}, y_{kp})$ can be computed as $kP=(x_{kp}, y_{kp})$, and then change the sign on the y-coordinate (y_{kp}). Thus, by kP the scalar reduction technique gets equivalent point tP through Eq. (5).

Methods	Cost	Precomputed points	W = 5	Directions
DA	$(l-1)D+\frac{(l-1)}{2}A$	0	....	⇆
NAF	$(l-1)D+\frac{(l-1)}{3}A$	0	....	→
MOF	$(l-1)D+\frac{(l-1)}{2}A$	0	....	⇆
RC1	$<(l-1)D+\frac{(l-1)}{3}A$	0	....	⇆
wNAF	$(l-1)D+\frac{l}{w+1}A$	$< 2^{w-1}-1$	10	→
wMOF	$(l-1)D+\frac{l}{w+1}A$	$<= 2^{w-1}-1$	7	⇆
wRC1	$<(l-1)D+\frac{l}{w+1}A$	$2^{w-1}-1$	15	⇆

Table 1.
Cost for computation and memory storage.

$$\begin{cases} 1.\ \text{If}\ \ k \in]\lfloor n \rfloor, n-1[, kP = tP \text{ where } t = (k-n) \\ 2.\ \text{If}\ \ k \in]0\left\lfloor \frac{n}{2} \right\rfloor, [, kP = tP \text{ where } t = k \end{cases} \tag{5}$$

For example, p =23 is a prime number, just to better explain this technique, but in reality p is much bigger than this. For an elliptic E over F23 defined by $E(F_{23})$, $y^2=x^3+x+1$, then # $E(F_{23})$=28, $E(F_{23})$ is a cyclic group, and P(0, 1) is a generator point. SR makes an equivalent representation on the set of points in $[\lfloor n/2 \rfloor+1, n-1]$, so that computing points 16P, 22P, and 27P can be, respectively, replaced by -12P, -7P, and -P. In this case, the computation of 27P is replaced by the calculation of -P and is almost free. For WSN or IoT embedded devices, replacing the calculation of kP by tP using Eq. (5.1) in $[\lfloor n \rfloor+1, n-1]$ can significantly accelerate scalar multiplication. From Eq. (6), all scalars can be scanned: In the interval $[\lfloor n \rfloor+1, n-1]$, for this example we have the following equivalence representations.

- [15]P = [13]P + 2([1]P)
- [16]P = [12]P + 2([2]P)
- [17]P = [11]P + 2([3]P)
- .....= .....+.....
- .....= .....+.....
- [26]P = [2]P + 2([12]P)
- [27]P = [1]P + 2([13]P)

It can be inferred that $\sum_{k=\lfloor n/2 \rfloor+1}^{n-1} kP = \sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + 2\sum_{1}^{\lfloor n/2 \rfloor-1} kP$. Thus

$$\sum_{k=1}^{n-1} kP = 2\sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + \left\lfloor \frac{n}{2} \right\rfloor P + \sum_{1=1}^{\lfloor n/2 \rfloor-1} kP \tag{6}$$

In SR technique, [15]P, [16]P, ..., [26]P, [27]P] can be replaced, respectively, by $[-13]P$, $[-12]P$,....., $[-2]P$, $[-1]P$ in interval $[\lfloor n \rfloor+1, n-1]$. The expression $\sum_{k=\lfloor n/2 \rfloor+1}^{n-1} kP$ can be replaced by

$$\sum_{k=1}^{n-1} kP = 2\sum_{k=1}^{\lfloor n/2 \rfloor-1} kP + \sum_{k=1}^{\lfloor n/2 \rfloor-1} |k|P + \left\lfloor \frac{n}{2} \right\rfloor P \tag{7}$$

The complexity of scalar multiplication can be determined by the bit length of k which is equal to $\lfloor log_2(k) \rfloor+1$, or $\log_2(k)$ if $k=2^x$, where x is an integer. In binary representation, $log_2(k)$ can be replaced in scalar reduction technique by

$$\log_2\left(k - 2\left(k - \frac{n}{2}\right)\right) = \log_2 k + \log_2\left(1 + \frac{n-2k}{k}\right) \tag{8}$$

Thus, the gain α in bit length is $\alpha = \left|\log_2\left(1 + \frac{n-2k}{k}\right)\right| = \left|\log_2\left(\frac{|t|}{k}\right)\right|$. (9)

NAF	SR	DA	Gain	$\frac{n}{6}$	$\frac{n}{3}$	$\frac{n}{2}$	$\frac{2n}{3}$	$\frac{5n}{6}$	n-1
		√		6579	6572	7604	6555	6931	7471
√				6282	6326	5317	6239	6698	5114
	√			6578	6573	7600	6416	6600	27
√	√			6279	6325	5320	6100	6556	27
	√	√	$\alpha_{(sr/da)}$				2.12%	4.77%	99.63%
√	√		$\alpha_{(sr/naf)}$					1.46%	99.47%
√	√	√	$\alpha_{(sr-naf/da)}$				6.94%	5.41%	99.63%

SR, scalar reduction.

Table 2.
Running times (ms) using affine coordinates.

SR technique is tested in affine coordinates. The scalars are in binary and NAF form combined with the scalar reduction scheme. The gain rate depends on the value of k. For comparison, if $(\alpha_{rs/da})$, $(\alpha_{rs/naf})$, and $(\alpha_{rs-naf/da})$ define, respectively, the gain rate of the scalar reduction (SR) method compared to double-and-add (DA), NAF and SR combined with NAF are compared to DA. The results are given in **Table 2**.

4. Parallelization of scalar multiplication on scalar arithmetic

Parallel computing is another choice for accelerating computation and balancing workload. For distributed system, a task can be divided into smaller ones which are then carried out simultaneously by different processors. The parallel computing for accelerating computation of scalar multiplication is a very hot research topic in cryptography. It can be also achieved through one or more arithmetic levels: on the formulas of operations such as addition and doubling, between the operations themselves, or on the scalar by partitioning it. In most current works, various solutions have been proposed in literature, but in this chapter we present works based on scalar arithmetic.

4.1 Efficient elliptic curve exponentiation

The efficient elliptic curve exponentiation based on point precomputation is proposed in [24]. To calculate Q = kP where Q and P are 2 points represented in Jacobian coordinates and k is a positive integer of 160 bits, a precomputed table which consists of 62 points is prepared.

$A[s] = \sum_{j=0}^{4} a_s, j^{2^{32j}} G3$ and $B[s] = \sum_{j=0}^{4} a_s, j^{2^{16+32j}} G3$ where $1 \leq s \leq 31$ and $a_s,0...,a_s,0$ is a binary representation of $s = \sum_{j=0}^{4} a_{s,j} 2^j$. Then calculation of kP is done by Algorithm 9.

Since this method is based on precomputation, a precomputed table is prepared, and the exponentiation loop can be performed separately by different processors.

4.2 Parallel scalar multiplication on two processors

In [27], two processors and a circular buffer are used to perform parallel scalar multiplication. A buffer acts as a communication channel between the two

Algorithm 9. Elliptic curve exponentiation based on precomputation.

Input: Data $k = \sum_{i=0}^{l-1} k_i 2^i P$
Output: kP,
Begin

for $0 \le j \le 15$ **do**
$u_i = \sum_{i=0}^{4} k_{32i+j} + 2^i$
$v_i = \sum_{i=0}^{4} k_{32i+16+j} + 2^i$
$A[0] = \infty$
$B[0] = \infty$
$T = \infty$
end
for i from 15 to 0 **do**
$T \leftarrow 2T$
$T \leftarrow 2T + A[u_i] + B[v_i]$
end
Return T
end

processors to reduce the average time of the scalar multiplication. As in the producer-consumer problem, the first processor initially reads P and then keeps scanning k_i and computing point doubling. It writes 2^iP into the buffer whenever a non-zero k_i is detected. The second processor reads 2^iP from the buffer and performs additions. The computation is terminated when there is no more 2^iP in the buffer.

4.3 Parallelization by partitioning the scalar

For other schemes, this technique of parallelization consists in partitioning the scalar k (represented on l bits) into *m* fixed-size blocks on SIMD architectures [28]. This partitioning generates precomputed points that need to be calculated and stored prior to starting parallel calculations.

Recent work [29], inspired by [30], uses this technique in *m* blocks of length *v* bits in wireless sensor networks. The scalar is represented on *l* bits and is divided into m blocks B_i of length $vb = l/m$ according to *m* sensors chosen to participate in the computation.

$$kP = B_0 2^{0v} P + B_1 2^{1v} P + B_2 2^{2v} P + \ldots\ldots\ldots + B_{m-1} 2^{(m-1)v} P \tag{10}$$

where $B_i = \sum_{iv}^{iv+v-1} l_i 2^j P$ with l_j the bit on position j on the binary sequence of length l.

This partitioning generates precomputed points $Pi = 2^{ib}P$. For example, consider a scalar k of 160 bits and point P; we want to compute kP on four sensors. The scalar k is broken down into four blocks of 40 bits:

$$kP = \underbrace{(B_{0.0}B_{0.1}\ldots\ldots B_{0.39})_2.2^0P}_{blockB_0} + \underbrace{(B_{1.40}B_{1.41}\ldots\ldots B_{1.79})_2.2^{40}P}_{blockB_1} +$$
$$\underbrace{(B_{2.80}B_{2.81}\ldots\ldots B_{2.119})_2 2^{80}P}_{blockB_2} + \underbrace{(B_{3.120}B_{3.121}\ldots\ldots B_{3.159})_2 2^{159}P}_{blockB_3}$$

Precomputed points are 2^{40}P, 2^{80}P, and 2^{120}P. Note that all parallelism techniques based on scalar partitioning generate pre-calculated points, which must first

be calculated and stored, thus leading to additional memory and energy consumption.

In [31], a parallel computation of *kP* between N sensor nodes is presented by partitioning the scalar k to *m* blocks of length $v = k/N$ bits, and each block is computed by one sensor node. A distributed algorithm (double-and-add, NAF, etc.) composed of m blocks is also proposed, and each block mi of the distributed algorithm operates on one block mi of the scalar. **Algorithms 10** and **11** show, respectively, block i for double-and-add and NAF algorithms.

Algorithm 10. Double-And-Add for node *i*

Input: $d=(d_{v-1},\ldots\ldots\ldots,d_1,d_0)_2$, $P \in E\ (Fp)$
Output: $Q=[d]P$
Begin
$Q \leftarrow \infty$
for $j \leftarrow 0$ *to* $v-1$ **do** // begin scanning bits from right-to-left.
if $d_j = 1$ **then**
$Q \leftarrow Q + 2^{vi}P$
end //2^{vi}P is the pre-computed point
$P \leftarrow 2P$
end
return (Q)
end

Algorithm 11. NAF method for *i*.

Input: $NAF(d)=(d_{v-1},\ldots\ldots\ldots,d_1,d_0)$, $P \in E\ (Fp)$
Output: $Q=[d]P$
Begin
$Q \leftarrow \infty$
for $j \leftarrow 0$ *to* $v-1$ **do** // begin scan from right to left step by step
$P \leftarrow 2Q$
if $(d_j = 1)$ **then** // compute point doubling
$Q \leftarrow Q + 2^{vi}P$ // 2^{vi}P is the pre-computed point
end
if $(d_j = -1)$ **then**
$Q \leftarrow Q - 2^{vi}P$
end
end
return (Q)
end

So as not to compromise security when partitioning scalar, the reliability and efficiency are taken into account. They demonstrate that after partitioning the scalar k to m blocks of length *v*, the node which leads calculation keeps one of the *m* blocks into its local memory and distributes *(m-1)* blocks to others nodes. In this case, a possibility is to send the *(m-1)* blocks securely by symmetric encryption. If blocks are sent randomly without encryption, the intruder, after gaining *(m-1)* blocks of the m blocks, must perform $(m!2^v)P$ to find the private scalar *k*. Moreover, if the intruder gains the *(m-1)* results sent by other nodes, security is not compromised; it has to deal against the ECDLP. So, it is as difficult to find *k* from kP

as k from the $(m\text{-}1)$ points derived from calculation of scalar multiplication on $(m\text{-}1)$ blocks. For each block d_iP, it needs to find d_i. And then after, it also needs to perform $(m!2^v)P$ before getting scalar k.

4.4 Performance measurement

The predominance of scalar multiplication in all operations makes the performance of the cryptosystem relatively based on this scalar operation. Theoretically, the efficiency of the formula using Jacobian coordinates can be determined by the number of multiplication (M) and of square (S) operations which compose it. Operations like addition, subtraction denoted by A, and multiplication with a constant are negligible when faced with square and multiplication of two variables. It is widely accepted that the cost of square is equivalent to 0.6–1 of the cost of multiplication [32–34]. Hence, for a scalar multiplication with a scalar of length of n bits, we can determine the ratio $(r=S/M)$ from which each approach justifies better performance.

5. Conclusion

To perform fast computation of scalar multiplication, which is the major computation involved in ECC, much research has been devoted to the point arithmetic level and the scalar arithmetic. In this chapter, we have presented only works on scalar arithmetic level. All the methods studied are almost based on scanning bits or digits of the scalar with a scan step. In the comparative studies, we found that calculations can be faster if the number of bits scanned is higher. However, scanning a number of bits greater than 1 results in precomputed points that need to be computed or stored before. In future works, we can explore mechanisms for accelerating calculation of precomputed points in order to avoid storing them. Like computing point doubling formula, we can consider effective point operation formulas which should allow to increase the scan step.

Author details

Youssou Faye[1*], Hervé Guyennet[2] and Ibrahima Niang[3]

1 University Assane Seck, Senegal

2 University of Franche Comte, Femto-St, France

3 University Cheikh Anta Diop, Senegal

*Address all correspondence to: yfaye@univ-zig.sn

IntechOpen

© 2019 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

References

[1] Robshaw MJB, Yin YL. Elliptic Curve Cryptosystems, An RSA Laboratories Technical Note. Revised June 27, 1997.

[2] Hakerson D, Menezes A, Vanston S. Guide to elliptic curve cryptography. In: Meloni N, Hasan MA, editors. Elliptic Curve Scalar Multiplication Combining Yaos Algorithm and Double Bases. CHES 2009. NY: Springer-Verlag; 2004. pp. 304-316

[3] Hankerson D, Menezes A, Vanstone S. Guide to Elliptic Curve Cryptography. London: Springer; 2004. pp. 95-123

[4] Muthukuru J, Sathyanarayana B. Fixed and variable size text based message mapping technique using ECC. Global Journal of Computer Science and Technology. 2012

[5] Trappe W. Introduction to Cryptography with Coding Theory. Pearson Education India; 2006

[6] Padma BH, Chandravathi D, Roja PP. Encoding and decoding of a message in the implementation of elliptic curve cryptography using Koblitz method. International Journal on Computer Science and Engineering. 2010;**2**(5): 1904-1907

[7] Sengupta A, Ray UK. Message mapping and reverse mapping in elliptic curve cryptosystem. Security and Communication Networks. 2016;**9**(18): 5363-5375

[8] Singh LD, Singh KM. Implementation of text encryption using elliptic curve cryptography. Procedia Computer Science. 2015;**54**: 73-82

[9] Gordon DM. A survey of fast exponentiation methods. Journal of Algorithms. 1998:129-146

[10] Khleborodov D. Fast elliptic curve point multiplication based on window non-adjacent form method. Applied Mathematics and Computation. 2018; **334**:41-59

[11] Dimitrov V, Imbert L, Mishra PK. Efficient and secure elliptic curve point multiplication using double-base chains. In: Advances in Cryptology, ASIACRYPT05, LNCS, Vol. 3788, Springer-Verlag. 2005. pp. 59-78

[12] Ciet M, Joye M, Lauter K, Montgomery PL. Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes, and Cryptography. 2006;**39**:189-206

[13] Meloni N, Hasan MA. Elliptic curve scalar multiplication combining yaos algorithm and double bases. In: CHES 2009. 2009. pp. 304-316

[14] Al Musa S, Xu G. Fast Scalar Multiplication for Elliptic Curves over Prime Fields by Efficiently Computable Formulas. 2018. Available from: https://eprint.iacr.org/2018/964.pdf

[15] Al Musa S. Multi-Base Chains for Faster Elliptic Curve Cryptography. 2018. Available from: https://dc.uwm.edu/etd/1970

[16] Tian M, Wang Y, Xu S. An efficient elliptic curves scalar multiplication algorithm suitable for wireles network. In: Second International Conference on Networks Security, Wireless Communication and trusted Computing. IEEE Computer Society; 2010. pp. 95-98

[17] Imai V, Masato E. Faster multi-scalar multiplication based on optimal double-base chains. In: World Congress on Internet Security (WorldCIS-2012). IEEE; 2012. pp. 93-98

[18] Khleborodov D. Fast elliptic curve point multiplication based on binary and binary non-adjacent scalar form methods. Advances in Computational Mathematics. 2018:1-19

[19] Tian M, Wang J, Wang Y, Xu S. An efficient elliptic curve scalar multiplication algorithm suitable for wireless network. In: 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing; Wuhan, China. April 2010. pp. 95-98

[20] Suppakitpaisarn V, Imai H, Masato E. Fastest multi-scalar multiplication based on optimal double-base chains. In: 2012 World Congress on Internet Security (WorldCIS), Guelph, Ontario, Canada: IEEE. June 2012. pp. 93-98

[21] Faye Y, Guyennet H, Niang I, Shou Y. Fast scalar multiplication on elliptic curve cryptography in selected intervals suitable for wireless sensor networks. In: Cyberspace Safety and Security. Zhangjiajie, China: Springer International Publishing; 2013. pp. 171-182

[22] Standard Specifications for Public Key Cryptography. IEEE Standard 1363; 2000

[23] Okeya K. Signed binary representations revisited. In: Proceedings of CRYPTO04. 2004. pp. 123-139

[24] Doche C, Imbert L. Extended double-base number system with applications to elliptic curve cryptography. In: International Conference on Cryptology in India. Berlin, Heidelberg: Springer; 2006. pp. 335-348

[25] Bernstein DJ, Lange T, Schwabe P. On the correct use of the negation map in the Pollard rho method. In: International Workshop on Public Key Cryptography. Taormia, Italy: Springer Berlin; 2011. pp. 128-146

[26] Morain F, Olivos J. Speeding up the computations on an elliptic curve using addition-subtraction chains. Theoretical Informatics and Applications Informatique theorique et Applications. 1990;**24**(6):531-543

[27] Ansari B, Wu H. Parallel scalar multiplication for elliptic curve cryptosystems. In: 2005 International Conference on Communications, Circuits and Systems, 2005. Proceedings. Vol. 1. IEEE. May 2005. pp. 71-73

[28] Wu K, Li D, Li H, Chen T, Yu F. Partitioned computation to accelerate scalar multiplication for elliptic curve cryptosystems. In: 2009 15th International Conference on Parallel and Distributed Systems (ICPADS); IEEE. December 2009. pp. 551-555

[29] Shou Y, Guyennet H, Lehsaini M. Parallel scalar multiplication on elliptic curves in wireless sensor networks. In: 14th International Conference on Distributed Computing and Networking (ICDCN), LNCS, Vol. 7730, 2013, Bombay, India. 2013. pp. 300-314

[30] Lim C, Lee P. More flexible exponentiation with pre-computation. In: Advances in Cryptology—CRYPTO94, LNCS, Vol. 839, Springer-Verlag. 1994. pp. 95-107

[31] Faye Y, Guyennet H, Yanbo S, Niang I. Accelerated precomputation points based scalar reduction on elliptic curve cryptography for wireless sensor networks. International Journal of Communication Systems. 2017;**30**(16): e3327

[32] Bernstein D, Hankerson D, López J, Menezes A. Software implementation of

the NIST elliptic curves over prime fields. In: Cryptographers Track at the RSA Conference; Springer. 2001. pp. 250-265

[33] Großschädl J, Avanzi RM, Savaş E, Tillich S. Energy-efficient software implementation of long integer modular arithmetic. In: International Workshop on Cryptographic Hardware and Embedded Systems; Springer. 2005. pp. 75-90

[34] Lim CH, Hwang HS. Fast Implementation of Elliptic Curve Arithmetic in GF (p n); 2000