

Constant-Time Conditions for Left-to-Right Scalar Multiplication

Sergey Agievich

Research Institute for Applied Problems of Mathematics and Informatics

Belarusian State University

agievich@bsu.by, gmail.com

Abstract

We consider methods for scalar multiplication on an elliptic curve where the scalar digits are processed from left to right, that is, from most significant to least significant. We analyze exceptions that may arise during point addition and doubling throughout the multiplication. Eliminating such exceptions is critical for achieving constant-time execution and preventing timing attacks. We establish conditions under which exceptions occur only at the final addition or not at all. Guided by these conditions, one can ensure constant-time behavior by performing all doublings and all intermediate additions using fast formulas that require no exception handling. We examine two variants of the Comb method: SAB-Set and LSB-Set. For both variants, we prove that the constant-time conditions are satisfied under mild restrictions on the base point order and Comb parameters. Additionally, we propose a convenient scalar recoding algorithm for the SAB-Set variant.

Keywords: elliptic curve cryptography, scalar multiplication, constant-time, Comb method.

1 Results

Let $\mathbb{G}$ be a group of points on an elliptic curve over a finite field, let $O \in \mathbb{G}$ be the point at infinity (the identity of the group), and let $P \in \mathbb{G}$ be a point of order $q > 1$. Scalar multiplication maps the point P and a scalar $d \in \{0, 1, \dots, q-1\}$ to the point

$$dP = \underbrace{P + P + \dots + P}_{d \text{ times}} \quad (0P = O).$$

Scalar multiplication is the core primitive of elliptic curve cryptography [5]. It is essential to use *constant-time* scalar multiplication methods—those whose execution time does not depend on d (see, e.g., [10]). This is because the scalar d is typically secret and variations in execution time can leak it through timing attacks [7].

In this paper, we analyze methods that process the pair (P, d) as follows.

1. Precomputations with P . Choose an *alphabet of digits* $\mathcal{D} \subset \mathbb{Z}$ and compute the *table of multiples* $\{P[j] \leftarrow jP : j \in \mathcal{D}\}$.

2. Recoding. Choose a power of two $B = 2^b$ with $b \geq 1$, and write d in the form

$$d = \sum_{i=0}^{h-1} d_i B^i, \quad d_i \in \mathcal{D}. \quad (\star)$$

Here $d_0, d_1, \dots, d_{h-1}$ are the *digits* of d , with d_{h-1} being the most significant digit. The total number of digits h is determined solely by q and is independent of d . Traditionally, digits are written from right to left, so d_0 is the rightmost digit and d_{h-1} is the leftmost.

We track two characteristics of the recoding scheme:

- $m = \min_d d_{h-1}$, the minimum value of the most significant digit over admissible d , and
- $M = \max_{\delta \in \mathcal{D}} |\delta|$, the maximum absolute value of any digit in $\mathcal{D}$.

3. Main Loop. Execute the following algorithm:

```

 $Q \leftarrow P[d_{h-1}]$ 
for  $i = h - 2, \dots, 1, 0$  do
   $Q \leftarrow 2^b Q$  { $b$  doublings}
   $Q \leftarrow Q + P[d_i]$ 
end for
  
```

(★★)

Upon termination, the variable Q contains the scalar multiple dP . Since the digits of d are processed in descending order from d_{h-1} down to d_0 , the scalar multiplication follows the left-to-right scheme.

Let us make several remarks. First, rather than tracking separate lower and upper bounds on digit values, we focus on the maximum absolute value M . This choice aligns with the typical central (or near-central) symmetry of the alphabet $\mathcal{D}$, where $j \in \mathcal{D}$ usually implies $-j \in \mathcal{D}$. Central symmetry of $\mathcal{D}$ is motivated by the fact that point negation in $\mathbb{G}$ is usually inexpensive: determining $-R$ from R is computationally cheap. Therefore, it is reasonable to use methods in which $\{P[j]\}$ contains both mutually inverse points, i.e., the indices j and $-j$ are either both included in $\mathcal{D}$ or both excluded. If central symmetry is maintained, the table size can be halved by storing only the first of two points $P[j]$ and $P[-j]$ and computing the second on the fly as $P[-j] = -P[j]$.

Second, some recoding algorithms impose constraints on the scalar d , for example, requiring d to be odd. When q is odd, as is typical in cryptographic applications, an even scalar d can be replaced with $q - d$, and the resulting point $Q = (q - d)P = -dP$ is then negated to obtain $-Q = dP$. To account for this possible replacement, we allow $d = q$.

Third, when P is a fixed base point, the table $P[j]$ can be precomputed in advance. Furthermore, the doublings $Q \leftarrow 2^b Q$ in the main loop can be eliminated entirely by precomputing a two-dimensional table $P[i][j] = (j(2^b)^i)P$ and simplifying the addition step to $Q \leftarrow Q + P[i][d_i]$. The cost of precomputation is irrelevant to our analysis and is not considered further.

Standard addition formulas on an elliptic curve operate differently depending on whether the operands are equal, whether one of the operands is O , or whether the result of the addition is O . Varying the addition rules based on operand values breaks constant-time guarantees. We therefore monitor the following *exceptions* during the execution of the main loop:

- an operand of an addition or a doubling is the point O ;

- addition of mutually inverse points: $Q = -P[d_i]$;
- the addition $Q + P[d_i]$ is actually a doubling: $Q = P[d_i]$.

In what follows, we establish conditions under which these exceptions do not occur at any step of the main loop, except possibly the final one. At the final step, the absence of exceptions is not guaranteed. To enforce constant-time execution at this step, special *complete* addition formulas, which process all possible operand pairs uniformly, should be used.

Complete formulas could be used at all steps of the main loop, thereby eliminating the need to care about exceptions. However, for some elliptic curves, for example, the Weierstrass curves widely used in cryptography, complete formulas are more costly and their use should be minimized.

Theorem 1. *Let scalar multiplication $(P, d) \mapsto dP$, $0 \leq d \leq q$, be performed using the method described above, and let the following conditions hold:*

- (a) q is odd;
- (b) $0 \notin \mathcal{D}$;
- (c) $M < q(B - 1)/(2B + 1)$;
- (d) $M \leq m(B - 1)$.

Then, in the main loop ($\star\star$), an exception can occur only at the final ($i = 0$) addition. This conclusion also holds if condition (d) is replaced by the following:

- (e) $\mathcal{D}$ contains only odd numbers.

Even if the conditions of the theorem are satisfied and complete formulas are used for the final addition, the scalar multiplication method is not automatically constant-time. It further requires that the following operations are constant-time:

- doubling a non-zero point;
- addition of non-zero, distinct and non-mutually inverse points;
- table lookups from $\{P[j]\}$.

These requirements are expected and relatively straightforward to satisfy.

Under stronger assumptions, the absence of exceptions throughout the entire main loop can be guaranteed.

Theorem 2. *Let conditions (a) – (c), (e) of Theorem 1 hold. If, in addition:*

- (f) d is odd and $d < q$;
- (g) $q + \delta \not\equiv B \pmod{2B}$ for negative $\delta \in \mathcal{D}$,

then no exceptions occur in the main loop ($\star\star$).

We illustrate the theorems with an example.

Example. The method of [9] uses $\mathcal{D} = \{\pm 1, \pm 3, \dots, \pm(B-1)\}$. Only odd scalars d are allowed. The number of digits in the representation of d is $h = 1 + \lfloor \log_B q \rfloor$.

The scalar d is initially written in the radix- B form:

$$d = \sum_{i=0}^{h-1} d_i B^i, \quad d_i \in \{0, 1, \dots, B-1\}.$$

Recoding is performed via the following algorithm:

```

for  $i = h-1, h-2, \dots, 1$  do
   $(d_i, d_{i-1}) \leftarrow (d_i + \text{even}(d_i), d_{i-1} - \text{even}(d_i)B)$ 
end for

```

Here $\text{even}(d_i)$ is 1 if d_i is even, and 0 otherwise.

We have $(m, M) = (1, B-1)$. For odd $q \geq 2B+3$, the conditions of Theorem 1 are satisfied, and an exception can occur only at the final addition. Furthermore, if $d < q$ and $q \bmod B = q \bmod 2B$, no exceptions occur by Theorem 2. Similar results are established in [2, Proposition 1] and [1].

The condition $q \geq 2B+3$ can be omitted. Indeed, if $B > 2$, then for $q < B$ no additions are performed in the main loop, whereas for $B < q < 2B+3$, exactly one addition takes place, which is necessarily the final one. If $B = 2$ and $q < 2B+3$, an intermediate addition can occur only when $q = 5$. A direct evaluation of the possible scalars $d = 1, 3, 5$ shows that no exceptions arise during this addition. $\square$

2 The Comb Method

In this section, we analyze the Comb method proposed in [8]. The method is parameterized by a positive integer w , the window width. From w and $\text{bitlen}(q) = 1 + \lfloor \log_2 q \rfloor$, the number of digits is determined as $h = \lceil \text{bitlen}(q)/w \rceil$.

The parameters w and h govern the resource cost of the method: 2^w points must be precomputed and stored, and $h-1$ doublings and the same number of additions must be performed in the main loop. Because computing the table of multiples costs comparably as the main loop, the Comb method should be used when the base point P is fixed.

Note that the method described in the example above can be configured for a fixed base point P so that computing dP requires $h-1$ additions and no doublings. However, this configuration increases the multiples table size by a factor of h . Thus, Comb provides a compromise between memory footprint and efficiency.

The digits of the scalar d are determined in three stages. First, the scalar is written in binary form:

$$d = \sum_{j=0}^{wh-1} e_j 2^j, \quad e_j \in \{0, 1\}.$$

Next, bit *slices* (also called *combs* or *combsets*) are formed:

$$E_i = (e_{(w-1)h+i}, e_{(w-2)h+i}, \dots, e_i), \quad i = 0, 1, \dots, h-1.$$

Finally, the digits are constructed from the slices E_i as

$$d_i = \sum_{k=0}^{w-1} e_{kh+i} 2^{kh}.$$

To ensure that Comb is constant-time, all slice digits must be made non-zero. One approach to achieving this is proposed in [6] and is known as SAB-Set (Signed All Bit Set).

SAB-Set processes only odd scalars d ($e_0 = 1$) and instead of bits e_j uses values from the set $\{\pm 1\}$. Let these values, in turn, be encoded via bits such that a bit ε_j represents the value $1 - 2\varepsilon_j$.

We propose to perform the recoding $(e_{wh-1}, e_{wh-2}, \dots, e_0) \mapsto (\varepsilon_{wh-1}, \varepsilon_{wh-2}, \dots, \varepsilon_0)$ as follows:

```

 $\varepsilon_{wh-1} \leftarrow 0$ 
for  $j = wh - 2, wh - 3, \dots, 0$  do
     $\varepsilon_j \leftarrow (e_{j+1} + 1) \bmod 2$ 
end for

```

Essentially, we merely invert and shifts the bits of the initial binary representation. The proposed recoding is simpler than the original one from [6].

The proposed recoding is correct because

$$\begin{aligned}
 \sum_{i=0}^{wh-1} (1 - 2\varepsilon_i)2^i &= 2^{wh-1} + \sum_{i=0}^{wh-2} (1 - 2\varepsilon_i)2^i \\
 &= 2^{wh-1} + \sum_{i=0}^{wh-2} (2e_{i+1} - 1)2^i \\
 &= 2^{wh-1} - \sum_{i=0}^{wh-2} 2^i + \sum_{i=1}^{wh-1} e_i 2^i \\
 &= 1 + \sum_{i=1}^{wh-1} e_i 2^i = d.
 \end{aligned}$$

After recoding, the bit slices take the form $(\varepsilon_{(w-1)h+i}, \varepsilon_{(w-2)h+i}, \dots, \varepsilon_i)$, yielding the digits

$$d_i = \sum_{k=0}^{w-1} (1 - 2\varepsilon_{kh+i})2^{kh}.$$

The resulting alphabet is

$$\mathcal{D} = \{\pm 1 \pm 2^h \pm \dots \pm 2^{(w-1)h}\}.$$

The representation $(\star)$ is thus valid with $B = 2$.

Suppose Comb/SAB-Set is used with an odd q and $h \geq w+3$. We show that the conditions of Theorem 1 are then satisfied, so an exception can occur in the main loop $(\star\star)$ only at the final addition.

Since the alphabet $\mathcal{D}$ consists of odd integers, it suffices to verify the inequality $M < q(B-1)/(2B+1)$. Here

$$M = \sum_{k=0}^{w-1} 2^{kh} = \frac{2^{wh} - 1}{2^h - 1}.$$

The inequality is satisfied:

$$M < \frac{2^{wh-w}}{5} < \frac{q}{5} = \frac{q(B-1)}{2B+1}.$$

We use the fact that $h - w \geq 3$ and therefore $5(2^{wh} - 1) < 2^{wh-w}(2^h - 1)$.

Another scheme for making Comb constant-time is LSB-Set, which forces the least significant coordinate of each slice to be non-zero [3, 4, 11]. To achieve this, an additional $(h+1)$ -th

digit is formed during recoding. As in Comb/SAB-Set, only odd scalars d are processed. The alphabet used is

$$\mathcal{D} = \{ \pm (1 + \beta_1 2^h + \dots + \beta_{w-1} 2^{(w-1)h}) : \beta_i \in \{0, 1\} \}.$$

For Comb/LSB-Set, Theorem 1 continues to hold when q is odd and $h \geq w + 3$, ensuring that an exception can only occur at the final addition. Moreover, if $d < q$ and $q \equiv 1 \pmod{4}$, Theorem 2 implies that an exception cannot occur even here. These results are established in [11] under the slightly stronger restriction $h \geq 2w + 5$.

The theorems apply to Comb/LSB-Set due to the following observations:

- M is the same as in Comb/SAB-Set, which again yields $M < q(B - 1)/(2B + 1)$;
- $\mathcal{D}$ consists only of odd digits;
- any negative digit $\delta \in \mathcal{D}$ satisfies $\delta \equiv 3 \pmod{4}$, so that $q + \delta \not\equiv 2 \pmod{4}$.

3 Proofs

3.1 Proof of Theorem 1

We need the following inequalities:

$$M < q/2, \quad m(B - 1)B^{h-1-i} - M(B^{h-1-i} - 1) > 0, \quad i = 1, 2, \dots, h - 2.$$

The first inequality follows directly from the assumption $M < q(B - 1)/(2B + 1)$, and the second—from $M \leq m(B - 1)$.

Let us analyze the main loop ($\star\star$). At step i , prior to the addition $Q + P[d_i]$, the variable Q has the form $s_i P$, where

$$s_i = \sum_{j=i+1}^{h-1} d_j B^{j-i}.$$

After the addition, Q takes the form $(s_i + d_i)P$, and after the subsequent b -fold doubling it becomes $((s_i + d_i)B)P$. In general,

$$s_{i-1} = (s_i + d_i)B, \quad i = h - 1, h - 2, \dots, 1,$$

with initial value $s_{h-1} = 0$ and final value $s_0 = d - d_0$.

We observe the following facts.

1. Since $d_i \neq 0$ and $|d_i| \leq M < q/2$, the initial point $Q = d_{h-1}P$ and all second operands in the additions $Q + P[d_i]$ are distinct from O .
2. For $i \geq 1$, we have $s_i < q - M$. Indeed,

$$\begin{aligned} s_i &= \sum_{j=i+1}^{h-1} d_j B^{j-i} = \frac{1}{B^i} \left(\sum_{j=i+1}^{h-1} d_j B^j \right) = \frac{1}{B^i} \left(d - \sum_{j=0}^i d_j B^j \right) \\ &\leq \frac{1}{B^i} \left(d + \frac{M(B^{i+1} - 1)}{B - 1} \right) \leq \frac{1}{B^i} \left(q + \frac{M(B^{i+1} - 1)}{B - 1} \right). \end{aligned}$$

The right-hand side achieves its maximum at $i = 1$, and this maximum is

$$\frac{1}{B}(q + M(B + 1)) < q - M.$$

The inequality holds because $M(2B + 1) < q(B - 1)$.

3. For $1 \leq i \leq h-2$, we also have $s_i > M$ since

$$\begin{aligned} s_i &\geq mB^{h-1-i} - M \sum_{j=i+1}^{h-2} B^{j-i} \\ &= mB^{h-1-i} - \frac{M(B^{h-1-i} - 1)}{B-1} + M \\ &= \frac{1}{B-1} (m(B-1)B^{h-1-i} - M(B^{h-1-i} - 1)) + M > M. \end{aligned}$$

4. For $1 \leq i \leq h-2$, from the inequalities

$$|d_i| \leq M < s_i < q - M \leq q - |d_i|$$

it follows that $s_i \not\equiv \pm d_i \pmod{q}$. Therefore, the operands of the addition $Q + P[d_i]$ are distinct and non-mutually inverse points. In particular, the sum cannot be O .

5. The doubling $Q \leftarrow 2Q$ cannot yield O because the point being doubled is a non-zero element of the group $\langle P \rangle$ of odd order q .

Together, these facts guarantee that no exceptions can arise during point additions and doublings, except possibly at $i = 0$. Specifically, there is no addition involving O , no doubling of O , no addition of mutually inverse points, and no instances of doubling instead of addition.

Finally, consider replacing condition (d) with (e). Although the lower bound $s_i > |d_i|$ no longer applies, the upper bound $s_i < q - |d_i|$ remains valid and can be strengthened to $|s_i| < q - M \leq q - |d_i|$. Consequently, the restriction $s_i \not\equiv \pm d_i \pmod{q}$ could only be violated if $s_i = d_i$ or $s_i = -d_i$. However, these equalities are impossible because s_i is even while d_i is odd.

3.2 Proof of Theorem 2

We adopt the formalism of the previous proof. At the final step of the loop ($\star\star$), two exceptions may arise: addition of mutually inverse points and doubling instead of addition. We show that the conditions (f) and (g) prevent both exceptions.

If mutually inverse points are added, the resulting sum is $Q = O$. This implies that $d = 0$ or $d = q$, which contradicts (f).

Doubling instead of addition occurs when $s_0 \equiv d_0 \pmod{q}$, or equivalently, when $d \equiv 2d_0 \pmod{q}$. Since $0 < d < q$ and $-q < 2d_0 < q$, only two cases are possible: $d = 2d_0$ with positive d_0 and $d = q + 2d_0$ with negative one. The first case is impossible because d is odd. It remains to examine the second case.

Here $s_0 = q + d_0$. By construction, s_0 is divisible by B , and the quotient $s_0/B = s_1 + d_1$ is odd being the sum of an even s_1 and an odd d_1 . However, the oddness of $s_0/B = (q + d_0)/B$ contradicts (g).

References

- [1] S. Agievich, S. Poruchnik, and V. Semenov. Small scalar multiplication on weierstrass curves using division polynomials. *Mat. Vopr. Kriptogr.* **13** (2) (2022), pp. 17–35.

- [2] J.M. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *J. Cryptogr. Eng.* **6** (2016). <https://eprint.iacr.org/2014/130>, pp. 259–286.
- [3] A. Faz-Hernández, P. Longa, and A.H. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV–GLS curves (extended version). *J Cryptogr. Eng.* **5** (2015), pp. 31–52.
- [4] M. Feng, B.B. Zhu, M. Xu, and S. Li. *Efficient Comb Elliptic Curve Multiplication Methods Resistant to Power Analysis*. Cryptology ePrint Archive, Paper 2005/222. 2005. URL: <https://eprint.iacr.org/2005/222>.
- [5] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2003.
- [6] M. Hedabou, P. Pinel, and L. Bénéteau. Countermeasures for preventing Comb method against SCA attacks. In: *Information Security Practice and Experience. ISPEC 2005*. Ed. by R.H. Deng, F. Bao, H. Pang, and J. Zhou. Vol. 3439. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 85–96.
- [7] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: *Advances in Cryptology – CRYPTO ’96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 104–113.
- [8] C.H. Lim and P.J. Lee. More flexible exponentiation with precomputation. In: *Advances in Cryptology – CRYPTO ’94*. Ed. by Y.G. Desmedt. Vol. 839. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1994, pp. 95–107.
- [9] K. Okeya and T. Takagi. The width- w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In: *Topics in Cryptology, The Cryptographers’ Track at the RSA Conference — CT-RSA 2003*. Ed. by M. Joye. Vol. 2612. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 328–342.
- [10] M. Rivain. *Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves*. Cryptology ePrint Archive, Report 2011/338. <https://eprint.iacr.org/2011/338>. 2011.
- [11] A. Russon. Exploiting dummy codes in elliptic curve cryptography implementations. In: *Symposium sur la sécurité des technologies de l’information et des communications — SSTIC*. https://www.sstic.org/media/SSTIC2020/SSTIC-actes/exploiting_dummy_codes_in_elliptic_curve_cryptogra/SSTIC2020-Article-exploiting_dummy_codes_in_elliptic_curve_cryptography_implementations-russon.pdf. France, Rennes, 2020, pp. 229–249.