A New Multiscalar Multiplication Method Resistant to Timing Attacks

Abhraneel Dutta[1], Veronika Kuchta[2], and Francesco Sica[2]

[1] Department of Mathematics and Statistics, Florida International University
[2] Department of Mathematics and Statistics, Florida Atlantic University

Abstract. Multiscalar multiplication (MSM) is a core operation in modern cryptographic systems, commonly used in various applications such as Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) and Homomorphic Encryption. In elliptic curve–based ZK-SNARK constructions, MSM accounts for up to 80–90% of the total proof generation time, making its optimization critical to improving overall protocol performance. Despite significant progress in accelerating MSM through algorithmic techniques such as Pippenger's method, existing implementations remain vulnerable to timing attacks due to irregular scalar representations and conditional operations on zero digits.

In this paper, we revisit the original Pippenger's MSM algorithm, proposing novel modifications that achieve resistance to timing attacks while at the same time increasing its performance by almost 25%. Our main contribution is a new scalar recoding algorithm that transforms conventional q-ary representations containing zero digits into equivalent non-zero representations. This ensures that all scalar digits are processed uniformly, eliminating timing-based side-channel leaks. Building on this recoding technique, we introduce a secure variant of Pippenger's bucket method, that avoids zero digits. Finally, we demonstrate that employing an endomorphism-based splitting yields shorter digit expansions and further efficiency gains. To the best of our knowledge, this is the first MSM algorithm explicitly designed to mitigate timing attacks within the Pippenger bucket method framework.

1 Introduction

Multiscalar multiplication is a fundamental operation in modern cryptographic systems, playing a critical role in applications such as Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) and Homomorphic Encryption. In ZK-SNARKs, which enable verifiable computations with minimal data revelation, multiscalar multiplication is often employed within elliptic curve-based constructions, such as [5]. Similarly, Homomorphic Encryption schemes, which allow computations on encrypted data, use multiscalar multiplication for efficient algebraic operations. The multiscalar multiplication problem is defined as follows: given a set of elliptic curve points P_i for $i \in \{1, \ldots, n\}$ and the corresponding scalars $a_i \in \mathbb{Z}$, compute the linear combination $\sum_{i=1}^{n} a_i P_i$.

This operation is computationally intensive, particularly for large n, and optimizing it is crucial for the performance of cryptographic protocols which are based on elliptic curves.

In the implementation of ZK-SNARKs, approximately 80% of the proof generation time is dedicated to multiscalar multiplication (MSM). Consequently, optimizing MSM is essential for enhancing the performance of ZK-SNARK protocols.

The primary challenges in multiscalar multiplication (MSM) zk-SNARKs include computational complexity, memory overhead, and scalability. MSM, defined as computing $\sum_{i=1}^{n} a_i P_i$ for elliptic curve points P_i and scalars a_i, exhibits a computational complexity of $O(n)$ for n points. Each operation involves computationally expensive elliptic curve point additions and doublings, making MSM a significant bottleneck in zk-SNARK proof generation, often accounting for 80–90% of the total time [15]. Additionally, MSM requires storing large sets of elliptic curve points and scalars, leading to substantial memory overhead, particularly in zk-SNARK constructions with high-dimensional arithmetic circuits. Regarding scalability, traditional MSM algorithms, such as the Pippenger algorithm, are inherently serial and fail to exploit modern parallel computing architectures like GPUs or distributed systems, limiting their efficiency for large-scale applications.

Recent research has focused on optimizing MSM to improve the performance of zk-SNARKs, particularly through algorithmic improvements and hardware acceleration. Key advancements include Pippenger's algorithm, a new parallel MSM algorithm [9], elastic MSM [16]. A GPU-based implementation of zk-SNARKs that accelerates MSM using a novel parallel algorithm (cuZK) was proposed in [9]. Unlike traditional methods that decompose MSM into smaller independent computations, cuZK treats MSM as a unified task, storing elements in a sparse matrix to achieve near-linear speedup over the Pippenger's algorithm.

Deployment of MSM methods in ZK-SNARK implementation. To demonstrate the role of MSM and the security requirements we briefly review the proof generation process of Groth16 [6]. A ZK-SNARK protocol consists of three algorithms $\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$ For a given circuit C, the Setup algorithm translates it to a Quadratic Arithmetic Program (QAP) instance, which consists of three sets of polynomials, denoted $\boldsymbol{A}(z) = \{A_i(z)\}, \boldsymbol{B}(z) = \{B_i(z)\}, \boldsymbol{C}(z) = \{C_i(z)\}, i \in [1, m]$ with coefficients in a finite field $\mathbb{F}$, and a target polynomial $Z(z) \in \mathbb{F}[z]$. Suppose F is a function describing the QAP, by taking as input n elements of $\mathbb{F}$ and outputting n' elements, i.e. in total there are $N = n + n'$ input/output elements. Then a QAP computes this function F if for a valid assignment $(x_1, \ldots, x_N) \in \mathbb{F}^N$ of inputs and outputs there exist coefficients $(x_{N+1}, \ldots, x_m)$ such that the target polynomial $Z(z)$ divides the polynomial $P(z) := \boldsymbol{A}(z) \cdot \boldsymbol{B}(z) - \boldsymbol{C}(z)$, i.e.

$$P(z) = \left(A_0(z) + \sum_{i=1}^{m} x_i A_i(z)\right) \cdot \left(B_0(z) + \sum_{i=1}^{m} x_i B_i(z)\right) - \left(C_0(z) + \sum_{i=1}^{m} x_i C_i(z)\right).$$

In other words this means that the target polynomial $Z(z)$ satisfies the following equation $\boldsymbol{A}(z) \cdot \boldsymbol{B}(z) - \boldsymbol{C}(z) = \boldsymbol{H}(z) \cdot \boldsymbol{Z}(z)$ for some polynomial $H(z) \in \mathbb{F}[z]$.

The size of the QAP is m and the degree is the degree of $Z(z)$.

In the Prove algorithm, the prover knows a valid assignment for the circuit C, which is denoted as $(\boldsymbol{x}, \boldsymbol{w})$, where $\boldsymbol{x} = (x_1, \ldots, x_N)$ is public input and the secret part $\boldsymbol{w} = (x_{N+1}, \ldots, x_m)$. The prover evaluates the polynomials $\{A_i(z)\}_{i\in[N+1,m]}, \{B_i(z)\}_{i\in[N+1,m]}, \{C_i(z)\}_{i\in[N+1,m]}$ for $i \in [N+1, m]$ at a random value s, which is provided by the `Setup` algorithm and computes $\tilde{A}(s) = \sum_{i=N+1}^{m} x_i A_i(s)$, $\tilde{B}(s) = \sum_{i=N+1}^{m} x_i B_i(s)$ and $\tilde{C}(s) = \sum_{i=N+1}^{m} x_i C_i(s)$. Finally, the prover encodes these values into elliptic curve points $g^{\tilde{A}(s)}$, $g^{\tilde{B}(s)}$, $g^{\tilde{C}(s)}$. The proof then contains these points and is output by the prover.
In the Verify algorithm, the verifier evaluates the polynomials $\boldsymbol{A}(z), (z), \boldsymbol{C}(z)$ at a randomly sampled value s, i.e. computes $\{A_i(s)\}_{i\in[1,N]}, \{B_i(s)\}_{i\in[1,N]}$, and $\{C_i(s)\}_{i\in[1,N]}$ for $i \in [1, N]$ and the corresponding polynomials $\bar{A}(s) = \sum_{i=1}^{N} x_i A_i(s)$, $\bar{B}(s) = \sum_{i=1}^{N} x_i B_i(s)$ and $\bar{C}(s) = \sum_{i=1}^{N} x_i C_i(s)$. Finally, the verifier encodes these values into elliptic curve points $g^{\bar{A}(s)}$, $g^{\bar{B}(s)}$, $g^{\bar{C}(s)}$ and checks the verification equations by performing computations over an elliptic curve. A major task of the proof generation algorithm Prove and the verification algorithm `Verify` is to encode the public key with the witness vector $(\boldsymbol{x}, \boldsymbol{w})$, which is essentially an MSM instance with the witness vector serving as scalars.

Our Contribution In this paper, we explore generalizations of Pippenger's MSM method. We propose thorough modifications of the Pippenger's method to avoid timing attacks. In [7] the authors present a new scalar recoding algorithm which allows for regular implementation of multiscalar multiplication algorithm. This recoding algorithm transforms conventional q-ary representation of scalars containing zeros into equivalent non-zero representations. It is important to note that the protocol in [7] only works for odd scalars. Given an odd integer of width c we map it into a non-zero, signed q-ary expansion $a = \sum_{j=0}^{L-1} d_j q^j$ with $d_j \in \{\pm 1, \pm 3, \pm 5, \ldots, \pm(2^c - 1)\}$. To deal with even scalars, we adapt our multiscalar multiplication by replacing each even a_i by $(a_i + 1)$. This modification leads to an additional sum $\sum_{a_i \in \mathcal{M}_e} P_i$, which will be computed separately and subtracted at the end of the computation. This specification is described in the preamble of section 3. This non-zero digit representation for odd integers is described in Section 2.1. After completing this procedure, we present a new algorithm which is based on the basic idea of Pippenger's bucket method; however, the new recoding algorithm provides security guarantees against timing attacks, that, to the best of our knowledge, has not been considered before. The main idea behind Pippenger's bucket method is to organize the points into buckets according to the digit values of their corresponding scalars. The scalars are represented in base q and points with even scalars are modified to get instead odd scalars which are then expanded using the JT algorithm [7]. This is the moment

where we apply our recoding algorithm from Section 3 to avoid timing attacks. In the end instead of skipping zero digits as it happens in Pippenger's bucket method, we process every scalar digit in a regular fashion. The new bucket filling algorithm with non-zero digits is derived and we show that it is possible to get a shorter digit expansion representation if we use a fast endomorphism ω attached to the usual j-invariant zero elliptic curve used in most applications; this is described in Section 3.1. Finally we provide a comparison of our algorithm with Pippenger's method and conclude that we achieve an improvement of 25%.

Organization of the paper. Section 2 reviews MSM preliminaries, a brief introduction to the Straus' [13] and Pippenger MSM [12] techniques. Section 2.1 surveys briefly zero-free signed-digit recoding methods for timing attack resistant MSM, introduces branchless bucketing templates, and explains the recoding choice we adopt to harden our MSM against timing attacks. In Section 3 we construct a new MSM algorithm resistant against timing attacks. Section 3.1 then leverages an efficient curve endomorphism to group bases and shorten digit expansions, yielding a $2n$-base, single-scan variant. Section 4 summarizes our contributions and situates them against classical Pippenger, emphasizing zero-free recoding, the endomorphism based grouping, and their combined effect on MSM cost. Section 4.1 provides an asymptotic cost comparison accounting for halved digit length, doubled bases, and even scalar compensation. Section 4.2 outlines next research steps and our rationale for implementing the method across multiple platforms.

1.1 Notations

In this section we collect the symbols and notations used throughout the paper. We provide a quick reference so readers can map the goals to the formal algorithms and analyses that follow.

$E/\mathbb{F}_p$ (curve) Elliptic curve over the prime field $\mathbb{F}_p$.
$\mathbb{F}_p$ (field) Prime field.
r (integer) Order of the subgroup $\mathcal{G}$.
$\mathcal{G} \subseteq E(\mathbb{F}_p)$ (group) Cyclic subgroup of order r used for MSM.
P_i (point) i-th base point (from the CRS/SRS).
n (integer) Number of base points (per MSM).
$a_i \in [0, r)$ (scalar) i-th scalar.
c (integer) Window width (bits).
$q = 2^c$ (integer) Radix for windowed recoding.
h (integer) Number of windows (typically $\lceil \log_q r \rceil$).
L (integer) Digit length used by algorithms (often $L = h$).
$d_{i,j}$ (digit) Zero-free signed digit at position j for scalar a_i.
B (set) Allowed absolute digit values (default $\{1, 3, \ldots, q-1\}$).
$B[k]$ (point) Bucket accumulating points with $|d_{i,j}| = k$.

S_j (point) Reverse-accumulated sum for window j.
ϕ (endomorphism) Efficient curve endomorphism on $\mathcal{G}$.
$\mathcal{O}(P)$ (set) ϕ-orbit of P: $\{P, \phi(P), \phi^2(P), \ldots\}$.
ω (constant) Cube root of unity with $\omega^2 + \omega + 1 = 0$.
$\nu \in \mathbb{Z}[\omega]$ (element) Eisenstein divisor used in scalar splitting.
JT Recoding$(\cdot)$ (map) Zero-free signed-digit recoding (e.g., Joye–Tunstall).

2 Preliminaries

We provide a concise overview of Pippenger's algorithm for multiscalar multiplication (MSM), building upon the foundational Straus method, which we briefly introduce first.

Straus' Method Let $\mathcal{G}$ be an elliptic curve subgroup of prime order r, $n \geqslant 1$, and $S_{n,r}$ be the following n-scalar multiplication over fixed points $P_1, \ldots, P_n \in E$,

$$S_{n,r} = \sum_{i=1}^{n} a_i P_i \ , \tag{1}$$

To compute $S_{n,r}$, the Straus method precomputes 2^{nc} points

$$\Big\{ \sum_{i=1}^{n} b_i P_i | \forall b_i \in [0, 2^c - 1], i \in [1, n] \Big\} \ ,$$

where c is a small integer. Next, the algorithm divides each a_i from (1) into segments of length c, i.e.

$$a_i = a_{i,h-1} \| a_{i,h-2} \| \cdots \| a_{i,1} \| a_{i,0} = \sum_{j=0}^{h-1} a_{i,j} 2^{jc}, i \in [1, n] \ ,$$

where $h = \lceil \log_2(r)/c \rceil$ and $0 \leqslant a_{ij} < 2^c$ for $j \in [1, h-1]$. The algorithm retrieves the point

$$S_{n,2^c} = \sum_{i=1}^{n} a_{i,h-1} P_i$$

from the precomputation table, doubles it c times, adds the precomputed point $\sum_{i=1}^{n} a_{i,h-2} P_i$ to obtain

$$S_{n,2^{2c}} = \sum_{i=1}^{n} (a_{i,h-1} \| a_{i,h-2}) P_i \ .$$

After $h-1$ repetitions, we obtain

$$S_{n,2^{hc}} = \sum_{i=1}^{n} (a_{i,h-1} \| a_{i,h-2} \| \ldots \| a_{i,0}) P_0 \ . \tag{2}$$

Pippenger's Bucket Method This method follows the same approach as the Straus method, differing only in the computation of

$$S_{n,2^c} = \sum_{i=1}^{n} a_{i,j} P_i \ ,$$

where $j \in [0, h-1], h = \lceil \log_2(r)/c \rceil$. Let $B = \{b_1, \ldots, b_{|B|}\}$ be a set of non-negative integers containing zero. Given scalars $a_i, 0 \leqslant a_i < r$, the Pippenger method first computes [10, Algorithm 6] a signed radix-q representation

$$a_i = \sum_{j=0}^{h-1} a_{ij} q^j \ ,$$

where $h = \lceil \log_q r \rceil$, $-q/2 \leqslant a_{ij} < q/2$ and for every $i \in [1, n], j \in [0, h-1]$,

$$a_{ij} = \epsilon_{ij} b_{ij}, \text{ where } \epsilon_{ij} \in \{\pm 1\}, b_{ij} \in B = \{0, 1, \ldots, q/2\}. \tag{3}$$

Then,

$$\begin{aligned}
S_{n,r} = \sum_{i=0}^{n} a_i P_i &= \sum_{j=0}^{h-1} q^j \sum_{i=1}^{n} a_{ij} P_i \\
&= \sum_{j=0}^{h-1} q^j \sum_{i=1}^{n} b_{ij} \epsilon_{ij} P_i \\
&= \sum_{j=0}^{h-1} q^j \sum_{i=1}^{n} \Big(\sum_{k=1}^{|B|} b_k \cdot \sum_{i \text{ s.t. } b_{ij}=b_k} \epsilon_{ij} P_i \Big) \\
&= \sum_{j=0}^{h-1} q^j \sum_{k=1}^{|B|} b_k S_k^{(j)} \\
&= \sum_{j=0}^{h-1} q^j \mathfrak{S}_j \ ,
\end{aligned}$$

where

$$S_k^{(j)} = \sum_{\substack{i \text{ s.t.} \\ b_{ij}=b_k}} \epsilon_{ij} P_i, \quad b_k \in B \ ,$$

and

$$\mathfrak{S}_j = \sum_{k=1}^{|B|} b_k S_k^{(j)} \ .$$

All $S_k^{(j)}$'s can be computed with at most $n - (|B| - 1)$ additions; $\mathfrak{S}_j$ is then computed using Algorithm 1 with at most $2(|B|-1)+d-3$ additions, where

Algorithm 1 Subsum accumulation algorithm [10]

Input: $B = \{b_1, b_2, \ldots, b_{|B|}\}, S_1, S_2, \ldots, S_{|B|}$
Output: $S = b_1 S_1 + \cdots + b_{|B|} S_{|B|}$

1: Define a length-$(d+1)$ array $\mathsf{tmp} = [0] \times (d+1)$ ▷ $d = \max |b_i - b_{i-1}|$
2: **for** $i = |B|$ to 1 by -1 **do**
3: $\quad \mathsf{tmp}[0] = \mathsf{tmp}[0] + S_i$ ▷ $\mathsf{tmp}[0] = \Sigma_i$
4: $\quad k = b_i - b_{i-1}$
5: $\quad \mathsf{tmp}[k] = \mathsf{tmp}[k] + \mathsf{tmp}[0]$ ▷ $\mathsf{tmp}[k] = \sum_{\substack{j \geqslant i \\ \beta_j = k}} \Sigma_j$
6: **return** $\sum_{k:\ |k| \leqslant d} k \cdot \mathsf{tmp}[k]$

d is the maximum difference between the two neighboring elements in B. The total cost of computing, via Horner's rule, of

$$S_{n,r} = \sum_{j=0}^{h-1} q^j \mathfrak{S}_j$$

is therefore at most

$$(h-1)(n + |B| + d - 3) + (h-1)(c+1) \approx (h-1)(n + |B|) \qquad (4)$$

elliptic curve additions, where $q = 2^c$. Optimal choice of parameters are when $|B| = q/2 = 2^{c-1} \approx n$.
We justify that Algorithm 1 works by the following analysis. Let $b_0 = 0$ and $\beta_i = b_i - b_{i-1}$ for $1 \leqslant i \leqslant |B|$. Define

$$\Sigma_i = \sum_{j=i}^{|B|} S_j \ ,$$

with $\Sigma_{|B|+1} = 0$. Then

$$\begin{aligned} \sum_{i=1}^{|B|} b_i S_i &= \sum_{i=1}^{|B|} b_i (\Sigma_i - \Sigma_{i+1}) = \sum_{i=1}^{|B|} b_i \Sigma_i - \sum_{i=0}^{|B|-1} b_i \Sigma_{i+1} \\ &= \sum_{i=1}^{|B|} (b_i - b_{i-1}) \Sigma_i = \sum_{i=1}^{|B|} \beta_i \Sigma_i \ . \end{aligned}$$

Remark. Pippenger's algorithm does not require any precomputation, only the storage of $P_1, \ldots, P_n$. Improvements by Luo, Fu and Gong [10] and Fan, Kuchta, Sica and Xu [3] have been presented requiring additional precomputation and storage. Therefore, our work should be compared to the original Pippenger algorithm. Moreover, previous cited work is still vulnerable to timing attacks.

2.1 Regular Recoding for MSM: Methods and Our Selection

When choosing a scalar-multiplication routine for ECC, the implementation must be regular *i.e* each loop iteration performing the same sequence of operations, so that trivial "skip vs. add" patterns do not leak scalar bits to simple power analysis; see the surveys in [1,8]. A common way to enforce regularity is to recode the scalar so that there is no zero window in the digit matrix scanned by the main loop. Signed-digit recodings are natural here because point negation is essentially free on elliptic curves. Notable examples include Möller's nonzero base-q recoding, replacing any zero digit by $-q$ and propagating a carry to higher positions, in [11], Okeya and Takagi's windowed recoding that yields SPA-resistant addition chains in [14], generalized signed digit recoding for all bases by Dutta, Hutchinson and Karabina in [2], and Joye–Tunstall's (JT-Recoding) base-$q = 2^c$ with $c \geqslant 1$ methods that produce either unsigned digits in $\{1, \ldots, q-1\}$ for all scalars or signed odd digits in $\{\pm 1, \pm 3, \ldots, \pm(q-1)\}$ for odd scalars, leading to regular single-dimensional loops [7].

In our grouped MSM, we can apply JT recoding to the GLV components α_i and β_i and then process a single list of $2n$ bases $(P_1, \ldots, P_n,\ \omega P_1, \ldots, \omega P_n)$. For each window j and each nonzero odd digit d, we perform a branchless bucket fill by adding $\mathrm{sgn}(d) \cdot \tilde{P}$ to bucket $u = (|d| + 1)/2$; since $|d|$ is odd, u ranges only over $\{1, \ldots, q/2\}$. Thus, compared to using all magnitudes $\{1, \ldots, q-1\}$, the odd-digit recoding halves the number of buckets per window, and grouping (α, β) into one pass means we do one scan instead of two. The net effect is branchless fills (no zero digits), a single cumulative scan over only $q/2$ buckets, and therefore a substantial reduction in bucket work and point-addition cost per window.

Given an odd integer a and a window width c (so the radix is $q = 2^c$), this routine maps the input to a zero-free, signed q-ary expansion

$$a \;=\; \sum_{j=0}^{L-1} d_j \, q^j,$$

where every output digit lies in the odd set $d_j \in \{\pm 1, \pm 3, \ldots, \pm(2^c - 1)\}$. In other words, it takes a single scalar and produces a vector of nonzero, odd-magnitude digits that exactly reconstruct the input under radix q, with length $L \approx \lceil \mathrm{bitlen}(a)/c \rceil$. The procedure is designed to be regular (branch-free) and constant-shape. If the caller holds an even scalar s, one typically converts it to an odd input (e.g., $a = s + 1$) and carries a one-time final compensation in the target group; the recoder itself always consumes odd inputs and outputs only odd signed digits.

3 A New Multiscalar Multiplication Algorithm Resistant against Timing Attacks.

In this section, we propose a new multiscalar multiplication algorithm in the context of zkSNARKs. The objective of the algorithm is to compute a multi-

Algorithm 2 Odd Signed-Digit Recoding (JT Recooding) [7]

Input: Odd integer $a \in \mathbb{Z}\backslash\{0\}$; window width $c \geqslant 1$; set $q \leftarrow 2^c$
Output: A digit vector $(d_0, \ldots, d_{L-1})$ with $d_j \in \{\pm 1, \pm 3, \ldots, \pm(2^c - 1)\}$ and $a = \sum_{j=0}^{L-1} d_j\, q^j$

1: $j \leftarrow 0$
2: **while** $a \neq 0$ **do**
3: $\quad t \leftarrow a \bmod 2^{c+1}$ $\quad \triangleright\ t \in \{0, 1, \ldots, 2^{c+1} - 1\}$
4: $\quad d \leftarrow t - 2^c$ $\quad \triangleright$ centered odd residue in $(-2^c, 2^c)$; since a is odd, $d \neq 0$ and is odd
5: $\quad d_j \leftarrow d$
6: $\quad a \leftarrow \dfrac{a-d}{2^c}$ $\quad \triangleright$ exact integer division; the new a remains odd
7: $\quad j \leftarrow j + 1$
8: $L \leftarrow j$
9: **return** $(d_0, \ldots, d_{L-1})$

scalar multiplication of the form

$$S_{n,r} = \sum_{i=1}^{n} a_i P_i,$$

where each scalar $a_i < r$. The scalars a_i are first expressed in base $q = 2^c$ where c is a small positive integer i.e. the scalars are decomposed into a q-ary form

$$a_i = \sum_{j=0}^{h-1} a_{i,j} q^j,\ 0 \leqslant a_{i,j} < q,$$

where $h = \lceil \log_q r \rceil$. Consequently, it follows

$$S_{n,r} = \sum_{i=1}^{n} a_i P_i = \sum_{i=1}^{n} \sum_{j=0}^{h-1} a_{i,j} q^j P_i = \sum_{j=0}^{h-1} q^j \sum_{i=1}^{n} a_{i,j} P_i.$$

In the bucket filling phase shown in the algorithm 3 just like in the Pippenger's algorithm, the objective is to organize the points of the elliptic curve according to the digit values of their corresponding scalars in a fixed base-q representation. For a given digit position j, each scalar digit $a_{i,j} \in \{0, 1, \ldots, q-1\}$ determines which of the $q-1$ available buckets the associated point P_i should be placed into. Specifically, if $a_{i,j} = k \neq 0$, then P_i is added to bucket B_k. Points with zero digits are ignored in this step. This process groups all points sharing the same digit value, creating partial sums that are later used in the accumulation phase. The output of this phase is a set of buckets $B[1], \ldots, B[q-1]$, where each bucket contains the sum of points with the same digit value.

In Algorithm 3 there is a conditional branch in step 3 that says whenever $a_{i,j}$ is non-zero then the points P_i's are added to the bucket, otherwise this step is skipped. This leaves this algorithm vulnerable to side channel attacks like timing

attack, and simple power analysis. To overcome this challenge, we can recode the scalars with the help of Algorithm 2 as follows:

$$a_i = (a_{i,h-1}, a_{i,h-2}, \ldots, a_{i,1}, a_{i,0})_q \xrightarrow{\text{JT Recoding}} (d_{i,h-1}, \ldots, d_{i,1}, d_{i,0})_q$$

where $q = 2^c$ for some integer $c \geqslant 1$ and $d_{i,j} \in \{\pm 1, \pm 3, \ldots, \pm(q-1)\}$. After this recoding, the bucket filling stage is modified to be uniform and resistant to timing attacks. Instead of conditionally skipping zero digits, we can process every scalar digit $d_{i,j} \in \{\pm 1, \pm 3, \ldots, \pm(q-1)\}$ in a regular fashion. For each digit, we add either P_i or its negation $-P_i$ to the bucket $B[d_{i,j}]$, depending on the sign. This ensures that all bucket accesses and additions follow a fixed pattern, independent of the scalar values, thereby eliminating data-dependent branches. Importantly, the cost of computing $-P_i$ is negligible, as elliptic curve point negation involves only inverting the y-coordinate, making the method both secure and efficient.

As the JT signed recoding works only on odd scalars we need to perform a certain modification for our algorithm. Let $\mathcal{S}$ be the set of scalars used in multiscalar algorithm. While considering using the scalar recoding algorithm, it should be noted that the scalars should not be even i.e. having a 0 in its least significant entry. Therefore, we divide the scalars into two different groups

The set of all even scalars denoted as $\mathcal{M}_e := \{a_i \in \mathcal{S} : a_i \bmod 2 = 0\}$

The set of all scalars that are odd denoted as $\mathcal{M}_o := \{a_i \in \mathcal{S} : a_i \bmod 2 = 1\}$

Therefore, we partition the scalar set as $\mathcal{S} = \mathcal{M}_e \cup \mathcal{M}_o$ with $\mathcal{M}_e \cap \mathcal{M}_o = \varnothing$. For scalars in $\mathcal{M}_o$, we can apply JT recoding technique directly to compute the partial sum:

$$S_{\mathcal{M}_o} = \sum_{a_i \in \mathcal{M}_o} a_i P_i.$$

On the other hand, for the scalars in $\mathcal{M}_e$ we add 1 to each $a_i \in \mathcal{M}_e$ to make them odd, updating the set to $\mathcal{M}_e + 1 := \{a_i + 1 : a_i \in \mathcal{M}_e\}$, which eliminates zero in the least significant digit. We then apply JT signed recoding method to generate a non-zero q-ary representation for each $a_i \in \mathcal{M}_q + 1$ and compute the partial sum

$$S_{\mathcal{M}_e+1} = \sum_{a_i \in \mathcal{M}_e} (a_i + 1) P_i.$$

This introduces an extra partial sum $\sum_{a_i \in \mathcal{M}_e} P_i$, which we compute separately. Finally, we recover the desired total scalar multiplication as

$$S_{n,r} = \sum_{i=1}^{n} a_i P_i = S_{\mathcal{M}_o} + S_{\mathcal{M}_e+1} - \sum_{a_i \in \mathcal{M}_e} P_i.$$

The partial sum $\sum_{a_i \in \mathcal{M}_e} P_i$ we call the compensation sum of $S_{n,r}$.

Algorithm 3 BucketFill (Unsigned) for window j

Input: Points $P_1, \ldots, P_n$; digits $a_{i,j} \in \{0, \ldots, q-1\}$; base $q = 2^w$
Output: Array $B[1..q-1]$ where $B[k] = \sum_{i:\, a_{i,j}=k} P_i$

for $k = 1$ to $q-1$ **do**
    $B[k] \leftarrow \mathcal{O}$
for $i = 1$ to n **do**
    $k \leftarrow a_{i,j}$
    if $k \neq 0$ **then**
        $B[k] \leftarrow B[k] + P_i$
return B

The Bucket Accumulation Algorithm. After the bucket filling phase, we obtain a set of buckets $B[1], B[2], \ldots, B[q-1]$, where each bucket $B[k]$ contains the sum of all points P_i whose scalar digit $a_{i,j}$ at position j equals k. The objective of the bucket accumulation algorithm is to compute the partial sum

$$S_j = \sum_{k=1}^{q-1} k \cdot B[k],$$

but without explicitly multiplying buckets by scalars. To achieve this, the algorithm proceeds in reverse order from $k = q-1$ down to $k = 1$ and uses a temporary accumulator `tmp` to keep track of the cumulative sum of buckets. In each iteration, the current bucket $B[k]$ is added to `tmp`, and then `tmp` is added to the running result S_j.
This approach guarantees that each bucket $B[k]$ contributes exactly k times to the final result. For example:

Example 1 (Bucket accumulation with $q = 4$). Assume at window j the only nonempty buckets are $B[1] = P_{a_1}$, $B[2] = P_{a_2}+P_{a_3}$, and $B[3] = \mathcal{O}$. Initialize $\texttt{tmp} \leftarrow \mathcal{O}$ and $S_j \leftarrow \mathcal{O}$, then sweep $k = 3, 2, 1$:

$$\begin{aligned}
k = 3: &\quad \texttt{tmp} \leftarrow \mathcal{O}+B[3] = \mathcal{O}, \quad S_j \leftarrow \mathcal{O}+\mathcal{O} = \mathcal{O},\\
k = 2: &\quad \texttt{tmp} \leftarrow \mathcal{O}+B[2] = P_{a_2}+P_{a_3}, \quad S_j \leftarrow \mathcal{O}+(P_{a_2}+P_{a_3}) = P_{a_2}+P_{a_3},\\
k = 1: &\quad \texttt{tmp} \leftarrow (P_{a_2}+P_{a_3})+B[1] = P_{a_2}+P_{a_3}+P_{a_1},\\
&\quad S_j \leftarrow (P_{a_2}+P_{a_3})+(P_{a_2}+P_{a_3}+P_{a_1}) = 2(P_{a_2}+P_{a_3}) + P_{a_1}.
\end{aligned} \tag{5}$$

Thus $S_j = 2 \cdot B[2] + 1 \cdot B[1] = \sum_{k=1}^{3} k \cdot B[k]$, and the bucket $B[2]$ appears exactly two times (once when $k = 2$ and again when $k = 1$), as desired.

3.1 Scalar Decomposition and Recoding in the Endomorphism Case

In proving systems the MSM $\sum_{i=1}^{n} a_i P_i$ dominates prover time, so any reduction in per-window bucket work or passes over the points has a significant impact. Implementations of MSM mostly use the j-invariant zero elliptic curve $y^2 =$

Algorithm 4 Bucket Accumulation Phase for Digit Position j

Input: Buckets $B[1 \dots q-1]$ where $B[k]$ stores the sum of points with digit value k at position j
Output: Partial sum $S_j = \sum_{k=1}^{q-1} k \cdot B[k]$

$S_j \leftarrow \mathcal{O}$ ▷ Initialize result to the identity point
tmp $\leftarrow \mathcal{O}$ ▷ Temporary accumulator
for $k \leftarrow q-1$ **to** 1 **by** -1 **do**
    tmp $\leftarrow$ tmp $+ B[k]$
    $S_j \leftarrow S_j +$ tmp
return S_j

$x^3 + b$ for some constant $b \in \mathbb{F}_p$. It comes equipped with an endomorphism ϕ defined by $\phi(x, y) = (\zeta_3 x, y)$, where $\zeta_3 \neq 1 \in \mathbb{F}_p$ is a cube root of unity. Hence $\phi^3 = \mathrm{id}$ and the endomorphism ring of our elliptic curve is isomorphic to $\mathbb{Z}[\omega]$, where $1 \neq \omega \in \mathbb{C}$ is a cube root of unity. In the endomorphism setting we can first shorten scalars by applying a Euclidean division in $\mathbb{Z}[\omega]$ to obtain $a_i \equiv \alpha_i + \omega\beta_i \bmod \nu$ with roughly half-size components. This already reduces the window count. We then recode α_i, β_i in base $q = 2^c$ using a regular (branch-free) signed-digit recoding in [7] that eliminates zero digits and supports constant-time bucket filling. Crucially, we group the two halves into a single windowed MSM over the unified bases $(P_1, \dots, P_n, \omega P_1, \dots, \omega P_n)$, so each window requires only one cumulative scan of the buckets instead of two. Intuitively, this design compresses the amount of work per window, keeping the fill linear in the number of bases while cutting the scan phase, and preserves a uniform control flow amenable to side-channel resistant implementations.

Firstly, we provide a step-by-step description of our MSM technique. Let the cyclic subgroup E have large order r. As shown in [3], $r \equiv 1 \pmod 3$ so that r splits in the Eisenstein ring $\mathbb{Z}[\omega]$:

$$r = \nu\bar{\nu} \ .$$

Therefore one of $\nu, \bar{\nu}$ (WLOG, ν) annihilates $\mathcal{G}$. Reduction modulo ν gives $\mathbb{Z}[\omega]/(\nu) \cong \mathbb{F}_r$ and sends $\omega \mapsto \lambda \in \mathbb{F}_r$ with $\lambda^2 + \lambda + 1 \equiv 0 \pmod r$. Performing Euclidean division in $\mathbb{Z}[\omega]$ to each scalar:

$$(\delta_i, \rho_i) \ \leftarrow \ \text{Euclidean Division}(a_i, \nu), \qquad a_i = \nu\delta_i + \rho_i, \quad |\rho_i|^2 \leqslant r/3.$$

We write $\rho_i \in \mathbb{Z}[\omega]$ in the $\{1, \omega\}$ basis,

$$\rho_i = \alpha_i + \omega\beta_i, \qquad \max(|\alpha_i|, |\beta_i|) \leqslant \sqrt{r} \ .$$

Consequently,

$$a_i \equiv \alpha_i + \lambda\beta_i \pmod r \quad \Longrightarrow \quad a_i P_i = \alpha_i P_i + \beta_i \phi(P_i) \ ,$$

yielding the GLV-style [4] decomposition over the two bases P_i and ωP_i with short integer coefficients (α_i, β_i) determined by Euclidean division in $\mathbb{Z}[\omega]$.

1. **Step 1 (Parameters and Decomposition).** Fix a radix $q = 2^c$ and an efficiently computable endomorphism ω on the curve (e.g., $\omega^3 = \mathrm{id}$). For each scalar a_i, choose a decomposition

$$a_i \equiv \alpha_i + \omega\beta_i \pmod{\nu} \ ,$$

and expand the two parts in base q using $L = h/2$ windows. Each α_i and β_i are given a q-ary expansion by the JT recoding method in [7]:

$$\alpha_i = \sum_{j=0}^{L-1} a_{i,j} q^j, \qquad \beta_i = \sum_{j=0}^{L-1} b_{i,j} q^j, \qquad a_{i,j}, b_{i,j} \in \{\pm 1, \pm 3 \ldots, \pm q-1\}.$$

The multi-scalar multiplication (MSM) target is

$$S \ = \ \sum_{i=1}^{n} a_i P_i \ = \ \sum_{j=0}^{L-1} q^j \left(\sum_{i=1}^{n} a_{i,j} P_i \ + \ \sum_{i=1}^{n} b_{i,j}\, \omega P_i \right). \tag{6}$$

For signed digits, write $d = \mathrm{sgn}(d)\,|d|$ with $|d| \in \{0, \ldots, q-1\}$ and $\mathrm{sgn}(0) = 0$.

2. **Step 2 (Grouping into a Single Windowed MSM).** For each window j, define $2n$ bases and digits by

$$\begin{aligned} (Q_1, \ldots, Q_{2n}) \ &:= \ (P_1, \ldots, P_n, \ \omega P_1, \ldots, \omega P_n), \\ (d_{1,j}, \ldots, d_{2n,j}) \ &:= \ (a_{1,j}, \ldots, a_{n,j}, \ b_{1,j}, \ldots, b_{n,j}). \end{aligned}$$

The j-th window contribution is

$$W_j \ = \ \sum_{k=1}^{2n} d_{k,j}\, Q_k, \tag{7}$$

and (6) becomes $S = \sum_{j=0}^{L-1} q^j W_j$.

3. **Step 3 (Bucket Filling).** For each $t \in \{1, \ldots, q-1\}$, define the bucket sum

$$B_t^{(j)} \ := \sum_{\substack{1 \leqslant k \leqslant 2n \\ |d_{k,j}| = t}} \mathrm{sgn}(d_{k,j})\, Q_k. \tag{8}$$

Then the window sum (7) can be rewritten as

$$W_j \ = \ \sum_{t=1}^{q-1} t\, B_t^{(j)}. \tag{9}$$

4. **Step 4 (Bucket Scan via Cumulative-Sum Identity).** Define cumulative buckets for $t = 1, \ldots, q-1$ by

$$C_t^{(j)} \ := \ \sum_{u=t}^{q-1} B_u^{(j)}. \tag{10}$$

Then the running-sum identity yields

$$\sum_{t=1}^{q-1} C_t^{(j)} \;=\; \sum_{t=1}^{q-1}\sum_{u=t}^{q-1} B_u^{(j)} \;=\; \sum_{u=1}^{q-1}\sum_{t=1}^{u} B_u^{(j)} \;=\; \sum_{u=1}^{q-1} u\, B_u^{(j)} \;=\; W_j. \tag{11}$$

Equivalently, define the backward partials $A_q^{(j)} := \mathcal{O}$ and

$$A_t^{(j)} \;:=\; A_{t+1}^{(j)} + B_t^{(j)} \qquad (t = q-1, \ldots, 1), \tag{12}$$

so that $C_t^{(j)} = A_t^{(j)}$ and

$$W_j \;=\; \sum_{t=1}^{q-1} A_t^{(j)}. \tag{13}$$

5. **Step 5 (Radix-q Stitching Across Windows).** Process windows from most significant to least significant. Define $S_L := \mathcal{O}$ and

$$S_j \;:=\; q\, S_{j+1} \;+\; W_j \qquad (j = L-1, \ldots, 0), \tag{14}$$

so that the final result is $S = S_0$. Since $q = 2^c$, multiplication by q is realized by c doublings:

$$q\, S_{j+1} \;=\; [2]^c S_{j+1}.$$

i.e per window: c doublings for the radix shift, plus one addition to add W_j.

4 Conclusion

Our goal here is to show, in simple terms, how our MSM algorithm performs over classical Pippenger's method and where it can go next. We first give an asymptotic cost comparison that isolates the three effects, shorter digit expansions, twice as many bases from the endomorphism split, resistance against timing attacks and small even-scalar fixups and show when the method is faster, tending toward a 25% saving as h grows. We then point to immediate next steps towards tighter non-asymptotic bounds, constant-time memory behavior, and portable implementations to guide adoption and further optimization.

4.1 Comparison

We provide a theoretical comparison of the savings incurred by our improvement of the Pippenger algorithm. From (4) we see that the cost of the original Pippenger algorithm is about

$$h(n + |B|)$$

with $n \approx |B| = q/2$. Our algorithm has the net effect of halving size of scalars at the cost of doubling the number of points to $2n$. Our new set $B = \{1, 3, \ldots, q-1\}$ has unchanged size $|B| = q/2$. However, the JT recoding only works for odd

Algorithm 5 Split and Recode with merged digit packing and divisibility screening (updated)

Input: Scalars $a_1, \ldots, a_n \in [0, r)$; points $P_1, \ldots, P_n \in \mathcal{G}$; endomorphism $\phi(P)=\omega P$ with $\omega^3=1$, $\omega^2+\omega+1=0$; divisor $\nu \in \mathbb{Z}[\omega]$; window size c with $q=2^c$; nonzero signed-digit recoder Recode

Output: Unified digits $D_{k,j} \in \{\pm 1, \ldots, \pm(q-1)\}$ for $k \in \{1, \ldots, 2n\}$, $j \in \{0, \ldots, L-1\}$ (no zeros); window count L; compensation sum $\mathsf{Comp} \in \mathcal{G}$

$L \leftarrow 0$; $\mathsf{Comp} \leftarrow \mathcal{O}$
for $i \leftarrow 1$ **to** n **do**
  $(\delta_i, \rho_i) \leftarrow \text{EuclideanDivision}(a_i, \nu)$ ▷ $a_i = \nu\delta_i + \rho_i$, ρ_i short
  Write $\rho_i = \alpha_i + \omega\beta_i$ with $(\alpha_i, \beta_i) \in \mathbb{Z}^2$
  $Q_i \leftarrow \phi(P_i) = \omega P_i$ ▷ no EC additions (endomorphism map)
  (Divisibility screen to ensure odd positive integer for recoding)
  if $\alpha_i \bmod 2 = 0$ **then**
    $\alpha_i \leftarrow \alpha_i + 1$; $\mathsf{Comp} \leftarrow \mathsf{Comp} + P_i$ ▷ store $+P_i$ to add/subtract at the end
  if $\beta_i \bmod 2 = 0$ **then**
    $\beta_i \leftarrow \beta_i + 1$; $\mathsf{Comp} \leftarrow \mathsf{Comp} + Q_i$ ▷ store $+\omega P_i$
  (Nonzero signed recoding in base q)
  $(A_{i,0}, A_{i,1}, \ldots) \leftarrow \text{JT Recoding}(q, \alpha_i)$ with $A_{i,j} \in \{\pm 1, \pm 3, \ldots, \pm(q-1)\}$
  $(B_{i,0}, B_{i,1}, \ldots) \leftarrow \text{JT Recoding}(q, \beta_i)$ with $B_{i,j} \in \{\pm 1, \pm 3 \ldots, \pm(q-1)\}$
  $L \leftarrow \max\big(L, \mathsf{len}(A_{i,\cdot}), \mathsf{len}(B_{i,\cdot})\big)$
  (Merge into unified digit matrix rows $1..2n$**)**
  for $j \leftarrow 0$ **to** $\mathsf{len}(A_{i,\cdot})-1$ **do**
    $D_{i,j} \leftarrow A_{i,j}$
  for $j \leftarrow 0$ **to** $\mathsf{len}(B_{i,\cdot})-1$ **do**
    $D_{n+i,j} \leftarrow B_{i,j}$
return $D_{k,j}$, L, Comp

scalars. Our workaround was to adjust the even scalars in each half, after division by ν, by adding one and subtracting the corresponding base point. On average, we would have to do this for n of the $2n$ points, resulting in a final addition of n points. Hence, our method has an average cost of

$$\frac{h}{2}\left(2n + \frac{q}{2}\right) + n \ ,$$

which, compared to $h(n + q/2)$, gives a relative improvement of

$$\frac{qh/4 - n}{nh + qh/2} \approx \frac{1}{4} - \frac{n}{qh} \approx \frac{1}{4} - \frac{1}{2h} \ ,$$

which, as h grows, approaches 25%.

4.2 Future Work

As a projected future work we can strengthen the theory with deriving tight and non-asymptotic bounds for the average and worst case costs under zero-free recoding, including the exact overhead of even scalar adjustments after division by

Algorithm 6 Grouped bucket MSM with endomorphism (single pass over $2n$ bases) + final compensation add

Input: Base points $P_1, \ldots, P_n \in \mathcal{G}$; endomorphism $\phi(P)=\omega P$; window size c with $q=2^c$; unified digits $D_{k,j}$, window count L, and compensation sum Comp from Alg. 5

Output: $S = \sum_{i=1}^{n} a_i P_i$

Build $2n$-point array (no EC additions): $\tilde{P}_k \leftarrow P_k$ for $k \leqslant n$; $\tilde{P}_k \leftarrow \phi(P_{k-n})$ for $k > n$
$S \leftarrow \mathcal{O}$
for $j \leftarrow L-1$ **down to** 0 **do** ▷ windows MSB→LSB
  Initialize $B[1, 3, \ldots, q-1] \leftarrow \mathcal{O}$
  for $k \leftarrow 1$ **to** $2n$ **do** ▷ branchless fill (nonzero digits)
    $d \leftarrow D_{k,j}$; $t \leftarrow |d|$; $X \leftarrow \mathrm{sgn}(d) \cdot \tilde{P}_k$; $B[t] \leftarrow B[t] + X$
  $A \leftarrow \mathcal{O}$; $W \leftarrow \mathcal{O}$ ▷ scan (running sum)
  for $t \leftarrow q-1, q-3, \ldots, 1$ **do**
    $A \leftarrow A + B[t]$; $W \leftarrow W + A$
  for $\ell \leftarrow 1$ **to** c **do**
    $S \leftarrow 2S$
  $S \leftarrow S + W$
Final compensation add: $S \leftarrow S - \mathsf{Comp}$
return S

ν. Furthermore, we expect to optimize window size c (hence $q = 2^c$) jointly with h and endomorphism parameters to minimize the full cost model. We can also extend the uniform or constant-time model to cover memory-access obliviousness and microarchitectural leakage. Finally, we aim to deliver portable, verified implementations on different platforms like AVX2/AVX-512, CUDA/HIP, FPGA etc. benchmarked on BLS12-381, BN254, and to measure end-to-end impact in zkSNARK/zkVM pipelines (arkworks, gnark, Halo2), including CRS orbit partitioning, MSM scheduling with FFT/NTT, and fixed-base or general-group extensions.

References

1. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F. (eds.): Handbook of Elliptic and Hyperelliptic Curve Cryptography. Discrete Mathematics and Its Applications, Chapman & Hall/CRC, Boca Raton, FL, 2nd edn. (2012)
2. Dutta, A., Hutchinson, A., Karabina, K.: Extending the signed non-zero bit and sign-aligned columns methods to general bases for use in cryptography. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology – INDOCRYPT 2020. pp. 248–270. Springer International Publishing, Cham (2020)
3. Fan, X., Kuchta, V., Sica, F., Xu, L.: Speeding up multi-scalar multiplications for pairing-based zkSNARKs. J. Cryptol. **38**(21) (2025), https://doi.org/10.1007/s00145-025-09540-x

4. Gallant, R.P., Lambert, J.L., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) Advances in Cryptology - Proceedings of CRYPTO 2001. Lecture Notes in Computer Science, vol. 2139, pp. 190–200. Springer (2001)
5. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
6. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)
7. Joye, M., Tunstall, M.: Exponent Recoding and Regular Exponentiation Algorithms. Lecture Notes in Computer Science **5580**, 334–349 (2009)
8. Karabina, K.: A survey of scalar multiplication algorithms. In: Chung, F., Graham, R., Hoffman, F., Mullin, R.C., Hogben, L., West, D.B. (eds.) 50 Years of Combinatorics, Graph Theory, and Computing, chap. 20, pp. 359–386. Discrete Mathematics and Its Applications, Chapman and Hall/CRC, Boca Raton, FL (2019). https://doi.org/10.1201/9780429280092-20
9. Lu, T., Wei, C., Yu, R., Chen, C., Fang, W., Wang, L., Wang, Z.k., Chen, W.: cuzk: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on gpus. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 194–220 (06 2023). https://doi.org/10.46586/tches.v2023.i3.194-220
10. Luo, G., Fu, S., Gong, G.: Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(2), 358–380 (2023), `https://doi.org/10.46586/tches.v2023.i2.358-380`
11. Moller, B.: Securing Elliptic Curve Point Multiplication Against Side-Channel Attacks. Information Security - ISC 2001, Lecture Notes in Computer Science **2200:**, 2001 (324–334)
12. Pippenger, N.: On the evaluation of powers and related problems. In: Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS 1976). pp. 258–263. IEEE Computer Society, Houston, TX, USA (Oct 1976). https://doi.org/10.1109/SFCS.1976.21, preliminary version
13. Straus, E.G.: Addition chains of vectors (problem 5125). The American Mathematical Monthly **71**(7), 806–808 (Sep 1964). https://doi.org/10.2307/2310929, problems and Solutions
14. Takagi, K.O.T.: The Width-w NAF Method Provides Small Memory and Fast Elliptic Curve Scalars Multiplications Against Side- Channel Attacks. T. Topics in Cryptology - CT-RSA 2003, Lecture Notes in Computer Science **2612:328–342** (2003)
15. Xavier, C.F.: PipeMSM: Hardware acceleration for multi-scalar multiplication. Cryptology ePrint Archive, Paper 2022/999 (2022), `https://eprint.iacr.org/2022/999`
16. Zhu, X., He, H., Yang, Z., Deng, Y., Zhao, L., Hou, R.: Elastic MSM: A fast, elastic and modular preprocessing technique for multi-scalar multiplication algorithm on GPUs. Cryptology ePrint Archive, Paper 2024/057 (2024), `https://eprint.iacr.org/2024/057`