# On the Optimization of Pippenger's Bucket Method with Precomputation

Guiwen Luo and Guang Gong

*Dedicated to Doug Stinson on the occasion of his 66th birthday.*

**Abstract** In an elliptic curve group the arithmetic of computing $n$ scalar multiplications then adding them together is called $n$-scalar multiplication. Pippenger's bucket method is the fastest algorithm to compute $n$-scalar multiplication when $n$ is large. In order to optimize the computation of $n$-scalar multiplication over fixed points with the help of precomputation, two new constructions of bucket set that can be utilized in the context of Pippenger-like algorithm are proposed. The first construction decreases the bucket size to $q/3$ for radix $q$. The second construction shrinks down the bucket size to about $q/(2\ell)$ for prime radix $q$ and small integer $\ell$. Those two bucket set constructions yield two faster algorithms computing $n$-scalar multiplication. When instantiating with a 256-bit group order, which approximately provides 128-bit security, for $2^{16} \leq n \leq 2^{20}$, our proposed constructions save more than 20% point additions comparing to the Pippenger's bucket method. Our second construction saves 4% to 10% point additions comparing to the variant of Pippenger's bucket method.

## 1 Introduction

Pippenger's bucket method was first introduced in [21] for the evaluation of powers and monomials in general number fields. With the increasing importance of privacy-preserving computation, it is now widely used to compute $n$-scalar multiplication,

Guiwen Luo
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, N2L 3G1, CANADA. e-mail: guiwen.luo@uwaterloo.ca

Guang Gong
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, N2L 3G1, CANADA. e-mail: ggong@uwaterloo.ca

the arithmetic that dominates the time consumption in paring-based zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [15].

In the pairing-based trusted setup zkSNARKs, it usually follows such a norm that the prover generates the proof by computing $n$-scalar multiplication over fixed points, where all fixed elliptic curve points are welded into the common reference string, and the verifier checks the proof by computing another multi-scalar multiplication, as well as several bilinear parings. The number of points $n$ is usually large. By the qualifier *large*, it means $n > 2^{15}$ in this paper. It covers most of the pairing-based trusted setup zkSNARK applications. For example, one of the most popular applications is using zkSNARK to prove the knowledge of preimage for SHA-256. A SHA-256 circuit has 22, 272 AND gates [10], so $n$ ought to be larger than 22, 272. The more complicated an arithmetic circuit is, the larger $n$ would be. When $n$ is large, the computation of $n$-scalar multiplication is time-consuming, hence taking up vast majority of the time in zkSNARKs for generating and verifying the proof.

There are various methods of calculating $n$-scalar multiplication. The trivial method is computing each single-scalar multiplication by doubling and addition algorithm (it is also known as square and multiplication algorithm in the multiplicative group) [16, Section 4.6.3], then sequentially adding $n$ intermediate results together. When the points are fixed, precomputation can be utilized, one of the representative methods is Knuth's window algorithm [16, 6]. Other attempts to either accelerate computation or do trade-off between time and memory include trying to represent the scalar with creative number systems such as basic digit sets [18, 7] and multi-base number systems [11, 23, 24], trying to do the point addition more efficiently by utilizing different coordinates such as projective coordinates and Jacobian coordinates, trying to use different differential addition chains [20, 4, 9, 22] together with Montgomery form that permits $x$-only arithmetic [19], trying to decompose the scalar into multiple small scalars using GLV method and GLS method [14, 13], and so on.

When it comes to the computation of $n$-scalar multiplication over fixed points with large $n$, Pippenger's algorithm and its variants would outperform other competitors by a large margin [21, 5]. As of today, some well-known zkSNARK applications and libraries that utilize Pippenger's algorithm to compute MSM over fixed points include Bellman[1], gnark [2], TurboPLONK [12], Zcash [3].

Those observations stimulate us to optimize $n$-scalar multiplication algorithms over fix points. We thus present our two main results. For positive integer $n$, we propose the first Pippenger-like algorithm that computes $n$-scalar multiplication in an elliptic curve group of order $r$ with at most $nh + q/3$ additions, with the help of $2nh$ precomputed points, where $q$ is a power of 2, and $q^{h-1} < r \leq (q/2-1)q^{h-1}$. We propose the second Pippenger-like algorithm that computes $n$-scalar multiplication with at most $nh + q/(2\ell)$ additions, with the help of $\ell nh$ precomputed points, where $q$ is a prime such that 2 is a primitive element in the finite field $\mathbb{F}_q$, and $q^{h-1} < r \leq 2^{\ell-1}q^{h-1}$.

The paper is organized as follows. We first review in Section 2 Pippenger's bucket method and its variant, laying the foundation for our newly proposed algorithms. Then in Section 3, we give the general methodology of computing $n$-scalar multiplication.

We use this methodology to present our new algorithms as well as explain the old. Section 4 is dedicated to our two constructions of bucket set and multiplier set, which yield two new algorithms that outperform existing Pippenger's bucket method and its variant. As a reference, we present the instantiation analysis in Appendix to demonstrate the power of our constructions.

Before diving into the content, let us first introduce the notations used throughout the paper.

**Notations.** Without special explanation hereinafter, let $E$ be an elliptic curve group and $r$ be its order. Let $\lfloor x \rfloor$ be the largest integer that is equal to or smaller than $x$, and $\lceil x \rceil$ be the smallest integer that is equal to or greater than $x$. Let $\mathrm{len}(x)$ be the bit length of $x$.

Notation $S_{n,r}$ represents the following $n$-scalar multiplication,

$$S_{n,r} = a_1 P_1 + a_2 P_2 + \cdots + a_n P_n, \tag{1}$$

where $a_i$'s are integers such that $0 \leq a_i < r$ and $P_i$'s are given points in $E$. Radix $q$ is a positive integer used to express a scalar in the radix $q$ representation. Positive integer $h$ is the length of a scalar in its radix $q$ representation, i.e., $h = \lceil \log_q r \rceil$. The term *addition* refers to the point addition arithmetic in $E$. Let us assume for simplicity that the computational cost of doubling and addition in $E$ are the same, denoted as $A$. This follows the norm because in Pippenger-like algorithms, the majority of the computation is addition. The memory size of a point in $E$ is denoted as $P$.

## 2 Pippenger's bucket method and its variants

In this section, Pippenger's bucket method and its variant are reviewed. Although more than 40 years have passed since the original paper [21] was published, Pippenger's bucket method and its variant are still the state-of-the-art algorithms to compute $S_{n,r}$ with large $n$.

### 2.1 Pippenger's bucket method

Here we review Pippenger's bucket method presented in [5, Section 4], which is a special instance of Pippenger's algorithm [21]. It was originally used to compute multi-scalar multiplication in the context of large batch signature verification.

If $r$ is small enough, then

$$\begin{aligned} S_{n,r} &= \sum_{i=1}^{n} a_i P_i \\ &= \sum_{i=1}^{n} \left( \sum_{k=1}^{r-1} k \cdot \sum_{i \ s.t. \ a_i=k} P_i \right) \\ &= \sum_{k=1}^{r-1} k \cdot \left( \sum_{i=1}^{n} \sum_{i \ s.t. \ a_i=k} P_i \right). \end{aligned} \tag{2}$$

Define intermediate subsum (or *bucket* sum) $S_k$,

$$S_k = \sum_{i=1}^{n} \sum_{i \ s.t. \ a_i=k} P_i, 1 \le k < r, \tag{3}$$

then

$$S_{n,r} = \sum_{k=1}^{r-1} k S_k. \tag{4}$$

$S_{n,r}$ is computed by first evaluating all $S_k$ $(1 \le k < r)$ with at most $n-(r-1)$ additions, because there are $n$ points being sorted into $r-1$ subsums, then by using the following method showed in Equation (5) with at most $2(r-2)$ additions to complete the computation,

$$\sum_{i=1}^{d} i S_i = \sum_{i=1}^{d} \sum_{j=1}^{i} S_i = \sum_{j=1}^{d} \sum_{i=j}^{d} S_i. \tag{5}$$

To sum up, when $r$ is small, the cost of computing $S_{n,r}$ is at most $n+r-3$ additions.

In practical zkSNARK applications, $r$ is usually very big, for example, $r$ is at least $2^{256}$ in order to achieve 128-bit security. To compute $S_{n,r}$ in this scenario, Pippenger's bucket method starts with breaking down the scalars. Let $q = 2^c$ be the radix, where $c$ is a small positive integer. For scalar $a_i$ $(0 \le a_i < r)$, suppose its $q$-ary representation is

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

where $h = \lceil \log_q r \rceil$. Then $S_{n,r}$ can be computed as follows,

$$\begin{aligned} S_{n,r} &= \sum_{i=0}^{n} a_i P_i = \sum_{i=1}^{n} \left( \sum_{j=0}^{h-1} a_{ij} q^j \right) P_i \\ &= \sum_{j=0}^{h-1} q^j \left( \sum_{i=1}^{n} a_{ij} P_i \right). \end{aligned} \tag{6}$$

Let us denote $Q_j = \sum_{i=1}^{n} a_{ij} P_i$. The computation of $Q_j$ boils down to the aforementioned case where every scalar is smaller than $q$, thus each $Q_j$ can be evaluated using at most

$$n + q - 3 \tag{7}$$

additions. The rest is

$$S_{n,r} = \sum_{j=0}^{h-1} q^j Q_j, \tag{8}$$

then $S_{n,r}$ can be evaluated by the following Equation (9) similar to Horner's rule,

$$S_{n,r} = Q_0 + q\left(Q_1 + q\left(Q_2 + q\left(Q_3 + \cdots + q\left(Q_{h-2} + qQ_{h-1}\right)\cdots\right)\right)\right). \tag{9}$$

The intermediate scalar multiplications like $q \cdot Q_{h-1}$ are evaluated by doubling and addition algorithm with $(\mathrm{len}(q) + \mathrm{hw}(q) - 1)$ additions, where $\mathrm{hw}(q)$ is the number of 1 in $q$'s binary expression, and this executes $h - 1$ times. So Equation (9) takes

$$(h-1) \cdot (\mathrm{len}(q) + \mathrm{hw}(q) - 1).$$

additions considering 1 extra addition to add $Q_j$ every time.

When $q = 2^c$, $\mathrm{len}(q) = c + 1$, $\mathrm{hw}(q) = 1$. The total cost would be at most

$$h(n + q - 3) + (h-1) \cdot (c+1) \approx h(n+q) \tag{10}$$

additions. Given $n$ and $r$, radix $q$ is carefully selected to minimize the cost of computing $S_{n,r}$.

## 2.2 The variant of Pippenger's bucket method

One drawback of Pippenger's bucket method presented in the previous section is that in order to compute all $Q_j$'s in Equation (8), the accumulation method showed in Equation (5) is executed $h$ times. This can be further optimized if precomputation is used. Here we introduce the well-known variant presented in [8] that bypasses this shortcoming.

Let $q = 2^c$ be the radix, where $c$ is a small positive integer. For scalar $a_i$ $(0 \le a_i < r)$, suppose its $q$-ary representation is

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

where $h = \lceil \log_q r \rceil$. Then $S_{n,r}$ can be computed as follows,

$$\begin{aligned} S_{n,r} &= \sum_{i=0}^{n} a_i P_i = \sum_{i=1}^{n} \left( \sum_{j=0}^{h-1} a_{ij} q^j \right) P_i \\ &= \sum_{i=1}^{n} \sum_{j=0}^{h-1} a_{ij} \cdot q^j P_i. \end{aligned} \tag{11}$$

If the following points

$$\{q^j P_i \mid 1 \le i \le n,\ 0 \le j \le h-1\},$$

are precomputed, which requires the storage cost of

$$nh \cdot P,$$

then $S_{n,r} = S_{nh,q}$ boils down to the case where all scalars are smaller than $q$. The cost is at most

$$nh + q - 3$$

additions.

## 2.3 Further optimization

In the elliptic curve group, $-P$ can be easily computed from $P$ by negating its $y$ coordinate. This observation can be utilized to further optimize the computation of $S_{n,r}$.

Suppose $q^{h-1} < r \le (q/2-1)q^{h-1}$ (the assumption ensures $a_{h-1} \le q/2$.), let us express scalar $a$ $(0 \le a < r)$ in such a way that the absolute value of every digit is no more than $q/2$, i.e.,

$$a = \sum_{j=0}^{h-1} a_j q^j, \text{ where } -\frac{q}{2} \le a_j \le \frac{q}{2}.$$

The cost of Pippenger's bucket method to compute $S_{n,r}$ would be at most approximately

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

additions, since Equation (7) is reduced to roughly $n + q/2$ when evaluating each $Q_j$. And the cost of its variant to compute $S_{n,r}$ becomes at most approximately

$$nh + \frac{q}{2}$$

additions due to the similar reason, with the help of

$$nh$$

precomputed points.

### 2.4 Comparisons of multi-scalar multiplication algorithms

We summarize in Table 1 the precomputation storage and the computational cost of computing $S_{n,r}$ over fixed points by the aforementioned methods, together with our new constructions proposed in Section 4.

Here $q$ is the radix, $h = \lceil \log_q(r) \rceil$, $\ell$ is a small integer such that $2^\ell < q$. Radix $q$ should be selected to minimize the computational cost. For Pippenger's bucket method, its variant and our first construction, it should be satisfied that $q$ is a power of 2, and $q^{h-1} < r \leq (q/2-1)q^{h-1}$. For our second construction, it should be satisfied that $q$ is a prime, 2 is a primitive element in the finite field $\mathbb{F}_q$, and $q^{h-1} < r \leq 2^{\ell-1}q^{h-1}$.

**Table 1** Comparison of different methods that computes $S_{n,r}$

| Method | Storage | Worst case cost |
|---|---|---|
| Pippenger [21, 5] | $n \cdot P$ | $h(n + q/2) \cdot A$ |
| Pippenger variant [8] | $nh \cdot P$ | $(nh + q/2) \cdot A$ |
| Construction I [this paper] | $2nh \cdot P$ | $(nh + q/3) \cdot A$ |
| Construction II [this paper] | $\ell nh \cdot P$ | $(nh + q/(2\ell)) \cdot A$ |

## 3 Methodology of computing multi-scalar multiplication

The goal of this section is to establish a methodology of computing $S_{n,r}$. This methodology is inspired by [8] and initially presented in [17][1]. It is used to derive our new algorithms, and it is suitable for explaining the old as well. Before we dive into the methodology, let us first present a new subsum accumulation method.

[1] The paper [17] is submitted to TCHES 2023 and is under revision following reviewers' comments. For the sake of completeness, we introduce in this paper the methodology. The reasoning and detailed analysis will be available once the revision of [17] is done.

### 3.1 A new subsum accumulation method

When computing $S_{n,r}$ over fixed points by Pippenger's bucket method, one first evaluates intermediate subsums $S_i's \{1 \le i \le m\}$, then utilizes the subsum accumulation method showed in Equation (5) to evaluate $\sum_{i=1}^{m} iS_i$ using $2(m-1)$ additions.

If we want to compute

$$S = b_1S_1 + b_2S_2 + \cdots + b_mS_m,$$

where $1 \le b_1 \le b_2 \le \cdots \le b_m$, and $\{b_i\}_{1\le i\le m}$ is not a sequence of consecutive integers, the method in Equation (5) still can handle it using $2(b_m - 1)$ additions by assigning some of the subsums to be 0. But we can do better. Here we introduce the method presented in [17] that handles this case.

Define $b_0 = 0$, $d = \max_{1\le i\le m}\{b_i - b_{i-1}\}$, $\delta_j = b_j - b_{j-1}$, then $b_i = \sum_{j=1}^{i} \delta_j$. $S$ can be computed by the following equation,

$$\begin{aligned} S &= \sum_{i=1}^{m} b_iS_i = \sum_{i=1}^{m}\left(\sum_{j=1}^{i}\delta_j\right)S_i \\ &= \sum_{j=1}^{m}\delta_j\left(\sum_{i=j}^{m}S_i\right) \\ &= \sum_{k=1}^{d} k \sum_{j=1,\delta_j=k}^{m}\left(\sum_{i=j}^{m}S_i\right). \end{aligned} \tag{12}$$

In Equation (12), every $\sum_{j=1,\delta_j=k}^{m}(\sum_{i=j}^{m} S_i)$ is computed first, then method in Equation (5) is utilized to complete the computation. It can be demonstrated that the cost of Equation (12) is at most $2m + d - 3$ additions.

### 3.2 The methodology

In order to compute $S_{n,r}$, one starts with breaking down the scalars. Let $M$ be a set of integers and $B$ a set of non-negative integers and $0 \in B$. Suppose scalar $a_i$ $(0 \le a_i < r)$ is given in its $q$-ary standard representation

$$a_i = \sum_{j=0}^{h-1} a_{ij}q^j,\ 0 \le a_{ij} < q,\ 1 \le i \le n.$$

If we can find a method to break down every $a_{ij}$ into the product of an element from $M$ and another element from $B$, i.e.,

$$a_{ij} = m_{ij}b_{ij}, m_{ij} \in M, b_{ij} \in B,$$

then $S_{n,r}$ can be rewritten as,

$$S_{n,r} = \sum_{i=1}^{n}(\sum_{j=0}^{h-1} a_{ij}q^j)P_i = \sum_{i=1}^{n}\sum_{j=0}^{h-1} b_{ij} \cdot m_{ij}q^j P_i. \tag{13}$$

If those $nh|M|$ points

$$\{P_{ij} \mid P_{ij} = mq^j P_i, 1 \leq i \leq n, 0 \leq j \leq h-1, m \in M\} \tag{14}$$

are precomputed, then $S_{n,r}$ is transformed into an $nh$-scalar multiplication where every scalar is from $B$. It can be computed by first computing all intermediate subsums corresponding to $b_{ij}$ with at most $nh - (|B| - 1)$ additions, since there are $nh$ points being sorted into $|B| - 1$ subsums. Then using Equation (12) with at most $2(|B| - 1) + d - 3$ additions (since $0 \in B$) to complete the computation, where $d$ is the maximum difference between two neighbor elements in $B$.

Following this methodology, $S_{n,r}$ can be evaluated using at most

$$nh + |B| + d - 4 \tag{15}$$

additions, given that those

$$nh|M| \tag{16}$$

points in Equation (14) are precomputed.

Because the precomputed points are those multiplied by the element from $M$, the set $M$ is called a *multiplier set*. Because all the subsums are corresponding to the scalars in $B$, the set $B$ is called a *bucket set*. Under this methodology, the variant of Pippenger's bucket method has $M = \{1\},\ B = \{0, 1, 2, ..., 2^c - 1\}$, or $M = \{-1, 1\},\ B = \{0, 1, 2, ..., 2^{c-1}\}$.

### 3.3 Trade-off between computational cost and precomputation

When the precomputation memory is limited, the following two popular techniques can be used. Those trade-offs are not adopted in our algorithms when we do the instantiation in Appendix.

#### Use the accumulation multiple times

One can precompute only the following $n|M|$ points

$$\{mP_i \mid 1 \leq i \leq n, m \in M\} \tag{17}$$

and then obtain an algorithm similar to the Pippenger's bucket method by using $h$ times the subsum accumulation Equation (12). The total cost is approximately at

most

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

additions.

### Endomorphism

Another popular technique to shrink down the precomputation size is endomorphism. One may reduce the precomputation size by approximately a factor of 2 using an easily computable endomorphism such as GLV endomorphism that enables us to compute half of the precomputed points on the fly [14].

Suppose GLV endomorphism $\psi : (x, y) \mapsto (\xi x, y)$, where $\xi^3 = 1$ is applicable in the elliptic curve group $E$ of order $r$. Denote $\psi(P) = \lambda P$, then endomorphism $\psi$ provides a shortcut to obtain $\lambda P$ with the cost of only one field multiplication. One can replace the single scalar multiplication $aP$ by the 2-scalar multiplication $a_0P + a_1\psi(P)$, where $a = a_0 + a_1\lambda$ and $|a_0|, |a_1| \approx \sqrt{r}$. It follows that

$$\begin{aligned} S_{n,r} &= \sum_{i=1}^{n} a_i P_i = \sum_{i=1}^{n} (a_{0,i} + a_{1,i}\lambda) P_i \\ &= \sum_{i=1}^{n} (a_{0,i} P_i + a_{1,i} \cdot \lambda P_i) = \sum_{i=1}^{n} (a_{0,i} P_i + a_{1,i} \cdot \psi(P_i)). \end{aligned} \tag{18}$$

Let us rearrange the above equation as a $2n$-scalar multiplication where the scalar is no more than $\sqrt{r}$, and renumber the scalars, so above equation is equivalent to the computation of

$$S_{2n,\sqrt{r}} = \sum_{i=1}^{n} a_i P_i + \sum_{i=1}^{n} a_{n+i} \cdot \psi(P_i). \tag{19}$$

Note that when we apply the methodology to $S_{2n,\sqrt{r}}$,

$$h' = \lceil \log_q \sqrt{r} \rceil = \lceil (\log_q r)/2 \rceil = \lceil h/2 \rceil.$$

And the following $|M|nh'$ points are precomputed,

$$\{mq^j P_i \mid m \in M, 1 \le i \le n, 0 \le j \le h'\}.$$

Given aforementioned precomputed points, one can compute

$$\{mq^j \psi(P_i) \mid m \in M, 1 \le i \le n, 0 \le j \le h'\}$$

on the fly with $\ell nh'$ field multiplications because $mq^j\psi(P_i) = \psi(mq^j P_i)$. The computation of $S_{n,r} = S_{2n,\sqrt{r}}$ would cost at most approximately

$$2n \cdot h' + |B| = n \cdot 2\lceil h/2 \rceil + |B|$$

additions.

## 4 The constructions of multiplier set and bucket set

In this section, we construct two pairs of multiplier set and bucket set $(M, B)$ that can be utilized to speed up the computation of $S_{n,r}$ over fixed points under the methodology presented in Section 3.2. The essential difficulty is to construct bucket set $B$ with smaller size.

Given a scalar $a$ $(0 \le a < r)$ in its standard $q$-ary representation

$$a = \sum_{j=0}^{h-1} a_j q^j,\ 0 \le a_j < q, \tag{20}$$

we will show that our constructions enable to convert scalar $a$ to its radix $q$ representation where every digit is the product of an element from $M$ and an element from $B$, thus yielding efficient $S_{n,r}$ computation algorithms by Section 3.2.

### 4.1 First construction

Let $q$ be a power of 2, $q = 2^c$, and let $q^{h-1} < r \le (q/2 - 1)q^{h-1}$. The multiplier set is defined as

$$M = \{-2, -1, 1, 2\}, \tag{21}$$

The corresponding bucket set is constructed as

$$B = \{0\} \cup \{\, i \mid \omega_2(i) \equiv 0 \bmod 2,\ 1 \le i \le q/2\}, \tag{22}$$

where $\omega_2(i)$ represents the exponent of factor 2 in $i$. Explicitly, if $i = 2^e k,\ 2 \nmid k$, then $\omega_2(i) = e$.

*Claim* For the multiplier set $M$ and bucket set $B$ defined in equations (21) (22), scalar $a$ $(0 \le a < r)$ can be expressed as follows

$$a = \sum_{j=0}^{h-1} m_j b_j q^j,\ m_j \in M, b_j \in B. \tag{23}$$

***Proof*** Let us first demonstrate that for arbitrary integer $t \in [0, q]$, it can be decomposed to

$$t = mb - \alpha q,\ m \in M,\ b \in B, \alpha \in \{-1, 0\}.$$

- If $t \in [0, q/2]$, from the construction of $B$ there exists an element $b \in B$ such that $t = b$ or $t = 2b$. In this case $\alpha = 0$.

- If $t \in (q/2, q]$, then $q - t$ lies in $[0, q/2]$, which goes back to the aforementioned case. Thus there exists an element $b \in B$ such that $q - t = b$ or $q - t = 2b$, which means $t = (-1) \cdot b - (-1) \cdot \alpha$ or $t = (-2) \cdot b - (-1) \cdot \alpha$. In this case $\alpha = -1$. □

Back to the claim, notice the following facts that
i) $a_0 \in [0, q-1]$,
ii) $-\alpha_j + a_{j+1} \in [0, q]$ for all $0 \le j \le h-3$,
iii) $-\alpha_{h-2} + a_{h-1} \in [0, q/2]$.
Given scalar $a$ in its standard $q$-ary representation defined in Equation (20), one first decomposes $a_i = m_i b_i - \alpha_i q$, then update $a_{i+1}$ to be $-\alpha_i + a_{i+1}$ for $0 \le i \le h-2$. The conversion is done.

For every $t \in [0, q]$, a hash table $H$ can be precomputed to store its decomposition, i.e., $H(t) = (m, b, \alpha)$ such that $t = mb - \alpha q, m \in M, b \in B, \alpha \in \{-1, 0\}$. Values $m$, $b$ and $\alpha$ can be retrieved from the hash table instead of being computed on the fly.

The size of $B$ is estimated by

$$\begin{aligned} |B| &= 1 + \sum_{i=1}^{\infty} (-1)^{i+1} \left\lfloor \frac{q}{2^i} \right\rfloor \\ &= 1 + \sum_{i=1}^{c} (-1)^{i+1} \cdot \frac{q}{2^i} \\ &= \frac{q + (-1)^{c+1}}{3} + 1 \\ &< \frac{q}{3} + 1. \end{aligned} \tag{24}$$

The maximum difference between two neighbor elements in $B$ is 2. By the computational cost estimation formula presented in Equation (15), the cost of computing $S_{n,r}$ over fixed points will be at most

$$nh + \frac{q}{3} - 1, \text{ where } h = \lceil \log_q r \rceil. \tag{25}$$

**Theorem 4.1** *Suppose $q$ is a power of 2 and $q^{h-1} < r \le (q/2 - 1)q^{h-1}$. The multiplier set and bucket set defined in equations (21) (22) yield an algorithm to compute $S_{n,r}$ over fixed points using at most*

$$nh + \frac{q}{3} - 1$$

*additions, with the help of $2nh$ precomputed points*

$$\left\{ mq^j P_i \mid 1 \le i \le n, 0 \le j \le h-1, m \in \{1, 2\} \right\}.$$

For point $P = (x, y)$ in $E$ with short Weierstrass form, $-P = (x, -y)$ can be obtained for almost no cost, so the precomputed points associated with negative elements in $M$ are excluded.

## 4.2 Second construction

Let $q$ be a prime such that 2 is a primitive element in the finite field $\mathbb{F}_q$, and $\ell$ a small positive integer such that $2^\ell < q$ and $q^{h-1} < r \leq 2^{\ell-1}q^{h-1}$. The multiplier set is picked as

$$M = \{2^i \mid 0 \leq i \leq \ell - 1\} \cup \{-2^i \mid 0 \leq i \leq \ell - 1\}, \tag{26}$$

and the corresponding bucket set is constructed as

$$B = \{i \mid 0 \leq i \leq 2^\ell\} \cup \left\{2^{i \cdot \ell} \bmod q \mid 0 \leq i \leq \left\lfloor \frac{(q-1)}{2\ell} \right\rfloor\right\}. \tag{27}$$

The idea behind the construction is that

$$\{i \mid -2^{\ell-1} \leq i \leq q + 2^\ell - 1\} \subseteq \{mb - \alpha q \mid m \in M, b \in B\}. \tag{28}$$

*Claim* For the multiplier set $M$ and bucket set $B$ defined in equations (26) (27), scalar $a$ $(0 \leq a < r)$ can be expressed (not necessarily uniquely) as follows

$$a = \sum_{j=0}^{h-1} m_j b_j q^j,\ m_j \in M, b_j \in B. \tag{29}$$

***Proof*** We first show the following lemma.

**Lemma 4.2** *Arbitrary integer $t \in [-2^{\ell-1}, q+2^\ell-1]$ can be expressed (not uniquely) as*

$$t = mb - \alpha q, m \in M, b \in B, -2^{\ell-1} \leq \alpha \leq 2^{\ell-1} - 1. \tag{30}$$

- If $t \in [-2^{\ell-1}, 2^\ell]$, then $t = 1 \cdot |t|$ or $t = -1 \cdot |t|$, thus $m = \pm 1 \in M, b = |t| \in B, \alpha = 0$.
- If $t \in [2^{\ell-1} + 1, q - 1]$, notice that $2^{(q-1)/2} = -1 \bmod q$ because 2 is a primitive element in $\mathbb{F}_q$, then

$$\begin{aligned}\{t \mid 1 \leq t \leq q-1\} &= \{2^i \bmod q \mid 0 \leq i \leq q-2\} \\ &= \left\{2^i \bmod q \mid 0 \leq i \leq (q-1)/2\right\} \\ &\quad \cup \left\{-2^i \bmod q \mid 1 \leq i \leq (q-3)/2\right\}.\end{aligned}$$

  If $t \in \{2^i \bmod q \mid 0 \leq i \leq (q-1)/2\}$, then

$$\begin{aligned}t &= 2^{i\ell+j} \bmod q \ \ (0 \leq i \leq \lfloor (q-1)/(2\ell) \rfloor, 0 \leq j \leq \ell - 1) \\ &= 2^j \cdot (2^{i\ell} \bmod q) \bmod q \\ &= mb - \alpha q, \text{ where } 0 \leq \alpha \leq 2^{\ell-1} - 1.\end{aligned}$$

  If $t \in \{-2^i \bmod q \mid 1 \leq i \leq (q-3)/2\}$, then $q - t \in \{2^i \bmod q \mid 0 \leq i \leq (q-1)/2\}$, so

$$q - t = mb - \alpha q, \text{ where } 0 \le \alpha \le 2^{\ell-1} - 1.$$

One can obtain that

$$t = q - (mb - \alpha q) = (-m)b - (-1 - \alpha)q = m'b - \alpha' q,$$

where $m' = -m \in M, b \in B$ and $\alpha' = -1 - \alpha \in [-2^{\ell-1}, -1]$.

- If $t \in [q, q + 2^{\ell} - 1]$, then $t = mb - \alpha q$, where $m = 1, b = t - q \in B, \alpha = -1$.

Back to the claim, one can convert scalar $a$ from its standard $q$-ary representation defined in Equation (20) to the radix $q$ representation defined in Equation (29) by first decomposing $a_i = m_i b_i - \alpha_i q$, then updating $a_{i+1}$ to be $-\alpha_i + a_{i+1}$ for $0 \le i \le h-2$. Using Lemma 4.2, one can check that

i) $a_0 \in [0, q-1]$,
ii) $-\alpha_j + a_{j+1} \in [1 - 2^{\ell-1}, q + 2^{\ell-1} - 1]$ for all $0 \le j \le h-3$,
iii) $-\alpha_{h-2} + a_{h-1} \in [1 - 2^{\ell-1}, 2^{\ell} - 1]$.

Those facts ensure the correctness of the conversion. □

Similar to the first construction, for every $t \in [-2^{\ell}, q + 2^{\ell} - 1]$, a hash table $H$ can be precomputed to store its decomposition, i.e., $H(t) = (m, b, \alpha)$ such that $t = mb - \alpha q, m \in M, b \in B, -2^{\ell-1} \le \alpha \le 2^{\ell-1} - 1$. In order to obtain the standard $q$-ary expression efficiently, $q$ can be selected from pseudo-Mersenne primes, i.e., $q = 2^c - e$, where $e$ is a small integer. The algorithms presented in [**?**, **?**] can convert scalar $a$ from its binary form to its standard $q$-ary expression using only additions, subtractions and bitwise operations.

For a small positive integer $\ell$, the size of $B$ is no more than $2 + 2^{\ell} + \lfloor (q-1)/(2\ell) \rfloor$. Denote maximum difference between two neighbor elements in sorted $B$ as $d$. By the computational cost estimation formula presented in Equation (15), the cost of computing $S_{n,r}$ over fixed points will be at most

$$nh + 2^{\ell} + \left\lfloor \frac{q-1}{2\ell} \right\rfloor + d - 2, \text{ where } h = \lceil \log_q r \rceil. \tag{31}$$

**Theorem 4.3** *Suppose $q$ is a prime such that 2 is the primitive element in the finite field $\mathbb{F}_q$, $\ell$ is a small positive integer such that $2^{\ell} < q$ and $q^{h-1} < r \le 2^{\ell-1} q^{h-1}$. The multiplier set and bucket set defined in equations (26) (27) permit an algorithm to compute $S_{n,r}$ over fixed points using at most*

$$nh + 2^{\ell} + \left\lfloor \frac{q-1}{2\ell} \right\rfloor + d - 2,$$

*additions, with the help of $\ell nh$ precomputed points*

$$\left\{ mq^j P_i \mid 1 \le i \le n, 0 \le j \le h-1, m \in \{2^i \mid 0 \le i \le \ell - 1\} \right\}.$$

The precomputed points associated with negative elements in $M$ are excluded. In practice, the integer $\ell$ can be firstly determined according to the available memory,

then radix $q$ is selected to minimize the computational cost. When $\ell$ is small enough, so is $d$, the computational cost is at most approximately

$$nh + \frac{q}{2\ell}.$$

**Acknowledgements.**

We would like to thank Chenkai Weng for providing us useful advice regarding the second construction.

## Appendix

We instantiate our constructions over elliptic curve group whose order $r = 2^{256}$, and present some comparisons against Pippenger's bucket method and its variant. The similar group size is typically used to provide around 128-bit security. For instance, the subgroup size of the popular curve BLS12-381 in which a pairing-based zkSNARK works is 255-bit.

For our constructions, we try to select radix $q$ that minimizes the number of additions when computing $S_{n,r}$ over fixed points. In order to permit fast scalar conversion from the binary form to the radix $q$ representation, the radix $q$ is chosen to be a power of 2 or a pseudo-Mersenne prime. We set $\ell = 6$ to demonstrate the effectiveness of our second construction, one can adjust $\ell$ according to the available memory size.

The choice of $q$ is summarized in Table 2, and the number of additions taken to compute $S_{n,r}$ is summarized in Table 3.

**Table 2** Radix $q$ employed by different methods to compute $S_{n,r}$

| $n$ | Pippenger | Pippenger variant | Construction 1 | Construction 2 |
|---|---|---|---|---|
| $2^{16}$ | $2^{13}$ | $2^{17}$ | $2^{18}$ | $2^{18} - 5$ |
| $2^{17}$ | $2^{14}$ | $2^{18}$ | $2^{19}$ | $2^{21} - 19$ |
| $2^{18}$ | $2^{15}$ | $2^{19}$ | $2^{20}$ | $2^{21} - 19$ |
| $2^{19}$ | $2^{17}$ | $2^{20}$ | $2^{20}$ | $2^{21} - 21$ |
| $2^{20}$ | $2^{18}$ | $2^{20}$ | $2^{20}$ | $2^{23} - 21$ |

## References

1. bellman: a crate for building zk-snark circuits. https://github.com/zkcrypto/bellman
2. gnark zk-snark library. https://github.com/ConsenSys/gnark

**Table 3** Number of additions taken to compute $S_{n,r}$

| $n$ | Pippenger | Pippenger variant | Construction 1 | Construction 2 |
|---|---|---|---|---|
| $2^{16}$ | $1.39 \times 10^6$ | $1.11 \times 10^6$ | $1.07 \times 10^6$ | $1.00 \times 10^6$ |
| $2^{17}$ | $2.65 \times 10^6$ | $2.10 \times 10^6$ | $2.01 \times 10^6$ | $1.88 \times 10^6$ |
| $2^{18}$ | $5.01 \times 10^6$ | $3.93 \times 10^6$ | $3.76 \times 10^6$ | $3.58 \times 10^6$ |
| $2^{19}$ | $9.44 \times 10^6$ | $7.34 \times 10^6$ | $7.17 \times 10^6$ | $6.99 \times 10^6$ |
| $2^{20}$ | $1.77 \times 10^7$ | $1.42 \times 10^7$ | $1.40 \times 10^7$ | $1.33 \times 10^7$ |


3. Zcash: Privacy-protecting digital currency. https://z.cash/
4. Bernstein, D.J.: Differential addition chains. URL: http://cr. yp. to/ecdh/diffchain-20060219. pdf (2006)
5. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.J.: Faster batch forgery identification. In: International Conference on Cryptology in India, pp. 454–473. Springer (2012)
6. Bos, J., Coster, M.: Addition chain heuristics. In: Conference on the Theory and Application of Cryptology, pp. 400–407. Springer (1989)
7. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation. In: Workshop on the Theory and Application of of Cryptographic Techniques, pp. 200–207. Springer (1992)
8. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation: algorithms and lower bounds. preprint, Mar **27** (1995)
9. Brown, D.R.: Multi-dimensional montgomery ladders for elliptic curves (2015). US Patent 8,958,551
10. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 229–243 (2017)
11. Doche, C., Kohel, D.R., Sica, F.: Double-base number system for multi-scalar multiplications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 502–517. Springer (2009)
12. Gabizon, A., J. Williamson, Z.: Proposal: The turbo-plonk program syntax for specifying snark programs (2020)
13. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. Journal of cryptology **24**(3), 446–469 (2011)
14. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Annual International Cryptology Conference, pp. 190–200. Springer (2001)
15. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques, pp. 305–326. Springer (2016)
16. Knuth, D.E.: The Art of Programming, vol. 2 (3rd ed.), Seminumerical algorithms. Addison Wesley Longman (1997)
17. Luo, G., Fu, S., Gong, G.: Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. In: Accepted paper in IACR Transactions on Cryptographic Hardware and Embedded Systems (2022)
18. Matula, D.W.: Basic digit sets for radix representation. Journal of the ACM (JACM) **29**(4), 1131–1143 (1982)
19. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)
20. Montgomery, P.L.: Evaluating recurrences of form xm+ n= f (xm, xn, xm- n) via lucas chains, 1983. Available at ftp. cwi. nl:/pub/pmontgom/Lucas. ps. gz (1992)
21. Pippenger, N.: On the evaluation of powers and related problems. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), pp. 258–263. IEEE Computer Society (1976)

22. Rao, S.R.S.: A note on schoenmakers algorithm for multi exponentiation. In: 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), vol. 4, pp. 384–391. IEEE (2015)
23. Suppakitpaisarn, V., Imai, H., Masato, E.: Fastest multi-scalar multiplication based on optimal double-base chains. In: World Congress on Internet Security (WorldCIS-2012), pp. 93–98. IEEE (2012)
24. Yu, W., Wang, K., Li, B., Tian, S.: Joint triple-base number system for multi-scalar multiplication. In: International Conference on Information Security Practice and Experience, pp. 160–173. Springer (2013)