# Fast Computation of Multi-Scalar Multiplication for Pairing-Based zkSNARK Applications

Guiwen Luo, Guang Gong
*Department of Electrical and Computer Engineering*
*University of Waterloo*
Waterloo, ON, Canada
{guiwen.luo, ggong}@uwaterloo.ca

2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 979-8-3503-1019-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICBC56567.2023.10174952

***Abstract*—The operation of computing $n$ scalar multiplications in an elliptic curve group and then adding them together is called $n$-scalar multiplication. $n$-scalar multiplication is the essential operation for proof generation and verification in pairing-based trusted setup zero-knowledge succinct non-interactive argument of knowledge protocols, which enable the privacy-preserving features in blockchain applications. This paper proposed a method to compute $n$-scalar multiplication taking advantage of $3n$ precomputed points. When instantiating over BLS12-381 curve, for $n = 2^c$ $(10 \leq c \leq 22)$, which covers the majority of our purported applications, the proposed method showed $2.59\% \sim 12.26\%$ theoretical speed improvement and demonstrated $1.63\% \sim 11.54\%$ experimental improvement against Pippenger's bucket method.**

***Index Terms*—Multi-scalar multiplication, zkSNARK, Privacy-preserving blockchain**

## I. Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) have been a hot topic lately because they can be used to build privacy-preserving blockchains. Known for their successful deployment in Zcash [1], zkSNARKs provide a convenient tool to construct blockchains supporting shielded transaction, in the sense that the sender, the receiver, and the coin being transacted remain confidential when verifying the validity of the transaction.

zkSNARKs enable one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing other information apart from the fact that the statement is true. Among the many zkSNARK schemes, there is a popular category of them, which are trusted setup zkSNARKs built upon elliptic curve bilinear pairings [2]–[6]. The representative in this category is Groth16 [4].

These pairing-based trusted setup zkSNARKs usually follow a common structure. There is a public string composed of an array of fixed points in an elliptic curve group called common reference string (CRS), which is generated in a trusted manner and accessible to all parties. The prover generates the zkSNARK proof by computing multi-scalar multiplications over the points in CRS. The verifier verifies the proof by computing multi-scalar multiplications and elliptic curve bilinear pairings.

The research is supported by NSERC SPG grant.

Multi-scalar multiplication, also called $n$-scalar multiplication if the number of points involved is clear, refers to the operation of computing multiple scalar multiplications in an elliptic curve group and then adding them together. For pairing-based trusted setup zkSNARKs, the cryptographic pairings are evaluated within several milliseconds, and the computation time of multi-scalar multiplications is usually measured in seconds. The overwhelming majority of time for proof generation and verification is thus spent on computing multi-scalar multiplications over the fixed points built in CRS. The number of points is very large. For examples, given an image of SHA256, if one wants to prove the knowledge of a 512-bit preimage in a zero-knowledge manner, the number of points in the corresponding multi-scalar multiplication is tens of thousands [7]. In order to hide the identity of a coin, Zcash utilizes Groth16 to prove the membership of the coin in a 64-layer Merkle tree built upon SHA256, and the number of points in the corresponding multi-scalar multiplication is more than a million [8].

Multi-scalar multiplication in pairing-based trusted setup zkSNARKs has two characteristics. First, all the points are fixed once CRS is generated. Second, the number of points is gigantic. In this paper, we target at $n = 2^c$ $(10 \leq c \leq 22)$, which covers the majority of applications. By now, all of the popular pairing-based zkSNARK implementations, such as Zcash, TurboPLONK and gnark, have adopted Pippenger's bucket method to compute multi-scalar multiplication.

Published in 1976 [9], Pippenger's bucket method has recently become an essential tool for privacy-preserving blockchains due to its effectiveness for computing multi-scalar multiplication in pairing-based zkSNARKs, elliptic-curve-based cryptographic commitments [10], BLS signature aggregations, and so on. A variant of Pippenger's bucket method that utilizes precomputation to speed up single scalar multiplication was introduced by Brickell *et al.* [11]. Their method can be easily applied to $n$-scalar multiplication with a large $n$. In order to achieve fast batch forgery signature verification, Bernstein *et al.* exploited Pippenger's bucket method to perform their essential computation, marking the start of extensive utilization of Pippenger's bucket method for computing multi-scalar multiplication. Luo *et al.* summarized a framework for computing multi-scalar multiplication [12], and proposed a method that improves the original Pippenger's

bucket method by $17\% \sim 40\%$ in terms of time complexity when instantiating it over BLS12-381 curve [13], with the help of a large precomputation table.

In light of the importance of multi-scalar multiplication in the process of proof generation and verification for pairing-based trusted setup zkSNARKs, a new method that utilizes precomputation to speed up multi-scalar multiplication over fixed points is proposed and validated by experiment in this paper.

*Our contributions.* Let us first introduce the notations. We denote $S_{n,r}$ as the following $n$-scalar multiplication over fixed points,

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

where $a_i's$ are integers, $0 \leq a_i < r$, and $P_i$'s are fixed points in an elliptic curve group. The integer $r$ is typically 256-bit in order to achieve 128-bit security. Given a radix $q = 2^c$, a scalar can be expressed in its radix $q$ expression. For efficiency reason, this paper only considers radix $q = 2^c$. Let $h = \lceil \log_q r \rceil$ be the length of the radix $q$ expression, and $r_{h-1} = \lfloor r/q^{h-1} \rfloor$ the maximum leading term in the standard $q$-ary expression. *Addition*, denoted as $A_E$, refers to the point addition in an elliptic curve group.

We have the following two contributions in this paper.

- By combining the ideas from both Pippenger's bucket method and LFG's method [12], we proposed a method that utilizes $1/h$ of the precomputation size compared to LFG's method, while gaining quite good speed improvement compared to Pippenger's bucket method. For most of the radixes, the proposed method computes $S_{n,r}$ using $h(n + 0.21q)$ additions, taking advantage of $3n$ precomputed points. The comparison among different methods is summarized in Table I, where $P$ represents a point in an elliptic curve group.
- For the computation of $S_{n,r}$ with $n = 2^c$ $(10 \leq c \leq 22)$, $r$ being the 255-bit subgroup order in BLS12-381 curve groups [13], the proposed method has $2.59\% \sim 12.26\%$ theoretical improvement against Pippenger's bucket method. We conducted the experiment, which showed $1.63\% \sim 11.54\%$ improvement, validating the effectiveness of the proposed method.

TABLE I
COMPARISON OF DIFFERENT METHODS THAT COMPUTES $S_{n,r}$

| Method | Memory | Worst-case time complexity |
|---|---|---|
| Pippenger's method [9], [14] | $n \cdot P$ | $h(n + 0.5q) \cdot A_E$ |
| LFG's method [12] | $3nh \cdot P$ | $(nh + 0.21q) \cdot A_E$ |
| Our method | $3n \cdot P$ | $h(n + 0.21q) \cdot A_E$ |

*Paper organization.* The remaining paper is organized as follows. Section II reviews Pippenger's bucket method and LFG's method. Section III presents the proposed method. Section IV instantiates the proposed method and Pippenger's bucket method over BLS12-381 curve, and tests their performance. Section V concludes the paper.

## II. REVIEW OF BACKGROUND KNOWLEDGE

### A. Pippenger's bucket method

Pippenger's bucket method chooses a radix $q = 2^c$, and expresses every scalar in its standard $q$-ary representation,

$$a_i = \sum_{i=0}^{h-1} a_{ij} q^j, \ 0 \leq a_{ij} < q. \tag{2}$$

It follows that

$$\begin{aligned} S_{n,r} &= \sum_{i=1}^{n} a_i P_i = \sum_{i=1}^{n} \sum_{j=0}^{h-1} a_{ij} q^j P_i \\ &= \sum_{j=0}^{h-1} q^j \cdot \sum_{i=1}^{n} a_{ij} P_i =: \sum_{j=0}^{h-1} q^j S_j. \end{aligned}$$

$S_j := \sum_{i=1}^{n} a_{ij} P_i$ is a special $n$-scalar multiplication whose every scalar is smaller than $q$. It can be computed by the following equation,

$$\begin{aligned} S_j &= \sum_{i=1}^{n} a_{ij} P_i = \sum_{k=1}^{q-1} k \cdot \sum_{i \text{ s.t. } a_{ij}=k} P_i \\ &= \sum_{\ell=1}^{q-1} \sum_{k=\ell}^{q-1} \sum_{i \text{ s.t. } a_{ij}=k} P_i. \end{aligned} \tag{3}$$

Equation (3) requires at most $n + q - 3$ additions. Once all intermediate $S_j$'s are computed, $S_{n,r}$ can be computed by a method similar to Horner's rule,

$$\begin{aligned} S_{n,r} &= \sum_{i=0}^{h-1} q^j S_j \\ &= S_0 + q(S_1 + \cdots + q(S_{h-2} + qS_{h-1}) \cdots ). \end{aligned} \tag{4}$$

$qS_j$ is computed by $c = \log_2 q$ additions (doubling is treated as addition for simplicity) given that $q = 2^c$. Equation (4) executes $(h-1)(c+1)$ additions. The time complexity of computing $S_{n,r}$ would be at most

$$h(n + q - 3) + (h - 1)(c + 1) \approx h(n + q). \tag{5}$$

If one notices that $-P$ can be easily obtained in an elliptic curve group by negating $P$'s $y$-coordinate, all $a_{ij}$'s in Equation (2) can be restricted to be $|a_{ij}| \leq q/2$. The time complexity of computing $S_{n,r}$ could be further reduced to approximately

$$h(n + 0.5q). \tag{6}$$

### B. LFG's method

On the basis of Pippenger's bucket method, Luo *et al.* recently proposed a new method to compute $S_{n,r}$ taking advantage of a large precomputation table [12].

Let radix $q = 2^c$, $M$ an integer set, $B$ a non-negative integer set with $0 \in B$, such that all scalar $a_i$ $(0 \leq a_i < r)$ can be expressed as

$$a_i = \sum_{j=0}^{h-1} b_{ij} m_{ij} q^j,\ b_{ij} \in B, m_{ij} \in M. \tag{7}$$

$M$ is called a multiplier set, $B$ a bucket set. We then have

$$\begin{aligned} S_{n,r} &= \sum_{i=1}^{n} a_i P_i = \sum_{i=1}^{n} \sum_{j=0}^{h-1} b_{ij} m_{ij} q^j P_i \\ &= \sum_{i=1}^{n} \sum_{j=0}^{h-1} b_{ij} \cdot m_{ij} q^j P_i. \end{aligned}$$

Denote $P_{ij} := m_{ij} q^j P_i$, suppose the following $nh|M|$ points

$$\{mq^j P_i \mid 1 \leq i \leq n, 0 \leq j \leq h-1, m \in M\} \tag{8}$$

are precomputed, then $S_{n,r}$ is equivalent to an $nh$-scalar multiplication whose all scalars are from set $B$. It follows that

$$S_{n,r} = \sum_{i=1}^{n} \sum_{j=0}^{h-1} b_{ij} \cdot P_{ij} = \sum_{b \in B} b \cdot \Big( \sum_{i,j \text{ s.t. } b_{ij}=b} P_{ij} \Big) =: \sum_{b \in B} b S_b,$$

where $S_b := \sum_{i,j \text{ s.t. } b_{ij}=b} P_{ij}$. All $S_b$'s can be computed with $nh - (|B| - 1)$ additions, because there are $nh$ points been sorted into $(|B| - 1)$ $S_b$'s according to their non-zero scalars.

Suppose $B = \{0, b_1, b_2, ..., b_m\}$ and $1 \leq b_1 \leq b_2 \leq \cdots \leq b_m$, define $\delta_j = b_j - b_{j-1}$, $d = \max_{1 \leq j \leq m} \{\delta_j\}$, then we have

$$\begin{aligned} S_{n,r} &= \sum_{b \in B} b S_b = \sum_{i=1}^{m} b_i S_{b_i} = \sum_{i=1}^{m} (\sum_{j=1}^{i} \delta_j) S_{b_i} \\ &= \sum_{j=1}^{m} \delta_j (\sum_{i=j}^{m} S_{b_i}) = \sum_{k=1}^{d} k \sum_{j \text{ s.t. } \delta_j = k} (\sum_{i=j}^{m} S_{b_i}) \\ &= \sum_{\ell=1}^{d} \sum_{k=\ell}^{d} \Big( \sum_{j \text{ s.t. } \delta_j = k} (\sum_{i=j}^{m} S_{b_i}) \Big). \end{aligned}$$

The logic is similar to Equation (3), but with one extra layer of sum operation to deal with the possible discontinuity of the elements in bucket set $B$. It is showed in [12] that the time complexity of this process is at most $2(|B| - 1) + d - 3$.

Combining those two parts together, the time complexity of computing $S_{n,r}$ would be at most

$$nh - (|B| - 1) + 2(|B| - 1) + d - 3 \approx nh + |B| + d, \tag{9}$$

using the help of

$$nh|M| \tag{10}$$

precomputed points defined in Equation (8).

LFG's method instantiates multiplier set $M$ as

$$M = \{-3, -2, -1, 1, 2, 3\}, \tag{11}$$

and bucket set $B$ is constructed by Algorithm 1, where $\omega_2(b)$ is the exponent of factor 2 in $b$, $\omega_3(b)$ is the exponent of factor 3 in $b$. For example, $b = 2^e k,\ 2 \nmid k$, then $\omega_2(b) = e$.

**Algorithm 1** Construction of bucket set $B$

**Input:** Radix $q = 2^c$.
**Output:** $B$.

```
B_0 = {0} ∪ {b | 1 ≤ b ≤ q/2, s.t. ω_2(b) + ω_3(b) ≡ 0 mod 2}
B_2 = {0} ∪ {b | 1 ≤ b ≤ r_{h-1} + 1, s.t. ω_2(b) + ω_3(b) ≡ 0 mod 2}
B_1 = B_0
for i = q/4 to q/2 − 1 by 1 do
    if i is in B_0 and q − 2 · i is in B_0 then
        B_1.remove(q − 2 · i)
for i = ⌊q/6⌋ to q/4 − 1 by 1 do
    if i is in B_0 and q − 3 · i is in B_0 then
        B_1.remove(q − 3 · i)
return B_1 ∪ B_2
```

It is proved that $M$ and $B$ defined above can be utilized to express every scalar as the representation defined in Equation (7). The biggest difference between two neighbor elements in $B$ is 6, i.e., $d \leq 6$. Bucket set size $|B| \approx 0.21q$ if $a_{h-1}/q$ is small, which is true in practice in most of the time. By Equations (9)(10), $M$ and $B$ defined above would yield a method of computing $S_{n,r}$ with time complexity

$$hn + 0.21q, \tag{12}$$

using the following $3nh$ precomputed points

$$\left\{mq^j P_i \mid 1 \leq i \leq n, 0 \leq j \leq h-1, m \in \{1, 2, 3\}\right\}. \tag{13}$$

Point $-mq^j P_i$ is omitted in the precomputed set because it can be easily obtained on the fly by negating the $y$-coordinate of $mq^j P_i,\ m \in \{1, 2, 3\}$.

## III. The Proposed Method

LFG's method shows an impressive performance, but one downside is that the precomputation size is too large. We thus propose a variant of LFG's method. The proposed method gains considerable speed improvement against Pippenger's bucket method, while using only $1/h$ of the precomputation size compared to LFG's method.

Let radix $q = 2^c$, $M$ an integer set, $B$ a non-negative integer set with $0 \in B$, such that all scalar $a_i$ $(0 \leq a_i < r)$ can be expressed as

$$a_i = \sum_{j=0}^{h-1} b_{ij} m_{ij} q^j,\ b_{ij} \in B, m_{ij} \in M. \tag{14}$$

$n$-scalar multiplication $S_{n,r}$ can be computed as follows,

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

Denote $P_{ij} := m_{ij} P_i$, suppose all the following $|M|n$ points

$$\{mP_i \mid 1 \leq i \leq n, m \in M\} \tag{15}$$

are precomputed, it follows that

$$S_j := \sum_{i=1}^{n} b_{ij} \cdot m_{ij} P_i = \sum_{i=1}^{n} b_{ij} \cdot P_{ij}$$

is an $n$-scalar multiplication whose all scalars are from $B$. By Equation (9) of LFG's method, every $S_j$ can be computed with $n + |B| + d$ additions, and the remaining part is

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

which can be computed using $(h-1)(c+1)$ additions by the same method introduced in Equation (4).

To sum up, the proposed method computes $S_{n,r}$ using at most

$$h(n + |B| + d) + (h-1)(c+1) \approx h(n + |B| + d) \quad (16)$$

additions. If instantiating our method with multiplier set $M$ and bucket set $B$ defined in Equation (11) and Algorithm 1 respectively, the proposed method will induce an algorithm of computing $S_{n,r}$ with worst-case time complexity

$$h(n + 0.21q) \quad (17)$$

for most of the $q$'s, with the help of the following $3n$ precomputed points

$$\{mP_i \mid 1 \le i \le n, m \in \{1, 2, 3\}\}. \quad (18)$$

## IV. Instantiation and experiment

We instantiate the proposed method over BLS12-381 curve, which has subgroups $\mathbb{G}_1$, $\mathbb{G}_2$ with a 255-bit prime order $r$. By running Algorithm 1, we can obtain Table II that lists all bucket sets used in our experiment. The bucket sets corresponding to some radixes are omitted because their performance is worse than some other bucket sets in Table II.

TABLE II
Bucket sets over BLS12-381 curve

| $q$ | $h$ | $r_{h-1}$ | $\lvert B\rvert$ | $d$ | $\lvert B\rvert/q$ |
|---|---|---|---|---|---|
| $2^{10}$ | 26 | 28 | 218 | 6 | 0.213 |
| $2^{11}$ | 24 | 3 | 427 | 6 | 0.208 |
| $2^{12}$ | 22 | 7 | 857 | 6 | 0.209 |
| $2^{13}$ | 20 | 231 | 1725 | 6 | 0.211 |
| $2^{14}$ | 19 | 7 | 3417 | 6 | 0.209 |
| $2^{16}$ | 16 | 29677 | 18343 | 6 | 0.280 |
| $2^{18}$ | 15 | 7 | 54618 | 6 | 0.208 |
| $2^{19}$ | 14 | 231 | 109244 | 6 | 0.208 |
| $2^{20}$ | 13 | 29677 | 220931 | 6 | 0.211 |

For a fixed $n$, there exists an optimal radix that minimizes Equation (6) and Equation (17) respectively. Table III shows the optimal radix $q$, its corresponding $h$, the number of additions taken to compute $S_{n,r}$, and the theoretical speed improvement.

We implemented the proposed method based on `blst`, a BLS12-381 signature library that contains the addition operation over $\mathbb{G}_1$ and $\mathbb{G}_2$, as well as the implementation of Pippenger's bucket method. The experiment is conducted on an Apple MacBook Pro with 3.2 GHz M1 Pro chip, 16 Gigabytes memory. The experiment runs single thread. The result is presented in Table IV.

TABLE III
$q$, $h$, and number of additions utilized to compute $S_{n,r}$

| | Pippenger | | | Our method | | | Improvement |
|---|---|---|---|---|---|---|---|
| $n$ | $q$ | $h$ | Additions | $q$ | $h$ | Additions | |
| $2^{10}$ | $2^8$ | 32 | $3.69 \times 10^4$ | $2^{10}$ | 26 | $3.23 \times 10^4$ | 12.26% |
| $2^{11}$ | $2^{10}$ | 26 | $6.66 \times 10^4$ | $2^{10}$ | 26 | $5.90 \times 10^4$ | 11.41% |
| $2^{12}$ | $2^{10}$ | 26 | $1.20 \times 10^5$ | $2^{11}$ | 24 | $1.09 \times 10^5$ | 9.35% |
| $2^{13}$ | $2^{11}$ | 24 | $2.21 \times 10^5$ | $2^{13}$ | 20 | $1.98 \times 10^5$ | 10.31% |
| $2^{14}$ | $2^{12}$ | 22 | $4.06 \times 10^5$ | $2^{13}$ | 20 | $3.62 \times 10^5$ | 10.67% |
| $2^{15}$ | $2^{13}$ | 20 | $7.37 \times 10^5$ | $2^{14}$ | 19 | $6.88 \times 10^5$ | 6.74% |
| $2^{16}$ | $2^{13}$ | 20 | $1.39 \times 10^6$ | $2^{14}$ | 19 | $1.31 \times 10^6$ | 5.92% |
| $2^{17}$ | $2^{16}$ | 16 | $2.62 \times 10^6$ | $2^{16}$ | 16 | $2.39 \times 10^6$ | 8.80% |
| $2^{18}$ | $2^{16}$ | 16 | $4.72 \times 10^6$ | $2^{16}$ | 16 | $4.49 \times 10^6$ | 4.89% |
| $2^{19}$ | $2^{16}$ | 16 | $8.91 \times 10^6$ | $2^{16}$ | 16 | $8.68 \times 10^6$ | 2.59% |
| $2^{20}$ | $2^{16}$ | 16 | $1.73 \times 10^7$ | $2^{19}$ | 14 | $1.62 \times 10^7$ | 6.31% |
| $2^{21}$ | $2^{19}$ | 14 | $3.30 \times 10^7$ | $2^{20}$ | 13 | $3.01 \times 10^7$ | 8.76% |
| $2^{22}$ | $2^{20}$ | 13 | $6.13 \times 10^7$ | $2^{20}$ | 13 | $5.74 \times 10^7$ | 6.43% |

TABLE IV
Experimental result of computing $S_{n,r}$

| | Pippenger | | Our method | | Improvement | |
|---|---|---|---|---|---|---|
| $n$ | $\mathbb{G}_1$ | $\mathbb{G}_2$ | $\mathbb{G}_1$ | $\mathbb{G}_2$ | $\mathbb{G}_1$ | $\mathbb{G}_2$ |
| $2^{10}$ | 15.1 ms | 37.0 ms | 13.7 ms | 34.0 ms | 8.98% | 8.13% |
| $2^{11}$ | 27.1 ms | 66.6 ms | 24.1 ms | 58.9 ms | 11.02% | 11.54% |
| $2^{12}$ | 48.7 ms | 122 ms | 44.3 ms | 110 ms | 9.05% | 10.05% |
| $2^{13}$ | 89.4 ms | 220 ms | 83.9 ms | 209 ms | 6.13% | 5.35% |
| $2^{14}$ | 164 ms | 401 ms | 149 ms | 363 ms | 9.44% | 9.62% |
| $2^{15}$ | 302 ms | 738 ms | 282 ms | 686 ms | 6.67% | 7.06% |
| $2^{16}$ | 553 ms | 1.35 s | 527 ms | 1.28 s | 4.79% | 5.26% |
| $2^{17}$ | 1.06 s | 2.64 s | 1.02 s | 2.52 s | 3.89% | 4.27% |
| $2^{18}$ | 1.94 s | 4.91 s | 1.84 s | 4.46 s | 5.12% | 5.45% |
| $2^{19}$ | 3.56 s | 8.87 s | 3.50 s | 8.63 s | 1.63% | 2.72% |
| $2^{20}$ | 6.85 s | 16.7 s | 6.66 s | 16.3 s | 2.71% | 2.26% |
| $2^{21}$ | 13.2 s | 32.2 s | 12.5 s | 30.4 s | 5.31% | 5.51% |
| $2^{22}$ | 24.8 s | 60.8 s | 23.2 s | 56.4 s | 6.23% | 7.26% |

## V. Conclusion

In this paper, we have proposed a method for computing $n$-scalar multiplication that takes advantage of $3n$ precomputed points. The performance is analyzed theoretically and verified by experiment. We expect that our method will improve the performance of privacy-preserving blockchain applications built upon pairing-based trusted setup zkSNARKs.

## References


[1] "Zcash: Privacy-protecting digital currency." https://z.cash/.

[2] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without pcps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 626–645, Springer, 2013.

[3] G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss, "Square span programs with applications to succinct nizk arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 532–550, Springer, 2014.

[4] J. Groth, "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.
[5] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2111–2128, 2019.
[6] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 738–768, Springer, Cham, 2020.
[7] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "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.
[8] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*, pp. 459–474, IEEE, 2014.
[9] N. Pippenger, "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.
[10] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *International conference on the theory and application of cryptology and information security*, pp. 177–194, Springer, 2010.
[11] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, "Fast exponentiation with precomputation," in *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 200–207, Springer, 1992.
[12] G. Luo, S. Fu, and G. Gong, "Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 358–380, 2023.
[13] S. Bowe, "BLS12-381: New zk-snark elliptic curve construction," 2017.
[14] D. J. Bernstein, J. Doumen, T. Lange, and J.-J. Oosterwijk, "Faster batch forgery identification," in *International Conference on Cryptology in India*, pp. 454–473, Springer, 2012.