Secure MSM Outsourcing Computation for Zero-knowledge Proof Generation *

Wujie Xiong[1], Arefeh Rahaei[1], Sangwon Shin[2], Xinxin Fan[3], Taeweon Suh[2], Veronika Kuchta[4], Francesco Sica[4], Weidong Shi[5], and Lei Xu[1]

[1] Kent State University, Kent, OH, USA
{wxiong1,arahaei,lxu12}@kent.edu
[2] Korea University, Seoul, Korea
{husask11,suhtw}@korea.ac.kr
[3] IoTeX, Menlo Park, CA, USA
xinxin@iotex.io
[4] Florida Atlantic University, Boca Raton, FL, USA
{vkuchta,sicaf}@fau.edu
[5] University of Houston, Houston, TX, USA
wshi3@central.uh.edu

Abstract. Zero-knowledge proof (ZKP) schemes enable a prover to convince a verifier of the validity of a statement without revealing the underlying secret. These schemes have found extensive applications in secure communications, privacy-preserving transactions and blockchain technologies. However, the computational cost of proof generation remains a major obstacle to practical deployment. Although various acceleration techniques have been proposed, they often rely on specialized hardware that may not be locally available. A promising yet underexplored alternative is to offload computation to a more powerful third party, such as a cloud server, in a secure and efficient manner. Rather than outsourcing the entire proof generation process, selectively offloading the most computationally intensive operations offers greater flexibility and simplicity. In this work, we propose a secure outsourcing scheme for multi-scalar multiplication (MSM), which is the most computationally expensive operation in many widely used ZKP protocols. Our scheme enables users to delegate MSM computations to a server while preserving the confidentiality of the secret inputs (i.e., the scalars) and allowing verification of the server's output. Our performance analysis shows that the proposed scheme significantly reduces the computational burden on the user while imposing only minimal overhead on the server.

1 Introduction

Zero-knowledge proofs (ZKP) have emerged as a cornerstone of modern cryptography, enabling a prover to demonstrate the validity of a statement without

* This work was partially supported by the NATO SPS Project G7200. We also gratefully acknowledge the reviewers and Dr. Majid Khabbazian for their valuable comments. Taeweon Suh is the corresponding author.

disclosing the underlying secret witness. This unique property has fueled the growth of privacy-centric technologies, ranging from verifiable computation and secure communication to confidential blockchain transactions [22, 24]. However, the widespread adoption of ZKP is hindered by a significant "prover bottleneck": generating a proof remains several orders of magnitude more computationally expensive than verifying one.

In state-of-the-art ZK-SNARK protocols such as Groth16, SONIC, and PLONK, the primary computational burden lies in Multi-Scalar Multiplication (MSM). MSM involves computing linear combinations of points on an elliptic curve, a task that often consumes 60% to 80% of the total proving time [25]. While algorithmic optimizations and specialized hardware (such as GPUs, FPGAs, and ASICs) can mitigate these costs, they introduce practical barriers. High-performance hardware is often prohibitively expensive for individual users and remains largely inaccessible in resource-constrained environments like IoT devices. Furthermore, because ZKP applications are often executed sporadically, local hardware investment frequently suffers from low utilization and poor cost-efficiency. To bridge this gap, outsourcing the heavy lifting of MSM to a powerful third-party service provider offers a compelling solution. Offloading these computations allows for high hardware utilization and provides "proving-as-a-service" to thin clients. Nevertheless, outsourcing introduces two fundamental security tensions: **(i)** *Scalar privacy.* The service provider must perform the computation without learning the scalars, which are directly derived from the user's secret witness. **(ii)** *Verifiability.* The user must be able to verify that the provider returned the correct result without re-performing the expensive MSM locally.

Existing researches on "whole-proof" outsourcing often rely on complex cryptographic machinery like multi-party computation (MPC). These approaches, while robust, typically incur massive communication overhead and significant design complexity. In this paper, we propose a lightweight and modular protocol for the secure outsourcing of MSM. Our key technical insight is a **dual-blinding matrix construction** that independently transforms the public base points and secret scalars into a randomized algebraic structure. Specifically, the prover expands the base points into two point matrices, M_P and M_T, using a random blinding matrix and a secret binding state $(\alpha, \boldsymbol{\rho})$. Simultaneously, the secret scalars are masked within a matrix M_k using per-row and per-entry blinding factors. This mechanism embeds an algebraic binding layer that enables the prover to recover the result via a simple de-scaling step while providing a built-in consistency check to detect malicious or incorrect outputs from the provider. In summary, our contributions include:

- *Strong scalar privacy and soundness.* We provide a blinding mechanism where the provider's view is statistically independent of the original scalars. Our per-index binding ensures that any malicious manipulation by the provider is detected by the prover with high probability.
- *Immediate result verification.* Unlike full outsourcing schemes that verify at the end of the proof pipeline, our protocol allows the prover to verify

the MSM result via a single de-scaling step. This prevents "wasted" local computation on downstream proof-generation tasks.
- *Modular, drop-in design.* Our scheme is agnostic to the specific ZKP protocol or hardware backend. It can be integrated as a "plug-and-play" module into existing pipelines like Groth16 or PLONK, utilizing any existing MSM hardware accelerator (GPU/FPGA) without modification.
- *Practical performance.* Experimental results indicate that our protocol reduces the prover's local workload by over 92% compared to local execution, effectively enabling high-speed ZKP generation on low-power devices.

2 Background

ZKP and ZK-SNARKs. Zero-knowledge proof (ZKP) protocols allow a prover to convince a verifier of a statement's validity without revealing the underlying secret witness [7]. A standard ZKP system consists of `Setup`, `Prove`, and `Verify` algorithms, satisfying the properties of completeness, soundness, and zero-knowledge. ZK-SNARKs are a prominent class of ZKPs that are *succinct* (small proofs and fast verification) and *non-interactive*. However, ZK-SNARKs suffer from asymmetric computation: while verification is efficient, proof generation is intensive. Specifically, Multi-Scalar Multiplication (MSM) often accounts for 60% to 80% of the total proving time [25, 1].

MSM computation. Mathematically, MSM is the sum of multiple scalar multiplications: $S_n = \sum_{i=1}^{n} k_i P_i$, where $k_i \in \mathbb{F}$ are scalars and P_i are points on an elliptic curve. For large n, this is computationally demanding. The primary acceleration strategy is Pippenger's algorithm [19], which uses precomputation and bucket aggregation to reduce the number of group operations. Beyond algorithmic improvements, specialized hardware such as GPUs, FPGAs, and ASICs are frequently used to parallelize the workload [26, 20].

MSM in proof generation. In ZK-SNARK protocols like Groth16 [8], the circuit is converted into a Quadratic Arithmetic Program (QAP). The prover generates the proof by encoding the witness vector $\boldsymbol{w}$ and public inputs $\boldsymbol{x}$ into elliptic curve points. This process—specifically evaluating $\sum x_i Y_i(s)$ and $\sum w_i Y_i(s)$—is fundamentally an MSM instance where the witness elements serve as scalars. Consequently, MSM in ZK-SNARKs has two critical security properties: (i) the base points P_i are public and often fixed; and (ii) the scalars k_i contain secret witness information and must remain hidden during computation.

3 Secure MSM Outsourcing Computation

We first formalize the problem of secure MSM outsourcing computation and then present a protocol to solve the problem.

3.1 Problem Statement and Security Assumptions

Participants and Security Assumptions. Our protocol involves two primary entities: the *Prover* ($\mathcal{P}$) and the *Service Provider* ($\mathcal{S}$). $\mathcal{P}$ is the trusted owner of the ZKP witness and the end-user of the computation. $\mathcal{S}$ is a resource-rich but untrusted entity that may deviate from the protocol to leak secret scalars or save costs by returning incorrect or partial results [12]. The interaction follows a three-step workflow: **(i)** *Offloading:* $\mathcal{P}$ transforms the MSM input into a randomized format and transmits it to $\mathcal{S}$; **(ii)** *Computation:* $\mathcal{S}$ executes the heavy MSM operations on the blinded data and returns the intermediate results; and **(iii)** *Verification & Recovery:* $\mathcal{P}$ performs a lightweight check to ensure result integrity and de-randomizes the output to obtain the final MSM result.

Requirements of secure MSM outsource computation. There are two essential requirements for secure MSM outsourcing computation: **(i)** *Scalar hiding.* The scalars involved in the MSM computation (i.e., $k_i, i \in [1, n]$) are closely connected with the statement $\mathcal{P}$ wants to prove and reflect the secret information the prover wants to hide. Therefore, the scalars should not be disclosed to the untrusted $\mathcal{S}$ directly. **(ii)** *Efficient MSM result verification.* Given that the computation is conducted by a third party $\mathcal{S}$ that is not entirely trustworthy, it is imperative for $\mathcal{P}$ to verify the accuracy of the computation result (i.e., S_n) efficiently. This verification process must be less resource-intensive than performing the MSM instance computation locally. It is important to note that the protocol is not required to ensure the correctness of the MSM computation result, as $\mathcal{S}$ can always provide arbitrary results, refuse the service request and even return a partially honest result to circumvent the verification logic.

For MSM result verification, a straightforward idea is to take the result returned by $\mathcal{P}$ without verification and use it in the proof generation algorithm directly. $\mathcal{P}$ can then run the ZKP verification algorithm `Verify` on the generated proof. As $\mathcal{P}$ is honest based on our security assumption, the verification of the proof will fail if the MSM computation result is wrong, assuming that MSM is the only outsourced computation. Otherwise, the returned MSM computation result is correct. Compared with an efficient direct verification, this delayed verification, using the ZKP scheme, faces several limitations: **(i)** *High latency.* $\mathcal{P}$ can only learn whether the MSM computation result is correct after the entire proof is generated and verified, which leads to high latency for the detection. **(ii)** *Potential waste of computation.* MSM contributes a significant portion of the proof generation, but there are also operations such as polynomial-related tasks. **(iii)** *Identification of error.* With this straightforward verification, it is hard to identify the error when the proof is invalid. For instance, Groth16 [8] proof generation involves 3 MSMs, so it is hard to tell which of these 3 computation tasks is wrong if the `Verify` algorithm fails.

3.2 Overview of Secure MSM Outsourcing Computation Protocol

We summarize the notations that will be used in the protocol in Table 1 for convenience, and then describe each step of the secure MSM outsourcing computation protocol.

Table 1: Notations used in the secure MSM outsourcing computation protocol.

Notation	Description
$\mathcal{P}$	The prover of the ZKP scheme.
$\mathcal{S}$	The service provider helps $\mathcal{P}$ for the MSM computation.
n	The number of scalar multiplications in the MSM computation, which is a positive integer.
ord	The order of the elliptic curve group EC and all elliptic curve points belong to this group.
$\boldsymbol{k}$	The scalar vector $(k_1, k_2, \ldots, k_n)$ in the MSM computation, $k_i \in \mathbb{Z}_{ord}$. When it is clear from the context, we use the terms integer and scalar interchangeably.
$\boldsymbol{P}$	The elliptic curve point vector $\{P_1, P_2, \ldots, P_n\}$ of a given MSM task, $P_i \in EC$.
α	The secret binding scalar used for public elliptic curve point, $\alpha \in \mathbb{Z}^*_{\text{ord}}$.
Q	The public, fixed, non-identity auxiliary point derived via hash-to-curve on a domain-separated string, $Q \in \text{EC} \setminus \{\mathcal{O}\}$.
$\boldsymbol{\rho}$	The secret binding vector $(\rho_1, \ldots, \rho_n)$, $\rho_i \in \mathbb{Z}_{\text{ord}}$.
t_i	The random blinding scalar used for i-th row blinding, $t_i \in \mathbb{Z}^*_{ord}$.
S_n	The result of the MSM computation $\boldsymbol{k} \cdot \boldsymbol{P}$, which is also an elliptic curve point in EC.

We first give an overview of the secure MSM outsourcing computation protocol in Algorithm 1, which outlines how $\mathcal{P}$ and $\mathcal{S}$ interact. Before invoking the algorithm, $\mathcal{P}$ samples a secret binding state $(\alpha, \boldsymbol{\rho}, Q)$ with $\alpha \in \mathbb{Z}^*_{\text{ord}}$, $\boldsymbol{\rho} \in \mathbb{Z}^n_{\text{ord}}$, and $Q \in \text{EC}$. We then describe each step in detail in Section 3.3, including the notations used in Algorithm 1.

3.3 Detailed Secure MSM Outsourcing Computation Protocol

We describe the details of the functions given in Algorithm 1 in this section.

Data preparation. $\mathcal{P}$ prepares for the MSM outsourcing computation in two steps:

- *Step 1: Elliptic curve points preprocessing.* $\mathcal{P}$ first expands the elliptic curve point vector $\boldsymbol{P}$ and constructs a secret binding $\boldsymbol{T} = (T_1, \ldots, T_n)$, where each $T_i = \alpha P_i + \rho_i Q$, based on the secret binding state $(\alpha, \boldsymbol{\rho}, Q)$. The vectors $\boldsymbol{P}$ and $\boldsymbol{T}$ are then expanded into two distinct matrices of elliptic curve points. Specifically, the *prover* generates a matrix of random integers $R = \{r_{i,j}\} \in \mathbb{Z}^{\ell \times n}_{ord}$, where ℓ denotes the number of rows and is a constant (typically $\ell = 2$ or $\ell = 3$). And for each row of R, the *prover* computes a

Algorithm 1 High-level description of the secure MSM outsourcing computation protocol

Require: Elliptic curve points $\boldsymbol{P} = (P_1, P_2, \ldots, P_n)$ and scalars $\boldsymbol{k} = (k_1, k_2, \ldots, k_n)$, which are only available to $\mathcal{P}$.
Ensure: MSM result $S_n = k_1P_1 + k_2P_2 + \cdots + k_nP_n$.

▷ *Data preparation executed by $\mathcal{P}$ with input $\boldsymbol{P}, \boldsymbol{k}$ and binding parameters $(\alpha, \boldsymbol{\rho}, Q)$* ◁
$\boldsymbol{T} \leftarrow \texttt{PointBinding}(\boldsymbol{P}, \alpha, \boldsymbol{\rho}, Q)$
$(M_{\boldsymbol{P}}, R) \leftarrow \texttt{PointPreProcessing}(\boldsymbol{P})$
$(M_{\boldsymbol{T}}, R) \leftarrow \texttt{PointPreProcessing}(\boldsymbol{T})$ ▷ *These functions only need to be done once.*
$(M_{\boldsymbol{k}}, \boldsymbol{t}) \leftarrow \texttt{ScalarPreProcessing}(\boldsymbol{k}, R)$
return $(M_{\boldsymbol{P}}, M_{\boldsymbol{T}}, M_{\boldsymbol{k}})$ to $\mathcal{S}$

▷ *MSM computation executed by $\mathcal{S}$ with input $M_{\boldsymbol{P}}, M_{\boldsymbol{T}}$ and $M_{\boldsymbol{k}}$* ◁
$\boldsymbol{Q} \leftarrow M_{\boldsymbol{P}} \star M_{\boldsymbol{k}}$
$\boldsymbol{V} \leftarrow M_{\boldsymbol{T}} \star M_{\boldsymbol{k}}$ ▷ *Details of $\star$ are provided in the later part of the section.*
return $(\boldsymbol{Q}, \boldsymbol{V})$ to $\mathcal{P}$

▷ *Postprocessing executed by $\mathcal{P}$ with input $(\boldsymbol{Q}, \boldsymbol{V}), \boldsymbol{t}$* ◁
$b \leftarrow \texttt{Verify}(\boldsymbol{Q}, \boldsymbol{V}, \boldsymbol{t})$ ▷ *Note that $\boldsymbol{t}$ is selected by $\mathcal{P}$ in a previous step.*
if b is true **then**
  return any element of $\boldsymbol{t}^{-1}\boldsymbol{Q}$
else
  return Incorrect

Hadamard product with $\boldsymbol{P}$ and $\boldsymbol{T}$. We denote the resulting matrix as $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$, and

$$M_{\boldsymbol{P}} = \begin{pmatrix} r_{1,1}P_1 & r_{1,2}P_2 & \cdots & r_{1,n}P_n \\ r_{2,1}P_1 & r_{2,2}P_2 & \cdots & r_{2,n}P_n \\ \vdots & \vdots & \ddots & \vdots \\ r_{\ell,1}P_1 & r_{\ell,2}P_2 & \cdots & r_{\ell,n}P_n \end{pmatrix},$$

$$M_{\boldsymbol{T}} = \begin{pmatrix} r_{1,1}T_1 & r_{1,2}T_2 & \cdots & r_{1,n}T_n \\ r_{2,1}T_1 & r_{2,2}T_2 & \cdots & r_{2,n}T_n \\ \vdots & \vdots & \ddots & \vdots \\ r_{\ell,1}T_1 & r_{\ell,2}T_2 & \cdots & r_{\ell,n}T_n \end{pmatrix} = \begin{pmatrix} r_{1,1}(\alpha P_1 + \rho_1 Q) & \cdots & r_{1,n}(\alpha P_n + \rho_n Q) \\ r_{2,1}(\alpha P_1 + \rho_1 Q) & \cdots & r_{2,n}(\alpha P_n + \rho_n Q) \\ \vdots & \ddots & \vdots \\ r_{\ell,1}(\alpha P_1 + \rho_1 Q) & \cdots & r_{\ell,n}(\alpha P_n + \rho_n Q) \end{pmatrix}.$$

The construction of $M_{\mathbf{P}}$ and $M_{\mathbf{T}}$ is performed entirely within $\mathcal{P}$'s local environment, keeping both the random coefficients in R and the long-lived secret state $(\alpha, \boldsymbol{\rho})$ hidden from $\mathcal{S}$. Notably, $M_{\mathbf{P}}$ and $M_{\mathbf{T}}$ are computed and uploaded to $\mathcal{S}$ only once, after which they can be reused for multiple subsequent MSM instances.

– *Step 2: Scalar preprocessing.* Given an MSM instance, $\boldsymbol{k}$ is fixed and known to $\mathcal{P}$ only. $\mathcal{P}$ needs to preprocess the scalars in $\boldsymbol{k}$ to hide them from $\mathcal{S}$ while still allowing $\mathcal{S}$ to contribute to the computation of the MSM instance. To

achieve this feature (scalar hiding), $\mathcal{P}$ expands the scalar vector $\boldsymbol{k}$ to a matrix in the following form before sending it to $\mathcal{S}$:

$$M_{\boldsymbol{k}} = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{\ell,1} & k_{\ell,2} & \cdots & k_{\ell,n} \end{pmatrix}.$$

For the i-th row of $M_{\boldsymbol{k}}$, the *prover* generates a random integer $t_i \in \mathbb{Z}_{ord}$, and calculates

$$(k_{i,1}, k_{i,2}, \cdots, k_{i,n}) \leftarrow t_i(k_1/r_{i,1}, k_2/r_{i,2}, \ldots, k_n/r_{i,n}), \quad i = 1, \ldots, \ell. \tag{1}$$

In total, $\mathcal{P}$ needs to choose ℓ random numbers for the generation of the matrix $M_{\boldsymbol{k}}$.

After generating the matrices $M_{\boldsymbol{P}}$, $M_{\boldsymbol{T}}$ and $M_{\boldsymbol{k}}$, $\mathcal{P}$ sends all of them to $\mathcal{S}$. Note that the matrices $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$ only need to be sent to $\mathcal{S}$ once when $\mathcal{P}$ needs to compute multiple MSM instances for the fixed set of elliptic curve points $\boldsymbol{P}$.

MSM computation on the side of $\mathcal{S}$. $\mathcal{S}$ receives the matrices $M_{\boldsymbol{P}}$, $M_{\boldsymbol{T}}$ and $M_{\boldsymbol{k}}$ from $\mathcal{P}$, and computes $M_{\boldsymbol{P}} \star M_{\boldsymbol{k}}$ and $M_{\boldsymbol{T}} \star M_{\boldsymbol{k}}$ as given in Equation (2) and Equation (3).

$$M_{\boldsymbol{P}} \star M_{\boldsymbol{k}} = \begin{pmatrix} \sum_{j=1}^{n} k_{1,j}(r_{1,j}P_j) \\ \sum_{j=1}^{n} k_{2,j}(r_{2,j}P_j) \\ \vdots \\ \sum_{j=1}^{n} k_{\ell,j}(r_{\ell,j}P_j) \end{pmatrix} = (Q_1, Q_2, \ldots, Q_\ell)^T. \tag{2}$$

$$M_{\boldsymbol{T}} \star M_{\boldsymbol{k}} = \begin{pmatrix} \sum_{j=1}^{n} k_{1,j}(r_{1,j}T_j) \\ \sum_{j=1}^{n} k_{2,j}(r_{2,j}T_j) \\ \vdots \\ \sum_{j=1}^{n} k_{\ell,j}(r_{\ell,j}T_j) \end{pmatrix} = (V_1, V_2, \ldots, V_\ell)^T. \tag{3}$$

The operator $\star$ denotes a row-wise inner product between two matrices. Although the ℓ rows of the matrix $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$ are derived from the same set of elliptic curve points $\boldsymbol{P}$, $\mathcal{S}$ cannot infer any relationships among them. Consequently, each element in the vector $(Q_1, Q_2, \ldots, Q_\ell, V_1, V_2, \ldots, V_\ell)^T$ must be computed independently, each representing a distinct MSM instance.

$\mathcal{S}$ can use any existing MSM computation algorithm to compute the 2ℓ MSM instances, or different algorithms for different MSM instances. The only difference is that when a precomputation-based MSM algorithm (e.g., [5, 19]) is used, $\mathcal{S}$ needs to maintain an precomputation table. The computation result (i.e., 2ℓ elliptic curve points) is sent back to $\mathcal{P}$.

Postprocessing of returned result. $\mathcal{P}$ receives the result from $\mathcal{S}$, which is a vector of 2ℓ elliptic curve points, i.e., $(Q_1, Q_2, \ldots, Q_\ell, V_1, V_2, \ldots, V_\ell)^T$ defined in Equation (2) and Equation (3). To ensure the correctness and soundness of the computation, $\mathcal{P}$ performs the following verification steps:

1. **Consistency check.** Compute $A_i = t_i^{-1} Q_i$ for each $i \in [1, \ell]$ and check $A_1 \stackrel{?}{=} A_2 \stackrel{?}{=} \cdots \stackrel{?}{=} A_\ell$. If not, reject.
2. **Per-index binding check.** For each row $i \in [1, \ell]$:
   (a) Compute the field inner product:

$$s_i = t_i \cdot \langle \mathbf{k}, \boldsymbol{\rho} \rangle = t_i \sum_{j=1}^{n} k_j \rho_j \in \mathbb{Z}_{\text{ord}}. \tag{4}$$

   (b) Check the binding equation, if any binding check fails, reject.

$$V_i \stackrel{?}{=} \alpha Q_i + s_i Q. \tag{5}$$

3. If both checks pass, output A_1 as the MSM result S_n.

4 Analysis of the MSM Outsource Computing Protocol

4.1 Structural Necessity Analysis

The two matrices $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$ are essential for the security of Algorithm 1. Intuitively, $M_{\boldsymbol{P}}$ serves as the payload to obtain the MSM result and $M_{\boldsymbol{T}}$ is the integrity tag for the MSM result. Furthermore, $M_{\boldsymbol{P}}$ can also serve as a pre-filter of incorrect outsourced computation.

4.2 Formal Security Analysis

In this section, we prove the correctness and analyze the security features of the MSM outsource computing protocol given in Section 3.

Correctness of the secure MSM outsourcing protocol. Correctness implies that the prover $\mathcal{P}$ can retrieve the correct MSM computation result if $\mathcal{S}$ follows the protocol. The formal proof of correctness is given in Appendix A.1.

Detection of incorrect MSM computation result (soundness). Based on the problem statement given in Section 3.1, there is no way to prevent a malicious $\mathcal{S}$ from sending incorrect computation results to $\mathcal{P}$. The best $\mathcal{P}$ can do is to verify whether $\mathcal{S}$ has sent an incorrect result. The soundness requirement ensures that a malicious $\mathcal{S}$ cannot cause $\mathcal{P}$ to accept an incorrect MSM result $S' \neq S_n$ with non-negligible probability. Specifically, the verification must be robust enough to detect partial-sum attacks, where a strategic server $\mathcal{S}$ returns an incomplete result $\sum_{j \in I} k_j P_j$ derived from only a subset I of the assigned indices. This verification must remain efficient for $\mathcal{P}$, requiring significantly less computational resource than performing the MSM locally. We demonstrate that the probability $\mathcal{P}$ accepts a wrong result is at most L/ord, where L is the number of reuse times, i.e., $\mathcal{P}$ asks $\mathcal{S}$ to calculate L MSM instances with the same setup (i.e., $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$). The formal definition of soundness and corresponding proofs are given in Appendix A.2 and Appendix A.3.

Hiding of original scalars (privacy). The privacy requirement is that the service provider $\mathcal{S}$ learns no information about the secret scalars $\boldsymbol{k} = (k_1, ..., k_n)$

from its view of the protocol. We model this using the simulation paradigm for secure computation. Since the protocol is essentially one-way from $\mathcal{P}$ to $\mathcal{S}$ for the input messages $(M_{\boldsymbol{P}}, M_{\boldsymbol{T}}, M_{\boldsymbol{k}})$, and $\mathcal{S}$'s response Q does not affect the privacy of k (i.e., Q is computed after $\mathcal{S}$ received the messages), we focus on simulating $\mathcal{S}$'s view $(M_{\boldsymbol{P}}, M_{\boldsymbol{T}}, M_{\boldsymbol{k}})$. The formal definition of privacy and corresponding proof are given in Appendix A.4. Note that in the security analysis, we do not consider the extra information a malicious $\mathcal{S}$ of concrete ZKP schemes, which may provide extra information to the attacker.

4.3 Side Channel Leakage Analysis

In practice, side channel attacks are a serious challenge for various cryptographic schemes. Under our security model, the service provider $\mathcal{S}$ is not trusted. Therefore, there is no side channel leakage on the side of $\mathcal{S}$. For the prover $\mathcal{P}$, there are two major operations that may suffer from side channel leakage: **(i)** *Elliptic curve points preprocessing.* This operation consists of group of scalar multiplications, and an adversary can utilize existing side channel attacks [10, 21] to obtain information regarding the matrix of random integers R. **(ii)** *Scalar preprocessing.* This operation consists of a group of modular multiplications, which also face different side channel attacks [16, 9] that can lead to leakage of involved operators. By combining information collected from these two operations, an adversary has a chance to recover the original scalars $\boldsymbol{k}$. Fortunately, existing mitigation mechanisms (e.g., [9]) can be easily adopted to mitigate such risk. Side channel resistant algorithms usually have higher computation cost. For instance, compared with standard Montgomery multiplication, the side channel resistant algorithm given in [9] introduced extra 8.5% overhead for 512 bits modulus. This only slightly increases the cost for $\mathcal{P}$, and the outsourced MSM solution can still save more than 90% of computation time compared with local computation.

5 Efficiency Analysis and Experiments

5.1 Theoretical Efficiency Analysis

In the protocol delineated in Section 3, certain operations are executed only once, such as the generation of the matrix of elliptic curve points $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$, transmission of $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$ to the third-party service provider $\mathcal{S}$, and generation of the MSM precomputation table associated with $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$ on the part of $\mathcal{S}$. This discussion primarily focuses on the recurring costs incurred by the user $\mathcal{P}$ and $\mathcal{S}$ for various MSM computation tasks. Table 2 provides a summary of the costs for $\mathcal{P}$ and $\mathcal{S}$ employing different methodologies.

To evaluate the improvement of our protocol and compare it with the w/o outsourcing computation, we conduct cost analysis and experiments on both $\mathcal{P}$ and $\mathcal{S}$ sides using the popular BLS12-381 curve with group order $ord =$ `0x73eda753299d7d483339d80809a1d80553bda402fffe5bfeffffffff00000001`, and the underlying field characteristic p is 381-bit. For a typical zkSNARK proof generation configuration, the value of n is typically between 2^{10} and 2^{21} [5].

Table 2: Comparison of the cost of MSM computation with different methods.

	w/o outsourcing	**w outsourcing**
Computation		
$\mathcal{P}$	1 MSM computation	$2 \times n \times \ell$ integer multiplications (preparation stage) ℓ modular inversions, 1 scalar dot product of size n, ℓ integer multiplications, $3 \times \ell$ scalar multiplications and ℓ point additions (verification stage)
$\mathcal{S}$	NA	$2 \times \ell$ MSM computation
Storage		
$\mathcal{P}$	n scalars, n points, and an MSM precomputation table	n scalars, $(n \times \ell + \ell + n + 1)$ integers
$\mathcal{S}$	NA	$2 \times n \times \ell$ points, $n \times \ell$ scalars, and an MSM precomputation table
Communication		
$\mathcal{P}$	None	$n \times \ell$ integers
$\mathcal{S}$	NA	$2 \times \ell$ points

[a] Scalars and integers in this table belong to $\mathbb{Z}_{ord}$.
[b] Points mean elliptic curve points.
[c] We only consider the amount of data sent by a party.

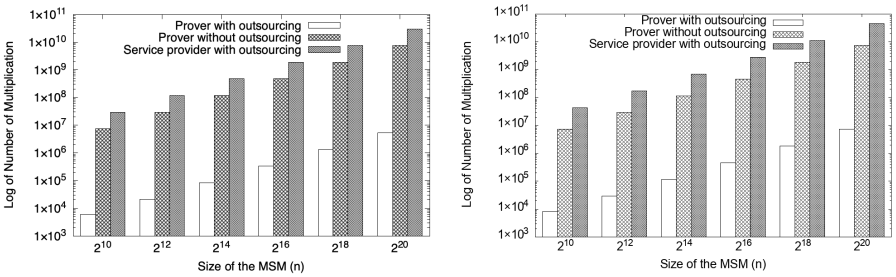
Computation cost. For each MSM computation task, $\mathcal{P}$ must finish two computation tasks: **(i)** *Preparation stage.* Generating the matrix of processed scalars $M_{\boldsymbol{k}}$ of size $\ell \times n$ requires $2n\ell$ integer multiplications. Note that $r_{i,j}^{-1}$ can be precomputed and stored so inversion is not needed in this stage. **(ii)** *Verification stage.* Verifying the MSM computation result requires ℓ modular inversions, 1 scalar dot product of size n, ℓ integer multiplications, 3ℓ scalar multiplications and ℓ point additions.

On the side of the service provider $\mathcal{S}$, it needs to compute 2ℓ MSM instances.

For simplicity, we consider a simple MSM computation algorithm, where each scalar multiplication $k_i P_i$ in S_n (defined as $S_n = \sum_{i=1}^{n} k_i P_i$) is computed separately using the doubling and addition method, then the intermediate results are added together to obtain the final result. We use the following computation cost model for comparison: **(i)** The computational cost of point doubling and addition are the same, denoted as $\mathtt{A}$. In the worst case, each scalar multiplication costs about $2 \cdot (\lfloor \log_2 ord \rfloor - 1)\,\mathtt{A}$, the total cost of computing S_n is $\left[2 \cdot (\lfloor \log_2 ord \rfloor - 1) \cdot n + (n-1)\right] \cdot \mathtt{A} \approx 2n \log_2 ord \cdot \mathtt{A}$. **(ii)** The unit cost $\mathtt{A}$ is equivalent to about 14 field multiplications, denoted by $\mathtt{M}$. Combining the above assumptions, the cost of every single MSM computation for $\mathcal{S}$ can be estimated as $28n \log_2 ord \cdot \mathtt{M}$, measured in terms of field multiplications. **(iii)** Similarly, the additional overhead for $\mathcal{P}$ can be estimated as $2\ell n \cdot \mathtt{M}$ for preparation and $(\ell \cdot \log_2 p + 57\ell + n) \cdot \mathtt{M}$ for verification.

Taking into account the probability that a malicious $\mathcal{S}$ returns a wrong MSM result but passes the verification (Section 4.2), the recommended choice is $\ell = 2$

or $\ell = 3$. Fig. 1 shows the theoretical analysis of the computation cost of one MSM task over BLS12-381 curve for $n = 2^k$ $(10 \leq k \leq 20)$. Compared with doing it locally, the computation cost on the side of $\mathcal{P}$ using our method is almost negligible. Specifically, our method is $1.22K \sim 1.43K$ times and $0.90K \sim 1.02K$ times faster than computing MSM locally for $\mathcal{P}$ with $\ell = 2$ and $\ell = 3$, respectively. Despite the increased computation time on the side of the service provider $\mathcal{S}$, the overhead is still acceptable in practice, as $\mathcal{S}$ typically has stronger computational power and more abundant resources.



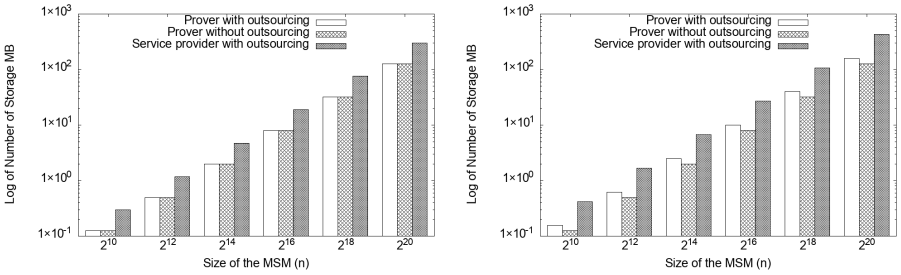
(a) Numbers of multiplications for $\ell = 2$ (b) Numbers of multiplications for $\ell = 3$

Fig. 1: Comparisons of computation cost.

Storage cost. We assume that each elliptic curve point is composed of two finite field elements (x, y), and the size of a point is given by $|P| = \lceil \log_2 p \rceil + 1$ bits after applying the compression scheme, where p is the characteristic of the underlying finite field. The size of a scalar/integer is $\lceil \log_2 ord \rceil$ bits, where ord denotes the order of the elliptic curve. With the simple doubling-addition MSM computation method, there is no storage cost for precomputation. In summary, for every MSM computation task, $\mathcal{P}$ needs to store: **(i)** The original n scalars containing secret information; and **(ii)** $(n \times \ell + \ell + n + 1)$ integers corresponding to the random matrix $R \in \mathbb{Z}_{ord}^{\ell \times n}$, random integer vector $\boldsymbol{t} = (t_1, t_2, \ldots, t_l)$ where $t_i \in \mathbb{Z}_{ord}$ and $1 \leq i \leq \ell$, secret binding vector $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$ where $\rho_j \in \mathbb{Z}_{\text{ord}}^n$ and $1 \leq j \leq n$, along with the secret binding scalar α. The service provider $\mathcal{S}$ needs to store: **(i)** $2n\ell$ elliptic curve points (matrix $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{T}}$); and **(ii)** $n \times \ell$ scalars (matrix $M_{\boldsymbol{k}}$).

When instantiating the protocol with the BLS12-381 curve, we summarize the storage-related results in Fig. 2. For $\ell = 2$, the prover $\mathcal{P}$ benefits from reduced storage requirements using our method compared to local computation (Fig. 2a), as it avoids storing the vector of elliptic curve points $\boldsymbol{P}$. For $\ell = 3$, the storage cost for $\mathcal{P}$ is nearly identical across both approaches, since additional randomness must be retained to support secure outsourced MSM computation. It is worth noting that the storage cost for the service provider is typically not a limiting factor, given its ample computational resources. Moreover, when employing a more efficient MSM algorithm such as Pippenger's method, the storage overhead is dominated by the precomputation table, which can reach several gigabytes. In such scenarios, the outsourcing protocol proposed in this

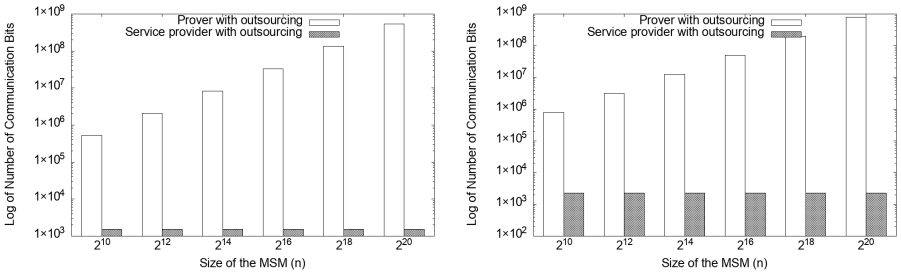
work becomes the only practical solution for a resource-constrained $\mathcal{P}$ who seeks to leverage high-performance MSM algorithms.



(a) Different storage costs for $\ell = 2$ (b) Different storage costs for $\ell = 3$

Fig. 2: Comparisons of storage cost.

Communication cost. Outsourcing introduces additional communication overhead; we evaluate its impact and demonstrate that this overhead remains acceptable for both the prover $\mathcal{P}$ and the service provider $\mathcal{S}$. In each MSM computation task, $\mathcal{P}$ transmits the matrix of processed scalars $M_{\boldsymbol{k}}$—of dimension $n \times \ell$—to $\mathcal{S}$. Upon completing the computation, $\mathcal{S}$ returns 2ℓ elliptic curve points $(Q_1, Q_2, \ldots, Q_\ell, V_1, V_2, \ldots, V_\ell)^T$ to $\mathcal{P}$ for verification. Fig. 3 presents the communication costs for both parties, considering only the cost of data transmission. For a fixed ℓ, the communication overhead incurred by $\mathcal{P}$ grows linearly with n, while the overhead for $\mathcal{S}$ remains constant and depends solely on ℓ, which is negligible for $\ell = 2, 3$. These findings collectively confirm that the communication overhead is modest, thereby supporting the practicality and feasibility of the proposed outsourcing protocol.



(a) Different communication costs for $\ell = 2$ (b) Different communication costs for $\ell = 3$

Fig. 3: Comparisons of communication cost.

5.2 Experimental Evaluation and Comparison

To further substantiate our theoretical analysis and compare it with other collaborative proof generation schemes [6, 14], we conducted a series of experiments

using `arkworks` cryptographic framework (`https://arkworks.rs`). The experiments were implemented in Rust and relied on the `ark-bls12-381`, `ark-ec`, and `ark-ff` libraries (v0.4.x), which provide optimized implementations of elliptic curve arithmetic over the BLS12-381 curve. In particular, MSM computations over $\mathbb{G}_1$ were performed using the built-in Pippenger algorithm provided by `arkworks`. All benchmarks were compiled in release mode and executed on an Amazon EC2 c7i.large instance equipped with 2 vCPUs and 4 GB RAM, running Ubuntu 24.04 LTS, and this environment is the same as HyperPlonk++ [14]. Each experiment was repeated 10 times, and the average execution time was reported to mitigate noise.

Experimental performance of our solution. The performance results of our solution are summarized in Table 3, which demonstrate substantial computational savings. Specifically, for varying values of n, the computation time required by $\mathcal{P}$ is reduced to just 0.57%–4.81% and 0.81%–7.62% of the time needed for local MSM computation for $\ell = 2$ and $\ell = 3$, respectively. Within the prover's computational overhead, the verification step incurs a constant cost and is negligible compared to the preparation phase with increase of n. Although the service provider $\mathcal{S}$ requires more time to complete each MSM task, this is not a limiting factor, as $\mathcal{S}$ is assumed to have access to significantly greater computational resources. Moreover, we note that the performance of the service provider $\mathcal{S}$ reflects execution on general-purpose CPU hardware, its performance can be expected to improve substantially when more efficient hardware such as GPUs or ASICs is employed.

Table 3: Summary of experimental performance results of our solution.

n	ℓ	$\mathcal{P}$ (w/o outsourcing)	$\mathcal{P}$ (outsourcing)		$\mathcal{S}$ (outsourcing)
			Preparation	Verification	
2^{10}	2	281 ms	0.593 ms	12.909 ms	1.122 s
	3	280 ms	0.892 ms	20.455 ms	1.682 s
2^{12}	2	905 ms	2.373 ms	13.356 ms	3.621 s
	3	906 ms	3.604 ms	20.907 ms	5.433 s
2^{14}	2	3.115 s	9.626 ms	15.166 ms	12.461 s
	3	3.102 s	14.439 ms	22.713 ms	18.612 s
2^{16}	2	10.388 s	37.977 ms	22.256 ms	41.551 s
	3	10.380 s	56.649 ms	29.749 ms	62.281 s
2^{18}	2	38.592 s	153.375 ms	51.108 ms	154.367 s
	3	37.943 s	250.927 ms	62.133 ms	227.655 s
2^{20}	2	134.739 s	603.804 ms	163.715 ms	538.956 s
	3	134.739 s	919.110 ms	173.496 ms	808.434 s

Experimental performance comparison with other collaborative proof generation methods. Unlike collaborative proof-generation schemes zkSaaS [6] and HyperPlonk++ [14] that handle the entire proof-generation process and involve multiple collaborators, our solution targets MSM computation specifically

and requires only one additional collaborator (the $\mathcal{S}$). To ensure a fair comparison, we make the following assumptions: **(i)** Since zkSaaS and HyperPlonk++ worked on different ZKP schemes, we use the acceleration ratio as the metric for computation efficiency comparison; and **(ii)** We assume a straightforward distributed implementation of $\mathcal{S}$ with 1 leader node and 127 worker nodes, where the original MSM instance is partitioned into 128 smaller sub-tasks evenly. The leader computes one sub-task and collects results from worker nodes to compute the final result.

Table 4: Comparison our solution with other collaborative proof generation methods [6] and [14].

Method		Computation in LAN	Computation in WAN	Communication
zkSaaS [6]	Leader	0.9×	0.1×	90 GB
	Worker	0.9×	0.1×	718 MB
HyperPlonk++ [14]	Leader/Worker	28.0×	25.5×	43 MB
Ours		94.15×[a]		$\mathcal{P}$: 4.28 MB $\mathcal{S}$: 69 KB

[a] Since the performance difference between LAN and WAN is negligible for our solution, we report a single ratio.

Table 4 summarizes the computation and communication costs of our solution compared to zkSaaS and HyperPlonk++, assuming a circuit size of 2^{21}, 128 parties, and $\ell = 2$. Overall, our MSM outsourcing approach demonstrates superior performance relative to existing MPC-based collaborative proof generation schemes: **(i)** *Higher computational efficiency.* MSM is inherently embarrassingly parallel, and state-of-the-art implementations achieve near-linear speedup [3]. Under our outsourcing model, the service provider $\mathcal{S}$ primarily executes one or more MSM instances, enabling straightforward parallelization and hardware acceleration. This design achieves a 94.15× speedup with 128 nodes. In contrast, MPC-based solutions exhibit limited scalability; for example, HyperPlonk++ reports only about 28× speedup with 128 parties. **(ii)** *Lower communication overhead.* MPC protocols typically incur significant communication costs. For zkSaaS and HyperPlonk++, the per-server communication volume ranges from 43 MB to 90 GB depending on the server's role. In comparison, our solution requires only 4.28 MB for $\mathcal{P}$ and 69 KB for $\mathcal{S}$.

6 Related Works

Efficient MSM computation. Efficient MSM computation has been intensively studied. When no extra information is provided (e.g., the relationship between elliptic curve points of the MSM instance), it is difficult to simplify the computation cost directly. From an algorithmic perspective, different strategies have

been proposed to achieve a trade-off between computation and storage, including Pippenger's algorithm [19] and its variants [5, 15]. The central idea of these algorithms is to store a set of pre-computed values and use them multiple times to reduce the overall computation cost. The other direction is to use different hardware platforms to speed up the computation through parallelism, such as GPU [11], FPGA [20], and ASIC [13]. These works are complementary to the MSM outsourcing method developed in this paper. By enabling secure MSM outsourcing, these MSM computation acceleration hardware can be deployed in a cloud computing environment and shared by multiple users to maximize resource utilization.

Collaborative proof generation using MPC. Smart and Alaoui [23] developed a framework that can convert a one-party protocol over an elliptic curve to an n-party protocol, which utilizes a commitment scheme and an SPDZ protocol [4]. Following this work, several efforts have been made to split the zk-SNARK proof generation workload among multiple participants [18, 6]. As an important component of the proof generation procedure, the traditional MSM computation is also converted into a distributed version. The original goal of [18] was to enable proof generation for a secret shared among multiple participants, and [6, 14] extended the idea to build a service for proof generation.

Unlike the MSM outsourcing approach proposed in this paper, prior works [18, 6] based on general MPC frameworks adopt the strategy of outsourcing the entire proof generation process. Although these approaches offer significant flexibility (e.g., supporting diverse statements and even multiple ZKP schemes), they incur substantial efficiency penalties for core operations such as MSM, as discussed in Section 5. Furthermore, our solution only assumes the prover itself is honest and generates the proof knowing the secret scalars (witness), while MPC-based collaborative proof generation approaches [18, 6] require the majority of participants to be honest to generate a correct proof and hide the secret witness.

Linear blinding and blind signatures. The concept of linear blinding has been widely applied in the construction of blind signature schemes, which aim to conceal the message from the signer [2, 17]. Similar to MSM outsourcing, a blind signature protocol typically involves three stages: **(i)** The user preprocesses the message and sends a "blinded" version to the signer; **(ii)** The signer applies its private key to sign the blinded message and returns the signature; and **(iii)** The user "unblinds" the received signature using the public key and the blinding information used during preprocessing. Despite this structural similarity, there are several fundamental differences: **(i)** *Goals.* Blind signatures aim solely to hide the message from the signer, whereas MSM outsourcing prioritizes reducing the prover's computational cost in addition to preserving secrecy; and **(ii)** *Settings.* Blind signature schemes rely on a public/private key pair, whereas MSM outsource computation assumes that the server holds no extra secret information.

7 Conclusion

In this work, we address the computational bottleneck posed by multi-scalar multiplication (MSM) in zero-knowledge proof (ZKP) generation by introducing a secure and efficient outsourcing protocol. Our approach enables a prover to delegate MSM tasks to a more powerful but untrusted service provider, while preserving the confidentiality of secret scalars and ensuring verifiability of the results. Through a novel randomization and de-randomization mechanism, the protocol achieves strong security guarantees, including scalar hiding and correctness verification, with a negligible probability of accepting incorrect outputs. Both theoretical and empirical analyses demonstrate that our scheme substantially reduces the prover's computational and storage burdens, while maintaining practical efficiency for the service provider. Moreover, the protocol is compatible with existing MSM acceleration techniques and ZKP frameworks, making it a scalable and cost-effective solution for real-world deployments. This work opens new directions for secure computation outsourcing in cryptographic systems and lays a foundation for the broader adoption of ZKP technologies in resource-constrained environments.

References

1. Botrel, G., El Housni, Y.: Faster montgomery multiplication and multi-scalar-multiplication for snarks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(3), 504–521 (2023)
2. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of Crypto 82. pp. 199–203. Springer (1983)
3. Chen, Y., Peng, C., Dai, Y., Luo, M., He, D.: Load-balanced parallel implementation on gpus for multi-scalar multiplication algorithm. IACR Transactions on Cryptographic Hardware and Embedded Systems (2), 522–544 (2024)
4. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Annual cryptology conference. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012)
5. Fan, X., Kuchta, V., Sica, F., Xu, L.: Speeding up multi-scalar multiplications for pairing-based zkSNARKs. Journal of Cryptology **38**(2), 21 (2025)
6. Garg, S., Goel, A., Jain, A., Policharla, G.V., Sekar, S.: zksaas: Zero-knowledge snarks as a service. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 4427–4444 (2023)
7. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology **7**(1), 1–32 (1994)
8. 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)
9. Hachez, G., Quisquater, J.J.: Montgomery exponentiation with no final subtractions: Improved results. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 293–301. Springer (2000)
10. Järvinen, K., Balasch, J.: Single-trace side-channel attacks on scalar multiplications with precomputations. In: International Conference on Smart Card Research and Advanced Applications. pp. 137–155. Springer (2016)

11. Ji, Z., Zhang, Z., Xu, J., Ju, L.: Accelerating multi-scalar multiplication for efficient zero knowledge proofs with multi-GPU systems. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. pp. 57–70. ACM (2024)
12. Khabbazian, M.: 2g2t: Constant-size, statistically sound msm outsourcing (2026), https://arxiv.org/abs/2602.23464
13. Liu, C., Zhou, H., Dai, P., Shang, L., Yang, F.: Priormsm: An efficient acceleration architecture for multi-scalar multiplication. ACM Transactions on Design Automation of Electronic Systems **29**(5), 1–26 (2024)
14. Liu, X., Zhou, Z., Wang, Y., Pang, Y., He, J., Zhang, B., Yang, X., Zhang, J.: Scalable collaborative zk-snark and its application to fully distributed proof delegation. In: 34th USENIX Security Symposium (USENIX Security 25) (2025)
15. Luo, G., Fu, S., Gong, G.: Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 358–380 (2023)
16. Mittmann, J., Schindler, W.: Timing attacks and local timing attacks against barrett's modular multiplication algorithm. Journal of Cryptographic Engineering **11**(4), 369–397 (2021)
17. Okamoto, T.: An efficient divisible electronic cash scheme. In: Annual international cryptology conference. pp. 438–451. Springer (1995)
18. Ozdemir, A., Boneh, D.: Experimenting with collaborative {zk-SNARKs}:{Zero-Knowledge} proofs for distributed secrets. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 4291–4308 (2022)
19. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
20. Pottier, X., de Ruijter, T., Bertels, J., Legiest, W., Van Beirendonck, M., Verbauwhede, I.: Optimsm: Fpga hardware accelerator for zero-knowledge msm. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(2), 489–510 (2025)
21. Sasdrich, P., Güneysu, T.: Implementing curve25519 for side-channel–protected elliptic curve cryptography. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **9**(1), 1–15 (2015)
22. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy. pp. 459–474. IEEE (2014)
23. Smart, N.P., Talibi Alaoui, Y.: Distributing any elliptic curve based protocol. In: IMA International Conference on Cryptography and Coding. pp. 342–366. Springer (2019)
24. Xing, Z., Zhang, Z., Zhang, Z., Li, Z., Li, M., Liu, J., Zhang, Z., Zhao, Y., Sun, Q., Zhu, L., et al.: Zero-knowledge proof-based verifiable decentralized machine learning in communication network: A comprehensive survey. IEEE Communications Surveys & Tutorials (2025)
25. Zhang, Y., Wang, S., Zhang, X., Dong, J., Mao, X., Long, F., Wang, C., Zhou, D., Gao, M., Sun, G.: Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). pp. 416–428. ACM/IEEE (2021)
26. 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. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(4), 258–284 (2024)

A Formal Security Analysis and Proofs

A.1 Proof of Correctness

Theorem 1. *If $\mathcal{S}$ follows the protocol honestly, then all verification checks pass and $\mathcal{P}$ obtains the correct MSM result S_n.*

Proof of Theorem 1. According to Equation (**??**) and Equation (1), we have:

$$(k_1, k_2, \ldots, k_n) = (\frac{k_{i,1}r_{i,1}}{t_i}, \frac{k_{i,2}r_{i,2}}{t_i}, \ldots, \frac{k_{i,n}r_{i,n}}{t_i}), \quad i = 1, 2, \ldots, \ell. \tag{6}$$

The correctness of the protocol can be verified by checking the following equation:

$$A_i = t_i^{-1}Q_i = \frac{1}{t_i}\sum_{j=1}^{n} k_{i,j}(r_{i,j}P_j) = \sum_{j=1}^{n} \frac{k_{i,j}r_{i,j}}{t_i}P_j = \sum_{j=1}^{n} k_j P_j, \quad i = 1, 2, \ldots, \ell. \tag{7}$$

The last equality holds due to Equation (6), and $\sum_{j=1}^{n} k_j P_j = S_n$ according to the definition of S_n.

For the binding check, when the server is honest:

$$\begin{aligned}
V_i &= \sum_{j=1}^{n} k_{i,j} \cdot T_{i,j} \\
&= \sum_{j=1}^{n} k_{i,j} \cdot \left[\alpha \cdot (r_{i,j}P_j) + r_{i,j}\rho_j \cdot Q\right] \\
&= \alpha \cdot \underbrace{\sum_{j=1}^{n} k_{i,j} \cdot (r_{i,j}P_j)}_{=Q_i} + \left(\sum_{j=1}^{n} k_{i,j} \cdot r_{i,j}\rho_j\right) \cdot Q \\
&= \alpha \cdot Q_i + \left(\sum_{j=1}^{n} \frac{t_i k_j}{r_{i,j}} \cdot r_{i,j}\rho_j\right) \cdot Q \\
&= \alpha \cdot Q_i + t_i \left(\sum_{j=1}^{n} k_j \rho_j\right) \cdot Q \\
&= \alpha \cdot Q_i + (t_i \cdot s) \cdot Q.
\end{aligned}$$

□

A.2 Proof of Soundness for Single Query

Lemma 1 (Binding). *Fix a row $i \in [\ell]$. Suppose the server computes Q_i' and V_i' using only the terms indexed by a strict subset $I \subsetneq [n]$. Then the binding check* (5) *fails except with probability at most* $1/\mathrm{ord}$ *over the randomness of $\boldsymbol{\rho}$.*

Proof. When the server restricts to subset I:

$$V_i' = \sum_{j \in I} k_{i,j} \cdot T_{i,j} = \alpha \cdot Q_i' + \left(\sum_{j \in I} t_i k_j \rho_j \right) \cdot Q.$$

For the binding check to pass, we need:

$$V_i' = \alpha \cdot Q_i' + (t_i \cdot s) \cdot Q,$$

which requires:

$$\sum_{j \in I} t_i k_j \rho_j = t_i \cdot \sum_{j=1}^{n} k_j \rho_j \iff \sum_{j \notin I} k_j \rho_j = 0.$$

Since $I \subsetneq [n]$, there exists at least one $j_0 \notin I$ with $k_{j_0} \neq 0$ (in the generic case). The sum $\sum_{j \notin I} k_j \rho_j$ is a non-trivial linear function of the ρ_j's for $j \notin I$. Since ρ_{j_0} is uniform over $\mathbb{Z}_{\text{ord}}$ and independent of everything else in the server's view (the trap points $T_{i,j}$ hide ρ_j behind the random term $r_{i,j}\rho_j Q$ plus a point involving α), the probability that this linear function evaluates to zero is exactly $1/\text{ord}$.

If all omitted scalars are zero (i.e., $k_j = 0$ for all $j \notin I$), then $\sum_{j \notin I} k_j P_j = \mathcal{O}$ and the partial sum equals the full MSM, so no cheating has occurred. □

Theorem 2 (Single-Query Statistical Soundness). *In the protocol (Algorithm 1), for any (possibly computationally unbounded) adversarial server S^* that returns $(Q_1', \ldots, Q_\ell', V_1', \ldots, V_\ell')$ such that the prover would accept an incorrect MSM result $A \neq S_n$, the probability of acceptance is at most* $1/\text{ord}$*, which is negligible.*

Proof. For the prover to accept, both the consistency check and all ℓ binding checks must pass. If $A \neq S_n$, then the consistency check forces $t_i^{-1} Q_i' = A$ for all i, i.e., $Q_i' = t_i A$ for all i. Define $Q_i^\circ = t_i S_n$ as the honest value. Since $A \neq S_n$, we have $Q_i' \neq Q_i^\circ$ for every i.

The binding check for row i requires $V_i' = \alpha \cdot Q_i' + (t_i \cdot s) \cdot Q$. The honest binding value satisfies $V_i^\circ = \alpha \cdot Q_i^\circ + (t_i \cdot s) \cdot Q$. Subtracting:

$$V_i' - V_i^\circ \;=\; \alpha \cdot (Q_i' - Q_i^\circ) \;=\; \alpha \cdot t_i (A - S_n).$$

Since $A \neq S_n$ and $t_i \neq 0$, the point $t_i(A - S_n) \neq \mathcal{O}$. Because EC has prime order, the map $\alpha \mapsto \alpha \cdot t_i(A - S_n)$ is a bijection on EC. Hence the equation fixes α to exactly one value.

By independence of α from the server's view (Lemma 2 in Appendix A.3), α is uniform over $\mathbb{Z}_{\text{ord}}$, so each row's binding check passes with probability $\leq 1/\text{ord}$. Since all ℓ rows must simultaneously pass with the *same* α, and each pins α to at most one value, the overall acceptance probability is at most $1/\text{ord}$. □

A.3 Proof of Soundness for Multiple Queries

In practice, the prover outsources *many* MSM computations to the same server using the same one-time secret key material. A realistic adversary is therefore *adaptive*: after observing the accept/reject outcome of each query, it can adjust its cheating strategy for subsequent queries. Here we prove tight multi-query (L-query) soundness bounds.

Definition 1 (L-Query Adaptive Adversary). *An L-query adaptive adversary S^* interacts with the prover/client in the following game:*

1. ***Setup.*** *The prover runs the one-time setup and publishes all public material (point matrices, trap matrices, merged-bases vectors, etc.). Secret key material is kept hidden.*
2. ***Queries.*** *For $j = 1, 2, \ldots, L$:*
   (a) *The prover prepares query j with scalar vector $\boldsymbol{k}^{(j)}$ and sends the (masked) query to the server.*
   (b) *S^* observes the query and all prior accept/reject decisions, and returns a response.*
   (c) *The prover runs the verification and announces accept/reject to S^*.*
3. ***Win condition.*** *S^* wins if* any *query $j \in [L]$ results in the prover accepting an incorrect MSM result.*

S^* may be computationally unbounded and may choose its responses as an arbitrary function of its entire view (public setup, all prior queries, all prior accept/reject bits, and its own internal state).

Lemma 2 (Independence of α from the server's view). *In the protocol (Algorithm 1), the binding secret α is statistically independent of (M_P, M_T). Formally, for every fixed $(m_P, m_T) \in \mathrm{EC}^{\ell \times n} \times \mathrm{EC}^{\ell \times n}$ and every $a \in \mathbb{Z}_{\mathrm{ord}}$:*

$$\Pr[\alpha = a \mid M_P = m_P, M_T = m_T] = \frac{1}{\mathrm{ord}}.$$

Proof. Fix $M_P = m_P$ (which determines each $r_{i,j}P_j$) and fix any $a \in \mathbb{Z}_{\mathrm{ord}}$. Conditioned on $\alpha = a$, the trap entry $T_{i,j} = a \cdot (r_{i,j}P_j) + r_{i,j}\rho_j \cdot Q$. Since $\rho_j \xleftarrow{\$} \mathbb{Z}_{\mathrm{ord}}$ independently, $r_{i,j} \neq 0$, and $Q \neq \mathcal{O}$, the term $r_{i,j}\rho_j \cdot Q$ is uniform in EC. Adding the fixed point $a \cdot (r_{i,j}P_j)$ is a translation, so $T_{i,j}$ is uniform in EC. Independence of the ρ_j's implies M_T is uniform in $\mathrm{EC}^{\ell \times n}$ regardless of a. Hence $\Pr[M_T = m_T \mid \alpha = a, M_P = m_P] = \Pr[M_T = m_T \mid M_P = m_P]$ for all a, and by Bayes' theorem $\Pr[\alpha = a \mid M_P, M_T] = 1/\mathrm{ord}$. □

Lemma 3 (Uniqueness of accepting α for an incorrect result). *Fix (M_P, M_T) and a query with scalars $\boldsymbol{k}^{(j)}$, per-row blinding $(t_1^{(j)}, \ldots, t_\ell^{(j)})$, and server response $(Q'_1, \ldots, Q'_\ell, V'_1, \ldots, V'_\ell)$. If the consistency check passes with output $A \neq S_n^{(j)}$, then there is* ***at most one*** *value of $\alpha \in \mathbb{Z}_{\mathrm{ord}}$ for which all ℓ binding checks simultaneously pass.*

Proof. Since the consistency check passes, $Q'_i = t_i^{(j)} A$ for all i. The binding check for row i is

$$V'_i \;=\; \alpha \cdot Q'_i + (t_i^{(j)} \cdot s^{(j)}) \cdot Q,$$

where $s^{(j)} = \langle \boldsymbol{k}^{(j)}, \boldsymbol{\rho} \rangle$. Let $Q_i^\circ = t_i^{(j)} S_n^{(j)}$ be the honest MSM output for row i. Define the "error" point $E_i = Q'_i - Q_i^\circ = t_i^{(j)}(A - S_n^{(j)})$. Since $A \neq S_n^{(j)}$ and $t_i^{(j)} \neq 0$, we have $E_i \neq \mathcal{O}$.

The honest binding value is $V_i^\circ = \alpha \cdot Q_i^\circ + (t_i^{(j)} s^{(j)})Q$. Subtracting:

$$V'_i - V_i^\circ \;=\; \alpha \cdot E_i.$$

For row i's check to pass, we need $V'_i - V_i^\circ = \alpha \cdot E_i$. Since $E_i \neq \mathcal{O}$ and EC has prime order, the map $\alpha \mapsto \alpha \cdot E_i$ is a bijection $\mathbb{Z}_{\text{ord}} \to \text{EC}$. Hence there is at most one α satisfying row i's constraint.

Since all ℓ rows must agree on the same α, the intersection of the ℓ singleton-or-empty solution sets contains at most one element. □

Theorem 3 (L-Query Adaptive Soundness of the Protocol). *In the protocol (Algorithm 1), for any L-query adaptive adversary S^* (Definition 1),*

$$\Pr\big[\exists\, j \in [L] :\ \textit{prover accepts incorrect } A^{(j)} \neq S_n^{(j)}\big] \;\leq\; \frac{L}{\text{ord}}.$$

The probability is over the one-time randomness $(\alpha, \boldsymbol{\rho}, R)$ and the per-query randomness $(t_1^{(j)}, \ldots, t_\ell^{(j)})_{j \in [L]}$.

Proof. We follow the candidate-set elimination framework of [12, Theorem 3], with α playing the role of r.

Fix an arbitrary realization (m_P, m_T) of (M_P, M_T) an condition on this event (and on S^*'s internal randomness, so that S^* is deterministic given the prover's accept/reject decisions). By Lemma 2, under this conditioning α is uniform over $\mathbb{Z}_{\text{ord}}$.

Define $\text{Bad}_{\leq} j$ as the event that an incorrect acceptance occurs in one of the first j queries, and set $\text{Bad} = \text{Bad}_{\leq} L$. We maintain a *candidate set* $\mathcal{A}_j \subseteq \mathbb{Z}_{\text{ord}}$ of values of α consistent with the transcript through query j, conditioned on $\neg\text{Bad}_{\leq} j$. Initialize $\mathcal{A}_0 = \mathbb{Z}_{\text{ord}}$.

Consider query $j \in [L]$ and condition on $\neg\text{Bad}_{\leq} j - 1$.

Case 1: $A^{(j)} = S_n^{(j)}$ (server returns correct MSM for query j). The consistency check and binding checks either both pass (honest computation) or the binding check fails due to an incorrect V'_i. In either case, no *incorrect acceptance* occurs in query j.

If the binding checks pass, then the check $V'_i = \alpha \cdot Q_i^{(j)} + (t_i^{(j)} s^{(j)})Q$ with $Q_i^{(j)} = t_i^{(j)} S_n^{(j)}$ is equivalent to $V'_i = MSM(M_T^{(i)}, \boldsymbol{k}_i^{(j)})$, which depends on M_T (fixed by conditioning) but *not* on α given M_T. Hence the accept/reject bit is independent of α and no candidate is eliminated: $\mathcal{A}_j = \mathcal{A}_{j-1}$.

If the binding checks fail (server gave correct A but wrong V_i), the prover rejects. Since $A^{(j)}$ is correct, this is not an incorrect acceptance. The rejection

may depend on α, but it does not constitute a failure event. In the worst case, the server may learn that α is *not* some specific value (from a failed binding check on a deliberately incorrect V_i'). However, since $A^{(j)}$ was correct, this query cannot contribute to the bad event. We conservatively set $\mathcal{A}_j = \mathcal{A}_{j-1}$ (no elimination needed for the soundness bound).

Case 2: $A^{(j)} \neq S_n^{(j)}$ (server returns incorrect MSM for query j).. By Lemma 3, the binding checks can pass for at most one value $\hat{\alpha}_j \in \mathbb{Z}_{\text{ord}}$. Conditioned on $\neg\text{Bad}_{\leq} j-1$ and $(M_P, M_T) = (m_P, m_T)$, the secret α is uniform over $\mathcal{A}_{j-1}$. Hence:

$$\Pr[\text{incorrect acceptance at query } j \mid \neg\text{Bad}_{\leq} j-1,\ (M_P, M_T) = (m_P, m_T)] \ \leq\ \frac{1}{|\mathcal{A}_{j-1}|}.$$

On the event $\neg\text{Bad}_{\leq} j$ (i.e., no incorrect acceptance through query j), the prover must have rejected, which implies $\alpha \neq \hat{\alpha}_j$. We set $\mathcal{A}_j = \mathcal{A}_{j-1} \setminus \{\hat{\alpha}_j\}$, so $|\mathcal{A}_j| \geq |\mathcal{A}_{j-1}| - 1$.

Aggregation. By construction, $|\mathcal{A}_0| = \text{ord}$ and each Case 2 query shrinks the candidate set by at most one. On $\neg\text{Bad}_{\leq} j-1$, we have $|\mathcal{A}_{j-1}| \geq \text{ord} - (j-1)$, so:

$$\Pr[\text{incorrect acceptance at query } j \mid \neg\text{Bad}_{\leq} j-1] \ \leq\ \frac{1}{\text{ord} - (j-1)}.$$

Let $p_j = \Pr[\text{Bad}_{\leq} j \mid (M_P, M_T)]$. Then:

$$p_j \ \leq\ p_{j-1} + (1 - p_{j-1}) \cdot \frac{1}{\text{ord} - (j-1)}.$$

With $p_0 = 0$, a standard induction yields $p_j \leq j/\text{ord}$ for all $j \leq L$. In particular, $p_L \ \leq\ \frac{L}{\text{ord}}$. Since (m_P, m_T) was arbitrary, the bound holds unconditionally. □

A.4 Proof of Scalar Privacy

Theorem 4 (Scalar Privacy). *The protocol Π achieves perfect privacy against a malicious server $\mathcal{S}^*$. Specifically, the server's view* $\mathsf{View} = (M_{\boldsymbol{P}}, M_{\boldsymbol{T}}, M_{\boldsymbol{k}})$ *is statistically independent of the secret scalar vector $\boldsymbol{k}$.*

Proof. The server's view consists of three components, each of which we analyze individually:

(i) $M_{\boldsymbol{P}}$: This matrix is constructed by applying row-wise randomization to the base point vector $\boldsymbol{P}$. Each entry is defined as$r_{i,j}P_j$, where $r_{i,j}$ is sampled uniformly and independently from $\mathbb{Z}^*_{ord}$. Since $P_j \neq \mathcal{O}$ and $r_{i,j}$ is a secret blinding factor known only to $\mathcal{P}$, each entry $r_{i,j}P_j$ is a uniformly random element in $\mathbb{EC}$. Consequently, $M_{\boldsymbol{P}}$ is a matrix of random group elements that reveals no information about the scalars.

(ii) $M_{\boldsymbol{T}}$: Each entry is defined as $T_{i,j} = \alpha \cdot (r_{i,j} P_j) + r_{i,j} \rho_j \cdot Q$, where $(\alpha, \boldsymbol{\rho})$ are the long-lived secret binding parameters. Conditioned on $r_{i,j} P_j$ (which is known to the server from $M_{\boldsymbol{P}}$), the term $r_{i,j} \rho_j \cdot Q$ is uniformly random in $\mathbb{EC}$. This follows because ρ_j is uniform and $r_{i,j} \neq 0, Q \neq \mathcal{O}$. Hence, $T_{i,j}$ is uniform conditioned on $M_{\boldsymbol{P}}$ and independent of $\boldsymbol{k}$.

(iii) $M_{\boldsymbol{k}}$: To hide the sensitive input $\boldsymbol{k}$, $\mathcal{P}$ blinds each scalar using per-row and per-column randomness: $k_{i,j} = t_i k_j r_{i,j}^{-1} \pmod{ord}$. Since the row-blinding factor t_i is sampled uniformly from $\mathbb{Z}_{ord}^*$ and is independent of $r_{i,j}$, the resulting product $t_i(k_j r_{i,j}^{-1})$ is uniformly distributed over $\mathbb{Z}_{ord}^*$ for any fixed non-zero k_j. Furthermore, because t_i is uniformly sampled for each row and $r_{i,j}$ is randomized for each entry, all $k_{i,j}$ are mutually independent.

Since the joint distribution of $(M_{\boldsymbol{P}}, M_{\boldsymbol{T}}, M_{\boldsymbol{k}})$ is identical to a set of independent and uniform samples from the respective domains, the server's entire view is completely independent of the secret scalars $\boldsymbol{k}$. □