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], Sica Francesco[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
fan.hecc@gmail.com
[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, blockchain technologies, and other domains. 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 proof (ZKP) is a type of cryptographic protocol that enables one party (the prover) to convince another party (the verifier) of the truth of a

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

statement without revealing any additional information. ZKPs have numerous applications in secure communication [30], privacy-preserving and verifiable computation [31], blockchain technologies [26], and cryptography [6, 1].

Despite their transformative potential, ZKP protocols often impose substantial computational costs, particularly during the proof generation phase. A key bottleneck in many modern ZKP systems, especially those based on elliptic curve cryptography (e.g., Groth16 [12], SONIC [19] and PLONK [9]), is the multi-scalar multiplication (MSM) operation. This operation computes a linear combination of elliptic curve points and dominates the runtime of proof generation. Many acceleration techniques have been proposed to speed up MSM computation, including algorithmic optimization [7] and dedicated hardware accelerators [33]. Given the inherently parallel and memory-bound nature of MSM, efficient hardware implementation remains the most practical and effective approach for achieving fast MSM computations. Unfortunately, leveraging specialized hardware accelerators for MSM computations presents two major challenges. First, availability is a significant barrier. Accelerators, such as GPUs or FPGAs, are often expensive and not universally accessible to all ZKP users. In constrained environments such as IoT devices, deploying such hardware is typically infeasible. Second, low utilization limits practical efficiency. MSM computation is only one component of the broader proof generation process, which itself is often embedded within larger applications. Moreover, users rarely run ZKP-based applications continuously, leading to sporadic usage of the accelerator and under-utilization of resources.

To address the limitations of local hardware acceleration, a promising yet underexplored direction is to outsource proof generation, particularly the most computationally intensive operations such as MSM, to a third-party service provider equipped with specialized hardware. Centralizing proof generation enables hardware accelerators to be shared across multiple users, thereby improving resource utilization and reducing individual costs. However, this outsourcing model introduces two critical security requirements. First, any secret information involved in proof generation must remain hidden from the service provider. Second, the user must be able to verify the correctness of the proof returned by the provider.

There are two main strategies for outsourcing proof generation. The first is to delegate the entire proof generation process to the service provider. While conceptually straightforward, this approach makes it difficult to satisfy the aforementioned security requirements due to the complexity of the computation. Achieving privacy and verifiability in this setting often requires expensive cryptographic techniques, such as secure multiparty computation [27], and several collaborative whole proof generation schemes have been proposed [10, 17]. An alternative strategy is to outsource only the most computationally demanding component like MSM while keeping the rest of the proof generation local. This not only reduces the complexity of the design, but also effectively mitigates the computation bottleneck. In this paper, we focus on outsourcing MSM computation and present a protocol that simultaneously achieves *scalar privacy* and *efficient result verifiability*. At a high level, we randomize the MSM instance by masking the public point set as well as the corresponding scalar vector. This blinding preserves the algebraic structure, conceals the witness scalars from the service provider, and enables lightweight de-randomization at the client side for verification. Unlike full outsourcing or MPC-based collaborative proving, our selective MSM outsourcing avoids heavy cryptographic machinery and high communication while remaining compatible with existing MSM accelerators. In summary, our MSM outsourcing scheme offers the following benefits: **(i)** *Immediate verification.* The prover can check the consistency of the returned MSM result via a single de-scaling step, detecting incorrect outputs before running subsequent proof-generation operations, thereby avoiding wasted computation. **(ii)** *Strong scalar privacy.* The service provider's view of the randomized inputs is independent of the original scalars, ensuring that the secret witness values are not leaked, even against a malicious provider. **(iii)** *Modular, drop-in integration.* The protocol is agnostic to the underlying ZKP and MSM implementation, i.e., it can be plugged into any ZKP that relies on MSM and is paired with existing algorithmic/hardware accelerators without modifying the rest of the proof-generation pipeline. **(iv)** *Practical efficiency.* Our analysis and experiments show that the prover's workload is reduced by more than 90% relative to local MSM computation.

2 Background of ZKP and MSM Computation

In this section, we briefly review the zero-knowledge proof technology, especially the associated computation tasks.

2.1 Zero-Knowledge Proof

ZKP is a cryptographic protocol that allows one party (the prover) to convince another party (the verifier) that a statement is true without revealing any additional information about the statement itself [11]. Formally, a ZKP system consists of three algorithms: `Setup`, `Prove`, and `Verify`.

- `Setup` takes as input a security parameter and outputs a public parameter that is shared between the prover and the verifier.
- `Prove` takes as input the public parameter, a secret witness, and a statement, and outputs a proof.
- `Verify` takes as input the public parameter, the statement, and the proof, and outputs either accept or reject.

A ZKP system needs to satisfy three properties, *completeness*, *soundness*, and *zero-knowledge*. ZK-SNARK is a special type of ZKP scheme that further meets the features *succinctness* and *non-interactive*. A ZK-SNARK system usually has an asymmetric computation cost, i.e., the proof generation is much more expensive than the proof verification, and MSM computation is often the most expensive part of the proof generation (i.e., 60% to 80% of the computation [33, 2]).

2.2 MSM Computation

Mathematically, MSM is defined as a sum of multiple scalar multiplications

$$S_n = \sum_{i=1}^{n} k_i P_i. \tag{1}$$

Here k_i is a positive integer, P_i is a point on the given elliptic curve defined on a finite field, and S_n is the calculation result, which is also a point on the same elliptic curve. Computationally, MSM is expensive when the number of scalar multiplications n and each scalar k_i are large.

The most popular strategy for fast MSM computation is using precomputation. By storing a set of precomputed values (usually in the form of elliptic curve points derived from points $P_1, P_2, \ldots, P_n$ in the original problem), the computation can be accelerated by reusing these values. Pippenger's algorithm [23] is a well-known algorithm that uses precomputation to speed up MSM computation, which breaks down a scalar into multiple chunks and precomputes scalar multiplications for all possible values of a chunk. Variants of Pippenger's algorithm have been proposed to further optimize MSM computation, which usually involves different strategies for precomputation and aggregation. Besides development of efficient algorithms, hardware is also utilized for MSM acceleration including GPU, FPGA, and ASIC [34, 15, 24, 32].

2.3 Overview of Proof Generation in ZK-SNARK

We briefly review the proof generation process of Groth16 [12] to demonstrate the role of MSM and the security requirements. For a given circuit C, the `Setup` algorithm translates it to a Quadratic Arithmetic Program (QAP) instance, which consists of three sets of polynomials $\vec{A}(z) = \{A_i(z)\}, \vec{B}(z) = \{B_i(z)\}, \vec{C}(z) = \{C_i(z)\}, i \in [1, m]$ with coefficients in a finite field $\mathbb{F}$, and a target polynomial $Z(z) \in \mathbb{F}[z]$. Suppose F is a function describing the QAP, by taking as input n elements of $\mathbb{F}$ and outputting n' elements, i.e. in total $N = n + n'$ input/output elements. Then a QAP computes F if for a valid assignment $(x_1, \ldots, x_N) \in \mathbb{F}^N$ of inputs and outputs there exist coefficients $(x_{N+1}, \ldots, x_m)$ s.t. the target polynomial $Z(z)$ divides the polynomial $P(z)$ defined as

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

This is equivalent to: $\vec{A}(z) \cdot \vec{B}(z) - \vec{C}(z) = \vec{H}(z) \cdot \vec{Z}(z)$ for some polynomial $H(z) \in \mathbb{F}[z]$. The prover knows a valid assignment for the circuit C, which is $(\vec{x}, \vec{w})$, where $\vec{x} = (x_1, \ldots, x_N)$ is public and $\vec{w} = (x_{N+1}, \ldots, x_m)$ is the secret input. For $i \in [N+1, m]$, the prover evaluates the polynomials at a random value s (provided by the `Setup` algorithm) and computes $\tilde{Y}(s) = \sum_{i=N+1}^{m} x_i Y_i(s)$, for $Y \in \{A, B, C, H\}$. These values are encoded into elliptic curve points $g^{\tilde{Y}(s)}$, with g being the generator of an elliptic curve group and the prover outputs them as the proof. For $i \in [1, N]$, the verifier evaluates the polynomials at s computes $\bar{Y}(s) = \sum_{i=1}^{N} x_i Y_i(s)$, with $Y \in \{A, B, C, H\}$. These values are encoded into elliptic curve points $g^{\bar{Y}(s)}$, and the verifier checks the verification equations by performing computations over an elliptic curve.

A major task of the proof generation algorithm `Prove` and the verification algorithm `Verify` is to encode the public key with the witness vector $(\vec{x}, \vec{w})$, which is essentially an MSM instance with the witness vector serving as scalars. Based on the proof generation process, it is easy to see that the MSM computation in the context of ZK-SNARKs has two security features: **(i)** The elliptic curve points are public information and usually fixed; and **(ii)** The scalars are related to the secret of the statement and need to be hidden.

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. Similar to other outsourcing computation tasks, MSM outsourcing computation involves two players, and we denote them as *prover* ($\mathcal{P}$) and *service provider* ($\mathcal{S}$): **(i)** $\mathcal{P}$ is the actual entity involved in the ZKP protocol for proof generation. Since $\mathcal{P}$ is the owner of the MSM instance (all inputs) and the end user of the computation result, it does not have motivation to deviate from the protocol and is fully trusted. **(ii)** $\mathcal{S}$ plays the role of computation service provider, who has more computation power and utilizes its resources to help $\mathcal{P}$ to handle computationally intensive tasks involved in proof generation. Here we only consider the MSM computation. $\mathcal{S}$ is not fully trusted, and it may deviate from the protocol to gain some benefits, such as learning the secret information associated with the computation task or returning an incorrect result.

The two players interact with each other through the following steps: **(i)** *Preparing and sending data.* $\mathcal{P}$ prepares the input data of the MSM instance (i.e., a set of integers and elliptic curve points) and sends them to $\mathcal{S}$ to compute; **(ii)** *Processing data and returning result.* $\mathcal{S}$ processes received data according to the protocol, and returns the computation result to $\mathcal{P}$; and **(iii)** *Postprocessing.* After receiving the result from $\mathcal{S}$, $\mathcal{P}$ further processes it to obtain the final MSM result.

Requirements of secure MSM outsource computation. There are two essential requirements for secure MSM outsourcing computation:

- *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.

– *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 or even refuse the service request.

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 [12] proof generation involves 3 MSMs, so it is hard to tell which of these 3 computation tasks are 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.
k_i	The i-th scalar 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.
$\boldsymbol{k}$	The scalar vector $(k_1, k_2, \ldots, k_n)$.
P_i	The i-th elliptic curve point in the MSM computation, $P_i \in EC$.
$\boldsymbol{P}$	The elliptic curve point vector $\{P_1, P_2, \ldots, P_n\}$ of a given MSM task.
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 that outlines the way $\mathcal{P}$ and $\mathcal{S}$ interact their tasks and then describe each step in detail in Section 3.3, including the notations used in Algorithm 1.

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 $\boldsymbol{P}, \boldsymbol{k}$* ◁
$(M_{\boldsymbol{P}}, R) \leftarrow$ `PointPreProcessing`$(\boldsymbol{P})$ ▷ *This function only needs to be done once.*
$(M_{\boldsymbol{k}}, \boldsymbol{t}) \leftarrow$ `ScalarPreProcessing`$(\boldsymbol{k}, R)$
return $(M_{\boldsymbol{P}}, M_{\boldsymbol{k}})$ to $\mathcal{S}$

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

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

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 points vector $\boldsymbol{P}$ associated with the MSM instance to a matrix of elliptic curve points. Specifically, the *prover* generates a matrix of random integers $R = \{r_{i,j}\} \in \mathbb{Z}_{ord}^{\ell \times n}$, and for each row of R, the *prover* computes a Hadamard product with $\boldsymbol{P}$. We denote the resulting matrix as $M_{\boldsymbol{P}}$, and

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

Note that the computation of $M_{\boldsymbol{P}}$ is done locally by $\mathcal{P}$, and $\mathcal{S}$ does not know the random numbers in R. Furthermore, the computation of $M_{\boldsymbol{P}}$ only needs to be done once and is used for multiple MSM instance computations.

– *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 $\mathcal{S}$:

$$M_{\boldsymbol{k}} = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{\ell,1} & k_{\ell,2} & \cdots & k_{\ell,n} \end{pmatrix}. \tag{3}$$

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

$$(k_{i,1}, k_{i,2}, \cdots, k_{i,n}) \leftarrow t_i(k_1/r_{i,1}, k_2/r_{i,2}, \ldots, k_n/r_{i,n}), \quad i = 1, \ldots, \ell. \tag{4}$$

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

After generating the matrices $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{k}}$, $\mathcal{P}$ sends both of them to $\mathcal{S}$. Note that the matrix $M_{\boldsymbol{P}}$ only needs 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 $M_{\boldsymbol{P}}$ and $M_{\boldsymbol{k}}$ from $\mathcal{P}$, and computes $M_{\boldsymbol{P}} \star M_{\boldsymbol{k}}$ as given in Equation (5).

$$M_{\boldsymbol{P}} \star M_{\boldsymbol{k}} = \begin{pmatrix} \sum_{j=1}^{n} k_{1,j}(r_{1,j}P_j) \\ \sum_{j=1}^{n} k_{2,j}(r_{2,j}P_j) \\ \vdots \\ \sum_{j=1}^{n} k_{\ell,j}(r_{\ell,j}P_j) \end{pmatrix} = (Q_1, Q_2, \ldots, Q_\ell)^T. \tag{5}$$

The operator $\star$ denotes a row-wise inner product between two matrices. Although the ℓ rows of the matrix $M_{\boldsymbol{P}}$ 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 $(Q_1, Q_2, \ldots, Q_\ell)^T$ must be computed independently, each representing a distinct MSM instance.

$\mathcal{S}$ can use any existing MSM computation algorithm to compute the ℓ MSM instances, or different algorithms for different MSM instances. The only difference is that when a precomputation-based MSM algorithm (e.g., [7, 23]) is used, $\mathcal{S}$ needs to maintain ℓ different precomputation tables. The computation result (i.e., ℓ 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 ℓ elliptic curve points, i.e., $(Q_1, Q_2, \ldots, Q_\ell)^T$ defined in Equation (5). $\mathcal{P}$ then computes the final MSM result S_n as follows:

$$S_n = t_i^{-1}Q_i, \quad i = 1, \ldots, \ell, \tag{6}$$

where t_i is the random integer chosen by $\mathcal{P}$ and used in the data preprocessing step. When $\mathcal{S}$ is honest and finishes the computation correctly, $t_i^{-1}Q_i == t_j^{-1}Q_j$ for any $i, j \in [1, \ell]$. Otherwise, $\mathcal{P}$ rejects the results.

4 Analysis of the MSM Outsource Computing Protocol

4.1 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.2.

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 is that a malicious $\mathcal{S}$ cannot cause $\mathcal{P}$ to accept an incorrect MSM result $S' \neq S_n$ with non-negligible probability, where the verification is efficient for $\mathcal{P}$. The formal definition of soundness and corresponding proof are given in 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 $k = (k_1, ..., k_n)$ 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 $(M_{\boldsymbol{P}}, M_{\boldsymbol{k}})$, 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 $(M_{\boldsymbol{P}}, M_{\boldsymbol{k}})$. The formal definition of privacy and corresponding proof are given in Appendix A.4.

Safety of reuse. In practical deployments, the prover $\mathcal{P}$ may reuse the point matrix $\mathbf{M}_P$ across multiple MSM computations with the same point vector $\boldsymbol{P}$ for different scalar vectors. Here we prove the security of Π in such cases. The formal security analysis is given in Appendix A.5.

4.2 Side Channel Leakage Analysis

In practice, side channel attacks is 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:

- *Elliptic curve points preprocessing.* This operation consists of group of scalar multiplications, and an adversary can utilize existing side channel attacks [14, 25] to obtain information regarding the matrix of random integers R.
- *Scalar preprocessing.* This operation consists of a group of modular multiplications, which also face different side channel attacks [20, 13] 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., [29, 13]) 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 [13] introduced extra 8.5% overhead for 512 bits modulus. This will only slightly increase 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 $M_{\boldsymbol{P}}$, transmission of $M_{\boldsymbol{P}}$ to the third-party service provider $\mathcal{S}$, and generation of the MSM precomputation table associated with $M_{\boldsymbol{P}}$ 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.

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

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

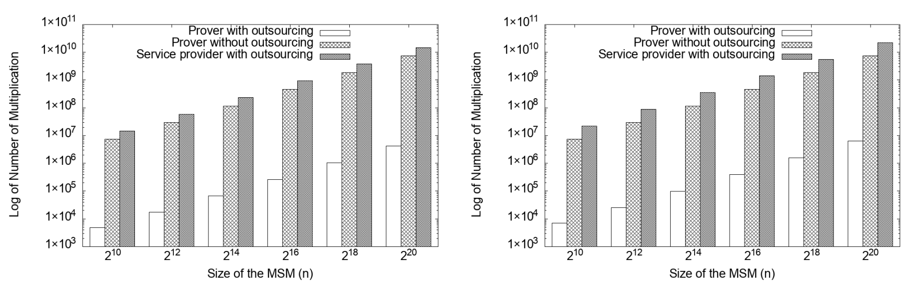
[a] Scalars and integers in this table belong to $\mathbb{Z}_{ord}$.
[b] Points mean elliptic curve points.
[c] We only consider the amount of data sent by a party.

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} [7].

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

For simplicity, we consider a simple MSM computation algorithm, where each scalar multiplication $k_i P_i$ in S_n (as defined in Equation (1)) 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 $\big[2 \cdot (\lfloor \log_2 ord \rfloor - 1) \cdot n + (n-1)\big] \cdot \mathtt{A} \approx 2n \log_2 ord \cdot \mathtt{A}$. **(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 $(\ell \cdot \log_2 p + 14\ell) \cdot \mathtt{M}$ for verification.

Taking into account the probability that a malicious $\mathcal{S}$ returns a wrong MSM result but passes the verification (Section 4.1), 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 $n = 2^k$ $(10 \le k \le 20)$. Compared with doing it locally, the computation cost on the side of $\mathcal{P}$ using our method is almost negligible. Specifically, our method is $1.50K \sim 1.78K$ times and $1.05K \sim 1.19K$ 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.

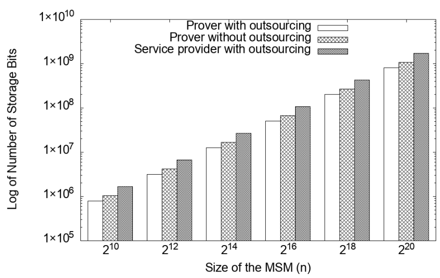


(a) Numbers of multiplications for $\ell = 2$ (b) Numbers of multiplications for $\ell = 3$

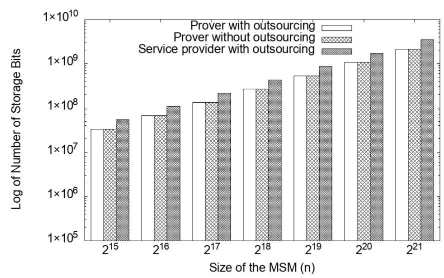
Fig. 1: Comparisons of computation cost.

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)** $n \times \ell + \ell$ integers corresponding to the random matrix $R \in \mathbb{Z}_{ord}^{\ell \times n}$ and random integer vector $\boldsymbol{t} = (t_1, t_2, \ldots, t_l)$ where $t_i \in \mathbb{Z}_{ord}$ and $1 \leq i \leq \ell$. The service provider $\mathcal{S}$ needs to store: **(i)** $n \times \ell$ elliptic curve points (matrix $M_{\boldsymbol{P}}$); 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 $\ell = 2$

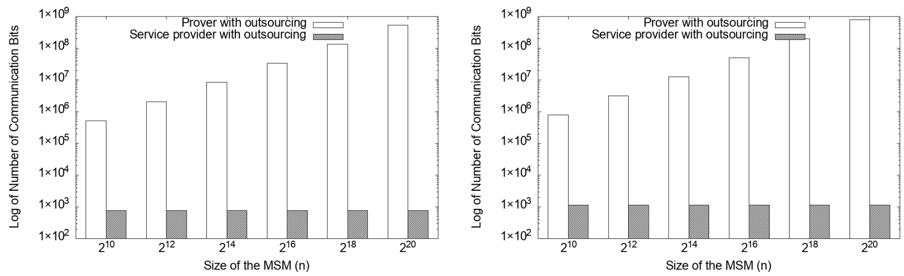


(b) Different storage costs for $\ell = 3$

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 ℓ elliptic curve points $(Q_1, Q_2, \ldots, Q_\ell)^T$ 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 [10, 17], 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 Hyerplonk++ [17]. 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 0.45%–1.75% and 0.69%–2.77% 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.

Experimental performance comparison with other collaborative proof generation methods. Unlike collaborative proof-generation schemes zkSaaS [10]

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

n	ℓ	$\mathcal{P}$ (w/o outsourcing)	$\mathcal{P}$ (outsourcing) Preparation	$\mathcal{P}$ (outsourcing) Verification	$\mathcal{S}$ (outsourcing)
2^{10}	2	281 ms	0.593 ms	4.322 ms	561 ms
	3	280 ms	0.892 ms	6.874 ms	841 ms
2^{12}	2	905 ms	2.373 ms	4.324 ms	1.810 s
	3	906 ms	3.604 ms	6.874 ms	2.716 s
2^{14}	2	3.115 s	9.626 ms	4.321 ms	6.231 s
	3	3.102 s	14.439 ms	6.873 ms	9.306 s
2^{16}	2	10.388 s	37.977 ms	4.323 ms	20.775 s
	3	10.380 s	56.649 ms	6.875 ms	31.140 s
2^{18}	2	38.592 s	153.375 ms	4.325 ms	77.184 s
	3	37.943 s	250.927 ms	6.879 ms	113.828 s
2^{20}	2	134.739 s	603.804 ms	4.326 ms	269.478 s
	3	134.739 s	919.110 ms	6.879 ms	404.217 s

and HyperPlonk++ [17] 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 [10] and [17].

Method		Computation in LAN	Computation in WAN	Communication
zkSaaS [10]	Leader	0.9×	0.1×	90 GB
	Worker	0.9×	0.1×	718 MB
HyperPlonk++ [17]	Leader/Worker	28.0×	25.5×	43 MB
Ours		99.5×[a]		$\mathcal{P}$: 4.28 MB $\mathcal{S}$: 34.5 KB

[a] Since the performance difference between LAN and WAN is negligible for our solution, we record 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 [4]. 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 99.5× 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 34.5 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 [23] and its variants [7, 18]. 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 [15], FPGA [24], and ASIC [16]. 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 [28] 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 [5]. Following this work, several efforts have been made to split the zk-SNARK proof generation workload among multiple participants [22, 10]. As an important component of the proof generation procedure, the traditional MSM computation is also converted to a distributed version. The original goal of [22] was to enable proof generation for a secret shared among multiple participants, and [10, 17] extended the idea to build a service for proof generation.

Unlike the MSM outsourcing approach proposed in this paper, prior works [22, 10] 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 [22, 10] 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 [3, 21]. 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. Aggarwal, T., Kumar, S., Singh, S.K., Gupta, B.B., Nedjah, N., Castiglione, A.: Zero-knowledge proofs and their Applications in Cryptography: Advancements, challenges, and future aspects. In: Innovations in Modern Cryptography, pp. 55–74. IGI Global (2024)

2. 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)
3. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of Crypto 82. pp. 199–203. Springer (1983)
4. 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)
5. 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)
6. Eshan, S., Shirish, A., Lav, U.: The power I know: Zero-Knowledge Proofs and their transformative role in the future of Cryptography. IEEE Access (2025)
7. Fan, X., Kuchta, V., Sica, F., Xu, L.: Speeding up multi-scalar multiplications for pairing-based zkSNARKs. Journal of Cryptology **38**(2), 21 (2025)
8. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 33–62. Springer International Publishing, Cham (2018)
9. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
10. 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)
11. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology **7**(1), 1–32 (1994)
12. 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)
13. 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)
14. 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)
15. 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)
16. 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)
17. 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)
18. 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)

19. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: 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. ACM (2019)
20. 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)
21. Okamoto, T.: An efficient divisible electronic cash scheme. In: Annual international cryptology conference. pp. 438–451. Springer (1995)
22. 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)
23. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
24. 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)
25. 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)
26. 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)
27. Shan, Z., Ren, K., Blanton, M., Wang, C.: Practical secure computation outsourcing: A survey. ACM Computing Surveys (CSUR) **51**(2), 1–40 (2018)
28. 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)
29. Walter, C.D.: Montgomery's multiplication technique: How to make it smaller and faster. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 80–93. Springer (1999)
30. Xi, N., Li, W., Jing, L., Ma, J.: Zama: A zkp-based anonymous mutual authentication scheme for the iov. IEEE Internet of Things Journal **9**(22), 22903–22913 (2022)
31. 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)
32. Yang, Y., Lu, Z., Zeng, J., Liu, X., Qian, X., Yu, Z.: Falic: An fpga-based multi-scalar multiplication accelerator for zero-knowledge proof. IEEE Transactions on Computers **73**(12), 2791–2804 (Dec 2024)
33. 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)
34. 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 Cryptographic Assumptions

Definition 1 (Discrete Logarithm (DL) Problem). *Let* $\mathbb{G}$ *be a cyclic group of prime order ord with generator* P*. Given* $(P,Q) \in \mathbb{G}^2$ *where* $Q = xP$ *for some* $x \xleftarrow{\$} \mathbb{Z}_{ord}^*$*, the DL problem is to compute* x*. The* ***DL advantage*** *of an algorithm* $\mathcal{B}$ *is:*

$$\mathsf{Adv}_{\mathbb{G}}^{DL}(\mathcal{B}) = \Pr[x \xleftarrow{\$} \mathbb{Z}_{ord}^*; Q \leftarrow xP : \mathcal{B}(P,Q) = x]$$

We say the DL assumption holds in $\mathbb{G}$ *if for all PPT algorithms* $\mathcal{B}$*:*

$$\mathsf{Adv}_{\mathbb{G}}^{DL}(\mathcal{B}) \leq \mathsf{negl}(\lambda)$$

where λ *is the security parameter.*

In the Algebraic Group Model [8], we restrict attention to *algebraic adversaries*: any PPT adversary $\mathcal{A}$ that outputs a group element $Q \in \mathbb{G}$ must also output a representation of Q in terms of all group elements it has received as input.

Definition 2 (Algebraic Group Model (AGM)). *If* $\mathcal{A}$ *receives group elements* $\{G_1, \ldots, G_m\} \subset \mathbb{G}$ *and outputs* $Q \in \mathbb{G}$*, it must also output coefficients* $(c_1, \ldots, c_m) \in \mathbb{Z}_{ord}^m$ *such that:*

$$Q = \sum_{i=1}^{m} c_i G_i$$

The AGM captures the fact that in practice, to compute a group element, one must perform group operations on known elements. It is a natural restriction that makes extraction possible while maintaining the hardness of DL.

A.2 Formal Proof of Correctness

Theorem 1. *If both* $\mathcal{P}$ *and* $\mathcal{S}$ *follow the protocol described in Section 3, then* $S_n = t_i^{-1} Q_i$ *for all* $i = 1, \ldots, \ell$ *and* $\mathcal{P}$ *can obtain the correct MSM computation result.*

Proof of Theorem 1. According to Equation (3) and Equation (4), we have:

$$(k_1, k_2, \ldots, k_n) = (\frac{k_{i,1} r_{i,1}}{t_i}, \frac{k_{i,2} r_{i,2}}{t_i}, \ldots, \frac{k_{i,n} r_{i,n}}{t_i}), \quad i = 1, 2, \ldots, \ell. \tag{7}$$

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

$$t_i^{-1} Q_i = \frac{1}{t_i} \sum_{j=1}^{n} k_{i,j} (r_{i,j} P_j) = \sum_{j=1}^{n} \frac{k_{i,j} r_{i,j}}{t_i} P_j = \sum_{j=1}^{n} k_j P_j, \quad i = 1, 2, \ldots, \ell. \tag{8}$$

The last equality holds due to Equation (7), and $\sum_{j=1}^{n} k_j P_j = S_n$ according to the definition of S_n. $\square$

A.3 Formal Proof of Soundness

Theorem 2 (Soundness). *Let $\mathcal{A}$ be a PPT adversary playing the role of a malicious server S^* in protocol Π. Define the soundness advantage of $\mathcal{A}$ as:*

$$\mathsf{Adv}_{\Pi}^{sound}(\mathcal{A}) = \Pr[P \text{ outputs } S' \neq S_n]$$

where the probability is over the random coins of P (sampling $\mathbf{R}$ and $\boldsymbol{t}$) and $\mathcal{A}$. Then under the Discrete Logarithm (DL) assumption in $\mathbb{G}$:

$$\mathsf{Adv}_{\Pi}^{sound}(\mathcal{A}) \leq \frac{\ell(\ell-1)}{2 \cdot ord} + \ell n \cdot \mathsf{Adv}_{\mathbb{G}}^{DL}(\lambda)$$

In practice, for $ord \approx 2^{255}$ (e.g., BLS12-381) and $\ell = 2$, we have $\mathsf{Adv}_{\Pi}^{\text{sound}}(\mathcal{A}) \leq 2^{-255} + 2n \cdot \mathsf{Adv}_{\mathbb{G}}^{\text{DL}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof of Theorem 2. We prove the theorem through a sequence of games.

Game 0: Real Attack

This is the real protocol execution with adversary $\mathcal{A}$ as malicious server S^*.

Algorithm A.2 Game 0 (Real Attack)

```
Input: Security parameter λ, MSM instance (k, P)
P samples R = (r_{i,j}) <-$ (Z*_{ord})^{ℓ×n}
P computes M_P = (r_{i,j} P_j)_{i∈[ℓ], j∈[n]}
P samples t = (t_1, ..., t_ℓ) <-$ (Z*_{ord})^ℓ
P computes k_{i,j} = t_i k_j r_{i,j}^{-1} mod ord for all i, j
P sends (M_P, M_k) to A
A returns Q' = (Q'_1, ..., Q'_ℓ)
P computes S'_i = t_i^{-1} Q'_i for all i ∈ [ℓ]
Compute correct result: S_n = Σ_{j=1}^{n} k_j P_j
if S'_1 = S'_2 = ··· = S'_ℓ and S'_1 ≠ S_n then
    return 1                                   ▷ Adversary succeeds
else
    return 0                                   ▷ Attack fails
```

Let $p_0 = \Pr[\text{Game } 0 \to 1]$. By definition, we have $p_0 = \mathsf{Adv}_{\Pi}^{\text{sound}}(\mathcal{A})$.

Game 1: Eliminating Row Collisions

In this game, we abort if two different rows of $\mathbf{M}_P$ produce the same MSM result when multiplied by their respective rows in $\mathbf{M}_k$. More precisely, we abort if there exist distinct $i_1, i_2 \in [\ell]$ such that:

$$\sum_{j=1}^{n} k_{i_1,j}(r_{i_1,j} P_j) = \sum_{j=1}^{n} k_{i_2,j}(r_{i_2,j} P_j),$$

despite the fact that the rows $(r_{i_1,1}, \ldots, r_{i_1,n})$ and $(r_{i_2,1}, \ldots, r_{i_2,n})$ being independently random.

Lemma 1 (Collision Bound). *Let* $p_1 = \Pr[\textit{Game 1} \to 1]$. *We have* $|p_1 - p_0| \leq \frac{\ell(\ell-1)}{2 \cdot ord}$

Proof of Lemma 1. For any two distinct rows $i_1, i_2 \in [\ell]$, consider the event E_{i_1,i_2} that:

$$\sum_{j=1}^{n} t_{i_1} k_j r_{i_1,j}^{-1} \cdot r_{i_1,j} P_j = \sum_{j=1}^{n} t_{i_2} k_j r_{i_2,j}^{-1} \cdot r_{i_2,j} P_j.$$

Above equation can be simplified:

$$t_{i_1} \sum_{j=1}^{n} k_j P_j = t_{i_2} \sum_{j=1}^{n} k_j P_j \Rightarrow t_{i_1} S_n = t_{i_2} S_n.$$

Since $S_n \neq 0$ (assuming not all $k_j = 0$, which is trivial case), this requires:

$$t_{i_1} = t_{i_2}.$$

But t_{i_1} and t_{i_2} are sampled independently and uniformly from $\mathbb{Z}_{ord}^*$. Therefore:

$$\Pr[E_{i_1,i_2}] = \Pr[t_{i_1} = t_{i_2}] = \frac{1}{ord}.$$

By union bound over all $\binom{\ell}{2}$ pairs:

$$\Pr[\text{Collision occurs}] \leq \binom{\ell}{2} \cdot \frac{1}{ord}.$$

Since Game 0 and Game 1 differ only when a collision occurs:

$$|p_1 - p_0| \leq \frac{\ell(\ell-1)}{2 \cdot ord}.$$

$\square$

Game 2: DL Reduction with AGM Extraction

Now we construct a reduction that uses an algebraic adversary succeeding in Game 1 to solve the DL problem.

Lemma 2 (DL Reduction in AGM). *There exists a PPT algorithm* $\mathcal{B}$ *such that:*

$$p_1 \leq \ell n \cdot \mathsf{Adv}_{\mathbb{G}}^{DL}(\mathcal{B}).$$

Proof of Lemma 2. We construct a reduction $\mathcal{B}$ that uses algebraic adversary $\mathcal{A}$ to solve the DL problem as shown in Algorithm A.3.

Algorithm A.3 DL Reduction $\mathcal{B}(P, H)$ using algebraic adversary $\mathcal{A}$

Input: DL challenge $(P, H = x^*P)$ where $x^* \in \mathbb{Z}^*_{ord}$ unknown
Output: x^* or $\perp$

▷ *Guess where to embed challenge* ◁
$i^* \xleftarrow{\$} [\ell]$, $j^* \xleftarrow{\$} [n]$

▷ *Setup blinding matrix* $\mathbf{M}_P$ ◁
for $i = 1$ to ℓ, $j = 1$ to n **do**
  Sample $\tilde{r}_{i,j} \xleftarrow{\$} \mathbb{Z}^*_{ord}$
  if $(i, j) = (i^*, j^*)$ **then**
    $M_P[i^*, j^*] \leftarrow H$
  else
    $M_P[i, j] \leftarrow \tilde{r}_{i,j} \cdot P_j$
    $r_{i,j} \leftarrow \tilde{r}_{i,j}$

▷ *Setup scalar matrix* $\mathbf{M}_k$ ◁
Sample $\boldsymbol{t} = (t_1, \ldots, t_\ell) \xleftarrow{\$} (\mathbb{Z}^*_{ord})^\ell$
Sample $\alpha \xleftarrow{\$} \mathbb{Z}^*_{ord}$
for $i = 1$ to ℓ, $j = 1$ to n **do**
  if $(i, j) = (i^*, j^*)$ **then**
    $k_{i^*,j^*} \leftarrow \alpha$
  else
    $k_{i,j} \leftarrow t_i \cdot k_j \cdot \tilde{r}^{-1}_{i,j} \bmod ord$

Send $(\mathbf{M}_P, \mathbf{M}_k)$ to $\mathcal{A}$

▷ *$\mathcal{A}$ is algebraic* ◁
Receive $\boldsymbol{Q'} = (Q'_1, \ldots, Q'_\ell)$ and $(c_{i,1}, \ldots, c_{i,m})$ for each Q'_i from $\mathcal{A}$

▷ *Verification check* ◁
for $i = 1$ to ℓ **do**
  $S'_i \leftarrow t_i^{-1} Q'_i$
Compute correct result $S_n = \sum_{j=1}^{n} k_j P_j$
if not $(S'_1 = \cdots = S'_\ell$ and $S'_1 \neq S_n)$ **then**
  return $\perp$

▷ *Extraction using AGM* ◁
Let $S' = S'_1$ and β be the coefficient of H in the representation of Q'_{i^*}

▷ *From the correct computation:* ◁
$Q^{\text{correct}}_{i^*} = t_{i^*} S_n + (\alpha x^* - t_{i^*} k_{j^*}) P$

▷ *But verification gives: $Q'_{i^*} = t_{i^*} S'$* ◁
▷ *The algebraic representation shows the coefficient of $H = x^*P$ is β* ◁
▷ *From consistency: $\beta = \alpha x^* - t_{i^*} k_{j^*}$. So* ◁
$x^* \leftarrow \alpha^{-1}(\beta + t_{i^*} k_{j^*}) \bmod ord$

return x^*

The key idea is that in the AGM, when $\mathcal{A}$ outputs Q'_{i^*}, it provides coefficients showing how Q'_{i^*} was computed from all input group elements. Since $H = x^* P$ appears in $M_P[i^*, j^*]$, and $\mathcal{A}$ is algebraic, the coefficient β of H in the representation directly gives us the relationship needed to extract x^*. More specifically, the honest computation would give contribution $\alpha H = \alpha x^* P$ from the embedded position and the adversary's algebraic representation shows actual coefficient β for H. So we can obtain $x^* = \alpha^{-1}(\beta + t_{i^*} k_{j^*})$ from the verification equations and consistency requirement.

The reduction in Algorithm succeeds if $\mathcal{A}$ succeeds in Game 1 and the guess (i^*, j^*) is correct. Hence we have

$$\mathsf{Adv}_{\mathbb{G}}^{\mathrm{DL}}(\mathcal{B}) \geq \frac{p_1}{\ell n}$$

□

From Lemmas 1 and 2, we obtain

$$\mathsf{Adv}_{\Pi}^{\mathrm{sound}}(\mathcal{A}) = p_0 \leq \frac{\ell(\ell-1)}{2 \cdot ord} + p_1 \leq \frac{\ell(\ell-1)}{2 \cdot ord} + \ell n \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathrm{DL}}(\mathcal{B})$$

Note that the above reduction has a loss factor of ℓn, where ℓ is small (typically 2 or 3) and n can be large (i.e., 2^{20} or more). However, the DL assumption has a substantial security margin. For practical parameters, the security level of the protocol Π remains well above 100 bits.

□

A.4 Proof of Perfect Privacy

Theorem 3 (Perfect Privacy). *The protocol Π realizes the ideal functionality $\mathcal{F}_{\textit{MSM-Outsource}}$ with perfect (information-theoretic) security against any malicious server S^* (even computationally unbounded). Formally, for any adversary $\mathcal{A}$ (not necessarily PPT), there exists a simulator* Sim *such that for all scalar vectors $\boldsymbol{k}, \boldsymbol{k}' \in \mathbb{Z}_{ord}^n$ and point vectors $\boldsymbol{P} \in \mathbb{G}^n$:*

$$\mathsf{View}_{\mathcal{A}}^{real}(\boldsymbol{k}, \boldsymbol{P}) \equiv \mathsf{View}_{\mathsf{Sim}}^{ideal}(\boldsymbol{k}', \boldsymbol{P})$$

where $\equiv$ denotes identical distribution (statistical distance zero), and the view consists of all messages received from P: $\mathsf{View} = (\mathbf{M}_P, \mathbf{M}_k)$.

Proof of Theorem 3. We construct a PPT simulator Sim that generates a view indistinguishable from the real protocol without knowledge of $\boldsymbol{k}$ as shown in Algorithm A.4.

Algorithm A.4 Simulator $\mathsf{Sim}(1^\lambda, n, \boldsymbol{P}, \mathbb{G}, ord, \ell)$

Input: Security parameter λ, public parameters $(n, \boldsymbol{P}, \mathbb{G}, ord, \ell)$
No access to: Secret scalars $\boldsymbol{k}$
Output: Simulated view $(\mathbf{M}_P^{\text{sim}}, \mathbf{M}_k^{\text{sim}})$

▷ *Simulate point matrix* ◁
for $i = 1$ to ℓ, $j = 1$ to n **do**
  Sample $r_{i,j}^{\text{sim}} \xleftarrow{\$} \mathbb{Z}_{ord}^*$
  $M_P^{\text{sim}}[i,j] \leftarrow r_{i,j}^{\text{sim}} \cdot P_j$

▷ *Simulate scalar matrix* ◁
for $i = 1$ to ℓ, $j = 1$ to n **do**
  Sample $k_{i,j}^{\text{sim}} \xleftarrow{\$} \mathbb{Z}_{ord}^*$
  $M_k^{\text{sim}}[i,j] \leftarrow k_{i,j}^{\text{sim}}$

return $(\mathbf{M}_P^{\text{sim}}, \mathbf{M}_k^{\text{sim}})$

We now prove that the simulated view has identical distribution to the real view. In the real protocol execution with input $(\boldsymbol{k}, \boldsymbol{P})$, we have

$$M_P^{\text{real}}[i,j] = r_{i,j} \cdot P_j, \quad r_{i,j} \xleftarrow{\$} \mathbb{Z}_{ord}^*$$
$$M_k^{\text{real}}[i,j] = t_i \cdot k_j \cdot r_{i,j}^{-1}, \quad t_i \xleftarrow{\$} \mathbb{Z}_{ord}^*,$$

whereas in the ideal world, we have

$$M_P^{\text{sim}}[i,j] = r_{i,j}^{\text{sim}} \cdot P_j, \quad r_{i,j}^{\text{sim}} \xleftarrow{\$} \mathbb{Z}_{ord}^*$$
$$M_k^{\text{sim}}[i,j] \xleftarrow{\$} \mathbb{Z}_{ord}^*$$

We can prove the following three lemmas.

Lemma 3 (Point Matrix Equivalence). *For all $i \in [\ell], j \in [n]$:*

$$M_P^{real}[i,j] \equiv M_P^{sim}[i,j]$$

Proof of Lemma 3. Both $M_P^{\text{real}}[i,j]$ and $M_P^{\text{sim}}[i,j]$ are computed as:

$$r \cdot P_j$$

where $r \xleftarrow{\$} \mathbb{Z}_{ord}^*$ is uniformly random and independent.

Since the randomness is sampled identically in both cases, and the point P_j is fixed:

$$M_P^{\text{real}}[i,j] \equiv M_P^{\text{sim}}[i,j]$$

Furthermore, all entries are independent, so:

$$\mathbf{M}_P^{\text{real}} \equiv \mathbf{M}_P^{\text{sim}}$$

□

Lemma 4 (Scalar Matrix Equivalence). *For all $i \in [\ell], j \in [n]$ and any fixed $k_j \in \mathbb{Z}^*_{ord}$:*
$$M_k^{real}[i,j] \equiv M_k^{sim}[i,j]$$

Proof of Lemma 4. For any fixed $i \in [\ell], j \in [n]$ and $k_j \in \mathbb{Z}^*_{ord}$, in the real protocol, we have
$$M_k^{\mathrm{real}}[i,j] = t_i \cdot k_j \cdot r_{i,j}^{-1} \bmod ord,$$

where $t_i, r_{i,j} \xleftarrow{\$} \mathbb{Z}^*_{ord}$ are independent and uniform. Consider the function:

$$f : \mathbb{Z}^*_{ord} \times \mathbb{Z}^*_{ord} \to \mathbb{Z}^*_{ord}, \quad f(t,r) = t \cdot k_j \cdot r^{-1} \bmod ord.$$

For fixed $k_j \neq 0$ and $r \in \mathbb{Z}^*_{ord}$, the map $t \mapsto t \cdot k_j \cdot r^{-1}$ is a bijection from $\mathbb{Z}^*_{ord}$ to $\mathbb{Z}^*_{ord}$. Since t is uniform, the output is uniform as well. More formally, for any $y \in \mathbb{Z}^*_{ord}$, we have

$$\Pr[M_k^{\mathrm{real}}[i,j] = y] = \Pr[t_i \cdot k_j \cdot r_{i,j}^{-1} = y] = \Pr[t_i = y \cdot r_{i,j} \cdot k_j^{-1}].$$

For any fixed value of $r_{i,j}$, the value $y \cdot r_{i,j} \cdot k_j^{-1}$ is some element in $\mathbb{Z}^*_{ord}$, and since t_i is uniform:
$$\Pr[t_i = y \cdot r_{i,j} \cdot k_j^{-1}] = \frac{1}{ord}$$
This holds for all y, so $M_k^{\mathrm{real}}[i,j]$ is uniformly distributed. Since $M_k^{\mathrm{sim}}[i,j]$ is also uniformly distributed in $\mathbb{Z}^*_{ord}$ (by construction):

$$M_k^{\mathrm{real}}[i,j] \equiv M_k^{\mathrm{sim}}[i,j].$$

□

Lemma 5 (Independence). *All entries of $\mathbf{M}_k^{real}$ are mutually independent.*

Proof of Lemma 5. We consider two distinct entries $(i_1, j_1) \neq (i_2, j_2)$. There are the following two cases.

- **Case 1:** $i_1 \neq i_2$ (i.e., different rows). In this case, we have $M_k^{\mathrm{real}}[i_1, j_1] = t_{i_1} \cdot k_{j_1} \cdot r_{i_1,j_1}^{-1}$ and $M_k^{\mathrm{real}}[i_2, j_2] = t_{i_2} \cdot k_{j_2} \cdot r_{i_2,j_2}^{-1}$. Note that $M_k^{\mathrm{real}}[i_1, j_1]$ and $M_k^{\mathrm{real}}[i_2, j_2]$ are computed using independent random variables $t_{i_1}, t_{i_2}, r_{i_1,j_1}, r_{i_2,j_2}$, respectively. Therefore, they are independent.
- **Case 2:** $i_1 = i_2$ but $j_1 \neq j_2$ (i.e., same row, different columns). In this case, we have $M_k^{\mathrm{real}}[i_1, j_1] = t_{i_1} \cdot k_{j_1} \cdot r_{i_1,j_1}^{-1}$ and $M_k^{\mathrm{real}}[i_1, j_2] = t_{i_1} \cdot k_{j_2} \cdot r_{i_1,j_2}^{-1}$. Note that $M_k^{\mathrm{real}}[i_1, j_1]$ and $M_k^{\mathrm{real}}[i_1, j_2]$ are computed using independent r_{i_1,j_1}, r_{i_1,j_2}, respectively. By the uniformity argument (Lemma 4), they are independent.

Hence, all entries of $\mathbf{M}_k^{\mathrm{real}}$ are mutually independent. □

Combining Lemmas 3, 4, and 5, we obtain: **(i)** $\mathbf{M}_P^{\mathrm{real}} \equiv \mathbf{M}_P^{\mathrm{sim}}$ (i.e., identical distribution); **(ii)** Each entry $M_k^{\mathrm{real}}[i,j] \equiv M_k^{\mathrm{sim}}[i,j]$ (i.e., uniform in $\mathbb{Z}^*_{ord}$); and

(iii) All entries are mutually independent in both real and simulated views. Therefore, we have

$$(\mathbf{M}_P^{\text{real}}, \mathbf{M}_k^{\text{real}}) \equiv (\mathbf{M}_P^{\text{sim}}, \mathbf{M}_k^{\text{sim}}).$$

Since the simulator Sim does not use $\boldsymbol{k}$ and the distributions are identical, i.e.,

$$\mathsf{View}_{\mathcal{A}}^{\text{real}}(\boldsymbol{k}, \boldsymbol{P}) \equiv \mathsf{View}_{\mathsf{Sim}}^{\text{ideal}}(\boldsymbol{P}).$$

The view is *completely independent* of the secret scalars $\boldsymbol{k}$.

□

A.5 Proof of Security Under Protocol Reuse

Theorem 4 (Security Under Reuse). *Let $\Pi_1, \Pi_2, \ldots, \Pi_m$ denote m sequential executions of the MSM outsourcing protocol with:*

- *Shared point vector $\boldsymbol{P}$ and point matrix $\mathbf{M}_P$ (computed once);*
- *Distinct scalar vectors $\boldsymbol{k}^{(1)}, \boldsymbol{k}^{(2)}, \ldots, \boldsymbol{k}^{(m)}$;*
- *Fresh randomness $\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, \ldots, \boldsymbol{t}^{(m)}$ for each execution.*

Then the following privacy and soundness claims hold:

- ***Privacy:*** *The joint view of a malicious server across all executions*

$$\mathsf{View} = (\mathbf{M}_P, \mathbf{M}_k^{(1)}, \mathbf{M}_k^{(2)}, \ldots, \mathbf{M}_k^{(m)})$$

reveals no information about any $\boldsymbol{k}^{(j)}$ beyond public parameters (i.e., perfect privacy).

- ***Soundness:*** *For each execution Π_j, the soundness error is*

$$\mathsf{Adv}_{\Pi_j}^{sound}(\mathcal{A}) \leq \frac{\ell(\ell-1)}{2 \cdot ord} + \ell n \cdot \mathsf{Adv}_{\mathbb{G}}^{DL}(\lambda)$$

(i.e., unchanged from single execution).

Proof of Theorem 4. We first prove the privacy claim. In the real world review, the server S^* observes:

$$\begin{aligned}
\mathbf{M}_P &= (r_{i,j} P_j)_{i\in[\ell], j\in[n]} && \text{(fixed, sent once)},\\
\mathbf{M}_k^{(1)} &= (t_i^{(1)} k_j^{(1)} r_{i,j}^{-1})_{i,j} && \text{(execution 1)},\\
\mathbf{M}_k^{(2)} &= (t_i^{(2)} k_j^{(2)} r_{i,j}^{-1})_{i,j} && \text{(execution 2)},\\
&\vdots \\
\mathbf{M}_k^{(m)} &= (t_i^{(m)} k_j^{(m)} r_{i,j}^{-1})_{i,j} && \text{(execution } m\text{)},
\end{aligned}$$

where $\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, \ldots, \boldsymbol{t}^{(m)} \xleftarrow{\$} (\mathbb{Z}_{ord}^*)^\ell$ are freshly sampled for each execution.

For each execution $j \in [m]$ and each entry (i, c), we have

$$M_k^{(j)}[i, c] = t_i^{(j)} \cdot k_c^{(j)} \cdot r_{i,c}^{-1}.$$

Since $t_i^{(j)} \xleftarrow{\$} \mathbb{Z}_{ord}^*$ is fresh and independent, we obtain

– $M_k^{(j)}[i,c]$ is uniformly distributed (by same argument as Theorem 3);
– $M_k^{(j)}[i,c]$ is independent of $k_c^{(j)}$;
– $M_k^{(j)}[i,c]$ is independent of all previous $M_k^{(1)}, \ldots, M_k^{(j-1)}$.

The simulator for multiple executions simply runs the single-execution simulator m times independently as shown in Algorithm A.5.

Algorithm A.5 Multi-Execution Simulator Sim_m

Input: Public parameters $(n, \boldsymbol{P}, \mathbb{G}, ord, \ell)$, number of executions m
No access to: Any $\boldsymbol{k}^{(j)}$

▷ *Simulate shared point matrix (once)* ◁
Sample $\mathbf{M}_P^{\text{sim}}$ as in Theorem 3

▷ *Simulate scalar matrices (independently for each execution)* ◁
for $j = 1$ to m **do**
  for $i = 1$ to ℓ, $c = 1$ to n **do**
    Sample $M_k^{\text{sim},(j)}[i,c] \xleftarrow{\$} \mathbb{Z}_{ord}^*$ // *Independent uniform*

return $(\mathbf{M}_P^{\text{sim}}, \mathbf{M}_k^{\text{sim},(1)}, \ldots, \mathbf{M}_k^{\text{sim},(m)})$

Applying the same reasoning as presented in Theorem 3, it follows that for each execution j, we have

$$M_k^{\text{real},(j)}[i,c] \equiv M_k^{\text{sim},(j)}[i,c] \equiv \text{Uniform}(\mathbb{Z}_{ord}^*)$$

Since the randomness $\boldsymbol{t}^{(j)}$ is fresh and independent across executions, the matrices $\mathbf{M}_k^{(1)}, \ldots, \mathbf{M}_k^{(m)}$ are mutually independent in both real and simulated views. Consequently, we derive

$$(\mathbf{M}_P, \mathbf{M}_k^{(1)}, \ldots, \mathbf{M}_k^{(m)})^{\text{real}} \equiv (\mathbf{M}_P, \mathbf{M}_k^{(1)}, \ldots, \mathbf{M}_k^{(m)})^{\text{sim}}$$

As a result, privacy is perfectly preserved across all m executions.

Next, we prove the soundness claim. For each execution Π_j, the soundness analysis from Theorem 2 applies independently due to: i) Fresh randomness $\boldsymbol{t}^{(j)} \xleftarrow{\$} (\mathbb{Z}_{ord}^*)^\ell$ is sampled for each execution; ii) The verification equations use this fresh randomness; iii) The adversary cannot predict $\boldsymbol{t}^{(j)}$ even knowing $\mathbf{M}_P$ and previous executions; and iv) Each execution's soundness error is independent of all other executions.

As a result, for each execution $j \in [m]$, we have

$$\mathsf{Adv}_{\Pi_j}^{\text{sound}}(\mathcal{A}) \leq \frac{\ell(\ell-1)}{2 \cdot ord} + \ell n \cdot \mathsf{Adv}_{\mathbb{G}}^{\text{DL}}(\lambda)$$

The reuse of $\mathbf{M}_P$ does not help the adversary because the fresh $\boldsymbol{t}^{(j)}$ maintains the randomization needed for soundness.

□