Secure Lookup Tables: Faster, Leaner, and More General

Chongrong Li*, Pengfei Zhu§, Yun Li†(✉) , Zhanpeng Guo†,
Jingyu Li†, Yuncong Hu*(✉) , Zhicong Huang†, Cheng Hong†

**Shanghai Jiao Tong University*, §*Tsinghua University*, †*Ant Group*

chongrongli@sjtu.edu.com, *zhupf321@gmail.com*, *liyun24@antgroup.com*, *zhanp.guo@gmail.com*,
ljy404490@antgroup.com, *huyuncong@sjtu.edu.cn*, *zhicong.hzc@antgroup.com*, *vince.hc@antgroup.com*

***Abstract*—Secure lookup table (LUT) protocols allow retrieving values from a table at secret indices, and have become a promising approach for the secure evaluation of non-linear functions. Most existing LUT protocols target the two-party setting, where the best protocols achieve a communication cost of $O(N)$ for a table of size N. MAESTRO (Morita et al., USENIX Security 2025) represents the state-of-the-art LUT protocol for AES in the three-party honest-majority setting, with a communication cost of $O(N^{1/2})$; malicious security is achieved with distributed zero-knowledge proofs. However, it only supports single-input tables over characteristic-2 fields $\mathbb{F}_{2^k}$ and lacks support for multi-input tables over rings $\mathbb{Z}_{2^k}$, which are more widely used in modern computation. Moreover, the $O(N^{1/2})$ cost remains expensive for large-scale applications; their efficient distributed zero-knowledge proofs are specialized for AES and cannot be easily applied to $\mathbb{Z}_{2^k}$.**

In this work, we present MARLUT, a new generalized and optimized LUT construction supporting multi-input tables over both rings $\mathbb{Z}_{2^k}$ and fields $\mathbb{F}_{2^k}$ with malicious security. We achieve this by (1) extending the semi-honest LUT protocol from MAESTRO, utilizing high-dimensional tensors to reduce its communication cost to $O(N^{1/3})$, and (2) designing a new distributed zero-knowledge proof for inner-product relations over $\mathbb{Z}_{2^k}$. Our distributed zero-knowledge proof is more efficient than the state-of-the-art work (Li et al., CCS 2024) and may be of independent interest. Experiments show that on a table of size 2^{16}, our semi-honest LUT protocol reduces the offline computational and communication cost by a factor of 5.95 and 3.23, respectively. Our distributed zero-knowledge proofs show up to $7.07\times$ and $4.97\times$ speedups over the state-of-the-art protocol on ring $\mathbb{Z}_{2^8}$ and $\mathbb{Z}_{2^{16}}$, respectively.

1. Introduction

Secure Multi-Party Computation (MPC) [1] is a prominent primitive in cryptography. It allows multiple distrustful parties to jointly compute a function on their private inputs and reveal only the output. Through decades of research, MPC has evolved into a powerful tool for protecting data privacy, being applied in Privacy-Preserving Machine Learning (PPML), private databases, and so on. However, many real-world applications require computing non-linear functions (such as activation functions in machine learning, S-box in AES, and floating-point arithmetic), where traditional MPC protocols are generally inefficient as these functions can be rather complicated to express as circuits. Recently, secure Lookup Table (LUT) protocols have become a promising alternative for such non-linear functions. With LUTs, the outputs are obtained by securely performing table lookups at secret indices corresponding to the private inputs, thus avoiding costly circuit evaluations.

Existing secure LUT protocols focus primarily on the two-party setting in the preprocessing model [2], [3], [4], [5], [6][1]. The state-of-the-art two-party LUT protocol FLUTE [5] is based on Additive Secret Sharing (ASS) over $\mathbb{F}_2$. For an n-to-1 table with input bit-width n and output bit-width 1, it achieves $O(1)$ communication cost in the online phase, but its offline communication cost is linear in the table size $N = 2^n$. A recent work MAESTRO [7] explores secure LUT protocols in the three-party setting for the S-box function in AES. It utilizes Replicated Secret Sharing (RSS) [8] over characteristic-2 fields $\mathbb{F}_{2^k}$. In the semi-honest setting, its main protocol achieves the same asymptotic communication complexity as the two-party FLUTE (when $k = 1$); furthermore, it proposes an improved protocol (Prot. 10) with only $O(N^{1/2})$ offline communication and computational costs, while the online communication cost remains $O(1)$. MAESTRO can also achieve malicious security by leveraging Distributed Zero-Knowledge Proofs (DZKPs) [9], [10], with an additional logarithmic communication cost.

Despite significant advancements, MAESTRO still has some limitations. First, as MAESTRO targets AES, it focuses on fields $\mathbb{F}_{2^k}$ (where $k = 1, 4$ or 8); however, many real-world applications involve non-linear functions over rings $\mathbb{Z}_{2^k}$, which are more widely used in modern computation. Second, in MAESTRO, the table is indexed by a *single* element from $\mathbb{F}_{2^k}$, while in practice many functions may take *multiple* elements as input from the underlying domain.[2] Third, although MAESTRO reduces the total communication cost to $O(N^{1/2})$, this is still expensive for massive lookup instances in practice.

1. Such protocols consist of an input-independent preprocessing/offline phase and an input-dependent online phase.

2. For instance, k-bit floating-point arithmetic can be modeled as an LUT which takes as input two elements from $\mathbb{Z}_{2^k}$ and outputs one element.

Furthermore, to achieve malicious security, MAESTRO presents highly specialized verification protocols for AES based on DZKP [9], [10]. However, in $\mathbb{Z}_{2^k}$ rings, these DZKP protocols cannot achieve reasonable soundness without resorting to large Galois rings, which substantially degrades their practical efficiency. The state-of-the-art work [11] mitigates this overhead by lifting the verification from $\mathbb{Z}_{2^k}$ to an expanded ring $\mathbb{Z}_{2^{k+s}}$, enabling significantly faster operations than Galois rings. Nevertheless, due to the limitations of ring structures, achieving 40-bit statistical security requires setting $s \approx 60$, much larger than k (8 or 16) for typical LUT protocols. In particular, the expanded ring exceeds 64 bits and becomes unfriendly to 64-bit CPUs. Experimental results show that operations on such rings are $2 \sim 4\times$ slower than those over $\mathbb{Z}_{2^{64}}$. Overall, the method of [11] remains costly. On the other hand, DZKPs over prime fields can provide comparable security with much smaller field sizes. Moreover, certain prime fields $\mathbb{F}_p$, especially Mersenne fields (where $p = 2^\ell - 1$), support fast modular reduction and offer similar efficiency to finite rings of similar sizes. This motivates us to design a new DZKP protocol that lifts verification over $\mathbb{Z}_{2^k}$ to (Mersenne) prime fields instead of expanded rings.

1.1. Our Contribution

In this work, we propose MARLUT, a novel highly-efficient maliciously secure LUT protocol over rings $\mathbb{Z}_{2^k}$ or characteristic-2 fields $\mathbb{F}_{2^k}$. Our contributions are as follows:

- We propose a generalized semi-honest LUT protocol for multi-input functions over $\mathbb{Z}_{2^k}$ or $\mathbb{F}_{2^k}$, with communication costs as low as $O(N^{1/3})$ where $N = 2^{nk}$. We first extend the LUT protocol of MAESTRO to multi-input tables over $\mathbb{Z}_{2^k}$. Then, by representing the size-N table as a c-dimensional tensor (where $c \geq 2$ is a constant), we achieve $O(c{\cdot}N^{1/c})$ offline communication and computational costs, and $O(N^{(c-2)/c})$ online communication cost. Setting $c = 3$ yields the optimal communication cost of $O(N^{1/3})$.
- We propose an efficient DZKP construction to attain malicious security for our LUT protocol over $\mathbb{Z}_{2^k}$. The core idea is to lift the verification from $\mathbb{Z}_{2^k}$ to Mersenne prime fields rather than expanded rings (as in [11]). In the context of LUT, this allows us to adopt a smaller and faster field while maintaining the same level of soundness. It can also be applied to achieving malicious security for general arithmetic circuits over $\mathbb{Z}_{2^k}$, which may be of independent interest.
- We fully implement MARLUT and conduct comprehensive evaluations. Experiments show our optimized LUT protocol can achieve up to $5.95\times$ speedup. The communication cost is reduced by a factor of 3.23 for size-2^{16} tables, while the online computational efficiency is superior to MAESTRO. Our DZKP protocol can be $7.07\times$ and $4.97\times$ faster than the State-Of-The-Art (SOTA) work [11] over $\mathbb{Z}_{2^8}$ and $\mathbb{Z}_{2^{16}}$, respectively. We also build two real-world applications for evaluating 8-bit floating-point multiplication and 16-bit Sigmoid activation function, which show practical throughput.

1.2. Technical Overview

A Generalized and Optimized LUT Construction. We first briefly review the LUT protocol of MAESTRO. Consider a size-2^k public table T and a secret input $v \in \mathbb{F}_{2^k}$ as the lookup index, held in a secret-shared form by the parties. In MAESTRO, the parties first generate secret-shared correlated randomness $(\boldsymbol{e}^{(r)}, r)$, where $\boldsymbol{e}^{(r)}$ is a size-2^k one-hot vector that has value 1 at index r and value 0 elsewhere. The parties then compute and reconstruct a masked value $m = v + r$, which is used as a public offset to construct a shifted table $\hat{T}$ s.t. $\hat{T}[j] = T[m + j]$ for $j \in \mathbb{F}_{2^k}$. Each party can then obtain a shared lookup result by locally computing shares of the inner-product $\hat{T} \cdot \boldsymbol{e}^{(r)}$ with the public $\hat{T}$ and shared $\boldsymbol{e}^{(r)}$, as $\hat{T} \cdot \boldsymbol{e}^{(r)} = \hat{T}[r] = T[(v + r) + r] = T[v]$ by construction.

We extend the above paradigm to n-input tables over $\mathbb{Z}_{2^k}$. Given a public table T of size $N := 2^{nk}$, the value indexed by the secret-shared input $\boldsymbol{v} \in \mathbb{Z}_{2^k}^n$ is $T[\mathsf{int}_k(\boldsymbol{v})]$ where $\mathsf{int}_k(\boldsymbol{x})$ is a mapping from $\boldsymbol{x} \in \mathbb{Z}_{2^k}^n$ to its corresponding integer representation in the range $[N]$. We now need a random vector $\boldsymbol{r} \in \mathbb{Z}_{2^k}^n$ to mask $\boldsymbol{v}$, and the one-hot vector $\boldsymbol{e}^{(r)}$ is now a length-N vector evaluating to 1 at index $r = \mathsf{int}_k(\boldsymbol{r})$. Correspondingly, the parties reveal a public vector $\boldsymbol{m} = \boldsymbol{v} + \boldsymbol{r}$, and the shifted table $\hat{T}$ is constructed s.t. $\hat{T}[\mathsf{int}_k(\boldsymbol{j})] = T[\mathsf{int}_k(\boldsymbol{m} - \boldsymbol{j})]$ for $\boldsymbol{j} \in \mathbb{Z}_{2^k}^n$. The lookup result then can be obtained by computing $\hat{T} \cdot \boldsymbol{e}^{(r)}$ as above.

The above construction incurs a communication cost linear in the table size N to prepare the length-N vector $\boldsymbol{e}^{(r)}$. Our key insight is that the length-N one-hot vector can be decomposed into c smaller ones $\boldsymbol{e}^{(r_0)}, \cdots, \boldsymbol{e}^{(r_{c-1})}$ each of length $N^{1/c}$, where $c \geq 2$ is an arbitrary constant. Accordingly, we reformulate the shifted table as a c-dimensional tensor $\hat{T}^{(c)}[j_0, \cdots, j_{c-1}]$ where $j_0, \ldots, j_{c-1} \in [N^{1/c}]$ (the superscript over $\hat{T}$ denotes its dimension). To compute the lookup result, we now perform c rounds of tensor contraction, where in each round the dimension of the tensor is reduced by 1: starting from $\hat{T}^{(c)}$, in each round $i \in [c]$ we derive a new tensor $\hat{T}^{(c-i-1)}$ from $\hat{T}^{(c-i)}$ by computing inner products of the smaller one-hot vector $\boldsymbol{e}^{(r_i)}$ and $\{\hat{T}^{(c-i)}[j_i, j_{i+1} \ldots, j_{c-1}]\}_{j_i \in [N^{1/c}]}$ for all j_{i+1}, $\cdots, j_{c-1} \in [N^{1/c}]$; then after c rounds of inner product computations, we can obtain a scalar as the lookup result.

This construction can be readily extended to fields $\mathbb{F}_{2^k}$ (more details in Sec. 4). The offline communication cost is $O(c{\cdot}N^{1/c})$ for the preparation of c size-$N^{1/c}$ smaller one-hot vectors, and the online communication cost is $O(N^{(c-2)/c})$ for computing all the inner products.

An Efficient DZKP Protocol for Inner-Product Relations over $\mathbb{Z}_{2^k}$. To achieve malicious security, we adopt the common paradigm where each party $\mathcal{P}_j$ first performs the semi-honest protocol and then proves that she executed the protocol faithfully. This is further reduced to showing the correctness of certain distributed inner-product triples involved in the semi-honest phase, where $\mathcal{P}_j$ (the prover) knows all the values in these triples while the other two parties (the verifiers) hold sharings of them. On a high level, our DZKP protocol has three steps: (1) batching the triples in

$\mathbb{Z}_{2^k}$ with random linear combinations, (2) lifting the batched triples from $\mathbb{Z}_{2^k}$ to a (sufficiently large) Mersenne prime field $\mathbb{F}_p$, and (3) verifying the new triples in $\mathbb{F}_p$ with well-known techniques [9], [10].

Given n distributed inner-product triples $\{([\![\boldsymbol{a}^{(i)}]\!], [\![\boldsymbol{b}^{(i)}]\!], c^{(i)})\}_{i\in[n]}$ in $\mathbb{Z}_{2^k}$ where $c^{(i)} = \boldsymbol{a}^{(i)} \cdot \boldsymbol{b}^{(i)}$, the first step follows [11], where we batch these triples with random coefficients from $\mathbb{Z}_2$. To achieve reasonable soundness we need to repeat the procedure λ times, where λ is a statistical security parameter. After this step, we obtain λ batched inner-product triples still in $\mathbb{Z}_{2^k}$ $\{([\![\boldsymbol{\mu}^{(j)}]\!], [\![\boldsymbol{\nu}^{(j)}]\!], w^{(j)})\}_{j\in[\lambda]}$.

In the second step, for each triple $([\![\boldsymbol{\mu}]\!], [\![\boldsymbol{\nu}]\!], w)$ (the superscripts are omitted), we lift them onto $\mathbb{F}_p$ by directly viewing all the shares in $\mathbb{Z}_{2^k}$ as ones in $\mathbb{F}_p$. Let $([\![\boldsymbol{\mu}']\!], [\![\boldsymbol{\nu}']\!], [\![w']\!])$ be the lifted triple in $\mathbb{F}_p$. Due to the construction of the semi-honest protocol, $\boldsymbol{\mu}, \boldsymbol{\nu}$ will remain unchanged after the shares are lifted to $\mathbb{F}_p$. But there may be a difference of 2^k between the original w and the lifted w' (more details in Sec. 5.2). Furthermore, the product of any two elements $\boldsymbol{\mu}'[j] \cdot \boldsymbol{\nu}'[j]$ may introduce additional differences in multiples of 2^k. Let $h = \boldsymbol{\mu} \cdot \boldsymbol{\nu}$ be the new inner-product result over $\mathbb{F}_p$. Assuming h doesn't exceed p (which we can ensure by restricting the size of the inner-product triples), there must exist some integer t s.t. $h - w' = t \cdot 2^k$. At first thought, we can ask the prover to compute and share t to the verifiers (as she knows all the original triples); then each verifier can compute $[\![h]\!] = [\![w']\!] + [\![t]\!] \cdot 2^k$. However, the inversion of 2^k exists in $\mathbb{F}_p$, so a malicious prover can eliminate any additive error e in the original triples by adding $-e \cdot (2^k)^{-1}$ to t. To prevent such attacks, we observe that maliciously crafted t need to be large to contain the inverse of 2^k. For correct triples, the range of t is limited if we revise the above equation to $h - w' + 2^{k+1} \equiv_p t \cdot 2^k$. More specifically, given a Mersenne prime field $\mathbb{F}_p$ where $p = 2^\ell - 1$, any attempt to cancel out a non-zero additive error $0 < e < 2^k$ will lead to a crafted t exceeding $\ell - k$ bits, while any honest t cannot exceed this size. Thus, we can detect malicious behaviors with an extra check on the bit length of t. In the last step, we follow well-known techniques from DZKPs [9], [10], [12] to verify the lifted inner-product triples over $\mathbb{F}_p$.

This construction allows us to use fast prime fields to verify distributed relations defined over small rings, and achieves notable speedups over the SOTA work [11].

2. Related Works

2.1. Secure LUT Protocols

Most previous works on secure LUT protocols focus on the two-party setting. Ishai et al. proposed the foundational work OTTT [2], but with substantial offline communication overhead. Dessouky et al. [4] proposed OP-LUT. For a table with n inputs, it reduces the offline communication cost to $O(2^{2n})$ by leveraging oblivious transfers. They also proposed SP-LUT, which improves the offline communication efficiency but at the cost of $O(2^n)$ online communication complexity. The SOTA work FLUTE [5] achieves an offline communication cost of $O(2^n)$ and an $O(1)$ online communication cost, but it's restricted to Boolean inputs and outputs. ROTL [6] further optimizes the computational cost of FLUTE but retains the same $O(2^n)$ offline communication overhead.

Another line of research explores LUT protocols based on function secret sharing (FSS), typically operating in a $\{2+1\}$-party setting with the assistance of a trusted dealer. Pika [13] designed an FSS-based LUT protocol with malicious security, and Sigma [14] adapted such protocols to PPML with improved computational efficiency through GPU acceleration. These FSS-based approaches achieve a low communication cost of $O(n)$, but require $O(2^n)$ invocations of a pseudorandom generator (PRG), which can be a major computational bottleneck.

In the three-party honest-majority setting, MAESTRO [7] explored LUT protocols for AES. They utilize random one-hot vectors and achieve constant online communication cost at the expense of an $O(2^n)$ offline cost. Furthermore, they presented an improved construction that reduces the offline communication cost to $O(2^{n/2})$. They additionally achieved malicious security using DZKPs, which we discuss below.

2.2. Malicious Security from DZKPs

Distributed Zero-knowledge Proofs (DZKPs) originated from the pioneering work [9] of Dan Boneh et al. in 2019. They allow a (single) prover to show the correctness of an NP statement that is distributed or secret-shared among multiple verifiers (so each verifier only knows a part of the statement), making them especially suitable for compiling semi-honest secret-sharing-based MPC protocols into maliciously secure ones. Starting from the seminal works [9], [10], this primitive has gradually become a paradigm for achieving malicious security with RSS-based protocols [9], [10], [11], [12], [15], [16], [17], [18], [19], [20].

2.2.1. DZKPs over Finite Fields. DZKPs typically operate on a sufficiently large finite field to guarantee negligible soundness error [9], [10], [12], [18], [20]. They build upon fully linear proof systems, and can achieve $O(\sqrt{N})$ or $O(\log N)$ communication cost for a circuit of size N. For Boolean circuits in $\mathbb{F}_2$ or arithmetic circuits in small fields, direct application of DZKPs would suffer a large soundness error. A workaround is to lift the relations onto a sufficiently large extension field (such as $\mathbb{F}_{2^{64}}$), and then apply DZKPs over this field. The SOTA three-party LUT protocol MAESTRO [7] follows this route. As MAESTRO primarily targets AES where k is rather small, it is able to perform the DZKP protocol over $\mathbb{F}_{2^{64}}$ with reasonable soundness. To improve efficiency, MAESTRO packs multiple multiplication triples in $\mathbb{F}_2$ into a single multiplication triple in the extension field. However, this relies on the specific structure of the multiplication triples and lacks generality.

Another recent work [18] observed that Mersenne prime fields can be faster than binary extension fields of similar sizes, and proposed to lift the verification of semi-honest AND computations in $\mathbb{F}_2$ to Mersenne prime fields $\mathbb{F}_p$ (e.g., $\mathbb{F}_{2^{61}-1}$). This is done by replacing the binary XOR operation

with an equivalent degree-2 expression in $\mathbb{F}_p$. However, it only applies to Boolean circuits and cannot deal with general inner-product computations over $\mathbb{Z}_{2^k}$ or $\mathbb{F}_{2^k}$ efficiently, as the degree of the resulting equation to be verified over $\mathbb{F}_p$ would scale linearly in the size of the inner-product relation.

2.2.2. DZKP over Rings. Traditional DZKP techniques [9], [10] fail to work over rings $\mathbb{Z}_{2^k}$ as rings lack some essential properties for polynomial interpolation and the Schwartz-Zippel lemma [21]. We can resort to Galois rings since they have the necessary properties. However, as observed in Li et al. [11], operations on Galois rings are quite inefficient, and thus they proposed to lift distributed inner-product relations defined in $\mathbb{Z}_{2^k}$ to a larger ring $\mathbb{Z}_{2^{k+s}}$, and recursively reduce the relations to achieve a $O(\log N)$ communication cost. However, as the structure of rings is not as ideal as that of fields, their protocol would require a larger ring size for a reasonable soundness error. For example, to attain 40-bit security for instance sizes larger than 2^{20} over $\mathbb{Z}_{2^k}$, the expanded ring size has to be roughly 2^{k+60}, while in the field case we only require p to be slightly larger than 2^{40}. Besides, their communication is quadratic in the recursive factor q, while in the field case it is linear in q.

3. Preliminaries

3.1. Notation

Let $\mathcal{P} = \{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2\}$ be a set of three parties. We use $\mathcal{P}_i$ where $i \in \{0, 1, 2\}$ to refer to one of the parties, and $\mathcal{P}_{i\pm1}$ denotes the next $(+)$ or previous $(-)$ party. Let $\mathbb{F}_p$ be the finite field of prime order p. In this work, we primarily focus on Mersenne prime fields, where $p = 2^\ell - 1$. Let $\mathbb{Z}_{2^k}$ be the ring of integers modulo 2^k. When the specific algebraic structure is irrelevant to or self-evident in the context, we use the standard symbol $=$ for equality; otherwise, we use $\equiv_p$ and $\equiv_k$ to denote congruence over $\mathbb{F}_p$ and $\mathbb{Z}_{2^k}$, respectively. We use $:=$ (or $:\equiv_p, :\equiv_k$ accordingly) for assignment.

Let λ be a statistical security parameter, and σ a computational security parameter. We will use $\mathrm{negl}(\cdot)$ to denote a negligible function in the security parameter. A PRG represents a pseudo-random generator with a key $K \in \{0,1\}^\sigma$. We use $[n]$ to denote the set $\{0, \ldots, n-1\}$ for $n \in \mathbb{N}$, and $[a, b]$ to denote $\{a, \ldots, b-1\}$ for $a, b \in \mathbb{Z}$.

Vector, matrix and tensor indices will begin at 0. We denote a vector as $\boldsymbol{x}$ and index it as $\boldsymbol{x}[i]$. The colon notation $\boldsymbol{x}[i:j] = (\boldsymbol{x}[i], \ldots, \boldsymbol{x}[j-1])$ is used to represent a slice of $\boldsymbol{x}$. A public truth table is denoted by T. The concatenation of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is written as $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$. For two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$, we use $\boldsymbol{x} \cdot \boldsymbol{y}$ and $\boldsymbol{x} \circ \boldsymbol{y}$ to denote their inner product and Hadamard product, respectively.

A one-hot vector is written as $\boldsymbol{e}^{(r)}$ where r is the index of the single "1" in the vector, i.e., $\boldsymbol{e}^{(r)}[r] = 1$ and $\boldsymbol{e}^{(r)}[i] = 0$ for $i \neq r$. For $x, y \in \mathbb{F}$, we define $\mathsf{eq}(x, y) = (1-x) \cdot (1-y) + x \cdot y$. Clearly, $x = y$ iff $\mathsf{eq}(x, y) = 1$; further, for any $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$, it holds that $\boldsymbol{x} = \boldsymbol{y}$ iff $\prod_{i\in[n]} \mathsf{eq}(\boldsymbol{x}[i], \boldsymbol{y}[i]) = 1$. For a length-$n$ vector $\boldsymbol{x} \in [2^t]^n$, we define its integer representation as $\mathsf{int}_t(\boldsymbol{x}) := \sum_{i\in[n]} \boldsymbol{x}[i] \cdot 2^{i \cdot t}$.

3.2. Security model

In this work, we follow the standard ideal/real simulation paradigm for security definitions. We focus on the three-party setting with an honest majority, where at most one party can be corrupted. We consider security with abort against a malicious adversary, which allows the adversary to arbitrarily deviate from the protocol specification and instruct the ideal functionality to send abort signals to the honest parties.

3.3. Secret Sharing Schemes

3.3.1. Additive Secret Sharing. In the three-party setting, additive secret sharing (ASS) splits a secret value x into three random values x_0, x_1, x_2 s.t. $x = x_0 + x_1 + x_2$. Each party $\mathcal{P}_i$ for $i \in [3]$ holds an additive share x_i. To reconstruct the secret x to some party $\mathcal{P}_i$, all the other parties reveal their individual share to $\mathcal{P}_i$, who then sums them up to obtain x.

3.3.2. Replicated Secret Sharing. Replicated secret sharing (RSS) [22] is an extension of the ASS scheme. It consists of the following two procedures:

- $\mathsf{Share}(x, D)$ allows a dealer D to distribute a secret x to the parties, such that each party gets a replicated share of x. In the three-party setting, the dealer samples random additive shares x_0, x_1, x_2 s.t. $x = x_0 + x_1 + x_2$; then for each $i \in [3]$, the dealer hands two additive shares (x_{i-1}, x_i) to $\mathcal{P}_i$ as her replicated secret share. We denote this sharing as $[\![x]\!] = (x_0, x_1, x_2)$.
- $\mathsf{Rec}([\![x]\!], \mathcal{P}_i)$ allows the parties to reveal the secret x to some party $\mathcal{P}_i$. To do this, both parties $\mathcal{P}_{i+1}, \mathcal{P}_{i-1}$ send x_{i+1} to $\mathcal{P}_i$, who then checks if the two received values are consistent; if they are, $\mathcal{P}_i$ computes $x := x_0 + x_1 + x_2$; otherwise $\mathcal{P}_i$ aborts. To reveal a secret x to all the parties, we simply run $\mathsf{Rec}([\![x]\!], \mathcal{P}_i)$ for each $i \in [3]$.

We denote as $[\![\cdot]\!]_p, [\![\cdot]\!]_k$ the replicated shares over $\mathbb{F}_p$ and $\mathbb{Z}_{2^k}$, and $[\![\cdot]\!]^{\mathcal{H}}, [\![\cdot]\!]^{\mathcal{C}}$ the replicated shares held by the honest and corrupted parties, respectively.

The RSS scheme has the following properties:

Pairwise Consistency. We say a replicated secret sharing has pairwise consistency if every common additive share held by a pair of two parties is consistent, i.e., for each $i \in [3]$, $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ hold the same additive share x_i. It can be checked by each pair of parties comparing their common shares, and the communication cost can be reduced by only sending and comparing a hash of them [23].

Linearity. The RSS scheme is linearly homomorphic. From two sharings $[\![x]\!], [\![y]\!]$ and any public values a, b, each $\mathcal{P}_i$ can *locally* compute a sharing $[\![z]\!] := [\![ax + by]\!]$ as $(z_{i-1}, z_i) := (ax_{i-1} + by_{i-1}, ax_i + by_i)$. For any value c known by some two parties $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$, they can locally obtain $[\![c]\!]$ by setting $c_i = c$ and $c_{i-1} = c_{i+1} = 0$.

Func. 1: $\mathcal{F}_{\text{Mult}}/\mathcal{F}_{\text{IP}}$ - Multiplication/Inner Product

Let $\mathcal{S}$ be the ideal world adversary.

1) $\mathcal{F}$ receives from honest parties their shares $[\![\boldsymbol{x}]\!]^{\mathcal{H}}$, $[\![\boldsymbol{y}]\!]^{\mathcal{H}}$, and reconstructs $\boldsymbol{x}$, $\boldsymbol{y}$. $\mathcal{F}$ is known as $\mathcal{F}_{\text{Mult}}$ if $\boldsymbol{x}, \boldsymbol{y}$ have length 1, and $\mathcal{F}_{\text{IP}}$ otherwise.
2) $\mathcal{F}$ computes the corrupted parties' shares $[\![\boldsymbol{x}]\!]^{\mathcal{C}}$ and $[\![\boldsymbol{y}]\!]^{\mathcal{C}}$, and sends them to $\mathcal{S}$.
3) $\mathcal{F}$ receives an error e and a set of shares $[\![z]\!]^{\mathcal{C}}$ from $\mathcal{S}$.
4) $\mathcal{F}$ computes $z := \boldsymbol{x} \cdot \boldsymbol{y} + e$ and samples honest parties' shares $[\![z]\!]^{\mathcal{H}}$ using z and $[\![z]\!]^{\mathcal{C}}$.
5) $\mathcal{F}$ sends the shares $[\![z]\!]^{\mathcal{H}}$ to the honest parties.

3.4. Basic Ideal Functionalities

We make use of some basic functionalities listed below, where R can be $\mathbb{Z}_{2^k}, \mathbb{F}_{2^k}$ or $\mathbb{F}_p$.

- $\mathcal{F}_{\text{rand}}(R)$: Samples a random $r \in R$, and distributes a sharing $[\![r]\!]$ to the parties. It can be instantiated non-interactively with a PRG and a shared key setup.
- $\mathcal{F}_{\text{coin}}(R)$: Samples a public random $r \in R$, and sends r to the parties. To instantiate this, the parties can call $\mathcal{F}_{\text{rand}}$ to obtain $[\![r]\!]$ and then open it.
- $\mathcal{F}_{\text{input}}(R)$: For a party $\mathcal{P}_i$ providing as input $x \in R$, the functionality distributes $[\![x]\!]$ to the parties. We can follow well-known techniques [8], [23], [24] to instantiate it.
- $\mathcal{F}_{\text{checkzero}}(\mathbb{F}_p)$: Checks if a shared value x is zero over $\mathbb{F}_p$. To instantiate it, the parties first call $\mathcal{F}_{\text{rand}}(\mathbb{F}_p)$ to obtain a shared randomness $[\![r]\!]$, then compute and open the masked product $[\![x \cdot r]\!]$, and finally check if the product is zero.
- $\mathcal{F}_{\text{bit}}(R)$: Samples a bit $r \in \{0, 1\} \subset R$, and distributes $[\![r]\!]$ to the parties over R. We show its instantiation in Sec. 5.

3.5. Polynomial Interpolation

We use $\mathsf{Poly}(\cdot)$ to denote the polynomial interpolation of a list of elements. More precisely, $\mathsf{Poly}(\{a_0, a_1, \ldots, a_{n-1}\}) = \sum_{i=0}^{n-1} a_i \cdot L_i(x)$, where $L_i(x) = \prod_{j \in [n], j \neq i} \frac{j-x}{j-i}$ is the i-th Lagrange polynomial. We'll only use this definition over large prime fields $\mathbb{F}_p$ with a small degree n, so an interpolation always exists.

3.6. Construction of Maliciously Secure MPC

A general paradigm for constructing a maliciously-secure MPC protocol with a linear secret sharing scheme is to (1) distribute shares of inputs to the parties, (2) evaluate the circuit, where additions and multiplication-by-constant are computed locally and multiplications require a semi-honest protocol, (3) check the correct execution of the multiplication protocol, and finally (4) reconstruct the outputs. We follow this template, and present some building blocks below.

3.6.1. Passive Multiplication Functionality. We define $\mathcal{F}_{\text{Mult}}$ as a functionality which takes two consistent sharings $[\![x]\!], [\![y]\!]$ and outputs $[\![x \cdot y + e]\!]$, where e is some additive error chosen by the adversary. We give a formal definition of $\mathcal{F}_{\text{Mult}}$ in Func. 1, and show its instantiation below.

Func. 2: $\mathcal{F}_{\text{VrfyIP}}$ - Verifying Inner-Product Triples

Let $\mathcal{S}$ be the ideal world adversary.

1) $\mathcal{F}_{\text{VrfyIP}}$ receives from the honest parties the prover's identity j, parameters k, n, and their shares of $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i \in [n]}$. $\mathcal{F}_{\text{VrfyIP}}$ checks whether honest parties' shares satisfy pairwise consistency. If not, $\mathcal{F}_{\text{VrfyIP}}$ sends j and honest parties' shares to $\mathcal{S}$. Then, $\mathcal{F}_{\text{VrfyIP}}$ receives an output from $\mathcal{S}$ for each honest party and hands them to the honest parties as the output of the functionality.
2) Otherwise, $\mathcal{F}_{\text{VrfyIP}}$ reconstructs all the sharings $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i \in [n]}$, and sends the identity j and the shares of corrupted parties to $\mathcal{S}$. In addition, if $\mathcal{P}_j$ is corrupted, $\mathcal{F}_{\text{VrfyIP}}$ also sends the whole sharings $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i \in [n]}$ to $\mathcal{S}$.
3) $\mathcal{F}_{\text{VrfyIP}}$ checks if $c^{(i)} \equiv_k \boldsymbol{a}^{(i)} \cdot \boldsymbol{b}^{(i)}$ holds for all $i \in [n]$.
   - If it doesn't hold for any $i \in [n]$, $\mathcal{F}_{\text{VrfyIP}}$ sends reject to all honest parties and $\mathcal{S}$.
   - Otherwise, $\mathcal{F}_{\text{VrfyIP}}$ receives a command $\mathsf{Out} \in \{\mathsf{accept}, \mathsf{reject}\}$ from $\mathcal{S}$ and sends Out to all honest parties.

Given two input sharings $[\![x]\!]$, $[\![y]\!]$, recall that each party $\mathcal{P}_i$ holds shares (x_{i-1}, x_i) and (y_{i-1}, y_i). $\mathcal{P}_i$ can locally compute an additive share $z_i := x_i \cdot y_i + x_i \cdot y_{i-1} + x_{i-1} \cdot y_i$ of z s.t. $z = z_0 + z_1 + z_2$. To convert the additive sharing into a replicated sharing, each $\mathcal{P}_i$ can pass its z_i to the next party $\mathcal{P}_{i+1}$. To preserve privacy, z_i should be masked with some random additive share of 0 before circulation.

To generate the masking, each party can $\mathcal{P}_i$ sample a random value ρ_i and send it to $\mathcal{P}_{i+1}$. Then each $\mathcal{P}_i$ locally sets $\alpha_i := \rho_i - \rho_{i-1}$; this ensures that $\alpha_0 + \alpha_1 + \alpha_2 = 0$. Now if each party computes

$$z_i := x_i \cdot y_i + x_i \cdot y_{i-1} + x_{i-1} \cdot y_i + \alpha_i, \tag{1}$$

z_i is still an additive share of z, but can now be passed to $\mathcal{P}_{i+1}$ without revealing information about x, y. Following [25], the correlated randomness $\{\alpha_i\}_{i \in [3]}$ can also be generated non-interactively with the help of PRG: each pair of parties $(\mathcal{P}_i, \mathcal{P}_{i+1})$ agree on a shared key K_i and locally generate $\rho_i := \mathsf{PRG}_{K_i}(\mathsf{cnt})$ with a global counter cnt. Then the parties can derive α_i from ρ_i and ρ_{i-1} as before.

3.6.2. Passive Inner-Product Functionality. In addition to multiplication, we also need to compute inner products of two length-n vectors in our protocol. We define the functionality as $\mathcal{F}_{\text{IP}}$, and instantiate it by extending the instantiation of $\mathcal{F}_{\text{Mult}}$ (in fact, $\mathcal{F}_{\text{Mult}}$ is a special case of $\mathcal{F}_{\text{IP}}$ where $n = 1$). Given two length-n shared vectors $[\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!]$ and correlated randomness $\{\alpha_i\}_{i \in [3]}$, each $\mathcal{P}_i$ locally computes

$$z_i := \sum_{j \in [n]} (\boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_i + \boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_{i-1} + \boldsymbol{x}[j]_{i-1} \cdot \boldsymbol{y}[j]_i) + \alpha_i. \tag{2}$$

As z_i is a masked additive share of the inner-product result z, it can be converted into the RSS form $[\![z]\!]$ as above.

3.6.3. Inner-Product Checking Functionality. Following previous works [10], [11], we use a functionality $\mathcal{F}_{\text{VrfyIP}}$ as the main building block for malicious security. It checks the correctness of distributed inner-product triples over $\mathbb{Z}_{2^k}$, which are known by a party $\mathcal{P}_j$ (the prover) and held in secret-shared form by the other parties (the verifiers). The formal description is presented in Func. 2. We'll show in Section 5.1 that we can use this functionality to check the correct executions of $\mathcal{F}_{\text{Mult}}$ and $\mathcal{F}_{\text{IP}}$.

4. A Generalized LUT Construction

In this section, we present a new semi-honest LUT protocol that generalizes and optimizes the LUT construction from MAESTRO [7].

4.1. Recap: LUT Protocol from MAESTRO

We first briefly review MAESTRO, where the public table T is defined over a finite field $\mathbb{F}_{2^k}$ for some k (e.g., $k = 4$ or 8), and indexed by a single field element. Given a secret-shared input $[\![v]\!]$ where $v \in \mathbb{F}_{2^k}$, the parties execute the LUT protocol to obtain a shared lookup result $[\![T[v]]\!]$.

The protocol starts by the parties generating correlated randomness during the offline phase, which consists of a shared random value $[\![r]\!]$ where $r \in \mathbb{F}_{2^k}$ and its corresponding length-2^k one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]$ that evaluates to a single 1 at index r and 0 elsewhere. During the online phase, all parties mask the input $[\![v]\!]$ using $[\![r]\!]$ by computing $[\![m]\!] := [\![v]\!] + [\![r]\!]$. Then they reconstruct m, and locally shift the lookup table T by the offset m to obtain a shifted table $\hat{T}$, defined as $\hat{T}[j] := T[m+j]$ for each $j \in \mathbb{F}_{2^k}$. The shifted table satisfies that $\hat{T}[r] = T[m + r] = T[v + r + r] = T[v]$, as $r + r = 0$ in $\mathbb{F}_{2^k}$. Consequently, the desired lookup result $[\![T[v]]\!]$ (or equivalently $[\![\hat{T}[r]]\!]$) can be retrieved by computing the inner product of the shifted table $\hat{T}$ and the one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]$.

The remaining task is the generation of correlated randomness $([\![r]\!], [\![\boldsymbol{e}^{(r)}]\!])$ in the offline phase. In MAESTRO, $[\![r]\!]$ is obtained by first invoking $\mathcal{F}_{\text{rand}}(\mathbb{F}_2)$ k times to generate shared bits $\{[\![r^{(i)}]\!]\}_{i\in[k]}$, which are then bitwise combined to form $[\![r]\!]$ over $\mathbb{F}_{2^k}$. To derive the corresponding $[\![\boldsymbol{e}^{(r)}]\!]$, recall that $\boldsymbol{e}^{(r)}$ is defined such that $\boldsymbol{e}^{(r)}[j] = 1$ iff $j = r$, and $\boldsymbol{e}^{(r)}[j] = 0$ otherwise. Equivalently,

$$\boldsymbol{e}^{(r)}[j] = \prod_{i\in[k]} \mathsf{eq}(r^{(i)}, \boldsymbol{j}[i]), \forall \boldsymbol{j} \in \{0,1\}^k, \tag{3}$$

where the binary vector $\boldsymbol{j}$ corresponds to the field element $j \in \mathbb{F}_{2^k}$ under the natural correspondence. Using the protocol proposed in [26], the entire vector $\boldsymbol{e}^{(r)}$ can be computed with $2^k - k - 1$ multiplications across $k - 1$ rounds, at a communication cost of $2^k - k - 1$ bits. The protocol starts by setting the initial vector $[\![\boldsymbol{f}^{(0)}]\!] := (1 - [\![r^{(0)}]\!], [\![r^{(0)}]\!])$. For each round $i \in [1, k]$, the parties iteratively compute the intermediate vectors as follows:

$$[\![\boldsymbol{f}^{(i)}]\!] := \left((1 - [\![r^{(i)}]\!]) \cdot [\![\boldsymbol{f}^{(i-1)}]\!], [\![r^{(i)}]\!] \cdot [\![\boldsymbol{f}^{(i-1)}]\!]\right).$$

Func. 3: $\mathcal{F}_{\text{LUT}}$ - Table Lookup

Let $\mathcal{S}$ be the ideal world adversary.

1) Upon receiving from honest parties a parameter n, their input shares $[\![\boldsymbol{v}]\!]_k^{\mathcal{H}} \in \mathbb{Z}_{2^k}^n$ and the public table T, $\mathcal{F}_{\text{LUT}}$ reconstructs $\boldsymbol{v}$ and defines $v = \mathsf{int}_k(\boldsymbol{v})$.
2) $\mathcal{F}_{\text{LUT}}$ computes the corrupted parties' shares $[\![\boldsymbol{v}]\!]_k^{\mathcal{C}}$ and sends them to $\mathcal{S}$.
3) $\mathcal{F}_{\text{LUT}}$ looks up $T[v]$ and receives $[\![T[v]]\!]_k^{\mathcal{C}}$ from $\mathcal{S}$.
4) $\mathcal{F}_{\text{LUT}}$ generate shares $[\![T[v]]\!]_k^{\mathcal{H}}$ from $T[v], [\![T[v]]\!]_k^{\mathcal{C}}$, and distributes them to the honest parties.

In the i-th round, computing $[\![\boldsymbol{f}^{(i)}]\!]$ incurs $2^i - 1$ multiplications between secret-shared values. In practice, we first compute the right half of $[\![\boldsymbol{f}^{(i)}]\!]$ by multiplying $[\![\boldsymbol{f}^{(i)}[j+2^i]]\!]$ with $[\![r^{(i+1)}]\!]$ for all $j \in [2^i - 1]$. The final element of the right half, $[\![\boldsymbol{f}^{(i)}[2^{i+1} - 1]]\!]$, is then computed locally as $[\![r^{(i)}]\!] - \sum_{j\in[2^i-1]} [\![\boldsymbol{f}^{(i)}[j + 2^i]]\!]$, since $\boldsymbol{f}^{(i-1)}$ is a one-hot vector and therefore $r^{(i)} = r^{(i)} \sum_j \boldsymbol{f}^{(i-1)}[j]$. The left half of $[\![\boldsymbol{f}^{(i)}]\!]$ can be locally derived from the right half. After $k - 1$ rounds, the resulting $[\![\boldsymbol{f}^{(k)}]\!]$ is the desired one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]$ corresponding to $[\![r]\!]$.

4.2. Generalizing to Multi-Input Tables over $\mathbb{Z}_{2^k}$

Now we extend the LUT protocol from MAESTRO to support *multi-input* functions defined over *rings* $\mathbb{Z}_{2^k}$. We first define a generalized ideal functionality $\mathcal{F}_{\text{LUT}}$ in Func. 3. Formally, it takes as input the number of inputs to the table n, a shared input vector $[\![\boldsymbol{v}]\!]_k \in \mathbb{Z}_{2^k}^n$, and the public lookup table T of size $N := 2^{n\cdot k}$. The lookup table is represented as a vector and indexed as $T[\mathsf{int}_k(\boldsymbol{j})]$ for $\boldsymbol{j} \in \mathbb{Z}_{2^k}^n$, where $\mathsf{int}_k(\boldsymbol{j})$ maps $\boldsymbol{j}$ to its integer representation in the range $[N]$. Then $\mathcal{F}_{\text{LUT}}$ outputs the lookup result $[\![T[\mathsf{int}_k(\boldsymbol{v})]]\!]_k$.

Basic Construction. Our basic construction adopts the high-level idea of MAESTRO, where the lookup result is derived from the inner product of the shifted table $\hat{T}$ and the one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]_k$. Similarly, all parties first generate correlated randomness during the offline phase. In the multi-input setting, masking the secret input vector $[\![\boldsymbol{v}]\!]_k \in \mathbb{Z}_{2^k}^n$ requires a random vector $[\![\boldsymbol{r}]\!]_k$ of the same length n, and the one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]_k$ should also match the size of T. Thus, the format of the correlated randomness is adapted accordingly, and the correspondence between $\boldsymbol{r}$ and $\boldsymbol{e}^{(r)}$ is established by letting $\boldsymbol{e}^{(r)}$ contain the single 1 at position $r = \mathsf{int}_k(\boldsymbol{r})$.

Then, during the online phase, the parties compute and reconstruct the masked value using the random vector $[\![\boldsymbol{r}]\!]_k$ as $\boldsymbol{m} := \boldsymbol{v} + \boldsymbol{r}$, where the addition is performed element-wise. The shifted table is now defined as $\hat{T}[\mathsf{int}_k(\boldsymbol{j})] = T[\mathsf{int}_k(\boldsymbol{m}-\boldsymbol{j})]$ for all $\boldsymbol{j} \in \mathbb{Z}_{2^k}^n$, which ensures that $\hat{T}[\mathsf{int}_k(\boldsymbol{r})] = T[\mathsf{int}_k(\boldsymbol{m}-\boldsymbol{r})] = T[\mathsf{int}_k(\boldsymbol{v})]$, as desired. We give a formal description of this protocol in App. B.

Correlated Randomness Generation. We now detail how to generate the required randomness $([\![\boldsymbol{r}]\!]_k, [\![\boldsymbol{e}^{(r)}]\!]_k)$. We formally define a functionality $\mathcal{F}_{\text{RandOHV}}$ in Func. 4 to capture

Func. 4: $\mathcal{F}_{\text{RandOHV}}$ - Generating Random One-Hot Vector

Let $\mathcal{S}$ be the ideal world adversary.

1) $\mathcal{F}_{\text{RandOHV}}$ receives $\hat{n}$ and $\hat{k}$ from all parties, and then receives $\{[\![\boldsymbol{r}]\!]_k^{\mathcal{C}}\}$ and $[\![\boldsymbol{e}^{(r)}]\!]_k^{\mathcal{C}}$ as the shares held by the corrupted parties.
2) $\mathcal{F}_{\text{RandOHV}}$ uniformly samples $\boldsymbol{r}$ from $[2^{\hat{k}}]^{\hat{n}}$, then computes the integer $r = \mathsf{int}_{\hat{k}}(\boldsymbol{r})$ and derives the corresponding $\boldsymbol{e}^{(r)}$ of length $2^{\hat{n}\cdot\hat{k}}$.
3) $\mathcal{F}_{\text{RandOHV}}$ generates shares $[\![\boldsymbol{r}]\!]_k^{\mathcal{H}}$ from $\boldsymbol{r}, [\![\boldsymbol{r}]\!]_k^{\mathcal{C}}$.
4) $\mathcal{F}_{\text{RandOHV}}$ generates shares $[\![\boldsymbol{e}^{(r)}]\!]_k^{\mathcal{H}}$ from $\boldsymbol{e}^{(r)}, [\![\boldsymbol{e}^{(r)}]\!]_k^{\mathcal{C}}$.
5) $\mathcal{F}_{\text{RandOHV}}$ distributes $[\![\boldsymbol{r}]\!]_k^{\mathcal{H}}, [\![\boldsymbol{e}^{(r)}]\!]_k^{\mathcal{H}}$ to the honest parties.

this procedure. It takes as input two parameters $(\hat{n}, \hat{k})$[3] and returns two correlated random values: a shared random vector $[\![\boldsymbol{r}]\!]_k$, where $\boldsymbol{r}$ is sampled uniformly from $[2^{\hat{k}}]^{\hat{n}}$, and a shared one-hot vector $[\![\boldsymbol{e}^{(r)}]\!]_k$ of length $2^{\hat{n}\hat{k}}$, where $\boldsymbol{e}^{(r)}$ evaluates to 1 at $r = \mathsf{int}_{\hat{k}}(\boldsymbol{r})$. All the sharings are defined over $\mathbb{Z}_{2^k}$.

To generate the shared random vector $[\![\boldsymbol{r}]\!]_k$, the parties first generate $\hat{n}\cdot\hat{k}$ shared random bits $\{[\![r^{(j)}]\!]_k\}_{j\in[\hat{n}\cdot\hat{k}]}$ by invoking the functionality $\mathcal{F}_{\text{bit}}(\mathbb{Z}_{2^k})$. When $k = 1$, this functionality is equivalent to $\mathcal{F}_{\text{rand}}(\mathbb{Z}_2)$ and can be easily instantiated. For $k > 1$, the functionality can be instantiated with techniques from [27], [28]. Concretely, we let two distinct parties (e.g., $\mathcal{P}_0$ and $\mathcal{P}_1$) sample and share two random bits a and b over $\mathbb{Z}_{2^k}$, respectively. Given the fact that $a \oplus b = a + b - 2ab$ holds for any $a, b \in \{0, 1\}$, the parties can obtain a shared random bit $[\![r]\!]_k$ by computing $[\![r]\!]_k := [\![a]\!]_k + [\![b]\!]_k - 2[\![ab]\!]_k$ over $\mathbb{Z}_{2^k}$.

The parties then locally combine each group of $\hat{k}$ bits into a random element from $[2^{\hat{k}}]$, yielding $\hat{n}$ shared elements $\{[\![\boldsymbol{r}[i]]\!]_k\}_{i\in[\hat{n}]}$ to form the random vector $[\![\boldsymbol{r}]\!]_k$. That is to say, the parties set $[\![\boldsymbol{r}[i]]\!]_k := \sum_{j\in[\hat{k}]}[\![r^{(i\cdot\hat{k}+j)}]\!]_k \cdot 2^j$ for each $i \in [\hat{n}]$. As each $r^{(i)}$ is uniformly distributed over $\{0, 1\}$, $\boldsymbol{r}[i]$ is uniformly distributed over $[2^{\hat{k}}]$. The one-hot vector $\boldsymbol{e}^{(r)}$ is then computed based on Eq. (3) similarly to Sec. 4.1. This procedure incurs $2^{\hat{n}\cdot\hat{k}} - \hat{n}\cdot\hat{k} - 1$ multiplications across $\hat{n}\cdot\hat{k} - 1$ rounds in the offline phase, and requires communicating $2^{\hat{n}\cdot\hat{k}} - \hat{n}\cdot\hat{k} - 1$ elements of $\mathbb{Z}_{2^k}$. The formal description of the protocol is presented in Prot. 1.

4.3. Optimizing Communication Costs via Tensors

While the above basic construction achieves an efficient online phase, it requires an expensive offline setup to generate a full-sized one-hot vector matching the table's size. To mitigate this overhead, we propose an optimization allowing the online lookup to be executed with several sublinear-sized one-hot vectors, and further reduce online computational cost using tensor reformulation. Consequently, we can achieve sub-linear offline costs with comparable online efficiency.

Achieving Square Root Offline Cost with Matrices. We begin with an illustrative example that reduces the offline cost to $O(N^{1/2})$ while incurring only an additional $N^{1/2}$ multiplications during the online phase.

3. The two parameters $\hat{n}, \hat{k}$ are defined to be flexible mainly for later usage. Here one can simply view $\hat{n} = n$ and $\hat{k} = k$.

Prot. 1: Π_{RandOHV} - Generating Random One-Hot Vector

All parties agree on parameters $\hat{n}$ and $\hat{k}$.

1) All parties invoke $\mathcal{F}_{\text{bit}}(\mathbb{Z}_{2^k})$ $\hat{n}\cdot\hat{k}$ times to get shares of random bits $\{[\![r^{(i)}]\!]_k\}_{i\in[\hat{n}\cdot\hat{k}]}$. The parties locally set $[\![\boldsymbol{f}^{(0)}]\!]_k := (1 - [\![r^{(0)}]\!]_k, [\![r^{(0)}]\!]_k)$.
2) For $i \in [1, \hat{n}\cdot\hat{k}]$:
   a) For $j \in [2^i - 1]$, all parties compute $[\![\boldsymbol{f}^{(i)}[j + 2^i]]\!]_k = \mathcal{F}_{\text{mult}}([\![\boldsymbol{f}^{(i-1)}[j]]\!]_k, [\![r^{(i)}]\!]_k)$ and set $[\![\boldsymbol{f}^{(i)}[2^{i+1} - 1]]\!]_k := [\![r^{(i)}]\!]_k - \sum_{j\in[2^i-1]}[\![\boldsymbol{f}^{(i)}[j + 2^i]]\!]_k$.
   b) For $j \in [2^i]$, all parties set $[\![\boldsymbol{f}^{(i)}[j]]\!]_k := [\![\boldsymbol{f}^{(i-1)}[j]]\!]_k - [\![\boldsymbol{f}^{(i)}[j + 2^i]]\!]_k$.
3) All parties set $[\![\boldsymbol{r}[i]]\!]_k := \sum_{j\in[\hat{k}]}[\![r^{(i\cdot\hat{k}+j)}]\!]_k \cdot 2^j, \forall i \in [\hat{n}]$.
4) All parties output $[\![\boldsymbol{r}]\!]_k$ and $[\![\boldsymbol{e}^{(r)}]\!]_k := [\![\boldsymbol{f}^{(\hat{n}\cdot\hat{k}-1)}]\!]_k$.

The construction builds on the key insight that a length-N one-hot vector can be reconstructed from two smaller one-hot vectors, each of length $N^{1/2}$. Without loss of generality, we assume nk is even. Then, following Eq. (3), each entry of $\boldsymbol{e}^{(r)}$ can be expressed as the product of two factors:

$$\boldsymbol{e}^{(r)}[j] = \prod_{i=0}^{nk/2-1} \mathsf{eq}(r^{(i)}, \boldsymbol{j}[i]) \cdot \prod_{i=nk/2}^{nk-1} \mathsf{eq}(r^{(i)}, \boldsymbol{j}[i]), \tag{4}$$

for all $\boldsymbol{j} \in \{0, 1\}^{nk}$ where $j = \mathsf{int}_1(\boldsymbol{j})$. These two factors naturally define two smaller one-hot vectors of length $N^{1/2}$:

$$\boldsymbol{e}^{(r_0)}[j_0] = \prod_{i\in[nk/2]} \mathsf{eq}(r^{(i)}, \boldsymbol{j}_0[i]), \forall \boldsymbol{j}_0 \in \{0, 1\}^{nk/2},$$
$$\boldsymbol{e}^{(r_1)}[j_1] = \prod_{i\in[nk/2]} \mathsf{eq}(r^{(i+nk/2)}, \boldsymbol{j}_1[i]), \forall \boldsymbol{j}_1 \in \{0, 1\}^{nk/2},$$

with $j_0 = \mathsf{int}_1(\boldsymbol{j}_0)$ and $j_1 = \mathsf{int}_1(\boldsymbol{j}_1)$. Conversely, each entry of $\boldsymbol{e}^{(r)}$ can be constructed from the corresponding entries of $\boldsymbol{e}^{(r_0)}$ and $\boldsymbol{e}^{(r_1)}$. Concretely, for all $j_0, j_1 \in [N^{1/2}]$,

$$\begin{aligned} &\boldsymbol{e}^{(r_0)}[j_0] \cdot \boldsymbol{e}^{(r_1)}[j_1] \\ &= \prod_{i\in[nk/2]} \mathsf{eq}(r^{(i)}, \boldsymbol{j}_0[i]) \cdot \prod_{i\in[nk/2]} \mathsf{eq}(r^{(i+nk/2)}, \boldsymbol{j}_1[i]) \\ &= \boldsymbol{e}^{(r)}[\mathsf{int}_1(\boldsymbol{j}_0, \boldsymbol{j}_1)] = \boldsymbol{e}^{(r)}[j_0 + j_1 \cdot N^{1/2}]. \end{aligned} \tag{5}$$

Using this equation, the online lookup can instead be performed with two smaller one-hot vectors as

$$\begin{aligned} \hat{T} \cdot [\![\boldsymbol{e}^{(r)}]\!]_k &= \sum_{j\in[N]} \hat{T}[j] \cdot [\![\boldsymbol{e}^{(r)}[j]]\!]_k \\ &= \sum_{j_0, j_1 \in [N^{1/2}]} \hat{T}[j_0 + j_1 \cdot N^{1/2}] \cdot [\![\boldsymbol{e}^{(r_0)}[j_0]]\!]_k \cdot [\![\boldsymbol{e}^{(r_1)}[j_1]]\!]_k. \end{aligned} \tag{6}$$

This approach ensures a more efficient offline setup, as only two length-$N^{1/2}$ one-hot vectors are required, therefore reducing the offline cost to $O(N^{1/2})$. However, naïvely computing the lookup result according to Eq. (6) incurs at least $2N$ multiplications, doubling the online cost of the basic construction. To eliminate redundant computation, we reformulate the shifted table $\hat{T}$ as a matrix $\hat{T}^{(2)}$ (where the superscript denotes its dimension as a tensor), by setting

Prot. 2: Π_{ExtLUT} - Extended Protocol for Securely Computing Lookup Table

All parties share $[\![\boldsymbol{v}]\!]_k$ as input. All parties agree on the public lookup table T over $\mathbb{Z}_{2^k}$ with n inputs, and all parties agree on constants c_1 and c_2. Let $c = c_1 \cdot c_2$.

1) All parties invoke $\mathcal{F}_{\text{RandOHV}}(n/c_1, k/c_2)$ c times to get
$$([\![\boldsymbol{r}_0]\!]_k, [\![\boldsymbol{e}^{(r_0)}]\!]_k), \ldots, ([\![\boldsymbol{r}_{c-1}]\!]_k, [\![\boldsymbol{e}^{(r_{c-1})}]\!]_k).$$
2) All parties set $[\![\boldsymbol{r}']\!]_k := ([\![\boldsymbol{r}_0]\!]_k, \ldots, [\![\boldsymbol{r}_{c-1}]\!]_k)$ and compute $[\![\boldsymbol{r}[j]]\!]_k := \sum_{i\in[c_2]} [\![\boldsymbol{r}'[i + j \cdot c_2]]\!]_k \cdot 2^{i \cdot k/c_2}, \forall j \in [n]$.
3) All parties reconstruct $\boldsymbol{m} := \mathsf{Rec}([\![\boldsymbol{v}]\!]_k + [\![\boldsymbol{r}]\!]_k)$.
4) Let $\hat{T}[\mathsf{int}(\boldsymbol{j})] := T[\mathsf{int}(\boldsymbol{m} - \boldsymbol{j})], \forall j \in [2^k]^n$ be the shifted table. All parties define a c-dimensional tensor $\hat{T}^{(c)}$ such that for all $j_0, \ldots, j_{c-1} \in [N^{1/c}]$ and $j = \sum_{i=0}^{c-1} j_i \cdot N^{i/c}$, $\hat{T}^{(c)}[j_0, \ldots, j_{c-1}] = \hat{T}[j]$.
5) In the first round, for all $j_1, \ldots, j_{c-1} \in [N^{1/c}]$, all parties locally compute $[\![\hat{T}^{(c-1)}[j_1, \ldots, j_{c-1}]]\!]_k = \sum_{j_0\in[N^{1/c}]} \hat{T}^{(c)}[j_0, j_1, \ldots, j_{c-1}] \cdot [\![\boldsymbol{e}^{(r_0)}[j_0]]\!]_k$.
6) In the i-th round for $i \in \{1, \ldots, c-1\}$, for all $j_{i+1}, \ldots, j_{c-1} \in [N^{1/c}]$, all parties perform
$$[\![\hat{T}^{(c-i-1)}[j_{i+1}, \ldots, j_{c-1}]]\!]_k := \mathcal{F}_{\text{IP}}((\![\hat{T}^{(c-i)}[j_i, j_{i+1}, \ldots, j_{c-1}]]\!]_k)_{j_i\in[N^{1/c}]}, [\![\boldsymbol{e}^{(r_i)}]\!]_k).$$
7) Finally, all parties output $[\![\hat{T}^{(0)}]\!]_k$ as $[\![T[\mathsf{int}_k(\boldsymbol{v})]]\!]_k$.

$\hat{T}^{(2)}[j_0, j_1] := \hat{T}[j_0 + j_1 \cdot N^{1/2}]$ for all $j_0, j_1 \in [N^{1/2}]$, and transform Eq. (6) into a vector-matrix-vector multiplication:

$$\begin{aligned}
&\sum_{j_0,j_1\in[N^{1/2}]} \hat{T}^{(2)}[j_0, j_1] \cdot [\![\boldsymbol{e}^{(r_0)}[j_0]]\!]_k \cdot [\![\boldsymbol{e}^{(r_1)}[j_1]]\!]_k \\
&= \sum_{j_1\in[N^{1/2}]} \Big(\sum_{j_0\in[N^{1/2}]} [\![\boldsymbol{e}^{(r_0)}[j_0]]\!]_k \cdot \hat{T}^{(2)}[j_0, j_1] \Big) \cdot [\![\boldsymbol{e}^{(r_1)}[j_1]]\!]_k \\
&= \sum_{j_1\in[N^{1/2}]} \Big([\![\boldsymbol{e}^{(r_0)}]\!]_k \cdot \hat{T}^{(2)}[:, j_1] \Big) \cdot [\![\boldsymbol{e}^{(r_1)}[j_1]]\!]_k \\
&= [\![\boldsymbol{e}^{(r_0)}]\!]_k \cdot \hat{T}^{(2)} \cdot [\![\boldsymbol{e}^{(r_1)}]\!]_k,
\end{aligned} \tag{7}$$

where the colon notation $\hat{T}^{(2)}[:, j_1]$ denotes the vector $(\hat{T}^{(2)}[0, j_1], \ldots, \hat{T}^{(2)}[N^{1/2} - 1, j_1])$. Under this reformulation, we can compute the lookup result in two rounds, where in each round the dimension of the tensor $\hat{T}^{(2)}$ is reduced by 1 (i.e., vector-matrix multiplication yields a vector which is a one-dimensional tensor, and vector-vector multiplication yields a scalar which is a zero-dimensional tensor). The computational cost is reduced to $N + N^{1/2}$ multiplications.

Generalizing to Arbitrary-Dimensional Tensors. The preceding example demonstrates the potential for developing a generalized construction. By decomposing the full-sized one-hot vector into several smaller vectors, the overall dimensionality is reduced, thereby lowering the offline communication and computational cost. Furthermore, representing the shifted table as a higher-dimensional tensor eliminates redundant operations required to reconstruct the full-sized one-hot vector, thus preserving online efficiency. In the following, we formalize this paradigm and present the generalized construction of our LUT protocol.

We start by decomposing the one-hot vector as c smaller one-hot vectors of length $N^{1/c}$ instead of just 2 smaller ones of length $N^{1/2}$. We denote them as $\{\boldsymbol{e}^{(r_i)}\}_{i\in[c]}$, and the correlated randomness is thus defined as $([\![\boldsymbol{r}]\!]_k, \{[\![\boldsymbol{e}^{(r_i)}]\!]_k\}_{i\in[c]})$. To generate this, we assume without loss of generality that nk is divisible by $c = c_1 \cdot c_2$, where n and k are divisible by c_1, c_2 respectively. All parties then invoke $\mathcal{F}_{\text{RandOHV}}$ c times with parameters $(n/c_1, k/c_2)$ to obtain a series of one-hot vectors $\{[\![\boldsymbol{e}^{(r_i)}]\!]_k\}_{i\in[c]}$ and the corresponding random vectors $\{[\![\boldsymbol{r}_i]\!]_k\}_{i\in[c]}$. Let $\boldsymbol{r}' = (\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{c-1})$ denote the concatenation of all $\boldsymbol{r}_i$. The required $[\![\boldsymbol{r}]\!]_k$ can then be computed as $[\![\boldsymbol{r}[j]]\!]_k := \sum_{i\in[c_2]} [\![\boldsymbol{r}'[i + j \cdot c_2]]\!]_k \cdot 2^{i \cdot k/c_2}, \forall j \in [n]$.

During the online phase, all parties reconstruct the masked value corresponding to the secret inputs as $\boldsymbol{m} := \boldsymbol{v} + \boldsymbol{r}$. Then we define a shifted table as $\hat{T}[\mathsf{int}_k(\boldsymbol{j})] := T[\mathsf{int}_k(\boldsymbol{m} - \boldsymbol{j})]$ for each $\boldsymbol{j} \in \mathbb{Z}_{2^k}^n$.

Now we show how to compute the lookup result utilizing the c smaller-sized one-hot vectors. Let $\boldsymbol{e}^{(r)}$ denote the length-N one-hot vector corresponding to $\boldsymbol{r}$, where $r = \mathsf{int}_k(\boldsymbol{r})$. Since each random vector $\boldsymbol{r}_i$ is chosen from $[2^{k/c_2}]^{n/c_1}$, we represent the associated nk/c random bits as $\{r^{(t+\frac{ink}{c})}\}_{t\in[nk/c]}$. Consequently, each entry of the corresponding one-hot vector $\boldsymbol{e}^{(r_i)}$ can be expressed as

$$\boldsymbol{e}^{(r_i)}[j_i] = \prod_{t\in[nk/c]} \mathsf{eq}(r^{(t+\frac{ink}{c})}, \boldsymbol{j}_i[t]), \forall \boldsymbol{j}_i \in \{0, 1\}^{nk/c},$$

where $j_i = \mathsf{int}_1(\boldsymbol{j}_i)$. This leads to the following identity:

$$\begin{aligned}
&\boldsymbol{e}^{(r_0)}[j_0] \cdots \boldsymbol{e}^{(r_{c-1})}[j_{c-1}] = \prod_{i\in[c]} \prod_{t\in[nk/c]} \mathsf{eq}(r^{(t+\frac{ink}{c})}, \boldsymbol{j}_i[t]) \\
&= \boldsymbol{e}^{(r)}[\mathsf{int}_1(\boldsymbol{j}_0, \ldots, \boldsymbol{j}_{c-1})] = \boldsymbol{e}^{(r)}[\sum_{i\in[c]} j_i \cdot N^{i/c}],
\end{aligned} \tag{8}$$

for all $j_0, \ldots, j_{c-1} \in [N^{1/c}]$. Given this relation, the lookup result $\hat{T} \cdot \boldsymbol{e}^{(r)}$, can be rewritten as

$$\sum_{j_0,\ldots,j_{c-1}\in[N^{1/c}]} \hat{T}[\sum_{i\in[c]} j_i \cdot N^{i/c}] \cdot \boldsymbol{e}^{(r_0)}[j_0] \cdots \boldsymbol{e}^{(r_{c-1})}[j_{c-1}]. \tag{9}$$

A direct computation of Eq. (9) would entail cN multiplications. We thus further reformulate the shifted table $\hat{T}$ into a c-dimensional tensor $\hat{T}^{(c)}$ where

$$\hat{T}^{(c)}[j_0, \ldots, j_{c-1}] := \hat{T}[\sum_{i\in[c]} j_i \cdot N^{i/c}], \forall j_0, \ldots, j_{c-1} \in [N^{1/c}].$$

This allows computing Eq. (9) inductively over c rounds, where in each round, the dimensional is reduced by 1. Starting from $\hat{T}^{(c)}$, in each round $i \in [c]$, the parties use the one-hot vector $\boldsymbol{e}^{(r_i)}$ to contract the first dimension of $[\![\hat{T}^{(c-i)}]\!]_k$ to obtain $[\![\hat{T}^{(c-i-1)}]\!]_k$. Concretely, for all $j_{i+1}, \cdots, j_{c-1} \in [N^{1/c}]$, all parties invoke $\mathcal{F}_{\text{ip}}$ to compute

$$\begin{aligned}
&[\![\hat{T}^{(c-i-1)}[j_{i+1}, \ldots, j_{c-1}]]\!]_k := \\
&\sum_{j_i\in[N^{1/c}]} [\![\boldsymbol{e}^{(r_i)}[j_i]]\!]_k \cdot [\![\hat{T}^{(c-i)}[j_i, j_{i+1}, \ldots, j_{c-1}]]\!]_k,
\end{aligned}$$

except in round 0, where the computation can be performed locally as $\hat{T}^{(c)}$ is public. After c rounds, the parties set the

final scalar $\hat{T}^{(0)}$ as the lookup result.

To see correctness of this construction, note that in each round i, every element indexed by $(j_i, \ldots, j_{c-1})$ is multiplied by $\boldsymbol{e}^{(r_i)}[j_i]$. Consequently, each entry $\hat{T}^{(c)}[j_0, \ldots, j_{c-1}]$ in the original tensor is ultimately multiplied by the product $\boldsymbol{e}^{(r_0)}[j_0] \cdots \boldsymbol{e}^{(r_{c-1})}[j_{c-1}]$. Given Eq. (8), we have

$$\begin{aligned}\hat{T}^{(0)} &= \sum_{j_0,\ldots,j_{c-1}\in[N^{1/c}]} \boldsymbol{e}^{(r_0)}[j_0] \cdots \boldsymbol{e}^{(r_{c-1})}[j_{c-1}] \\ &\quad \cdot \hat{T}^{(c)}[j_0, \ldots, j_{c-1}] \\ &= \sum_{j\in[N]} \boldsymbol{e}^{(r)}[j] \cdot \hat{T}[j] = \boldsymbol{e}^{(r)} \cdot \hat{T} = T[\mathsf{int}_k(\boldsymbol{v})].\end{aligned}$$

We formally present our construction in Prot. 2, and have the following theorem, which we prove in App. B.

Theorem 1. *The protocol Π_{ExtLUT} in Prot. 2 securely computes $\mathcal{F}_{LUT}$ in the $\{\mathcal{F}_{RandOHV}, \mathcal{F}_{ip}\}$-hybrid model in the presence of a semi-honest adversary controlling one corrupted party.*

Complexity Analysis. In the offline phase, all parties invoke $\mathcal{F}_{\text{RandOHV}}(n/c_1, k/c_2)$ a total of c times. When instantiated with Prot. 1, this incurs both communication and computation overheads of $c \cdot (N^{1/c} - \log N^{1/c} - 1)$. During the online phase, in round 0, all parties compute N local multiplications between public table $\hat{T}^{(c)}$ and the secret one-hot vector $[\![\boldsymbol{e}^{(r_0)}]\!]_k$. The dominant cost of the subsequent rounds arises in round 1, where all parties compute $N^{(c-1-1)/c}$ inner products of length $N^{1/c}$ over secret values. This yields a computational cost of $N^{(c-1)/c}$ multiplications on secret values and a communication cost of $N^{(c-2)/c}$ $\mathbb{Z}_{2^k}$ elements.

Extending our Construction to $\mathbb{F}_{2^k}$. In addition to $\mathbb{Z}_{2^k}$, our scheme also supports multi-input tables in $\mathbb{F}_{2^k}$. Intuitively, the natural map between $\mathbb{F}_{2^k}$ and $\mathbb{F}_2^k$ converts an n-input table over $\mathbb{F}_{2^k}$ into an nk-input table over $\mathbb{F}_2 = \mathbb{Z}_2$, which can then be directly dealt with our LUT protocol. Consequently, our protocol can be viewed as a generalization of MAESTRO.

Remark 1 (Comparison with MAESTRO)**.** *Prot. 10 of MAESTRO [7] adopts a similar idea to our illustrative example by replacing the length-N one-hot vector with two one-hot vectors of length $N^{1/2}$. However, the protocol essentially follows the naïve computation of Eq.* (6) *in the online phase, which results in a doubled online overhead.*

Remark 2 (Cases when nk is not divisible by c)**.** *The divisibility assumption is made for presentation clarity. We can readily handle the non-divisible cases by using one-hot vectors of varying lengths. This would only influence the size of the inner-product computation in each round.*

Remark 3 (Multiple tables with identical inputs)**.** *Some applications that may require lookup into multiple tables at the same indices. In this situation, naïvely invoking $\mathcal{F}_{LUT}$ multiple times entails generating multiple sets of one-hot vectors, which is unnecessary. In fact, we only need one set of one-hot vectors and all parties only need to reconstruct one $\boldsymbol{m}$. Then each table is shifted with the same offset.*

5. Achieving Malicious Security

In this section, we present our DZKP protocol for verifying distributed inner-product triples defined over $\mathbb{Z}_{2^k}$, which allows our LUT protocols to attain malicious security.

5.1. From Semi-honest to Malicious Security

Recall that in our LUT protocol, the parties invoke $\mathcal{F}_{\text{Mult}}$ to compute multiplications in the offline phase, and $\mathcal{F}_{\text{IP}}$ to compute inner products in the online phase. To obtain malicious security, it suffices for the parties to verify the correct execution of the semi-honest multiplication and inner-product protocols, since all other operations are local and their correctness is ensured by the replicated secret sharing scheme. Following previous works [7], [9], [10], the honest execution of $\mathcal{F}_{\text{Mult}}$ and $\mathcal{F}_{\text{IP}}$ can be further reduced to the correctness of certain distributed inner-product triples over $\mathbb{Z}_{2^k}$, which can be verified with $\mathcal{F}_{\text{VrfyIP}}$ defined in Func. 2.

We briefly review the construction of these distributed inner-product triples. As discussed in Sec. 3.6, to compute $[\![z]\!]_k = [\![x]\!]_k \cdot [\![y]\!]_k$, each party $\mathcal{P}_i$ computes an additive share z_i according to Eq. (1). If the computation is faithfully conducted, then it must hold that

$$z_i - x_i \cdot y_i - \rho_i + \rho_{i-1} \equiv_k x_i \cdot y_{i-1} + x_{i-1} \cdot y_i. \tag{10}$$

To verify the correct execution of the multiplication protocol, it suffices for each party $\mathcal{P}_i$ to show Eq. (10) indeed holds. A key feature of Eq. (10) is that $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$ each knows part of the values involved, while $\mathcal{P}_i$ knows all of them. Concretely, let w be the LHS of Eq. (10), we set $w_i :\equiv_k z_i - x_i \cdot y_i - \rho_i$ and $w_{i-1} :\equiv_k \rho_{i-1}$ s.t. $w = w_i + w_{i-1}$. It's clear that $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ each know w_i, while $\mathcal{P}_i$ and $\mathcal{P}_{i-1}$ know w_{i-1}. Now we have

$$w \equiv_k x_i \cdot y_{i-1} + x_{i-1} \cdot y_i. \tag{11}$$

Since $\mathcal{P}_{i+1}$ and $\mathcal{P}_{i-1}$ know x_i, y_i and x_{i-1}, y_{i-1} respectively and $\mathcal{P}_i$ knows all of them, it follows that the parties can locally obtain replicated secret sharings of all values in Eq. (11).[4] Let $[\![\boldsymbol{x}]\!]_k = ([\![x_i]\!]_k, [\![x_{i-1}]\!]_k)$, $[\![\boldsymbol{y}]\!]_k = ([\![y_{i-1}]\!]_k, [\![y_i]\!]_k)$. Then the verification of Eq. (10) is reduced to the verification of a length-2 distributed inner-product triple $([\![\boldsymbol{x}]\!]_k, [\![\boldsymbol{y}]\!]_k, [\![w]\!]_k)$, satisfying the form required by $\mathcal{F}_{\text{VrfyIP}}$.

This construction can be easily extended to verifying the inner-product protocol. For a length-n secret inner-product computation, the resulting distributed inner-product triple having size $2n$ instead of 2 (see App. F for more details).

5.2. Our Instantiation of $\mathcal{F}_{\text{VrfyIP}}$

We now present our instantiation of $\mathcal{F}_{\text{VrfyIP}}$. Suppose the prover holds n inner-product triples $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k,$

4. Take x_i for example: $[\![x_i]\!]$ can be locally obtained by $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ setting their shares to $(0, x_i)$ and $(x_i, 0)$ respectively, and $\mathcal{P}_{i-1}$ setting her share to $(0, 0)$. For w, $\mathcal{P}_{i+1}, \mathcal{P}_{i-1}$ set their shares to $(w_i, 0)$ and $(0, w_{i-1})$, while $\mathcal{P}_i$ sets her share to (w_{i-1}, w_i).

$[\![c^{(i)}]\!]_k)\}_{i\in[n]}$. For simplicity of presentation, we assume the triples have the same size d, i.e. $|\boldsymbol{a}^{(i)}| = |\boldsymbol{b}^{(i)}| = d, \forall i \in [n]$.[5]

Step 1: Batching all inner-product triples. Similarly to [11], [12], we first batch the inner-product triples with random linear combinations. Specifically, we combine n triples $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i\in[n]}$ each of size d into a single inner-product triple of size $d \cdot n$ with a length-n random coefficient vector $\boldsymbol{\gamma}$:

$$\sum_{i=0}^{n-1} \boldsymbol{\gamma}[i] \cdot [\![\boldsymbol{a}^{(i)}]\!]_k \cdot [\![\boldsymbol{b}^{(i)}]\!]_k \equiv_k \sum_{i=0}^{n-1} \boldsymbol{\gamma}[i] \cdot [\![c^{(i)}]\!]_k.$$

As established in [11], [29], since the triples are defined over $\mathbb{Z}_{2^k}$, it suffices to choose the random coefficients from $\mathbb{Z}_2$ instead of $\mathbb{Z}_{2^k}$, as they offer the same soundness error of $1/2$.[6] To boost the soundness, we can repeat this procedure λ times, leading to a soundness error of $2^{-\lambda}$. In the computational security setting, the parties can generate these random coefficients by expanding a random seed (received from $\mathcal{F}_{\text{coin}}$) with a PRG.

Let $\{\boldsymbol{\gamma}^{(j)}\}_{j\in[\lambda]}$ be the λ random coefficient vectors each of length n. Now the parties locally derive λ inner-product triples $\{\boldsymbol{\mu}^{(j)}, \boldsymbol{\nu}^{(j)}, w^{(j)}\}_{j\in[\lambda]}$ each of size $d' := n \cdot d$, where

$$\begin{aligned}
[\![\boldsymbol{\mu}^{(j)}]\!]_k &:= (\boldsymbol{\gamma}^{(j)}[0] \cdot [\![\boldsymbol{a}^{(0)}]\!]_k, \ldots, \boldsymbol{\gamma}^{(j)}[n-1] \cdot [\![\boldsymbol{a}^{(n-1)}]\!]_k),\\
[\![\boldsymbol{\nu}^{(j)}]\!]_k &:= ([\![\boldsymbol{b}^{(0)}]\!]_k, \ldots, [\![\boldsymbol{b}^{(n-1)}]\!]_k),\\
[\![w^{(j)}]\!]_k &:= \textstyle\sum_{i=0}^{n-1} \boldsymbol{\gamma}^{(j)}[i] \cdot [\![c^{(i)}]\!]_k,
\end{aligned}$$

The verification task is now reduced to checking these new λ inner-product triples $\{([\![\boldsymbol{\mu}^{(j)}]\!]_k, [\![\boldsymbol{\nu}^{(j)}]\!]_k, [\![w^{(j)}]\!]_k)\}_{j\in[\lambda]}$.

Step 2: Lifting Inner-Product Triples to $\mathbb{F}_p$. Our protocol differs from previous work [11] mainly at this step, where we aim to lift the above λ distributed inner-product triples from $\mathbb{Z}_{2^k}$ to $\mathbb{F}_p$. It consists of the following three sub-steps.

(2.1) Local lifting of $([\![\boldsymbol{\mu}]\!]_k, [\![\boldsymbol{\nu}]\!]_k, [\![w]\!]_k)$**.** Below we'll omit the superscript (j) over the triple $([\![\boldsymbol{\mu}^{(j)}]\!]_k, [\![\boldsymbol{\nu}^{(j)}]\!]_k, [\![w^{(j)}]\!]_k)$, and use subscript k or p to specify the sharing domain. We first perform a local lifting on these triples, by directly viewing the original shares in $\mathbb{Z}_{2^k}$ as those in $\mathbb{F}_p$. Specifically, for any replicated secret sharing $[\![v]\!]_k \equiv_k (v_0, v_1, v_2)$ in $\mathbb{Z}_{2^k}$, we now view it as a replicated secret sharing $[\![v']\!]_p \equiv_p (v_0, v_1, v_2)$ in $\mathbb{F}_p$ where $v' \equiv_p v_0 + v_1 + v_2$. Let's first consider $[\![\boldsymbol{\mu}]\!]_k, [\![\boldsymbol{\nu}]\!]_k$, which are formed by concatenating all vectors $\{\boldsymbol{\gamma}[i] \cdot [\![\boldsymbol{a}^{(i)}]\!]_k\}_{i\in[n]}$ (where $\boldsymbol{\gamma}[i] \in \{0, 1\}$) and $\{[\![\boldsymbol{b}^{(i)}]\!]_k\}_{i\in[n]}$, respectively. As analyzed in Sec. 5.1, each element in $\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}$ come from some value on the RHS of Eq. (11), and its sharing must consist of two zero additive shares, e.g., $[\![\boldsymbol{a}^{(i)}[j]]\!]_k \equiv_k (\boldsymbol{a}^{(i)}[j], 0, 0)$ for some $j \in [n]$. Thus, the lifted secret value defined over $\mathbb{F}_p$ must be consistent with the original one defined over $\mathbb{Z}_{2^k}$ provided that $p > 2^k$. Hence, directly defining two new vectors $[\![\boldsymbol{\mu'}]\!]_p :\equiv_p [\![\boldsymbol{\mu}]\!]_k$, $[\![\boldsymbol{\nu'}]\!]_p :\equiv_p [\![\boldsymbol{\nu}]\!]_k$ yields $\boldsymbol{\mu'}, \boldsymbol{\nu'}$ that are consistent with the original $\boldsymbol{\mu}, \boldsymbol{\nu}$.

Now let's consider w. Similarly, we can perform a local lifting on $[\![w]\!]_k$ to obtain $[\![w']\!]_p$, where $[\![w']\!]_p :\equiv_p [\![w]\!]_k$. However, there is only one zero additive share in $[\![w]\!]_k$, and the sum of the two non-zero additive shares may exceed 2^k, leading to a difference of 2^k between w' and w over $\mathbb{F}_p$. Furthermore, the equation $w' \equiv_p \boldsymbol{\mu'} \cdot \boldsymbol{\nu'}$ may no longer hold over $\mathbb{F}_p$, as $\boldsymbol{\mu'}[j] \cdot \boldsymbol{\nu'}[j]$ can overflow multiple 2^k. Let $h :\equiv_p \boldsymbol{\mu'} \cdot \boldsymbol{\nu'}$ be the new correct inner-product result over $\mathbb{F}_p$. Our next sub-step is to derive a correct $[\![h]\!]_p$ from $[\![w']\!]_p$.

(2.2) Deriving correct inner-product result $[\![h]\!]_p$ ***from*** $[\![w']\!]_p$. To ensure our protocol works properly, we first enforce that h won't overflow p. Since each multiplication can be at most $(2^k - 1)^2$, the new result h is at most $(2^k - 1)^2 \cdot d'$. Thus, we assume $\ell > 2k$ and add a restriction on the size of the triples that $d' \le 2^{\ell-2k}$ s.t. $(2^k - 1)^2 \cdot d' < p = 2^\ell - 1$. [7]

Note that h can only be computed by the prover (since only she knows all the secrets). To enable the verifiers to derive $[\![h]\!]_p$, we can let the prover share the necessary information to the verifiers. As in the previous work [11], we make the observation that if h is correctly computed, then the difference between h and w' must be a multiple of 2^k, i.e., there must exist an integer t s.t.

$$h - w' = t \cdot 2^k. \tag{12}$$

In [11], the target lifting domain is $\mathbb{Z}_{2^{k+s}}$, so the prover can share t in $\mathbb{Z}_{2^s}$, and the parties can compute $[\![h]\!]_{k+s} := [\![w']\!]_{k+s} + [\![t]\!]_s \cdot 2^k$ as desired.

However, things become tricky when working in $\mathbb{F}_p$. The first problem is that t can be -1 in some cases, which would entail a wraparound over $\mathbb{F}_p$. To see this, note that $t = (h - w')/2^k$ counts the difference between the number of 2^k-overflows in the computation of $\boldsymbol{\mu} \cdot \boldsymbol{\nu}$ and the number of overflows in the lifting of w. Thus, a valid t satisfies $-1 \le t \le d'(2^k - 2) + \lfloor d'/2^k \rfloor$. [8] Another problem arises from the fact that the inverse of 2^k exists in $\mathbb{F}_p$. If we naively let the prover share any t in $\mathbb{F}_p$, a malicious prover would be able to introduce an additive error e into w in $\mathbb{Z}_{2^k}$ and then cancel it out in h over $\mathbb{F}_p$ by adding some crafted value.

We resolve these problems by adjusting and restricting the range of t. We add a constant 2^{k+1} to the LHS of Eq. (12), and transform it into

$$h - w' + 2^{k+1} \equiv_p \hat{t} \cdot 2^k. \tag{13}$$

It's clear that $\hat{t} = t+2$. As we have established that $-1 \le t \le d'(2^k-2)+\lfloor d'/2^k \rfloor$, we have $1 \le \hat{t} \le d'(2^k-2)+\lfloor d'/2^k \rfloor + 2$. Given $\ell > 2k$ and $d' \le 2^{\ell-2k}$, it follows that $d'(2^k-2)+\lceil d'/2^k \rceil + 2 < 2^{\ell-k}$ (we give a deduction of the inequality in App. E). Thus, any legal $\hat{t}$ is in the range $[1, 2^{\ell-k}]$, and Eq. (13) won't overflow p under this restriction. Given this,

5. Below we identify field elements in $\mathbb{Z}_{2^k}$ with integers in $[2^k]$, and elements in $\mathbb{F}_p$ with integers in $[p]$. We may say that a field element is smaller or larger than another; such comparisons are in relation to their corresponding integer values.

6. Consider the case where the prover introduces an additive error of 2^{k-1} into an inner-product triple. Then for every even coefficient w.r.t. this triple, the error will be canceled out, which happens with probability $1/2$.

7. This is a minor restriction, as p is typically a large prime (for example, $2^{61} - 1$ in our evaluation). We can verify approximately 2^{28} multiplications when $k = 16$, which is sufficient for practical settings.

8. Specifically, $t = d'(2^k-2)+\lfloor d'/2^k \rfloor$ when $\boldsymbol{\mu} \cdot \boldsymbol{\nu} = (2^k-1)^2 \cdot d'$ and $w_{i-1} + w_{i+1} < 2^k$; $t = -1$ when $w_{i-1} + w_{i+1} \ge 2^k$ and $\boldsymbol{\mu} \cdot \boldsymbol{\nu} < 2^k$.

we let the prover compute and distribute $[\![\hat{t}]\!]_p$ to the verifiers, and the parties can derive $[\![h]\!]_p :\equiv_p [\![w']\!]_p + ([\![\hat{t}]\!]_p - 2) \cdot 2^k$ as desired; additionally, they have to conduct a range check on $\hat{t}$ to validate $\hat{t} < 2^{\ell-k}$. Now by allowing $\hat{t} = 1$, we can cover the case where $t = -1$, and by bounding $\hat{t} < 2^{\ell-k}$, we can prevent the malicious prover from introducing non-zero errors into the triples, which we elaborate below.

(2.3) Range Check on t. We first show why a range check on the shared value $\hat{t}$ is sufficient to rule out additive errors. Assume that there is a non-zero additive error $e \in [1, 2^k]$ in the triple $([\![\boldsymbol{\mu}]\!]_k, [\![\boldsymbol{\nu}]\!]_k, [\![w^*]\!]_k)$ s.t. $w^* \equiv_k \boldsymbol{\mu} \cdot \boldsymbol{\nu} + e$. Recall that the lifting of $[\![\boldsymbol{\mu}]\!]_k, [\![\boldsymbol{\nu}]\!]_k, [\![w]\!]_k$ is local. Let $\tilde{w}$ be the lifted value after incorporating the error term e; since there is no overflow, we have $\tilde{w} \equiv_p w' + e$, where w' is the lifted value of w^*. The goal of the malicious prover is to craft a t^* such that there is no additive error in the lifted relation $([\![\boldsymbol{\mu'}]\!]_p, [\![\boldsymbol{\nu'}]\!]_p, [\![h]\!]_p)$. Specifically, t^* has to satisfy

$$h - \tilde{w} + 2^{k+1} \equiv_p h - w' + 2^{k+1} - e \equiv_p t^* \cdot 2^k. \quad (14)$$

Let $\hat{t} = (h - w') \cdot (2^k)^{-1} + 2$. This is the value an honest prover will produce from Eq. (13) if $e = 0$. To make Eq. (14) hold, the malicious prover has to set $t^* :\equiv_p \hat{t} - e \cdot (2^k)^{-1}$. Given that $p = 2^\ell - 1$, we have $(2^k)^{-1} \equiv_p 2^{\ell-k}$, and thus $t^* \equiv_p \hat{t} - e \cdot 2^{\ell-k}$. However, for any non-zero error $1 \le e \le 2^k - 1$, we have $-2^\ell + 2^{\ell-k} \le -e \cdot 2^{l-k} \le -2^{\ell-k}$, where elements in $\mathbb{F}_p$ are identified with $[-p, 0]$. This is equivalent to $2^{\ell-k} - 1 \le -e \cdot 2^{\ell-k} \le 2^\ell - 2^{\ell-k} - 1$, where elements are identified with $[p]$ (recall that $p = 2^\ell - 1$). Further, as the honestly computed $\hat{t}$ must satisfy $1 \le \hat{t} < 2^{\ell-k}$, we have

$$2^{\ell-k} \le t^* < 2^\ell - 1 = p.$$

Thus, the crafted t^* will always exceed $2^{\ell-k} - 1$, and any malicious attempt will be detected with the range check.

We now present how to perform the range check efficiently. While a legal $\hat{t}$ is always in $[1, 2^{\ell-k}]$, we can relax the check to $\hat{t} \in [0, 2^{\ell-k}]$, as the above analysis shows that any maliciously crafted t^* will never be zero. Therefore, we only need to check that the bit length of $\hat{t}$ is bounded by $\ell - k$. We ask the prover to additionally share a vector $[\![\hat{\boldsymbol{t}}]\!]_p :\equiv_p ([\![\hat{t}_0]\!]_p, \cdots, [\![\hat{t}_{\ell-k-1}]\!]_p)$ of length $\ell - k$, claimed to be the shares of the bit-decomposition of $\hat{t}$. Upon receiving these shares, the parties check whether the purported bit-decomposition is legal, by locally verifying if $[\![\hat{t}]\!]_p \equiv_p \sum_{i=0}^{\ell-k-1} [\![\hat{t}_i]\!]_p \cdot 2^i$ and also if $\hat{t}_i \in \{0, 1\}$ for each $i \in [\ell - k]$, which can be expressed as $\hat{t}_i \cdot (1 - \hat{t}_i) \equiv_p 0$. Since each $\hat{t}_i$ is shared by the verifiers and all shares are known by the prover, this is simply another distributed multiplication triple and can be verified with traditional DZKP techniques. Additionally, we can use a random linear combination $\boldsymbol{\eta}$ to batch them, and instead verify $\sum_{i=0}^{\ell-k-1} \boldsymbol{\eta}[i] \cdot \hat{t}_i \cdot (1 - \hat{t}_i) \equiv_p 0$. This is a distributed inner-product triple $(\boldsymbol{\eta} \circ \hat{\boldsymbol{t}}, \mathbf{1} - \hat{\boldsymbol{t}}, 0)$ of size $\ell - k$, where $\mathbf{1} = (1, \cdots, 1)$.

After obtaining the lifted triples and the extra triples from the range check over $\mathbb{F}_p$, the remaining step follows well-known DZKP techniques [9], [10], [12] that apply to prime fields. We describe it below for completeness.

Step 3. Proving Inner-Product Triples over $\mathbb{F}_p$. We first batch all the triples $\{([\![\boldsymbol{\mu'}^{(j)}]\!]_p, [\![\boldsymbol{\nu'}^{(j)}]\!]_p, [\![h^{(j)}]\!]_p)\}_{j\in[\lambda]}$ and $\{(\boldsymbol{\eta}^{(j)} \circ [\![\hat{\boldsymbol{t}}^{(j)}]\!]_p, \mathbf{1} - [\![\hat{\boldsymbol{t}}^{(j)}]\!]_p, 0)\}_{j\in[\lambda]}$ using a random linear combination over $\mathbb{F}_p$. Concretely, we let the parties generate a random coefficient vector $\boldsymbol{\theta}$ of length 2λ, and define

$$\begin{aligned}
[\![\boldsymbol{x}]\!]_p &:= (\boldsymbol{\theta}[0] \cdot [\![\boldsymbol{\mu'}^{(0)}]\!]_p, \cdots, \boldsymbol{\theta}[\lambda - 1] \cdot [\![\boldsymbol{\mu'}^{(\lambda-1)}]\!]_p, \\
&\qquad \boldsymbol{\theta}[\lambda] \cdot (\boldsymbol{\eta}^{(0)} \circ \hat{\boldsymbol{t}}^{(0)}), \cdots, \boldsymbol{\theta}[2 \cdot \lambda - 1] \cdot (\boldsymbol{\eta}^{(\lambda-1)} \circ \hat{\boldsymbol{t}}^{(\lambda-1)})), \\
[\![\boldsymbol{y}]\!]_p &:= ([\![\boldsymbol{\nu'}^{(0)}]\!]_p, \cdots, [\![\boldsymbol{\nu'}^{(\lambda-1)}]\!]_p, 1 - [\![\hat{\boldsymbol{t}}^{(0)}]\!]_p, \cdots, 1 - [\![\hat{\boldsymbol{t}}^{(\lambda-1)}]\!]_p), \\
[\![z]\!]_p &:= \textstyle\sum_{j=0}^{\lambda-1} \boldsymbol{\theta}[j] \cdot [\![h^{(j)}]\!]_p.
\end{aligned}$$

The parties then turn to verify the new batched triple $([\![\boldsymbol{x}]\!]_p, [\![\boldsymbol{y}]\!]_p, [\![z]\!]_p)$ which is of size $D := \lambda \cdot (d' + \ell - k)$.

Given a parameter q, the parties recursively reduce the size of the triple, each time by a factor of q. At a high level, $\boldsymbol{x}, \boldsymbol{y}$ are divided into D/q sub-vectors of the same length q, i.e., $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{D/q-1})$ and $\boldsymbol{y} = (\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{D/q-1})$. Then, we define two vectors of degree-$(q-1)$ polynomials $\boldsymbol{f}, \boldsymbol{h}$ s.t. $\boldsymbol{f}[i] \equiv_p \mathsf{Poly}(\boldsymbol{x}_i)$, and $\boldsymbol{g}[i] \equiv_p \mathsf{Poly}(\boldsymbol{y}_i)$ for all $i \in [D/q]$. For example, $\boldsymbol{f}$ consists of D/q polynomials, and each polynomial is of degree $(q-1)$, interpolated from the length-q vector $\boldsymbol{x}_i$.

Now we construct a degree-$(2q-2)$ polynomial $H \equiv_p \boldsymbol{f} \cdot \boldsymbol{g}$, the inner product of the two polynomial vectors $\boldsymbol{f}, \boldsymbol{g}$. By construction, we have $z \equiv_p \boldsymbol{x} \cdot \boldsymbol{y} \equiv_p \sum_{j=0}^{q-1} \boldsymbol{f}(j) \cdot \boldsymbol{g}(j) \equiv_p \sum_{j=0}^{q-1} H(j)$. The original relation can now be reduced to two checks: (1) $z \equiv_p \sum_{j=0}^{q-1} H(j)$, and (2) $H \equiv_p \boldsymbol{f} \cdot \boldsymbol{g}$. To check (1), we ask the prover to compute and share the coefficients of H to the verifiers; they then locally compute $\{[\![H(j)]\!]_p\}_{j\in[q]}$, and invoke $\mathcal{F}_{\text{checkzero}}$ on $[\![z]\!] - \sum_{j=0}^{q-1} [\![H(j)]\!]_p$. To check (2), it suffices for the verifiers to sample a random challenge $r \in \mathbb{F}_p$ and check whether $h(r) \equiv_p \boldsymbol{f}(r) \cdot \boldsymbol{g}(r)$ holds based on the Schwartz-Zippel lemma [21]. Note that holding $[\![\boldsymbol{x}]\!]_p, [\![\boldsymbol{y}]\!]_p$, the verifiers can locally derive their shares of the coefficients of $\boldsymbol{f}, \boldsymbol{g}$, and since they have already received the shares of the coefficients of H from the prover, they can locally compute $([\![\boldsymbol{f}(r)]\!]_p, [\![\boldsymbol{g}(r)]\!]_p, [\![h(r)]\!]_p)$. This means that the original triple is reduced to a smaller one $([\![\boldsymbol{f}(r)]\!]_p, [\![\boldsymbol{g}(r)]\!]_p, [\![h(r)]\!]_p)$, with size D/q reduced by a factor of q. Therefore, we can recursively apply this process until the size is reduced to one, at which point the parties can verify the relation by direct opening the shares with the help of a random triple.

We denote the protocol by Π_{VrfyIP}, and have the following theorem. We give a detailed description of Π_{VrfyIP} and a proof the theorem in App. D.

Theorem 2. *Assume that λ is the security parameter. Let $\mathbb{F}$ be a prime field of size $|\mathbb{F}| = 2^{\Omega(\lambda)}$. The protocol Π_{VrfyIP} securely computes $\mathcal{F}_{\mathit{VrfyIP}}$ with abort in the $\{\mathcal{F}_{\mathit{input}}, \mathcal{F}_{\mathit{coin}}, \mathcal{F}_{\mathit{checkzero}}\}$−hybrid model against a malicious adversary controlling one corrupted party and achieves soundness error* $\mathrm{negl}(\lambda)$.

5.3. Further Optimizations

(Almost) Eliminating the Multiplicative Overhead of λ. Recall that the final inner-product triple $([\![\boldsymbol{x}]\!]_p, [\![\boldsymbol{y}]\!]_p, [\![z]\!]_p)$

has size $\lambda \cdot (d' + \ell - k)$ where λ is the statistical security parameter, and $d' = nd$ is the total size of all input triples to be verified. We can utilize the techniques from [11] to reduce the size of the final triple to $d' + \lambda(\ell - k)$ by merging the random coefficients in Step 1 and Step 3 together. As $\ell - k$ is generally much smaller than d' in large-scale applications, this can significantly reduce the computational cost in practice. We present more details in App. A.

Merging Triples with Same Multiplier. Besides the above general optimization technique, we can further reduce the size of the triples to be verified in our LUT protocol. The key observation is that many inner-product triples involved in our LUT protocol share the same multipliers. Specifically, during the offline phase, all triples to be verified in the i-th round, $\{(\llbracket \boldsymbol{f}^{(i-1)}[j] \rrbracket_k, \llbracket b^{(i)} \rrbracket_k, \llbracket \boldsymbol{f}^{(i-1)}[j + 2^i] \rrbracket_k)\}_{j \in [2^i - 1]}$, share the same multiplier $\llbracket b^{(i)} \rrbracket_k$. Similarly, in the online phase, each party must verify the inner product between $\boldsymbol{e}^{(r_i)}$ and $\{\llbracket \hat{T}^{(c-i)}[j_i, j_{i+1}, \cdots, j_{c-1}] \rrbracket_k\}_{j_i \in [N^{1/c}]}$ for all $(j_{i+1}, \cdots, j_{c-1}) \in [N^{1/c}]^{c-i-1}$, where the multiplier is always $\llbracket \boldsymbol{e}^{(r_i)} \rrbracket_k$. This allows a substantial reduction in the size of the final triple to be verified, as triples sharing the same multiplier can be merged with random linear combinations.

With the above optimizations, each round of offline computation yields a triple of size 1, and each round of online computation yields a triple of size $N^{1/c}$. This is substantially smaller than the actual number of multiplications performed, which is helpful as the malicious-security phase is much more costly than the semi-honest protocol.

Remark 4 (Optimization for Multiplication Verification over $\mathbb{Z}_2$)**.** *The structure of $\mathbb{Z}_2$ enables further optimizations to Π_{VrfyIP} using the techniques from [18], which performs polynomial evaluation with faster table lookups. This is applicable to our LUT protocols in $\mathbb{Z}_2$ and $\mathbb{F}_{2^k}$ (see Sec. 4.3). Especially, it can be applied to the verification of general binary circuits, which may be of independent interest.*

6. Evaluation

6.1. Experimental Setup

We have fully implemented MARLUT based on the released framework [30] of MAESTRO in Rust.[9] As far as we know, MARLUT is the first *maliciously secure* LUT protocol supporting multi-input tables over *both* $\mathbb{Z}_{2^k}$ and $\mathbb{F}_{2^k}$, we thus use the most state-of-the-art LUT work MAESTRO [7] and DZKP protocol LEDH+24 [11] as our baselines. We performed three kinds of experiments to evaluate:

- **The semi-honest LUT protocol.** To evaluate our semi-honest LUT protocol, we implemented Prot. 10 of MAESTRO (with semi-honest security) as the baseline, which achieves the lowest communication costs among all LUT protocol presented in MAESTRO (see Remark 1).
- **The DZKP protocol.** We conduct two experiments: (1) comparison with the DZKP protocol from LEDH+24 [11] (which also targets $\mathbb{Z}_{2^k}$), where we compare the two protocols verifying multiplication gates in arithmetic circuits over $\mathbb{Z}_{2^k}$, and (2) comparison with the DZKP protocol from MAESTRO (which targets $\mathbb{F}_{2^k}$), where we verify inner-product triples yielded by OHV generation on $\mathbb{Z}_{2^k}$ and $\mathbb{F}_{2^k}$, respectively.
- **End-to-End Performance.** We evaluate the end-to-end performance of MARLUT by measuring its throughput for floating-point multiplication (over $\mathbb{Z}_{2^8}$) and the widely-adopted Sigmoid activation function (over $\mathbb{Z}_{2^{16}}$).

To achieve 40-bit statistical security, we use $p = 2^{61} - 1$ for our protocol, and $s = 64$ for LEDH+24. We set $c = 3$ in our protocol to achieve the best concrete efficiency. [10] All the experiments were run on three Alibaba ecs.r8i.8xlarge cloud servers located in a Local Area Network (LAN) with 32 vCPUs and 256 GB memory using a single thread. We also use a simulated Wide Area Network (WAN) with 40 MBps bandwidth and 20 ms delay using the Linux *tc* command. The results are averaged over five runs.

6.2. Performance of Semi-Honest LUT Protocol

To align with MAESTRO, we use a single-input table of size 2^8 over $\mathbb{F}_{2^8}$; to further evaluate scalability, we adapt Prot. 10 of MAESTRO to ring $\mathbb{Z}_{2^k}$ (using the same techniques presented in Sec. 4), and test the two protocols on a larger size-2^{16} table over $\mathbb{Z}_{2^{16}}$. We perform 2^{18} ($\sim 262,000$) lookups in the two tables, and present the results in Tab. 1.

The results demonstrate that our protocol significantly reduces offline overhead while achieving superior online efficiency compared to MAESTRO, particularly for large tables. For table size 2^{16}, in the LAN setting, our protocol achieves $5.95\times$ speedup in the offline phase with $3.23\times$ lower communication cost, owing to the use of smaller one-hot vectors; the online efficiency is also improved by $1.44\times$ as the tensor representation eliminates the redundant computations. Although the online communication increases slightly, the total communication cost is still reduced by a factor of 2.7. In the WAN setting, where the benefits of computational savings are less pronounced, our protocol still shows $2.23\times$ speedup in the offline phase, and continues to outperform MAESTRO in online efficiency.

6.3. Performance of DZKP Protocol

We set the recursion parameter $q = 8$ in our protocol. For LEDH+24 and MAESTRO, we adopt their default parameters ($q = 8$ and 2, respectively). As all three DZKP protocols have communication costs logarithmic in instance size, which is negligible compared with the linear communication cost of the semi-honest protocol, we conduct our experiments in the LAN setting to evaluate their computational efficiency.

Comparison with LEDH+24. To fairly conduct the comparison, we follow LEDH+24 using our DZKP protocol to verify

9. Our code can be found in the anonymous repository: https://anonymous.4open.science/r/marlut-0594.

10. Specifically, for tables of size 2^8 and 2^{16}, we set the size of dimensions to $(2^3, 2^3, 2^2)$ and $(2^6, 2^5, 2^5)$, respectively.

TABLE 1: Evaluation of semi-honest protocols of ours and MAESTRO [7] (Prot. 10) on 2^{18} lookups in tables over $\mathbb{F}_{2^8}$ and $\mathbb{Z}_{2^{16}}$.

Table Size	Protocol	LAN Time (s)		WAN Time (s)		Commu. (MB)		
		Offline	Online	Offline	Online	Offline	Online	Total
2^8 ($\mathbb{F}_{2^8}$)	Ours	0.062 (**1**×)	0.118 (**1**×)	0.614 (**1**×)	0.272 (**1**×)	10.25 (**1**×)	2.5	12.75 (**1**×)
	MAESTRO	0.112 (**1.81**×)	0.134 (**1.14**×)	1.211 (**1.97**×)	0.348 (**1.28**×)	13.5 (**1.32**×)	0.5	14 (**1.1**×)
2^{16} ($\mathbb{Z}_{2^{16}}$)	Ours	0.495 (**1**×)	2.089 (**1**×)	5.439 (**1**×)	2.835 (**1**×)	86.5 (**1**×)	17	103.5 (**1**×)
	MAESTRO	2.945 (**5.95**×)	3.01 (**1.44**×)	12.113 (**2.23**×)	3.22 (**1.14**×)	279 (**3.23**×)	1	280 (**2.7**×)

TABLE 2: Evaluation of DZKP protocols of ours and LEDH+24 [11] on verifying multiplication gates over $\mathbb{Z}_{2^8}$ and $\mathbb{Z}_{2^{16}}$.

Protocol	Circuit Size	Base Ring $\mathbb{Z}_{2^8}$			Base Ring $\mathbb{Z}_{2^{16}}$		
		Time (s)	Commu. (MB)		Time (s)	Commu. (MB)	
			Passive	DZKP		Passive	DZKP
Ours	2^{20}	0.246 (**1**×)	1	0.0338	0.355 (**1**×)	2	0.0289
	2^{22}	1.012 (**1**×)	4	0.0339	1.501 (**1**×)	8	0.0290
	2^{24}	4.511 (**1**×)	16	0.0340	6.635 (**1**×)	32	0.0291
LEDH+24	2^{20}	1.739 (**7.07**×)	1	0.023	1.763 (**4.97**×)	2	0.026
	2^{22}	6.851 (**6.77**×)	4	0.093	6.873 (**4.58**×)	8	0.103
	2^{24}	27.125 (**6.01**×)	16	0.37	27.25 (**4.11**×)	32	0.41

TABLE 3: Evaluation of DZKP protocols of ours and MAESTRO [7] on verifying triples yielded by OHV generation for 2^{18} lookups in size-2^8 tables over $\mathbb{Z}_{2^8}$ and $\mathbb{F}_{2^8}$.

Table Size	Protocol	# of Triples	Time (s)	Commu. (MB)
2^8 ($\mathbb{Z}_{2^8}$)	Ours	5	0.390	2.318
2^8 ($\mathbb{F}_{2^8}$)	MAESTRO	4	0.223	0.687

TABLE 4: Evaluation of MARLUT for computing floating-point multiplication over $\mathbb{Z}_{2^8}$ and Sigmoid over $\mathbb{Z}_{2^{16}}$.

App.	Table Size	LAN Throughput		WAN Throughput	
		Offline	Online	Offline	Online
FP Mul	2^{16} ($\mathbb{Z}_{2^8}$)	71,290	6,973	30,542	6,718
Sigmoid	2^{16} ($\mathbb{Z}_{2^{16}}$)	527,441	12,368	48,201	11,333

multiplication gates for general arithmetic circuits over $\mathbb{Z}_{2^k}$, and test the two protocols with circuits of various sizes over $\mathbb{Z}_{2^8}$ and $\mathbb{Z}_{2^{16}}$. Concretely, we use circuits of depth $1, 4$ and 16, with each layer consisting of 2^{20} multiplication gates (the total circuit sizes are $2^{20}, 2^{22}$ and 2^{24}). We present the total runtime, as under LAN environments the time spent in communication is negligible.

As shown in Tab. 2, compared with LEDH+24, our protocol can achieve up to $7.07\times$ speedup in $\mathbb{Z}_{2^8}$, and $4.97\times$ speedup in $\mathbb{Z}_{2^{16}}$. This mainly arises from the faster Mersenne prime field adopted in our protocol (and implementation-level optimizations). In terms of communication, our DZKP protocol is comparable to or more efficient than LEDH+24, especially for large-scale instances. This is because the communication cost of our field-based DZKP protocol is linear in the recursion parameter q, while the expanded ring-based protocol of LEDH+24's is quadratic in q.

Comparison with MAESTRO. As MAESTRO aims to build an end-to-end AES application, we only test the two protocols for verifying distributed triples obtained in the offline OHV generation process.[11] We compare with an optimized DZKP protocol of MAESTRO, which is customized for OHV verification and verifies 4 triples, while ours verifies 5 triples with the optimizations presented in Sec. 5.3. We run 2^{18} OHV generation instances for a size-2^8 table over $\mathbb{Z}_{2^8}$ and $\mathbb{F}_{2^8}$ using our protocol and MAESTRO's, respectively.

As shown in Tab. 3, our protocol is comparable to MAESTRO's, taking into account the number of triples verified. This shows that our ring-target DZKP protocol (which is more challenging to construct) is on par with well-known field-target protocols [9], [10]. We clarify that the higher total communication cost of our protocol arises from the semi-honest phase, where MAESTRO can directly operate over $\mathbb{F}_2$ and we have to use $\mathbb{Z}_{2^8}$ due to the deficiencies of the ring structure. (In fact, our protocol generates smaller one-hot vectors, so the total communication cost is less than $8\times$ that of MAESTRO.) We emphasize that the communication costs incurred by the two DZKP protocols are comparable, as they are both logarithmic in the triple size.

6.4. E2E Performance

We build two applications atop MARLUT: 8-bit floating-point multiplication with a two-input table over $\mathbb{Z}_{2^8}$, and 16-bit Sigmoid activation function with a single-input table over $\mathbb{Z}_{2^{16}}$. Both applications require a table of size 2^{16}.

11. We omit the result of verifying random bits over $\mathbb{Z}_{2^k}$ in our protocol, which is not involved in the offline phase of MAESTRO as they target $\mathbb{F}_{2^k}$.

We use throughput (i.e., number of instances executed per second) to evaluate the performance of MARLUT.

We present the result in Tab. 4. MARLUT shows high practical throughput on the two applications, especially in the offline phase; this is because we utilize high-dimensional tensors to reduce offline costs. For reference, MAESTRO did not evaluate applications with table sizes larger than 2^8, and the reported offline and online throughput [30] of MAESTRO in evaluating the S-Box function (with a single-input table of size 2^8) is $50,482$ and $11,381$ blocks/s, respectively. This shows the great potential of MARLUT for modern real-world ring-based applications.

Ethics considerations

Societal Benefits of Our Research. Our work advances efficient secure computation by presenting a generalized and optimized lookup table protocol with malicious security. Our framework makes strong, accountable privacy protections more practical for data holders in healthcare, finance, and other sensitive domains.

Compliance with Legal, Ethical, and Open-Science Standards. This study involves no human subjects, user studies, or personal/regulated data, and thus requires no IRB review. Microbenchmarks use synthetic/random inputs; all evaluations use simple mathematical tables. All experiments were run on machines we control; WAN conditions were emulated locally using traffic control, and no unsolicited traffic was sent to third parties. We complied with institutional policies and cloud provider terms of service. We will release our code and experiment scripts under an open-source license and we will publish configuration parameters and seeds to support reproducibility.

Disclosure and Security Posture. Our work does not target deployed systems and does not disclose exploitable vulnerabilities. Our implementation is built on standard cryptographic libraries; any issues encountered during development were reported upstream when applicable. However, we stress that any code we release is on an "as-is" basis and has not been audited for security and defects.

Dual-Use and Misuse Considerations. Our proposed techniques are intended to enhance privacy and integrity in secure computation, rather than to obscure or enable malicious behavior. We explicitly avoid releasing any tooling that could facilitate evasion of accountability.

LLM usage considerations

LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

References

[1] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, ser. SFCS '82, 1982, pp. 160–164.

[2] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, ser. Lecture Notes in Computer Science, A. Sahai, Ed., vol. 7785. Springer, 2013, pp. 600–620. [Online]. Available: https://doi.org/10.1007/978-3-642-36594-2_34

[3] I. Damgård and R. W. Zakarias, "Fast oblivious AES A dedicated application of the minimac protocol," in *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval, A. Nitaj, and T. Rachidi, Eds., vol. 9646. Springer, 2016, pp. 245–264. [Online]. Available: https://doi.org/10.1007/978-3-319-31517-1_13

[4] G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/pushing-communication-barrier-secure-computation-using-lookup-tables/

[5] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame, "Flute: Fast and secure lookup table evaluations," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 515–533.

[6] X. Hou, J. Liu, J. Li, J. Zhang, and K. Ren, "Faster lookup table evaluation with application to secure llm inference," *Cryptology ePrint Archive*, 2024.

[7] H. Morita, E. Pohle, K. Sadakane, P. Scholl, K. Tozawa, and D. Tschudi, "Maestro: Multi-party aes using lookup tables," in *34 USENIX Security Symposium 2025*, 2025.

[8] A. N. Baccarini, M. Blanton, and C. Yuan, "Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning," *IACR Cryptol. ePrint Arch.*, p. 1577, 2020. [Online]. Available: https://eprint.iacr.org/2020/1577

[9] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear pcps," in *Annual International Cryptology Conference*. Springer, 2019, pp. 67–97.

[10] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 869–886.

[11] Y. Li, D. Escudero, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Sublinear distributed product checks on replicated secret-shared data over z2k without ring extensions," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 825–839.

[12] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Efficient fully secure computation via distributed zero-knowledge proofs," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 244–276.

[13] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proceedings on Privacy Enhancing Technologies*, 2022.

[14] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "SIGMA: secure GPT inference with function secret sharing," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 4, pp. 61–79, 2024. [Online]. Available: https://doi.org/10.56553/popets-2024-0107

[15] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger, "Blaze: Fast snarks from interleaved raa codes," *Cryptology ePrint Archive*, p. 1609, 2024.

[16] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "{SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2651–2668.

[17] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2183–2200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

[18] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5377–5394. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun

[19] M. Hamilis, E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Preprocessing for life: Dishonest-majority mpc with a trusted or untrusted dealer," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 41–41.

[20] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Sublinear gmw-style compiler for MPC with preprocessing," in *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Springer, 2021, pp. 457–485. [Online]. Available: https://doi.org/10.1007/978-3-030-84245-1_16

[21] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," vol. 27, no. 4, pp. 701–717, 1980.

[22] I. Mitsuru, S. Akira, and N. Takao, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[23] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10211, 2017, pp. 225–255. [Online]. Available: https://doi.org/10.1007/978-3-319-56614-6_8

[24] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, "Fast large-scale honest-majority MPC for malicious adversaries," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Springer, 2018, pp. 34–64. [Online]. Available: https://doi.org/10.1007/978-3-319-96878-0_2

[25] A. Toshinori, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016, pp. 805–817.

[26] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek, "Faster secure multi-party computation of aes and des using lookup tables," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 229–249.

[27] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, 2020. [Online]. Available: https://doi.org/10.2478/popets-2020-0077

[28] ——, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. D. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2183–2200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

[29] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively secure 4pc for secure training and inference," in *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/auto-draft-202/

[30] Maestro. [Online]. Available: https://github.com/KULeuven-COSIC/maestro

Appendix A. Detailed Description of Our Optimization

In this section, we elaborate on our general optimization for reducing the size of the final verification triple in Π_{VrfyIP}. For clarity, we partition the vectors into two disjoint components: $(\llbracket \boldsymbol{x}^{(0)} \rrbracket_p, \llbracket \boldsymbol{y}^{(0)} \rrbracket_p)$ of size $\lambda \cdot d'$, constructed from input triples, and $(\llbracket \boldsymbol{x}^{(1)} \rrbracket_p, \llbracket \boldsymbol{y}^{(1)} \rrbracket_p)$ of size $\lambda \cdot (\ell - k)$, from the range check on $\hat{t}$. The second part is relatively small and the first part is dominant. A direct application of our DZKP protocol would result in $(\llbracket \boldsymbol{x}^{(0)} \rrbracket_p, \llbracket \boldsymbol{y}^{(0)} \rrbracket_p)$ being λ times larger than the input triples, as we need to repeat the batching process λ times for soundness. However, this factor of λ can be eliminated using techniques similar to [11].

To see this, we shall recall how $(\llbracket \boldsymbol{x}^{(0)} \rrbracket_p, \llbracket \boldsymbol{y}^{(0)} \rrbracket_p)$ is obtained. In the first step of Π_{VrfyIP}, given the n input triples $\{(\llbracket \boldsymbol{a}^{(i)} \rrbracket_k, \llbracket \boldsymbol{b}^{(i)} \rrbracket_k, \llbracket c^{(i)} \rrbracket_k)\}_{i\in[n]}$ each of length d, the parties generate λ binary coefficient vectors $\{\boldsymbol{\gamma}^{(j)}\}_{j\in[\lambda]}$, and use them to combine the input triples into λ batched inner-product triples $\{(\llbracket \boldsymbol{\mu}^{(j)} \rrbracket_k, \llbracket \boldsymbol{\nu}^{(j)} \rrbracket_k, \llbracket w^{(j)} \rrbracket_k)\}_{j\in[\lambda]}$, each of length $d' = nd$. These batched triples are then lifted to $\mathbb{F}_p$ as $\{(\llbracket \boldsymbol{\mu'}^{(j)} \rrbracket_p, \llbracket \boldsymbol{\nu'}^{(j)} \rrbracket_p, \llbracket h^{(j)} \rrbracket_p)\}_{j\in[\lambda]}$. The λ triples are then combined with another linear combination using random coefficients $\boldsymbol{\theta}[0:\lambda]$ in $\mathbb{F}_p$. The key observation is that the two sets of coefficients are only applied to $\llbracket \boldsymbol{a}^{(i)} \rrbracket_k$, while $\llbracket \boldsymbol{b}^{(i)} \rrbracket_k$ is left untouched and simply concatenated, repeated λ times, and lifted to $\mathbb{F}_p$. Therefore, each pair of $\llbracket \boldsymbol{a}^{(i)} \rrbracket_k$ and $\llbracket \boldsymbol{b}^{(i)} \rrbracket_k$ appears λ times in the final instance, with varying coefficients. We can merge those terms by adding up the coefficients to obtain

$$\llbracket \boldsymbol{x}^{(0)} \rrbracket_p := (\delta^{(0)} \cdot \llbracket \boldsymbol{a}^{(0)} \rrbracket_p, \cdots, \delta^{(n-1)} \cdot \llbracket \boldsymbol{a}^{(n-1)} \rrbracket_p),$$
$$\llbracket \boldsymbol{y}^{(0)} \rrbracket_p := (\llbracket \boldsymbol{b}^{(0)} \rrbracket_p, \ldots, \llbracket \boldsymbol{b}^{(n-1)} \rrbracket_p),$$

where $\delta^{(i)} = \sum_{j\in[\lambda]} \boldsymbol{\gamma}^{(j)}[i] \cdot \boldsymbol{\theta}[j]$ for all $i \in [n]$. Subsequently, the size of $\llbracket \boldsymbol{x}^{(0)} \rrbracket_p, \llbracket \boldsymbol{y}^{(0)} \rrbracket_p$ is reduced to $d' = nd$. The total size of the final triple is reduced to $d' + \lambda \cdot (\ell - k)$.

Appendix B. Formal Description of Π_{LUT}

We give a formal description of Π_{LUT} in Prot. 3.

Appendix C. Proof of Theorem 1

Proof. Let $\mathcal{S}$ be the ideal world adversary and $\mathcal{A}$ the real world adversary controlling a single corrupted party.

1) $\mathcal{S}$ first receives from $\mathcal{F}_{\text{LUT}}$ the shares of $\llbracket \boldsymbol{v} \rrbracket_k^{\mathcal{C}}$.

Prot. 3: Π_{LUT} - Securely Computing Lookup Table

All parties hold a secret-shared length-n vector $[\![\boldsymbol{v}]\!]_k$ in $\mathbb{Z}_{2^k}$ and agree on a public lookup table T of size $N = 2^{n \cdot k}$.

1) All parties invoke $\mathcal{F}_{\text{RandOHV}}(n, k)$ to get $([\![\boldsymbol{r}]\!]_k, [\![\boldsymbol{e}^{(r)}]\!]_k)$.
2) All parties reconstruct $\boldsymbol{m} := \mathsf{Rec}([\![\boldsymbol{v}]\!]_k + [\![\boldsymbol{r}]\!]_k)$.
3) All parties locally define a shift table $\hat{T}$ s.t. $\hat{T}[\mathsf{int}_k(\boldsymbol{j})] = T[\mathsf{int}_k(\boldsymbol{m} - \boldsymbol{j})], \forall \boldsymbol{j} \in [2^k]^n$. Then they locally compute $[\![\boldsymbol{e}^{(r)}]\!]_k \cdot \hat{T}$, and output it as $[\![T[\boldsymbol{v}]]\!]_k$.

2) When invoking $\mathcal{F}_{\text{RandOHV}}$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{RandOHV}}$ and receives from $\mathcal{A}$ the shares of $\{([\![\boldsymbol{r}_i]\!]_k^{\mathcal{C}}, [\![\boldsymbol{e}^{(r_i)}]\!]_k^{\mathcal{C}})\}_{i \in [c]}$. $\mathcal{S}$ then computes the share of $[\![\boldsymbol{r}]\!]_k^{\mathcal{C}}$ according to all $[\![\boldsymbol{r}_i]\!]_k^{\mathcal{C}}$, and sets $[\![\boldsymbol{m}]\!]_k^{\mathcal{C}} := [\![\boldsymbol{v}]\!]_k^{\mathcal{C}} + [\![\boldsymbol{r}]\!]_k^{\mathcal{C}}$.
3) $\mathcal{S}$ randomly samples a set of shares $[\![\boldsymbol{m}]\!]_k^{\mathcal{H}}$ held by honest parties and sends them to the adversary. $\mathcal{S}$ then locally reconstructs the value $\boldsymbol{m}$, builds $\hat{T}^{(c)}$, and computes $[\![\hat{T}^{(c-1)}[j_1, \ldots, j_{c-1}]]\!]_k^{\mathcal{C}} = \sum_{j_0 \in [N^{1/c}]} [\![\boldsymbol{e}^{(r_0)}[j_0]]\!]_k^{\mathcal{C}} \cdot \hat{T}^{(c)}[j_0, j_1, \ldots, j_{c-1}]$, for all $j_1, \ldots, j_{c-1} \in [N^{1/c}]$.
4) In the i-th round for $i \in \{1, \ldots, c-1\}$, for all $j_{i+1}, \ldots, j_{c-1} \in [N^{1/c}]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{ip}}$ with input sharings $[\![\boldsymbol{e}^{(r_i)}]\!]_k^{\mathcal{C}}$ and $([\![\hat{T}^{(c-i)}[j_i, j_{i+1}, \ldots, j_{c-1}]]\!]_k^{\mathcal{C}})_{j_i \in [N^{1/c}]}$ and obtains the shares of $[\![\hat{T}^{(c-i-1)}[j_{i+1}, \ldots, j_{c-1}]]\!]_k^{\mathcal{C}}$.
5) Finally, $\mathcal{S}$ outputs $[\![\hat{T}^{(0)}]\!]_k^{\mathcal{C}}$ to $\mathcal{F}_{\text{LUT}}$.

The adversary's view consists of (1) shares of the input $\boldsymbol{v}$, shares of all correlated randomness $\{(\boldsymbol{r}_i, \boldsymbol{e}^{(r_i)})\}_{i \in [c]}$, and shares of all values $\hat{T}^{(c-i)}[j_i, \ldots, jc-1]$, as well as (2) the revealed value $\boldsymbol{m}$. Due to the secrecy of the RSS scheme, view (1) is the same in the simulation and the real execution. Moreover, the revealed value $\boldsymbol{m}$ is uniformly distributed in both executions. Hence, view (2) is also identically distributed in the real and ideal worlds. □

Appendix D. Detailed Description of Π_{VrfyIP}

We give the detailed description of Π_{VrfyIP} in Prot. 4, and give a proof of Theorem 2 below.

Proof. Let $\mathcal{S}$ be the ideal world adversary and $\mathcal{A}$ the real world adversary controlling a single corrupted party. $\mathcal{S}$ first receives the prover's identity j from $\mathcal{F}_{\text{VrfyIP}}$. We consider the following two cases.

Case 1: the prover $\mathcal{P}_j$ is corrupted. In this case, $\mathcal{S}$ also receives the shares of the corrupted parties and the whole sharings $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i \in [n]}$ of the corrupted prover from $\mathcal{F}_{\text{VrfyIP}}$. $\mathcal{S}$ proceeds as follows.

1) $\mathcal{S}$ emulates the role of $\mathcal{F}_{\text{coin}}$ by sampling and handing random binary coefficients $\boldsymbol{\gamma}^{(0)}, \ldots, \boldsymbol{\gamma}^{(\lambda-1)}$ to $\mathcal{A}$. Then $\mathcal{S}$ follows the protocol and computes the sharings $([\![\boldsymbol{\mu}^{(j)}]\!]_k, [\![\boldsymbol{\nu}^{(j)}]\!]_k, [\![w^{(j)}]\!]_k)$ for all $j \in [\lambda]$.
2) $\mathcal{S}$ receives from $\mathcal{A}$ the honest parties' shares of $\hat{\boldsymbol{t}}^{(i)}$ for each $j \in [\lambda]$. $\mathcal{S}$ locally computes $[\![h^{(j)}]\!]_p$ held by the honest parties for each $j \in [\lambda]$.
3) $\mathcal{S}$ plays the role of $\mathcal{F}_{\text{coin}}$ by sampling and handing a length-2λ random vector $\boldsymbol{\theta}$ to $\mathcal{A}$.
4) As $\mathcal{S}$ is given the corrupted party' shares and all shares in $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i \in [n]}$ and $\{\hat{\boldsymbol{t}}^{(i)}\}_{i \in [\lambda]}$, it deduces the shares of $([\![\boldsymbol{x}]\!]_p, [\![\boldsymbol{y}]\!]_p, [\![z]\!]_p)$ held by the honest parties with the randomnesses θ_i as described in the protocol.
5) $\mathcal{S}$ repeats the following procedure. It receives the shares of the coefficients of H the honest parties should have from $\mathcal{A}$. Then it simulates $\mathcal{F}_{\text{coin}}$ by sending a random r in $\mathbb{F}_p$ to $\mathcal{A}$, and updates the honest parties' shares of $([\![\boldsymbol{x}]\!]_p, [\![\boldsymbol{y}]\!]_p, [\![z]\!]_p)$ according to the protocol. $\mathcal{S}$ then goes to the next iteration until the dimension of the vectors becomes 1.
6) In the final check step, $\mathcal{S}$ receives the honest parties' shares of x', y', z' from $\mathcal{A}$. Then it simulates $\mathcal{F}_{\text{coin}}$ by handing a random r in $\mathbb{F}_p$ to $\mathcal{A}$.
7) $\mathcal{S}$ computes the shares of the final multiplication triple $([\![x']\!]_p, [\![y']\!]_p, [\![z']\!]_p)$ held by the honest parties, and then reconstruct the triple (x', y', z') from the computed shares. Then $\mathcal{S}$ simulates the reconstruction procedure by handing the honest parties' shares of the multiplication triple to $\mathcal{A}$.
8) If $\mathcal{F}_{\text{VrfyIP}}$ doesn't send reject to $\mathcal{S}$, then $\mathcal{S}$ sends accept to $\mathcal{F}_{\text{VrfyIP}}$ if $([\![x']\!]_p, [\![y']\!]_p, [\![z']\!]_p)$ is a correct multiplication triple, and sends reject otherwise. Otherwise, if $\mathcal{S}$ receives reject from $\mathcal{F}_{\text{VrfyIP}}$ and the final multiplication triple is incorrect, i.e., $x' \cdot y' \neq_p z'$, then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. Otherwise, if it receives reject from $\mathcal{F}_{\text{VrfyIP}}$ but the final multiplication triple is correct, then $\mathcal{S}$ outputs fail and halts.

Observe that $\mathcal{S}$ knows exactly the honest parties' shares in this case, and thus the above simulation is nearly perfect. The only difference between the simulation and the real execution is the event that $\mathcal{S}$ outputs fail, where $\mathcal{F}_{\text{VrfyIP}}$ outputs reject to the honest parties but the honest parties in the real execution output accept. Note that this situation arises only when there exists several incorrect triples $c_i \neq_k \boldsymbol{a}^{(i)} \cdot \boldsymbol{b}^{(i)}$ for some $i \in [n]$ but the final multiplication triple is correct, i.e., $x' \cdot y' \equiv_p z'$. Several steps may contribute to such a simulation failure, namely Step 1, Step 2.2 and Step 3. For each of these steps, we show that the failure probability is negligible in λ, provided that $|\mathbb{F}_p| = 2^{O(\lambda)}$ and $\lambda = \Omega(nd)$. By applying the union bound, the overall failure probability is therefore also negligible.

We first consider Step 1, where all input triples are batched. As previously discussed, the probability that a malicious prover cancels the additive error is at most $2^{-\lambda}$, which is negligible. For Step 2.2, as explained in Step 2.3, if all bit decompositions $\{\hat{\boldsymbol{t}}^{(j)}\}_{j \in [\lambda]}$ are generated correctly, then the malicious prover cannot transform an invalid triple into a valid one after lifting to $\mathbb{F}_p$. Hence, the only way for a malicious prover to cause $\mathcal{S}$ to output fail in this step is by constructing an invalid bit decomposition that nonetheless passes the binary check on the shared bits. Since this binary check is deferred to Step 3, we analyze both steps jointly.

Prot. 4: Π_{VrfyIP} - Verifying Secret-Shared Inner-Product Triples

All parties share n inner-product triples $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i=0}^{n-1}$ each of size d. The prover in addition learns all shares of $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i=0}^{n-1}$. All parties agree on parameters λ and q.

Step 1: Batching All Inner-Product Triples.

1) All parties invoke $\mathcal{F}_{\text{coin}}$ to obtain λ random binary coefficient vectors $\boldsymbol{\gamma}^{(0)}, \cdots, \boldsymbol{\gamma}^{(\lambda-1)} \in \{0,1\}^n$.
2) For all $j \in [\lambda]$, all parties locally set $[\![\boldsymbol{\mu}^{(j)}]\!]_k = (\boldsymbol{\gamma}^{(j)}[0] \cdot [\![\boldsymbol{a}^{(0)}]\!]_k, \cdots, \boldsymbol{\gamma}^{(j)}[n-1] \cdot [\![\boldsymbol{a}^{(n-1)}]\!]_k)$, $[\![\boldsymbol{\nu}^{(j)}]\!]_k = ([\![\boldsymbol{b}^{(0)}]\!]_k, \cdots, [\![\boldsymbol{b}^{(n-1)}]\!]_k)$, and $[\![w^{(j)}]\!]_k = \sum_{i=0}^{n-1} \boldsymbol{\gamma}^{(j)}[i] \cdot [\![c^{(j)}]\!]_k$.

Step 2: Lifting Inner-Product Triples to $\mathbb{F}_p$.

1) All parties locally define $[\![\boldsymbol{\mu'}^{(j)}]\!]_p :\equiv_p [\![\boldsymbol{\mu}^{(j)}]\!]_k, [\![\boldsymbol{\nu'}^{(j)}]\!]_p :\equiv_p [\![\boldsymbol{\nu}^{(j)}]\!]_k$ and $[\![w'^{(j)}]\!]_p :\equiv_p [\![w^{(j)}]\!]_k$ by directly viewing all the shares as elements of $\mathbb{F}_p$ for all $j \in [\lambda]$.
2) The prover computes $h^{(j)} :\equiv_p \boldsymbol{\mu'}^{(j)} \cdot \boldsymbol{\nu'}^{(j)}$, and $\hat{t}^{(j)} :\equiv_p (h^{(j)} - w'^{(j)}) \cdot (2^k)^{-1} + 2$ for all $j \in [\lambda]$.
3) The prover distributes replicated shares $[\![\hat{t}^{(j)}]\!]_p$ and its bit-decomposition $[\![\hat{\boldsymbol{t}}^{(j)}]\!]_p := ([\![t_0^{(j)}]\!]_p, \cdots, [\![t_{\ell-k-1}^{(j)}]\!]_p)$ for all $j \in [\lambda]$.
4) The two verifiers check that the length of each $[\![\hat{\boldsymbol{t}}^{(j)}]\!]_p$ is $\ell - k$. Then they call $\mathcal{F}_{\text{checkzero}}$ on $[\![\hat{t}^{(j)}]\!]_p - \sum_{i\in[\ell-k]} [\![t_i^{(j)}]\!]_p \cdot 2^i$ to check if it equals zero over $\mathbb{F}_p$. If all the checks are passed, the verifiers compute $[\![h^{(j)}]\!]_p$ as $[\![w'^{(j)}]\!]_p + [\![\hat{t}^{(j)}]\!]_p \cdot 2^k$; otherwise they abort.

Step 3: Proving Inner-Product Triples over $\mathbb{F}_p$.

- ***Step 3.1: Merging Into One Inner-Product Triple.***
  1) All parties invoke $\mathcal{F}_{\text{coin}}$ to sample a random coefficient vector $\boldsymbol{\theta} \in \mathbb{F}_p^{2\lambda}$.
  2) All parties set $[\![\boldsymbol{x}]\!]_p := (\boldsymbol{\theta}[0] \cdot [\![\boldsymbol{\mu'}^{(0)}]\!]_p, \cdots, \boldsymbol{\theta}[\lambda-1] \cdot [\![\boldsymbol{\mu'}^{(\lambda-1)}]\!]_p, \boldsymbol{\theta}[\lambda] \cdot (\boldsymbol{\eta}^{(0)} \circ \hat{\boldsymbol{t}}^{(0)}), \cdots, \boldsymbol{\theta}[2\lambda-1] \cdot (\boldsymbol{\eta}^{(\lambda-1)} \circ \hat{\boldsymbol{t}}^{(\lambda-1)}))$, $[\![\boldsymbol{y}]\!]_p := ([\![\boldsymbol{\nu'}^{(0)}]\!]_p, \cdots, [\![\boldsymbol{\nu'}^{(\lambda-1)}]\!]_p, 1 - [\![\hat{\boldsymbol{t}}^{(0)}]\!]_p, \cdots, 1 - [\![\hat{\boldsymbol{t}}^{(\lambda-1)}]\!]_p)$, and $[\![z]\!]_p := \sum_{j=0}^{\lambda-1} \boldsymbol{\theta}[j] \cdot [\![h^{(j)}]\!]_p$.
- ***Step 3.2: Recursively Verifying the Inner-Product Triple.***

  Let $D := \lambda \cdot (nd + \ell - k)$. Without loss of generality, we assume D is divisible by q.
  1) All parties locally divide $[\![\boldsymbol{x}]\!]_p$ and $[\![\boldsymbol{y}]\!]_p$ each into D/q sub-vectors of size q, i.e., $[\![\boldsymbol{x}]\!]_p = ([\![\boldsymbol{x}_0]\!]_p, \cdots, [\![\boldsymbol{x}_{D/q-1}]\!]_p)$, $[\![\boldsymbol{y}]\!]_p = ([\![\boldsymbol{y}_0]\!]_p, \cdots, [\![\boldsymbol{y}_{D/q^j-1}]\!]_p)$.
  2) All parties locally compute $[\![\boldsymbol{f}]\!]_p$ and $[\![\boldsymbol{g}]\!]_p$ s.t. $[\![\boldsymbol{f}[i]]\!]_p = [\![\mathsf{Poly}(\boldsymbol{x}_i)]\!]_p$ and $[\![\boldsymbol{g}[i]]\!]_p = [\![\mathsf{Poly}(\boldsymbol{y}_i)]\!]_p$ for all $i \in [D/q]$.
  3) The prover computes $H := \boldsymbol{f} \cdot \boldsymbol{g}$ of degree $2q-2$, and share the coefficients of H to the verifiers.
  4) All parties locally compute $[\![z]\!]_p - \sum_{j=0}^{q-1} [\![H(j)]\!]_p$ and invoke $\mathcal{F}_{\text{checkzero}}$ on it to check if it is zero in $\mathbb{F}_p$.
  5) All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_p)$ to randomly sample $r \in \mathbb{F}_p$.
  6) All parties locally compute and set $[\![\boldsymbol{x}]\!]_p := [\![\boldsymbol{f}(r)]\!]_p, [\![\boldsymbol{y}]\!]_p := [\![\boldsymbol{g}(r)]\!]_p, [\![z]\!]_p := [\![h(r)]\!]_p$, and $D := D/q$.
- ***Step 3.3: Checking the Final Multiplication Triple.***

  All parties hold an inner-product triple $([\![x]\!]_p, [\![y]\!]_p, [\![z]\!]_p)$ of size 1.
  1) The prover generate a random multiplication triple $a, b, c \in \mathbb{F}_p$ where $c \equiv_p a \cdot b$, and distribute shares $[\![a]\!]_p, [\![b]\!]_p, [\![c]\!]_p$ to the verifiers.
  2) All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_p)$ to randomly sample $r \in \mathbb{F}_p$.
  3) All parties locally set $[\![x']\!]_p := [\![x]\!]_p + r \cdot [\![a]\!]_p, [\![y']\!]_p := [\![y]\!]_p + r \cdot [\![b]\!]_p$ and $[\![z']\!]_p := [\![z]\!]_p + r \cdot [\![c]\!]_p$.
  4) All parties recover x', y', z' and check if $x' \cdot y' \equiv_p z'$. If not, all parties abort, otherwise output accept.

Finally, for Step 3, the analysis follows from the standard DZKP protocol in [9], [10]. According to the results therein, as the final triple has dimension $\lambda \cdot (n \cdot d + \ell - k)$, the probability that the malicious prover passes the protocol undetected is at most $O(q \cdot \log_q(\lambda \cdot (n \cdot d + \ell - k))/|\mathbb{F}_p|)$, which is again negligible.

Combining these three parts, we conclude that the overall probability of simulation failure remains negligible.

Case 2: the prover $\mathcal{P}_j$ is honest. In this case, $\mathcal{S}$ receives the corrupted parties' shares $\{([\![\boldsymbol{a}^{(i)}]\!]_k, [\![\boldsymbol{b}^{(i)}]\!]_k, [\![c^{(i)}]\!]_k)\}_{i\in[n]}$. It works as follows.

1) $\mathcal{S}$ emulates the role of $\mathcal{F}_{\text{coin}}$ by sampling and handing random binary coefficients $\boldsymbol{\gamma}^{(0)}, \ldots, \boldsymbol{\gamma}^{(\lambda-1)}$ to $\mathcal{A}$. Then $\mathcal{S}$ follows the protocol and computes the shares of $([\![\boldsymbol{\mu}^{(j)}]\!]_k, [\![\boldsymbol{\nu}^{(j)}]\!]_k, [\![w^{(j)}]\!]_k)$ for all $j \in [\lambda]$ hold by the corrupted party.
2) $\mathcal{S}$ receives the shares of $\hat{\boldsymbol{t}}^{(j)}$ for each $j \in [\lambda]$ held by the corrupted party.
3) $\mathcal{S}$ plays the role of $\mathcal{F}_{\text{coin}}$ by handing a random length-2λ vector over $\mathbb{F}_p$ to $\mathcal{A}$.
4) $\mathcal{S}$ repeats the following procedure. $\mathcal{S}$ receives the shares of the coefficients of $[\![H]\!]_p$ of the corrupted party. Next it simulates $\mathcal{F}_{\text{coin}}$ by sending a random r to $\mathcal{A}$. Finally, $\mathcal{S}$ follows the protocol and computes the shares

$(\llbracket \boldsymbol{x} \rrbracket_p, \llbracket \boldsymbol{y} \rrbracket_p, \llbracket \boldsymbol{z} \rrbracket_p)$ the corrupted party should hold. $\mathcal{S}$ goes to the next iteration until the length of the vectors becomes 1.

5) $\mathcal{S}$ receives the corrupted party's shares of x', y' and z'. $\mathcal{S}$ simulates $\mathcal{F}_{\text{coin}}$, handing a random r to $\mathcal{A}$. Now $\mathcal{S}$ follows the protocol and computes the shares of $(\llbracket x' \rrbracket_p, \llbracket y' \rrbracket_p, \llbracket z' \rrbracket_p)$ that the corrupted party should hold. Then $\mathcal{S}$ picks a random triple (a, b, c) over $\mathbb{F}_p$ s.t. $a \cdot b \equiv_p c$. Since there is exactly one corrupted party, the shares of honest parties are fully determined by the secret value and the share of the corrupted party. Thus, $\mathcal{S}$ could calculate the honest parties' shares of $(\llbracket a \rrbracket_p, \llbracket b \rrbracket_p, \llbracket c \rrbracket_p)$. Finally, it simulates the reconstruction procedure by sending the honest parties' shares to the corrupted party. Also, it receives the corrupted party' shares of $(\llbracket x' \rrbracket_p, \llbracket y' \rrbracket_p, \llbracket z' \rrbracket_p)$ from $\mathcal{A}$. $\mathcal{S}$ follows the rest of the protocol to check the triple $(\llbracket x' \rrbracket_p, \llbracket y' \rrbracket_p, \llbracket z' \rrbracket_p)$. If the received shares are inconsistent, it sends reject to $\mathcal{F}_{\text{VrfyIP}}$; otherwise it sends accept to $\mathcal{F}_{\text{VrfyIP}}$. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Observe that the adversary's view consists of (1) shares of $\{\hat{\boldsymbol{t}}^{(j)}\}$ for $j \in [\lambda]$, shares of the coefficients of H, shares of a, b, c from the prover, (2) the revealed triple (x', y', z'). Due to the secrecy of the RSS scheme, view (1) is the same in the simulation and the real execution. As the prover is honest, the final triple in view (2) is always a random triple under the condition that $a \cdot b \equiv_p c$, and thus view (2) is also the same in both executions. □

Appendix E.
Deduction of Inequality in Sec. 5

Given that $d' \leq 2^{\ell-2k}$ and $\ell > 2k$ (or equivalently, $\ell \geq 2k+1$), we have

$$\begin{aligned}
&d'(2^k - 2) + \lceil d'/2^k \rceil + 2 \\
\leq &2^{\ell-k} - 2^{\ell-2k+1} + 2^{\ell-3k} + 2 \\
= &2^{\ell-k} + 2^{\ell-3k}(1 + 2^{1-\ell+3k} - 2^{k+1}) \\
\leq &2^{\ell-k} + 2^{\ell-3k}(1 + 2^k - 2^{k+1}) \\
< &2^{\ell-k}.
\end{aligned}$$

Appendix F.
Extending the Transformation of Inner Product Triples

In this section, we present the construction of distributed inner-product triples for $\mathcal{F}_{\text{VrfyIP}}$ from inner-product computations performed with $\mathcal{F}_{\text{IP}}$. This is a direct extension of the scalar multiplication case.

Recall that, when computing $\llbracket z \rrbracket_k := \llbracket \boldsymbol{x} \rrbracket_k \cdot \llbracket \boldsymbol{y} \rrbracket_k$, if each party $\mathcal{P}_i$ honestly computes the additive share z_i, the following equality holds from Eq. (2)

$$\begin{aligned}
&z_i - \sum_{j\in[n]} (\boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_i) - \rho_i + \rho_{i-1} \\
\equiv_k &\sum_{j\in[n]} (\boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_{i-1} + \boldsymbol{x}[j]_{i-1} \cdot \boldsymbol{y}[j]_i). \qquad (15)
\end{aligned}$$

To verify the correct execution of $\mathcal{F}_{\text{IP}}$, it suffices for each party $\mathcal{P}_i$ to show that Eq. (15) indeed holds. Let w be the LHS of Eq. (15), shared between the verifiers by $w_i :\equiv_k z_i - \sum_{j\in[n]} (\boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_i) - \rho_i$ and $w_{i-1} :\equiv_k \rho_{i-1}$. Eq. (15) can thus be expressed as

$$w \equiv_k \sum_{j\in[n]} (\boldsymbol{x}[j]_i \cdot \boldsymbol{y}[j]_{i-1} + \boldsymbol{x}[j]_{i-1} \cdot \boldsymbol{y}[j]_i). \qquad (16)$$

Let

$$\begin{aligned}
\llbracket \boldsymbol{x}' \rrbracket_k &:= (\llbracket \boldsymbol{x}[0]_i \rrbracket_k, \llbracket \boldsymbol{x}[0]_{i-1} \rrbracket_k, \ldots, \llbracket \boldsymbol{x}[n-1]_i \rrbracket_k, \llbracket \boldsymbol{x}[n-1]_{i-1} \rrbracket_k), \\
\llbracket \boldsymbol{y}' \rrbracket_k &:= (\llbracket \boldsymbol{y}[0]_{i-1} \rrbracket_k, \llbracket \boldsymbol{y}[0]_i \rrbracket_k, \ldots, \llbracket \boldsymbol{y}[n-1]_{i-1} \rrbracket_k, \llbracket \boldsymbol{y}[n-1]_i \rrbracket_k).
\end{aligned}$$

The verification of Eq. (15) is reduced to the verification of a length-$2n$ distributed inner-product triple $(\llbracket \boldsymbol{x}' \rrbracket_k, \llbracket \boldsymbol{y}' \rrbracket_k, \llbracket w \rrbracket_k)$, which is the input to $\mathcal{F}_{\text{VrfyIP}}$.