Unconditionally Secure MPC for Boolean Circuits with Constant Communication

Yubo Zeng[1,2], Kang Yang[2(✉)], Dengguo Feng[1], and Min Zhang[1]

[1] Laboratory of Trusted Computing and Information Assurance, Institute of Software, Chinese Academy of Sciences, Beijing, China
zengyubo21@mails.ucas.ac.cn
fengdg@263.net
zhangmin@iscas.ac.cn
[2] State Key Laboratory of Cryptology, Beijing, China
yangk@sklc.org

Abstract. The communication complexity of unconditionally Secure Multi-Party Computation (MPC) protocols has been studied by a series of works in the honest-majority setting. For evaluating an arbitrary Boolean circuit, the state-of-the-art MPC protocol by Goyal et al. (Crypto 2021 and Crypto 2022) achieves the total communication cost of $O(\log n)$ bits per gate, where n is the number of parties. In this work, we present the first unconditional MPC protocol for any Boolean circuit with communication of $O(1)$ bits per gate. We first construct an unconditionally secure protocol in the presence of semi-honest adversaries, and then strengthen it to guarantee security against malicious adversaries with the same communication efficiency.

1 Introduction

Secure Multi-Party Computation (MPC) [24,37,38] enables a set of n parties to jointly compute an arbitrary function on their private inputs, while preserving the privacy of the inputs and guaranteeing the correctness of the outputs. Unconditionally (or information-theoretically) secure MPC protocols [3,6,11,33] can be constructed in the honest-majority setting where the corruption threshold $t < n/2$, and guarantee security in the presence of either a semi-honest adversary who follows the protocol specification, or a malicious adversary who performs an arbitrary attack strategy. The approaches used in unconditionally secure protocols tend to be computationally much more efficient than the cryptographic machinery in the computational setting. This makes unconditionally secure protocols very attractive from a computational point of view, but they incur more communication cost than computationally secure protocols.

A lot of work is devoted to reducing the communication complexity of unconditionally secure MPC protocols. For the optimal corruption threshold $t = \lfloor (n-1)/2 \rfloor$, the work by Damgård and Nielsen [17] constructed an unconditional MPC protocol with communication of $O(n)$ elements per gate for arithmetic circuits. The concrete communication cost of their protocol is improved

in subsequent works [26,29], and the security is also strengthened from semi-honest security to active security (tolerating malicious corruptions) using the so-called distributed zero-knowledge proofs [7,8,29]. For Boolean circuits that are a better representation for comparison, equality test, truncation, and so on (compared to arithmetic circuits), the best-known work [32] achieves $O(n)$ bits of communication per gate in the presence of malicious adversaries. The linear communication complexity is necessary for unconditionally secure MPC in the optimal corruption-threshold regime [16]. To break the lower bound of communication, the recent unconditional MPC protocols [19,30,39] achieve $O(1)$ *online* communication complexity, but still require $O(n)$ communication cost in total. In this work, we focus on the total communication cost of unconditionally secure MPC protocols.

To achieve lower communication cost, the sub-optimal corruption threshold (i.e., $t = (1/2 - \epsilon)n$) for a constant $0 < \epsilon < 1/2$ has already been considered. In this setting, Packed Secret Sharing (PSS) [21] is often used for packing multiple secrets in a single sharing and computing a group of gates in parallel. Existing works focus on the case of arithmetic circuits. Specifically, prior works [5,22] constructed unconditional MPC protocols for arithmetic circuits with highly repetitive structures (e.g., SIMD circuits) with $O(1)$ elements per gate. The recent MPC protocols [27,28] develop some interesting techniques to solve the so-called network routing problem, which aims to compute input sharings of each layer using the output sharings from previous layers, and is crucial for unstructured circuits. Based on the network-routing technique, these unconditionally secure protocols achieve $O(1)$ elements of communication per gate for general (unstructured) arithmetic circuits. When these protocols are used to compute general Boolean circuits by simulating AND and XOR operations with arithmetic operations, they require $O(\log n)$ bits per gate of communication, as the size of the finite field underlying the arithmetic circuit must be at least $O(\log n)$. For Boolean circuits with the SIMD structure, one can combine Reverse Multiplication-Friendly Embedding (RMFE) [9] with PSS to achieve the communication cost of $O(1)$ bits per gate. However, it is infeasible to apply the approach to computing general (unstructured) Boolean circuits. In conclusion, the following natural question remains open (even for the case of semi-honest adversaries):

Is it possible to construct an unconditionally secure MPC protocol for evaluating a single Boolean circuit with the amortized communication complexity of $O(1)$ bits per gate?

1.1 Our Contribution

We answer the above question in the affirmative by constructing two unconditionally secure MPC protocols for any Boolean circuit with communication of $O(1)$ bits per gate in the honest-majority setting with corruption threshold $t = (1/2 - \epsilon)n$ for any constant $0 < \epsilon < 1/2$ if the width of the circuit is at least $O(n \log n)$. The first protocol is secure in the presence of semi-honest adversaries,

and the second protocol is obtained by compiling the first protocol to guarantee security against malicious adversaries.

Our unconditional MPC protocol with semi-honest security combines PSS with RMFE. The main technical contribution is a new network-routing technique (that carefully handles the compatibility between PSS and RMFE) to support the evaluation of general (unstructured) Boolean circuits. The semi-honest secure protocol is described in detail in Section 5. Here, we provide a theorem.

Theorem 1. *For any Boolean circuit C with a width of at least $O(n \log n)$ and constant $\epsilon \in (0, 1/2)$, there exists an unconditionally secure MPC protocol that securely realizes C in the presence of a semi-honest adversary controlling up to $t = (1/2 - \epsilon)n$ parties. Moreover, the amortized communication cost of the protocol is $O(1)$ bits per gate.*

Then, we compile the semi-honest protocol into an unconditional MPC protocol with active security by designing a verification protocol to detect possible malicious behaviors. In particular, we extend the distributed zero-knowledge proof (DZKP) techniques to check our packed sharing in the form $[\![f(\boldsymbol{x})]\!]$ over binary fields, where $\boldsymbol{x}$ represents the bit secret vector and f is a $\mathbb{F}_2$-linear map acting on $\boldsymbol{x}$. Consequently, the resulting actively secure protocol only introduces negligible additional overhead relative to the total communication cost, and still achieves the communication cost of $O(1)$ bits per gate. The details of the MPC protocol with active security are shown in Section 6. Here, we provide the following theorem.

Theorem 2. *For any Boolean circuit C with a width of at least $O(n \log n)$ and constant $\epsilon \in (0, 1/2)$, there exists an unconditionally secure MPC protocol that securely realizes C in the presence of a malicious adversary controlling up to $t = (1/2 - \epsilon)n$ parties. The overall additional communication cost to achieve active security is bounded by $O(\log |C|)$, where $|C|$ denotes the circuit size. Moreover, the amortized communication cost of the protocol remains $O(1)$ bits per gate, achieving the same level as the semi-honest setting.*

1.2 Related Work

RMFE-based MPC: Apart from lifting the Boolean circuit to an extension field for computation, an alternative solution is to rely on the algebraic notion of reverse multiplication-friendly embedding (RMFE) that encodes a binary vector to an extension field element by translating the coordinate-wise vector product to multiplication of extension field elements. [9] first defined RMFE and gave a construction about how to generate a family of $\mathbb{F}_q$-RMFE. [9] also constructs the first linear communication protocol for the Boolean circuit by evaluating $O(\log n)$ copies of the same circuit. [10] extends the work of [9] to the dishonest majority setting, and induces an extra overhead of $\log |C|$ when evaluating a single Boolean circuit. [32] proposes a separation approach that transforms a random sharing over $\mathbb{F}_{2^m}$ to a batch of random bit sharings, where we can

generate random bit triples with only linear communication overhead. Then [32] presents a protocol that evaluates for a single circuit without imposing any constraint on the circuit structure, such as the SIMD circuit. [20] extends the standard degree-2 RMFE to degree-D RMFE that preserves the coordinate-wise product of D vectors. [30] achieves the constant communication complexity by modifying the protocol TURBOPACK [19] into the Boolean circuit.

AG Codes-based MPC: Another line of work, introduced by Hao Chen and Ronald Cramer [12], used Algebraic Geometric (AG) codes to construct secret-sharing schemes directly over small fields. Unlike conventional linear secret-sharing schemes that require the field size to exceed the number of parties, such schemes exploit algebraic curves with many rational points to support a larger number of parties over a small field. However, the additional algebraic structure of AG codes often leads to more complex multiplication than those based on Shamir secret sharing. To reduce the resulting overhead, existing works [1,2,13,14] typically employ techniques such as Galois extensions and RMFEs to amortize the communication and computation costs.

Packed Secret Sharing-based MPC: To achieve better efficiency, [21] introduces packed secret sharing that contains multiple secrets to achieve a batch of operations by calculating only once. The most challenging part of applying the packed secret sharing scheme is network routing, i.e., putting each secret in the correct position during computations. [15] and [22] decide to transform a single circuit such that it has a highly repetitive structure, which can be viewed as the circuit containing k copies of the same small circuits. In particular, this transformation technique will result in an additional factor of $\log |C|$. The subsequent work [5,25] removes this factor to achieve overall constant communication in this special case. For evaluating a single arithmetic circuit, Goyal et al. [27] successfully restricted the types of required randomness by using Hall's Marriage Theorem, and then first achieved the constant communication overhead. [28] further optimizes the generation of randomness to [27] by generating a batch of random sharings to different sharing schemes in parallel. [19] allocates a random packed secret sharing to each multiplication gate such that it can efficiently select proper secrets from previous sharings for network routing.

Verification: As our paper works in the base field $\mathbb{F}_2$ and the extension field $\mathbb{F}_{2^m}$, we present a brief survey respectively.

- Verification over arithmetic circuits: In the honest majority setting, the most efficient check is introduced from [7] as the notion of *distributed zero-knowledge proof*, which recursively verifies multiplication triples round by round with total sublinear communication cost. A recent line of works [7,31,36] further develop this technique, leading to better efficiency and a broader range of application scenarios.
- Verification over Boolean circuits: We usually use additive sharing over the binary field, where the shares to the honest party can not determine the whole sharing. A standard method is that we use MAC to authenticate bit

sharing as in the dishonest majority setting over an extension field. Details can refer to [10,18,32,35].

2 Technical Overview

This section provides a brief overview of the starting points and techniques used in our paper. In this paper, we denote $n = 2t + 1$ to be the number of parties and allow the adversary to control at most $t' = t - k + 1$ corrupted parties for any $k = O(n) \leq t$. Our protocol achieves security with abort in the information-theoretic setting in the presence of both semi-honest and malicious adversaries.

In the following, we present and analyze the communication bottlenecks of existing schemes in Section 2.1. Then, Section 2.2 identifies the core design challenge in addressing Boolean network routing and elaborates on the technical difficulties involved. We propose our main construction in the semi-honest setting in Section 2.3. Finally, we briefly discuss how to advance our semi-honest protocol to the malicious setting in Section 2.4.

2.1 Communication Bottlenecks over Boolean Circuits

There are typically two technical routes for evaluating Boolean circuits. The first is to construct secret-sharing schemes directly over the Boolean field using AG codes [12]. This approach relies on richer algebraic structures, making operations such as multiplication and reconstruction more complex and often resulting in higher communication and computational costs. Subsequent works [1,2] have incorporated classical MPC techniques, including Beaver triples [4] and DN multiplications [17], to reduce the communication overhead. Despite these improvements, the best-known result [14] still requires $O(n)$ communication complexity.

The second route leverages extension-field MPC to evaluate Boolean circuits. A direct method embeds the field elements 0 and 1 into an extension field and then executes the corresponding extension-field MPC protocol. Nevertheless, this incurs an expansion factor of $\log n$, resulting in a cost of at least $O(\log n)$ bits per gate. An alternative approach maps Boolean vectors to extension field elements using a powerful tool known as reverse multiplication-friendly embedding (RMFE), introduced in [9], which effectively transforms parallel computations over the small field into a single computation over the extension field. Specifically, this technique allows us to evaluate circuits like the first approach while completely avoiding field expansion and additional costs.

Nevertheless, such batch execution has a major drawback: the circuit is not structured to guarantee that each layer can be evaluated in a group-by-group manner (similar to using packed secret sharings over extension fields). To address this problem, [9] restricted their framework to evaluating $O(\log n)$ copies of the same Boolean circuits in parallel. Later, [32] eliminated this single circuit limitation by converting extension field correlations into base field Beaver triples. Although both works achieved linear communication, their gate-by-gate

evaluation paradigm unfortunately suffers from a strict linear lower bound on communication.

Thus, achieving total constant communication over Boolean circuits requires more techniques for cost amortization. Two recent works [30] and [39] further employed packed secret sharing (PSS) to pack RMFE extended-field elements to reduce communication. The integration of PSS and RMFE enables parallel operations on more bit instances, thereby achieving $O(1)$ bit communication costs per addition and multiplication gate. Regrettably, adopting PSS inevitably introduces the problem of network routing, which becomes substantially more difficult in the Boolean setting. These two works fell short of providing an equally efficient network routing protocol to match their gate evaluation performance. We will elaborate on this challenge in detail in the following subsection.

2.2 Challenges of Boolean Network Routing

Unlike SIMD circuits, in arbitrary and unstructured circuits, the input PSS of the next layer may not be identical to any prior output sharing. Instead, the following challenging scenarios frequently arise:

- The required secrets in the input sharing are distributed among various preceding output sharings, and individual secrets may be duplicated in the input sharing.
- Even if the input sharing contains the exact required secrets, these secrets are not arranged in the correct order.

For example, the prior output sharings may be packed as $[\![x_1, x_2, x_3]\!]$ and $[\![x_4, x_5, x_6]\!]$, but an input sharing to the next layer requires the form of $[\![x_1, x_3, x_4]\!]$ (selection), $[\![x_1, x_1, x_5]\!]$ (repeated secrets) or $[\![x_1, x_3, x_2]\!]$ (disorder). We categorize this network routing problem into two distinct challenges, and first provide a brief review of existing solutions.

Review of Prior Routing Solutions. Typically, resolving the network routing problem requires implementing operations like selection (extracting secrets from multiple output sharings into a single input sharing) and permutation (properly reordering the input secrets). These operations can usually be transformed into a DN-style operation, except that they demand special types of randomness (similar to random double sharings in [17]). Efficiently preparing this randomness is a major bottleneck.

[15,22] introduced a circuit transformation process to limit the number of different randomness types (thereby achieving a lower amortized cost), but at the expense of adding an extra $\log |C|$ factor to the communication overhead. Other works [5] rely on highly repetitive circuit structures to generate this randomness. This bottleneck remained unresolved until [27] used Hall's Marriage Theorem to effectively minimize the type number of needed randomness. They modularized the routing process into a standardized pipeline of three operations: selection, fan-out gates (copying secrets), and permutation, first successfully achieving a total constant communication overhead for any single circuit. Following this, a

subsequent work [28] optimized the randomness generation of [27] by introducing an efficient sharing transformation technique.

Additionally, another study [19] refined the routing techniques of [27,28], but at the cost of increasing the total overhead to linear communication. Specifically, their approach relies on a preprocessing phase to insert a random value λ_α into each wire α via an individual PSS instance $[\![\lambda_\alpha \cdot \mathbf{1}]\!]$, meaning that generating these sharings inevitably incurs a linear communication overhead. This very paradigm was also adopted in [30] and [39], which explains why their total communication complexity remains strictly linear even in the Boolean setting.

Consequently, in the pursuit of Boolean constant communication, the protocols of [27,28] remain the only strategic starting points. They currently stand as the only two works that successfully attain an $O(|C|)$ communication complexity.

Incompatibility of [27,28] with Boolean Circuits. While [27,28] provide an elegant solution for extension fields, directly porting their techniques into the Boolean setting introduces several technical barriers: first, their protocols require full extension-field (denoted as $\mathbb{F}_{2^m}$ for simplicity) linear operations, but RMFE mappings only support $\mathbb{F}_2$-linearity. This barrier renders some efficient operations incompatible with Boolean circuits, such as using the Vandermonde matrix to generate random sharing.

Second, the joint employment of PSS and RMFE yields a two-tier hierarchical structure where current techniques can only function at the field element $\phi(\boldsymbol{x})$ rather than interacting with the individual bits inside $\boldsymbol{x}$. As a direct result, one cannot implement bit-level alternatives for selection, fan-out, or permutation operations, even setting aside all considerations regarding communication cost.

Furthermore, several steps in [27,28] are restricted by the field size. For instance, [28] requires $|\mathbb{F}_{2^m}| > |C|$ to assign each secret a distinct slot. This field size dependency forces prior works [30,39] to choose other frameworks like [19] to handle routing problems. Ultimately, these unresolved issues culminate in non-trivial challenges for addressing Boolean network routing, while simultaneously demanding strict efficiency optimizations to achieve constant communication.

2.3 Our Main Construction and Solution

In this subsection, we present our primary construction and network routing solutions. At a high level, our approach intricately couples and extends techniques from [27,28,32] to the Boolean settings. Concretely, we deconstruct current solutions to isolate field independent operations that can be adapted to Boolean circuits, together with translating extension field arithmetic via bit decomposition and RMFE multiplicative homomorphism. In addition, we port the three routing operations to the bit-level version and extend the sharing transformation framework in [28] to efficiently generate such randomness. Before diving into the details of our construction, we first give a quick review of the RMFE and its associated calculations.

Review of Reverse Multiplication-friendly Embedding. Let $\mathbb{F}_2^\ell$ be a vector space of $\mathbb{F}_2$ of dimension ℓ, and $\mathbb{F}_{2^m}$ be an extension field of $\mathbb{F}_2$ of degree m.

A reverse multiplication-friendly embedding (RMFE) is a pair of $\mathbb{F}_2$-linear maps (ϕ, ψ) satisfying for all two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^\ell$,

$$\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})),$$

where $\phi : \mathbb{F}_2^\ell \to \mathbb{F}_{2^m}, \psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^\ell$ and $*$ denotes the coordinate-wise product.

According to [9], it has been proved that there exists a family of RMFEs such that $\frac{m}{\ell}$ is bounded by a positive constant. Therefore, a degree-t packed secret sharing in the form $[\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t$ can group $k \cdot \ell = O(m \cdot n)$ bit secrets. Furthermore, RMFE enables the seamless lifting of Boolean circuit evaluations to arithmetic computations over extension fields, which can be summarized as follows:

1. ADD: $[\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t + [\![\phi(\boldsymbol{y}^1), \ldots, \phi(\boldsymbol{y}^k)]\!]_t \to [\![(\phi(\boldsymbol{x}^1 + \boldsymbol{y}^1), \ldots, \phi(\boldsymbol{x}^k + \boldsymbol{y}^k)]\!]_t$.
2. MULT: $[\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t \cdot [\![\phi(\boldsymbol{y}^1), \ldots, \phi(\boldsymbol{y}^k)]\!]_t \to [\![\phi(\boldsymbol{x}^1) \cdot \phi(\boldsymbol{y}^1), \ldots, \phi(\boldsymbol{x}^k) \cdot \phi(\boldsymbol{y}^k)]\!]_t$ by preparing a pair of random $([\![r_1, \ldots, r_k]\!]_t, [\![r_1, \ldots, r_k]\!]_{2t})$.

Decode and Encode. However, the intended output of our multiplication is $\phi(\boldsymbol{x}^i * \boldsymbol{y}^i)$, rather than $\phi(\boldsymbol{x}^i) \cdot \phi(\boldsymbol{y}^i)$, which exactly represents the output of a batch of bit multiplication gates in this work. Therefore, we need to transform $\phi(\boldsymbol{x}^i) \cdot \phi(\boldsymbol{y}^i)$ to $\phi(\boldsymbol{x}^i * \boldsymbol{y}^i)$. Drawing inspiration from [9,32], an extension of the "decode-then-encode" idea is proposed to facilitate this transformation.

Let $c_i = \phi(\boldsymbol{x}^i) \cdot \phi(\boldsymbol{y}^i) \in \mathbb{F}_{2^m}$ for simplicity. Our goal is to compute $\phi(\psi(c_i)) = \phi(\psi(\phi(\boldsymbol{x}^i) \cdot \phi(\boldsymbol{y}^i))) = \phi(\boldsymbol{x}^i * \boldsymbol{y}^i)$. The solution is to prepare a pair of random double sharings in a specific form $([\![r_1, \ldots, r_k]\!]_t, [\![\phi(\psi(r_1)), \ldots, \phi(\psi(r_k))]\!]_t)$ in advance, then a designated party P_1 (acting as the King party P_{king}) can reconstruct $c_i + r_i$ for all $i \in [k]$ and distribute $[\![\phi(\psi(c_1 + r_1)), \ldots, \phi(\psi(c_k + r_k))]\!]_t$ to all parties, allowing for local calculations as $\phi(\boldsymbol{x}^i * \boldsymbol{y}^i) = \phi(\psi(c_i + r_i)) - \phi(\psi(r_i))$. The remaining challenge, however, lies in efficiently generating $([\![r_1, \ldots, r_k]\!]_t, [\![\phi(\psi(r_1)), \ldots, \phi(\psi(r_k))]\!]_t)$ without compromising the constant-overhead communication.

Extended Randomness Generation Over Binary Fields. Similar to multiplication, our construction necessitates various types of correlated random sharings to mask intermediate values. Regrettably, the underlying functions (correlated to RMFE) governing these sharings typically support only $\mathbb{F}_2$-linear operations, rendering standard extension-field randomness generation methods incompatible.

Toward the efficient generation of random sharings in the form $[\![f(\boldsymbol{r})]\!]_d$ or a pair of $([\![f_1(\boldsymbol{r})]\!]_{d_1}, [\![f_2(\boldsymbol{r})]\!]_{d_2})$, where f, f_1, f_2 are $\mathbb{F}_2$-linear maps acting on secret vectors and $d, d_1, d_2 \in \{t, t + k - 1\}$, we observe that a powerful tool called "the Tensoring-up Lemma" in [9], allowing extension-field arithmetic to operate on these sharings via a folded representation. Informally speaking, the m-fold interleaved sharing tuple $[\![\boldsymbol{r}]\!]^{\times m} = ([\![f(\boldsymbol{r}_1)]\!]_d, \ldots, [\![f(\boldsymbol{r}_m)]\!]_d)$ supports $\mathbb{F}_{2^m}$-linear operations for these tuples. Hence, we can adopt the extended double-sharing generation method in [17] to generate random tuples over $\mathbb{F}_{2^m}$ and subsequently

split them into the required random sharings. This approach ensures that the amortized cost per random sharing is $O(n \cdot m)$ bits. More details are provided in Section 4.1.

Thus far, we have completed the evaluation of addition and multiplication gates with a communication complexity of $O(1)$ bits per gate. We then proceed to address the Boolean network routing problem. Our construction follows the paradigm of [27] and streamlines randomness generation by incorporating techniques from [28], thereby solving two problems. First, it removes the need to use Hall's Marriage Theorem to restrict the number of types of fan-out gates and permutation operations in order to amortize the generation cost for finite types of randomness in [27]. Second, it enables us to bypass the field-size requirements for selection operations in [28]. The fundamental reason this framework works is that routing operations can be viewed as directly operating on the secret vectors. So, as long as the involved computations are successfully converted into bit-level alternatives without loss of efficiency, the protocol can be considered to scale the secret vectors up by a factor of ℓ.

Realizing Boolean Selection. Extracting secrets from PSS is generally predicated on the assumption that the targeted secrets are distributed exactly across distinct secret slots. One primary contribution of [27] is to ensure that this assumption holds for each selection. Specifically, they used fan-out gates to secrets so that each is consumed exactly once. Through Hall's Marriage Theorem, they proved that a proper permutation exists for each duplicated sharing, which guarantees the output sharings are properly permuted to meet this property before each selection.

Our key observation is that, assuming the availability of bit-level fan-out and permutation operations, our work can be reduced to realizing bit-level selection for the case where extracted bit secrets are in different positions, since the remaining steps only contain proper circuit transformation and the use of Hall's Marriage Theorem, which are independent of the field size and can be naturally extended into the present framework, with the only difference that the underlying secret dimension expands from k to $k \cdot \ell$.

Let $\boldsymbol{y}$ be the input sharing vector and $\{[\![\phi(\boldsymbol{x}_i)]\!]_t\}_{i=1}^{k\cdot\ell}$ be the previous sharings such that the i-th secret in $\boldsymbol{y}$ equals the i-th secret in $\boldsymbol{x}_i$, i.e., $\boldsymbol{y} = \sum_{i=1}^{k\cdot\ell} \boldsymbol{e}_i * \boldsymbol{x}_i$, where $\boldsymbol{e}_i$ is the unit vector of $\mathbb{F}_2^{k\cdot\ell}$, whose i-th position is 1 and all other components are 0. In the extension field setting, a k-dimensional unit vector directly converts to a degree-$(k-1)$ PSS, a property that fails for our $k \cdot \ell$-dimensional vector. Nevertheless, note that under the coordinate-wise product, the product of higher-dimensional vectors can be partitioned into smaller blocks and then concatenated. This allows us to encode $\boldsymbol{e}_i$ to $[\![\phi(\boldsymbol{e}_i)]\!]_{k-1} = [\![\phi(\boldsymbol{e}_i^1), \ldots, \phi(\boldsymbol{e}_i^k)]\!]_{k-1}$ with $\boldsymbol{e}_i = \boldsymbol{e}_i^1 \parallel \cdots \parallel \boldsymbol{e}_i^k$ for $i \in [k \cdot \ell]$. Then we can obtain $\phi(\boldsymbol{e}_i * \boldsymbol{x}_i)$ similarly to our multiplication protocol. Meanwhile, we need to prepare a pair of random $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$ to mask the extracted secrets in each selection. An efficient selection can be summarized as follows:

1. All parties locally compute $[\![\boldsymbol{y}']\!]_{t+k-1} = \sum_{i=1}^{k\cdot\ell} [\![\phi(\boldsymbol{e}_i)]\!]_{k-1} \cdot [\![\phi(\boldsymbol{x}_i)]\!]_t + [\![\boldsymbol{r}]\!]_{t+k-1}$ and send their shares of $[\![\boldsymbol{y}']\!]_{t+k-1}$ to P_1.
2. P_1 reconstructs the secret vector $\boldsymbol{y}'$, computes $\phi(\psi(\boldsymbol{y}'))$, and then distributes $[\![\phi(\psi(\boldsymbol{y}'))]\!]_t$ to all parties.
3. All parties locally set $[\![\phi(\boldsymbol{y})]\!]_t := [\![\phi(\psi(\boldsymbol{y}'))]\!]_t - [\![\phi(\psi(\boldsymbol{r}))]\!]_t$.

Here, $\psi(\boldsymbol{y}') = (\sum_{i=1}^{k\cdot\ell} \psi(\phi(\boldsymbol{e}_i) \cdot \phi(\boldsymbol{x}_i))) + \psi(\boldsymbol{r}) = (\sum_{i=1}^{k\cdot\ell} \boldsymbol{e}_i * \boldsymbol{x}_i) + \psi(\boldsymbol{r}) = \boldsymbol{y} + \psi(\boldsymbol{r})$. Crucially, generating this type of randomness can be efficiently amortized via our extended approach described above, which restricts the communication cost of each selection operation to $O(n \cdot m)$ bits. Next, we demonstrate how to ensure that the stated assumption can be satisfied before performing each selection, and first transform the circuit to hold the following properties:

- Every output wire of the input layer and all intermediate layers is used exactly once as an input wire of a later layer (which may not be the next layer).
- The gates of the same type in each layer are divided into groups of $k \cdot \ell$.

The first property can be satisfied by using fan-out gates to copy each secret to each output wire according to the number of times it will be used in later layers. Meanwhile, the second property is achieved through inserting additional gates of the same type with public inputs in each layer. Based on these two properties, all gates in each layer are precisely grouped by PSS, and the number of all input PSS of the output layer and all intermediate layers (denoted by u) equals the number of output PSS of the input layer and all intermediate layers in the circuit. We then label all input sharings as $1, 2, \ldots, u$ and label the output sharings similarly.

Now we consider a matrix $N = (N_{i,j})_{i\in[u],j\in[k\cdot\ell]}$ where $N_{i,j} \in [u]$ is the position number of the input sharing that the j-th secret of the i-th output sharing should go to. For each $v \in [u]$, there exist exactly $k \cdot \ell$ entries $N_{i,j}$ that are equal to v. We will prove the following theorem:

Theorem 3. *Let N be a $u \times k\ell$ matrix in $\{1, \ldots, u\}^{u\times k\ell}$ such that there exist exactly $k \cdot \ell$ entries $N_{i,j}$ that are equal to v for all $v \in [u]$. Then, we can find u permutations $p_1, \ldots, p_u$ over $\{1, \ldots, k \cdot \ell\}$ such that after operating p_i on the i-th row of N, each column in the new matrix N' forms a permutation over $\{1, \ldots, u\}$. Additionally, $p_1, \ldots, p_u$ can be found efficiently, i.e., in polynomial time.*

Theorem 3 first guarantees the existence and the efficient search of each p_i and ensures that for all $v \in [k\cdot\ell]$, the v-th secrets of all output sharings need to go to different input sharings, which perfectly realizes the above assumption. Up to this point, the task of efficiently achieving bit-level selection is now equivalent to the efficient instantiation of bit-level fan-out and permutation operations, which we will elaborate on below.

Realizing Boolean Permutation and Fan-Out Gates. Since the fan-out gate or permutation can be viewed as an $\mathbb{F}_2$-linear map $f : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$, then

the corresponding operation is equivalent to transforming $[\![\phi(\boldsymbol{x})]\!]_t$ to $[\![\phi(f(\boldsymbol{x}))]\!]_t$, which can be naturally reduced to an extension of DN-style computations with a pair of random $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$:

1. All parties locally calculate $[\![\phi(\boldsymbol{x}+\boldsymbol{r})]\!]_t$ and send their shares of it to P_1.
2. P_1 reconstructs the secret vector $\boldsymbol{x}+\boldsymbol{r}$, computes $f(\boldsymbol{x}+\boldsymbol{r})$, and then distributes $[\![\phi(f(\boldsymbol{x}+\boldsymbol{r}))]\!]_t$ to all parties.
3. All parties locally set $[\![\phi(f(\boldsymbol{x}))]\!]_t := [\![\phi(f(\boldsymbol{x}+\boldsymbol{r}))]\!]_t - [\![\phi(f(\boldsymbol{r}))]\!]_t$.

Consequently, the overarching Boolean network routing task can be fully reduced to the efficient generation of such paired sharing for each required map f. One would perhaps expect to generate them by previous generation methods. Unfortunately, this path proves unviable because it relies on batch producing sharings to amortize communication costs. If the map f is required only once in the circuit, such amortization becomes inherently infeasible. Moreover, since the number of types of f can be relevant to $|C|$, pushing us to find an alternative to support such tailored randomness generation.

Generating Random $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$. To efficiently generate random sharings $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$ for each required type f, we extend the sharing transformation technique introduced in [28] to the Boolean circuit setting. At a high level, this technology achieved amortized communication by concurrently generating multiple required secret sharing instances. Specifically, they first prepared a sufficient number of random PSS instances and considered a secret matrix, where each row corresponds to an underlying secret vector in sequence. Within this structure, the number of columns equals the total number of sharing schemes, while the number of rows exceeds the maximum vector dimension required by any of the involved share algorithms. Given that the sharing algorithms consist of linear operations, the resulting sharings are inherently random whenever both the input tape and the randomness tape are fed with random secrets in each row. Then all parties can perform local parallel evaluation over these random PSS using constant vectors determined by the corresponding share algorithms, enabling each party to obtain a designated PSS that contains exactly its random shares computed by all share algorithms. Therefore, this mechanism relies fundamentally on two points:

- Identifying an efficiently generated PSS as the consumed randomness.
- The consumed randomness should support coordinate-wise multiplication with a constant vector.

In the classical setting, the original packed secret sharing scheme indeed meets both requirements; in the Boolean setting, we present an alternative paradigm combined with RMFE. First, we denote $[\![\phi(\boldsymbol{r})]\!]_t = [\![\phi(\boldsymbol{r}^1), \ldots, \phi(\boldsymbol{r}^k)]\!]_t$ as our base sharing (the consumed randomness), which has been previously demonstrated to be efficiently generated. Concurrently, for any constant vector $\boldsymbol{c} \in \mathbb{F}_2^{k\cdot\ell}$, it can be encoded into $[\![\phi(\boldsymbol{c})]\!]_{k-1} = [\![\phi(\boldsymbol{c}^1), \ldots, \phi(\boldsymbol{c}^k)]\!]_{k-1}$ similar to the selection process. Then we can execute $[\![\phi(\boldsymbol{c})]\!]_{k-1} \cdot [\![\phi(\boldsymbol{r})]\!]_t \to [\![\phi(\boldsymbol{c} * \boldsymbol{r})]\!]_{t+k-1}$ to

realize coordinate-wise multiplication with a constant vector under the map of RMFE.

However, even if the two properties are fully satisfied, adapting this generation method to the Boolean setting still encounters distinct roadblocks. A primary challenge is that this technique is applied to extension-field elements such as $\phi(\boldsymbol{r}^1)$, whereas our target implementation requires bit-level operations on the secret vectors. Compounding this, the resulting shares of $[\![\phi(f(\boldsymbol{r}))]\!]_t$ are defined over $\mathbb{F}_{2^m}$, and the main computation in each share algorithm still requires $\mathbb{F}_{2^m}$-linearity, whereas f may only support Boolean computations.

To address these issues, our solution employs bit decomposition to bypass the extension field during the primary computation by operating entirely over the base field, and then restoring the final extension-field shares via a field isomorphism. Specifically, for a series of $k \cdot \ell$ $\mathbb{F}_2$-linear maps $f_1, \ldots, f_{k\cdot\ell}$ defined by permutations or fan-out gates, and letting $\prod_p$ be the secret sharing scheme to generate random $[\![\phi(f_p(\boldsymbol{r}))]\!]_t$, the share algorithm for each $\prod_p$ can be identified as: $\mathbf{share}_p : \mathbb{F}_2^{k\cdot\ell} \times \mathbb{F}_2^{(t+1-k)\cdot m} \to \mathbb{F}_2^{n\cdot m}$ for $p \in [k \cdot \ell]$, where the dimension of random tapes and sharing space is $(t + 1 - k) \cdot m$ and $n \cdot m$ respectively.

Then we can prepare $n \cdot m$ random $[\![\phi(\boldsymbol{r}_i)]\!]_t$ for $i \in [n \cdot m]$ as consumed randomness. For a fixed $w \in [m]$, a sub-algorithm $\mathbf{share}_p^{j,w}$ that outputs the w-th bit of the share can be naturally induced from $\mathbf{share}_p$ for each party P_j and each scheme $\prod_p$. Consequently, when applying these sub-algorithms to the secret vectors of each column (still considering the matrix composed of secret vectors), we can concurrently compute the bit shares at the same position across all $k \cdot \ell$ schemes for each party. After repeating this process m times, all parties are enabled to obtain the full shares via field reconstruction.

If we want to generate a pair of random $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f_p(\boldsymbol{r}))]\!]_t)$, we can define another secret sharing scheme $\prod'$ to generate $[\![\phi(\boldsymbol{r})]\!]_t$. Then we introduce a new scheme $\prod'_p = (\prod_p, \prod')$ that outputs one $\prod_p$-sharing and one $\prod'$-sharing based on their algorithms, respectively, but uses the same secret vector.

Finally, we evaluate the communication overhead of this proposed scheme. Generating $k\cdot\ell$ random sharings for each scheme requires consuming $n\cdot m$ random $[\![\phi(\boldsymbol{r}_i)]\!]_t$, $n \cdot m$ random zero sharings to refresh secrets and m openings for each party, leading to an overall communication complexity of $O(n^2 \cdot m^2)$ bits; thus preparing a random $\prod'_p$-sharing consumes $\frac{O(n^2\cdot m^2)}{k\cdot\ell} = O(n \cdot m)$ bits.

Achieving the Overall Constant-Overhead. According to the previous analysis, an efficient generation scheme of $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$ yields efficient fan-out and permutation operations, which in turn enable efficient selection. Consequently, the communication overhead for a single sharing instance $[\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t$ in this work is bounded by $O(n\cdot m)$ bits, resulting in a communication complexity of $\frac{O(n\cdot m)}{k\cdot\ell} = O(1)$ bits per gate.

2.4 Towards Malicious Security

This section presents an external verification phase that empowers our semi-honest protocols to resist malicious adversaries. The security paradigm of [27]

has already extended the resilience against additive attacks in [23] to make it still hold for PSS-based protocols. This ensures that our main components, including the generation of a variety of random sharings, all routing operations, and multiplication, are secure up to additive attacks since the main computation is executed over $\mathbb{F}_{2^m}$, not $\mathbb{F}_2$. Hence, the deviation of a fully malicious adversary can be reduced by introducing additive errors on each wire or by distributing an inconsistent degree-t PSS. Similar to the classical setting, the parties first run the semi-honest protocol, followed by a verification protocol to check that no malicious behavior is launched before the output reconstruction, thereby ensuring malicious security.

First and foremost, the consistency check for all sharings involved in the computation is merged into the verification of multiplication and routing, so it suffices to conduct the consistency check on generated randomness. Such consistency checks typically require taking random linear combinations, which often need to be repeated sufficiently many times to minimize the soundness error for Boolean circuits. To reduce the communication cost, our solution involves aggregating the sharings $[\![f(\boldsymbol{x}_i)]\!]$ to form an m-fold interleaved structure $[\![\boldsymbol{x}]\!]^{\times m} = ([\![f(\boldsymbol{x}_1)]\!], \ldots, [\![f(\boldsymbol{x}_m)]\!])$. By the Tensoring-up Lemma in [9], we can perform $\mathbb{F}_{2^m}$-linear checking operations on each group holistically.

Multiplication verification over Boolean circuits typically doubles the communication cost, which is commonly driven by either SPDZ-like protocols or the alternative approach of repeated random bit combinations. In this work, the m-fold interleaved structure can transform verified multiplication triples, denoted as $([\![\phi(\boldsymbol{x}_i)]\!]_t, [\![\phi(\boldsymbol{y}_i)]\!]_t, [\![\phi(\boldsymbol{z}_i)]\!]_t)_{i\in[h]}$, into m-fold multiplication triples $([\![\boldsymbol{a}_j]\!]^{\times m}, [\![\boldsymbol{b}_j]\!]^{\times m}, [\![\boldsymbol{c}_j]\!]^{\times m})_{j\in[\frac{h}{m}]}$, where we can extend distributed zero-knowledge proof (DZKP) techniques to check the consistency and additive errors during multiplications.

For routing, this process does not change values. We only need to check the consistency of each pair of secrets between the front sharing and the latter sharing after network routing. Inspired by [27], we can construct $k^2 \cdot \ell^2$ lists $L(i, j, u, v), i, j \in [k], u, v \in [\ell]$ to denote the sharing pairs that the u-th bit secret of the i-th share in the first sharing should equal the v-th bit secret of the j-th share in the second sharing. For each list $L(i, j, u, v)$, we can also perform a random $\mathbb{F}_{2^m}$-linear combination check. Note that if any sharing is inconsistent during routing, this check will fail with overwhelming probability. Consequently, this check inherently covers the consistency verification of the sharings within the routing phase.

In summary, when all checks pass, the parties reconstruct the output in the same way as in the semi-honest setting. Now, to compute the verification cost, the primary overhead derives from checking the correctness of the multiplications, which can be reduced to $O(\log |C|)$ bits by employing the extended DZKP in [27]. Therefore, if the circuit size $|C|$ is sufficiently large, our protocols in the malicious setting maintain the same communication complexity as the semi-honest setting. We defer the comprehensive details to Section 6.

3 Preliminaries

3.1 Notation

Let $\mathbb{F}_q$ be a small field and s be the statistical security parameter. We denote $\mathbb{F}_q^\ell$ to be an ℓ-dimensional vector space over $\mathbb{F}_q$ and $\mathbb{F}_{q^m}$ to be an extension field of $\mathbb{F}_q$. In this paper, we consider a special case for $q = 2$ and assume that $|\mathbb{F}_{q^m}| \geq 2^s$ for simplicity.

We will use bold lower-case letters like $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ for row vectors. For two vectors $\boldsymbol{x}, \boldsymbol{y}$, $\boldsymbol{x} * \boldsymbol{y}$ denotes the coordinate-wise product as $\boldsymbol{x} * \boldsymbol{y} = (x_1 y_1, \ldots, x_\ell y_\ell)$. For a circuit C, we denote $|C|$ to be the circuit size.

Given two non-empty sets U and I, we denote U^I to be the indexed Cartesian product $\prod_{i \in I} U := \{f : I \to U\}$. If I is finite, we can consider $\prod_{i \in I} U$ as a vector space with dimension $|I|$ over U. For a non-empty set $A \subset I$, the natural projection π_A maps a tuple $(u_i)_{i \in I} \in U^I$ to $(u_i)_{i \in A} \in U^A$.

For a $(k \cdot \ell)$-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_{k \cdot \ell}) \in \mathbb{F}_2^{k \cdot \ell}$, we denote $[\![\phi(\boldsymbol{x})]\!]_d$ to be a degree-d packed secret sharing $[\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_d$, where $\boldsymbol{x}^i = (x_{(i-1) \cdot \ell + 1}, \ldots, x_{i \cdot \ell}) \in \mathbb{F}_2^\ell$.

3.2 Security Model

We use the *client-server* model in this work, where clients offer inputs to and receive outputs from the Functionality, while the servers carry out the primary computation. The servers carry out the main computation. The same model has been widely employed in [26,27,28]. Let c be the number of clients and $n = 2t+1$ be the number of servers. In this paper, the adversary A is allowed to corrupt at most $t' = t-k+1$ servers. For all clients and servers, we assume that each pair of them is connected by a secure synchronous channel. The overall communication cost is measured by the number of bits via these secure channels.

Security definition. Given an ideal Functionality F, we say that a protocol π securely realizes F if there exists a simulator S, such that for any adversary A, the output distribution in the real world is *statistically close* to the output distribution in the ideal world. We refer the reader to [23] for a formal and detailed description.

Benefits of the Client-Server Model. One benefit of the client-server model is that we only need to consider the maximum corruption, i.e., adversaries that corrupt exactly $t' = t - k + 1$ servers. For some adversary A corrupting $t'' < t-k+1$ servers, we can construct a new adversary A' that controls an additional $t - k + 1 - t''$ servers and operates as follows:

1. For a party controlled by A, A' follows A.
2. For a party not corrupted by A, but controlled by A', A' instructs it to behave as an honest party.

If a protocol is secure against A', then this protocol is also secure against A since the additional $t - k + 1 - t''$ servers behave honestly in both cases. Hence, we assume that the adversary corrupts exactly $t' = t-k+1$ servers in this paper.

3.3 Reverse Multiplication Friendly Embeddings

Definition 1 ([9]). *Let ℓ, m be two positive integers and $\mathbb{F}_q$ be a small field. A pair map (ϕ, ψ) is called an $(\ell, m)_q$ reverse multiplication-friendly embedding (RMFE) if $\phi : \mathbb{F}_q^\ell \to \mathbb{F}_{q^m}$ and $\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^\ell$ are two $\mathbb{F}_q$-linear maps that satisfy*

$$\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$$

for all vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^\ell$.

Note that $\boldsymbol{x} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\mathbf{1}))$ if we let $\boldsymbol{y} = \mathbf{1} = (1, \ldots, 1)$. Then ϕ is injective, and we can define a reverse map $\phi^{-1} : \mathbb{F}_{q^m} \to \mathbb{F}_q^\ell, x \to \psi(x \cdot \phi(\mathbf{1}))$. It is easy to check that ϕ^{-1} is also $\mathbb{F}_q$-linear and $\phi^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{F}_q^\ell$.

The work in [9] proved that there exists a family of rates with a constant upper bound RMFE, as Lemma 1 shows.

Lemma 1 ([9]). *There exists a constructive family of $(\ell, m)_2$-RMFE with $\ell \to \infty$ and $\frac{m}{\ell} \to 4.92$.*

3.4 Secret Sharing

Here we use the packed secret sharing (denoted by PSS for simplicity) from [21] over the extension field $\mathbb{F}_{2^m}$. The packed secret sharing that contains multiple secrets in one sharing is an extension of the Shamir Secret sharing scheme from [34]. Denote $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_k$ be the $n+k$ non-zero field elements over $\mathbb{F}_{2^m}$. Given a secret vector $\boldsymbol{x} = (x_1, \ldots, x_k)$, we call $(w_1, \ldots, w_n)$ a degree-d ($d \geq k-1$) packed secret sharing if there exists a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ with degree at most d satisfying $f(\alpha_i) = w_i$ for $i \in [n]$ and $f(\beta_j) = x_j$ for $j \in [k]$. We will use $[\![\boldsymbol{x}]\!]_d$ to denote a degree-d PSS and set w_i to be the share held by the party P_i. For a random $[\![\boldsymbol{x}]\!]_d$, any $d - k + 1$ shares are independent of the secret $\boldsymbol{x}$.

The PSS scheme also inherits some linear homomorphism properties, as the Shamir secret sharing scheme does:

1. **Addition:** $[\![\boldsymbol{x}]\!]_d + [\![\boldsymbol{y}]\!]_d = [\![\boldsymbol{x} + \boldsymbol{y}]\!]_d$
2. **Local Multiplication:** $[\![\boldsymbol{x}]\!]_{d_1} \cdot [\![\boldsymbol{y}]\!]_{d_2} = [\![\boldsymbol{x} * \boldsymbol{y}]\!]_{d_1+d_2}$ with $d_1 + d_2 \leq n - 1$.

We will also introduce two abstract sharing schemes for later constructions. One is the general linear secret sharing scheme (GLSSS) from [9] (with slight modifications). The other is the arithmetic secret sharing scheme (ASSS) from [1] (with slight modifications).

Definition 2 ($\mathbb{K}$-GLSSS). *Given a field $\mathbb{K}$, a general $\mathbb{K}$-linear secret sharing scheme Σ consists of the following data:*

- *A set of parties $I = \{1, \ldots, n\}$.*
- *A finite-dimensional $\mathbb{K}$-vector space Z as the secret space.*
- *A finite-dimensional $\mathbb{K}$-vector space U as the share space.*
- *A $\mathbb{K}$-linear subspace $C \subset U^I$ as the sharing space.*

- *A surjective $\mathbb{K}$-linear map $\Phi : C \to Z$, its defining map.*

Definition 3. *Let $A \subset I$ be non-empty. We call A a privacy set if the $\mathbb{K}$-linear map $(\Phi, \pi_A) : C \to Z \times \pi_A(C), x \to (\Phi(x), \pi_A(x))$ is surjective. We call A a construction set if $\pi_A(x) = 0 \Rightarrow \Phi(x) = 0$ for all $x \in C$.*

We give a brief explanation of the privacy and construction sets. The privacy property requires that the shares of A are independent of the secrets. Since the map (Φ, π_A) is surjective, then no matter what shares of parties in A are, the secrets can be all possible values in the secret space. The construction property requires that the shares in this set can only recover one secret. For a construction set A and two different sharings y, y', if $\pi_A(y) = \pi_A(y')$, then we have $\pi_A(y - y') = 0 \Rightarrow \Phi(y - y') = 0 \Rightarrow \phi(y) = \phi(y')$, which implies that these two sharings correspond to the same secret.

Lemma 2 ([9]). *Let L be an extension field of $\mathbb{K}$ of degree m. Let V be a $\mathbb{K}$-vector space. Then the following holds:*

1. *Let $\mathbb{K}^{m,m}$ denote the matrix algebra over $\mathbb{K}$ consisting of all $m \times m$ matrices with entries in $\mathbb{K}$, with the usual addition and $\mathbb{K}$-scalar multiplication. Then there is a (non-unique) injective $\mathbb{K}$-algebra morphism*

$$\Phi : L \to \mathbb{K}^{m,m}; \lambda \to \Phi^{(\lambda)}.$$

   , where a $\mathbb{K}$-algebra is a ring having the field $\mathbb{K}$ as a subring and a $\mathbb{K}$-algebra morphism $R \to S$ is a ring morphism that fixes $\mathbb{K}$. In particular, the image of L is a field isomorphic to it.
2. *Each such Φ induces an L-vector space structure on V^m by defining L-scalar multiplication, for each $\lambda \in L$, as*

$$\lambda \cdot V^m \to V^m; w \to \Phi^{(\lambda)}(w),$$

   where the action of $\mathbb{K}^{m,m}$ on V^m is natural one, i.e., multiplication of an $m \times m$ matrix with an m-column vector.
   If $\lambda \in \mathbb{K}$, it restricts to

$$\lambda \cdot V^m \to V^m; w \to \lambda \cdot w,$$

   Since Φ fixes $\mathbb{K}$. Hence, this structure is compatible with the standard $\mathbb{K}$-vector space structure on V^m, i.e., as a direct sum over V.

Proof. We leave the formal proof in Appendix D. Details refer to the proof of Lemma 13.

Corollary 1 ([9]). *Let L be an extension field of $\mathbb{K}$ of degree m and Σ be a $\mathbb{K}-GLSSS$. Then the m-fold interleaved $\mathbb{K}-GLSSS$ $\Sigma^{\times m}$ is naturally viewed as an L-GLSSS, compatible with its $\mathbb{K}$-linearity.*

Let $[x]$ denote a sharing in Σ. This Corollary allows us to define $\lambda : \Sigma^{\times m} \to \Sigma^{\times m}$ for every $\lambda \in L$ and all $[\boldsymbol{x}] = ([x_1], \ldots, [x_m]) \in \lambda : \Sigma^{\times m}$:

– for all $\lambda \in \mathbb{K}$, $\lambda \cdot [\boldsymbol{x}] = (\lambda \cdot [x_1], \ldots, \lambda \cdot [x_m])$;
– for all $\lambda_1, \lambda_2 \in L$, $\lambda_1 \cdot [\boldsymbol{x}] + \lambda_2 \cdot [\boldsymbol{x}] = (\lambda_1 + \lambda_2) \cdot [\boldsymbol{x}]$;
– for all $\lambda_1, \lambda_2 \in L$, $\lambda_1 \cdot (\lambda_2 \cdot [\boldsymbol{x}]) = (\lambda_1 \cdot \lambda_2) \cdot [\boldsymbol{x}]$.

Definition 4 ($\mathbb{F}$-ASSS). *Let $\mathbb{F}$ be a field. An $\mathbb{F}$-arithmetic secret sharing scheme Σ consists of the following data:*

- *A set of parties $I = \{1, \ldots, n\}$.*
- *A secret space $Z = \mathbb{F}^k$, where k is the number of secrets packed within Σ.*
- *A share space $Z = \mathbb{F}^\ell$, where ℓ is the number of the share size.*
- *A sharing space $Z = U^I$.*
- *An injective $\mathbb{F}$-linear map: $\boldsymbol{share} : Z \times \mathbb{F}^r \to C$, which maps a secret $\boldsymbol{x} \in Z$ and a random tape $\boldsymbol{\rho} \in \mathbb{F}^r$, to a sharing $\boldsymbol{X} = \boldsymbol{share}(\boldsymbol{x}, \boldsymbol{\rho}) \in C$. $\boldsymbol{share}$ denotes as the sharing map of Σ.*
- *A surjective $\mathbb{F}$-linear map: $\boldsymbol{rec} : C \to Z$, which takes a sharing $\boldsymbol{X} \in C$ as the input and outputs a secret $\boldsymbol{x} \in Z$. $\boldsymbol{rec}$ denotes as the reconstruction map of Σ.*

The scheme Σ satisfies $\mathbf{rec}(\mathbf{share}(\boldsymbol{x}, \boldsymbol{\rho})) = \boldsymbol{x}$ for all $\boldsymbol{x} \in Z, \boldsymbol{\rho} \in \mathbb{F}^r$. We may refer to Σ as the tuple $\Sigma = (n, Z, U, C, \mathbf{share}, \mathbf{rec})$. We can also define the privacy and reconstruction set as GLSSS does similarly.

Definition 5. *Let $A \subset I$ be non-empty. We call A a privacy set if for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in Z$ and $\boldsymbol{u} \in U^A$,*

$$Pr_{\boldsymbol{\rho}}(\pi_A(\boldsymbol{share}(\boldsymbol{x}_1, \boldsymbol{\rho})) = \boldsymbol{u}) = Pr_{\boldsymbol{\rho}}(\pi_A(\boldsymbol{share}(\boldsymbol{x}_2, \boldsymbol{\rho})) = \boldsymbol{u}).$$

We call A a construction set if there exists an F-linear map $\boldsymbol{rec}_A : \pi_A(C) \to Z$, such that for all $\boldsymbol{X} \in C$,

$$\boldsymbol{rec}_A(\pi_A(\boldsymbol{X})) = \boldsymbol{rec}(\boldsymbol{X}).$$

We can also extend the threshold into the $\mathbb{F}$-ASSS. Concretely, given a positive number $t < n$, a threshold-t $\mathbb{F}$-ASSS satisfies that a subset $A \subset I$ is private as long as $|A| < t$. In this work, we are also interested in doing scalar multiplication on the $\mathbb{F}$-ASSS, therefore, we need the following Multiplication-friendly property.

Definition 6. *We say $\Sigma = (n, Z = \mathbb{F}^k, U, C, \boldsymbol{share}, \boldsymbol{rec})$ is multiplication-friendly if there exists another $\mathbb{F}$-ASSS $\Sigma' = (n, Z = \mathbb{F}^k, U', C', \boldsymbol{share}', \boldsymbol{rec}')$ and n functions $\{f_i : \mathbb{F}^k \times U \to U'\}_{i=1}^n$ such that for all $\boldsymbol{c} \in \mathbb{F}^k$ and for all $\boldsymbol{X} \in U$,*

- *$\boldsymbol{Y} = (f_1(\boldsymbol{c}, X_1), \ldots, f_n(\boldsymbol{c}, X_n)) \in C'$ is a sharing of Σ'.*
- *$\boldsymbol{rec}'(\boldsymbol{Y}) = \boldsymbol{c} * \boldsymbol{rec}(\boldsymbol{X})$.*

Intuitively, for a multiplication-friendly scheme Σ, if all parties hold a public vector $\boldsymbol{c} \in \mathbb{F}^k$ and a sharing of Σ with the secret $\boldsymbol{x}$, they can locally compute a sharing of Σ' with the secret $\boldsymbol{c} * \boldsymbol{x}$. In addition, a threshold-t $\mathbb{F}$-ASSS Σ with multiplication-friendly property implies that its corresponding scheme Σ' holds threshold t as well.

Lemma 3. *If Σ is a multiplication-friendly threshold-t $\mathbb{F}$-ASSS, and Σ' being the $\mathbb{F}$-ASSS defined in Definition 6, then Σ' holds threshold t.*

Proof. We leave the formal proof in Appendix D. Details refer to the proof of Lemma 12.

3.5 Building Blocks

In this paper, we need some Functionalities that will be used in our protocol. Since most of them are standard Functionalities that are used to evaluate over arithmetic circuits, we only give a brief description of these Functionalities in this Section. Details and proofs are formalized in Appendix A. Let $\mathcal{F}_{\mathsf{zero}}$ be an ideal Functionality that distributes a random packed zero sharing over $\mathbb{F}_{2^m}$ to all parties. Let $\mathcal{F}_{\mathsf{coin}}$ be an ideal Functionality that samples a random element $r \in \mathbb{F}_{2^m}$ to all parties. Let $\mathcal{F}_{\mathsf{verifymult}}$ be an ideal Functionality introduced from [27] that checks the correctness of all packed multiplication tuples during computation.

4 Generating Random Sharings

This section presents our efficient construction for generating randomness used in this work. At a high level, the required randomness can be categorized into two types: the first needs to be generated in bulk, whereas the second is demanded in limited amounts per instance. To minimize communication overhead, we propose an amortized approach in Section 4.1 for the first category, and extend the sharing transformation technique of [28] in the subsequent section to generate a batch of types for the second category.

4.1 Extended Randomness Generation

For each type in the first category of randomness, we use an abstract secret-sharing scheme called the general linear secret sharing scheme (GLSSS) to avoid defining redundant ideal functionalities and constructing their corresponding realized protocols in this subsection. Since all the required randomness in the first category is in the form of $[\![f(\boldsymbol{r})]\!]_d = [\![f(\boldsymbol{r}^1), \ldots, f(\boldsymbol{r}^k)]\!]_d$, where $f : \mathbb{F}_2^\ell \to \mathbb{F}_{2^m}$ or $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ is a $\mathbb{F}_2$-linear map and $d \in \{t, t+k-1\}$, their underlying schemes inherently constitute many $\mathbb{F}_2$-GLSSS.

Given a set $A \subset I$ and a set of shares $(a_i)_{i\in A}$ held by parties in A, we denote $\Sigma(A, (a_i)_{i\in A}) = \{[x] | [x] \in \Sigma, \pi_A([x]) = (a_i)_{i\in A}\}$ and define $\mathcal{F}_{\mathsf{rand}}(f, d)$ to be the ideal Functionality that distributes a random $[\![f(\boldsymbol{r})]\!]_d$ to all parties. However, generating random $[\![f(\boldsymbol{r})]\!]_d$ poses a challenge that f only supports $\mathbb{F}_2$-linearity, making it incompatible with the algebraic operations required over $\mathbb{F}_{2^m}$.

To overcome this barrier, according to Corollary 1, we can leverage the m-fold interleaved tuple $\Sigma_f^{\times m} = ([\![f(\boldsymbol{r}_1)]\!], \ldots, [\![f(\boldsymbol{r}_m)]\!])$ to perform $\mathbb{F}_{2^m}$-linear operations. This enables us to extract such random tuples via a Vandermonde matrix over $\mathbb{F}_{2^m}$, which can subsequently be split into the desired individual sharing instances. Now we illustrate $\mathcal{F}_{\mathsf{rand}}(f, d)$ in Figure 1 and present our generation protocol $\Pi_{\mathsf{extend\text{-}rand}}$ in Figure 2.

Theorem 4. *Protocol $\Pi_{\mathsf{extend\text{-}rand}}$ shown in Figure 4 securely realizes the Functionality $\mathcal{F}_{\mathsf{rand}}(f,d)$ shown in Figure 1 against a semi-honest adversary controlling $t-k+1$ parties.*

Proof. Let A be the adversary. We construct a simulator S to simulate the behaviors of honest parties. **Simulation for $\Pi_{\mathsf{extend\text{-}rand}}$:** In the first step, for each honest party P_i, for each sharing in the m-fold interleaved tuple, S samples $(t-k+1)$ field random elements $(a_j)_{j\in C}$ as the shares of corrupted parties such that $\Sigma_{f,d}(C,(a_j)_{j\in C})$ is non-empty, and sends them to the adversary. For each corrupted P_i, for each $u\in[m]$, S receives the shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ held by honest parties. Note that S learns at least $(d+1)$ shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$, which determines the whole sharing. S computes the shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ held by corrupted parties.

In the second step, for all $j\in[n-t']$ and $u\in[m]$, S computes the shares of each $[\![f(\boldsymbol{r}_{j,u})]\!]_d$ held by corrupted parties and passes these shares to $\mathcal{F}_{\mathsf{rand}}(f,d)$.

Now we claim that S perfectly simulates the behaviors of honest parties. Consider the following hybrid arguments:

Hybrid$_0$: This is the real world execution.

Hybrid$_1$: In this hybrid, S changes the way of preparing a random sharing for each honest party P_i:

- S first randomly samples the shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ as $(a_j)_{j\in C}$ such that $\Sigma_{f,d}(C,(a_j)_{j\in C})$ is non-empty.
- Then, S samples a random $[\![f(\boldsymbol{s}_{i,u})]\!]_d\in\Sigma_{f,d}$ such that $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ belongs in $\Sigma_{f,d}(C,(a_j)_{j\in C})$ and distributes the shares to the rest of honest parties.

For each corrupted P_i, S locally compute the shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ held by corrupted parties for each $u\in[m]$.

Note that this does not change the distribution of the random sharings generated by honest parties. The distribution of **Hybrid**$_0$ is identical to the distribution of **Hybrid**$_1$.

Hybrid$_2$: In this hybrid, S omits the second step when preparing a random sharing $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ for each honest party P_i in **Hybrid**$_1$. And S computes the shares of $[\![f(\boldsymbol{s}_{i,u})]\!]_d$ held by corrupted parties when P_i is corrupted. Then for each $j\in[n-t']$ and $u\in[m]$, S computes the shares of each $[\![f(\boldsymbol{r}_{j,u})]\!]_d$ held by corrupted parties and passes these shares to $\mathcal{F}_{\mathsf{rand}}(f,d)$. We next show that the distribution of **Hybrid**$_2$ is identical to the distribution of **Hybrid**$_1$.

Let M^H be the sub-matrix of M consisting of all columns of M with indices in H and M^C similarly. By the property of Vandermonde matrices, M^H is invertible. Therefore, given the m-fold interleaved sharing $\{[\![f(\boldsymbol{s}_i)]\!]_d^{\times m}\}_{i\in C}$ dealt by corrupted parties, there is a bijective between $\{[\![f(\boldsymbol{s}_i)]\!]_d^{\times m}\}_{i\in H}$ and $\{[\![f(\boldsymbol{r}_j)]\!]_d^{\times m}\}_{j\in[n-t']}$. Note that the only difference between **Hybrid**$_1$ and **Hybrid**$_2$ is that, in **Hybrid**$_1$, $\{[\![f(\boldsymbol{s}_i)]\!]_d^{\times m}\}_{i\in H}$ is randomly sampled based on the shares which have been sent to corrupted parties, while in **Hybrid**$_2$, $\mathcal{F}_{\mathsf{rand}}(f,d)$ directly generates $\{[\![f(\boldsymbol{r}_j)]\!]_d^{\times m}\}_{j\in[n-t']}$ based on the shares that corrupted parties should hold.

However, this does not change the distribution of the shares of $\{[\![f(\boldsymbol{r}_j)]\!]_d^{\times m}\}_{j\in[n-t']}$ held by honest parties. Therefore, the distribution of **Hybrid**$_2$

Functionality $\mathcal{F}_{\mathsf{rand}}(f,d)$

1. Upon receiving the shares of $s_{\ell\{\ell\in C\}}$ from corrupted parties. $\mathcal{F}_{\mathsf{rand}}(f,d)$ randomly samples a $[\![f(\boldsymbol{r})]\!]_d \in \Sigma_{f,d}$ such that $[\![f(\boldsymbol{r})]\!]_d$ belongs in $\Sigma_{f,d}(C,(s_\ell)_{\ell\in C})$.
2. $\mathcal{F}_{\mathsf{rand}}(f,d)$ sends shares of $[\![f(\boldsymbol{r})]\!]_d$ to the honest parties.

Fig. 1: **Extended Random sharings**

Protocol $\Pi_{\mathsf{extend\text{-}rand}}$

Input: Let M^T be a $n \times (n-t')$ Vandermonde matrix in $\mathbb{F}_{2^m}$.

Protocols:

1. For each $i \in [n]$, the party P_i distributes a random $\Sigma_{f,d}^{\times m}$-sharing $[\![f(\boldsymbol{s}_i)]\!]_d^{\times m}$ to all parties.
2. All parties locally compute $([\![f(\boldsymbol{r}_1)]\!]_d^{\times m},\ldots,[\![f(\boldsymbol{r}_{n-t'})]\!]_d^{\times m})^T = M \cdot ([\![f(\boldsymbol{s}_i)]\!]_d^{\times m},\ldots,[\![f(\boldsymbol{s}_n)]\!]_d^{\times m})^T$
3. For each $j \in [n-t']$, all parties decompose and then output m random $\Sigma_{f,d}$-sharings as $[\![f(\boldsymbol{r}_j)]\!]_d^{\times m} = ([\![f(\boldsymbol{r}_{j,1})]\!]_d,\ldots,[\![f(\boldsymbol{r}_{j,m})]\!]_d)$.

Fig. 2: **Protocol for generating extended randomness**

is identical to the distribution of $\mathbf{Hybrid}_1$. Note that $\mathbf{Hybrid}_2$ is the execution in the ideal world, and the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_0$, the execution in the real world. □

If we want to generate a pair of random sharings like $([\![f_1(\boldsymbol{r})]\!]_{d_1},[\![f_2(\boldsymbol{r})]\!]_{d_2})$, we can define a joint scheme $\Sigma_{\boldsymbol{f},\boldsymbol{d}} = (\Sigma_{f_1,d_1},\Sigma_{f_2,d_2})$ and call $\mathcal{F}_{\mathsf{rand}}(f,d)$ to generate a random $\Sigma_{\boldsymbol{f},\boldsymbol{d}}$-sharing. In addition, to use $\mathcal{F}_{\mathsf{rand}}(f,d)$ generate random sharings, we need to prove the scheme $\Sigma_{f,d}$ can realize sampling efficiently, i.e., given a set $A \subset I$ and a set of shares $(a_i)_{i\in A}$ held by parties in A, there exists an efficient algorithm that outputs either a random sharing in $\Sigma(A,(a_i)_{i\in A})$ or $\Sigma(A,(a_i)_{i\in A}) = \emptyset$.

For each type of randomness generated by $\mathcal{F}_{\mathsf{rand}}(f,d)$, we will provide a proof in Appendix A.3 that its scheme satisfies this requirement. In summary, generating an extended random sharing costs $\frac{n^2\cdot m^2}{(n-t')\cdot m} = O(n\cdot m)$ bits in communication.

4.2 Boolean Sharing Transformation

In this part, we employ another abstracted secret sharing scheme called the arithmetic secret sharing scheme (ASSS) to represent the second category of randomness. Most types of this randomness are used to handle fan-out and permutation operations in network routing, where certain operations may occur

Functionality $\mathcal{F}_{\mathsf{rand}}(\Pi)$

1. Upon receiving the shares of $s_{\ell\{\ell\in C\}}$ from corrupted parties. $\mathcal{F}_{\mathsf{rand}}(\Pi)$ randomly samples a Π-sharing $\boldsymbol{R}$ such that $\pi_C(R) = s_{\ell\{\ell\in C\}}$. If such a sharing does not exist, $\mathcal{F}_{\mathsf{rand}}(\Pi)$ sends **abort** to all honest parties and halts.
2. $\mathcal{F}_{\mathsf{rand}}(\Pi)$ waits for the reply from the adversary.
   - If the adversary sends **abort**, $\mathcal{F}_{\mathsf{rand}}(\Pi)$ sends **abort** to the honest parties.
   - If the adversary sends **continue**, $\mathcal{F}_{\mathsf{rand}}(\Pi)$ sends shares of $\boldsymbol{R}$ to the honest parties.

Fig. 3: **Random sharings for $\mathbb{F}_2$-ASSS**

only once. So we cannot amortize their communication overhead using the first approach.

The required randomness in the second category is all instantiated in the form $[\![\phi(f(\boldsymbol{r}))]\!]_t$, where $f : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$ is an $\mathbb{F}_2$-linear map, indicating its underlying scheme to be an $\mathbb{F}_2$-ASSS. While a wide range of schemes are involved in this work, the required quantity for each might be limited. This asymmetry naturally motivates us to consider the sharing transformation technique from [28], which has the ability to parallelize the generation of individual random sharing instances across different schemes. However, a structural challenge persists in the Boolean setting: this technique not only requires $\mathbb{F}_{2^m}$-linear operations, but also only acts on the field element $\phi(\boldsymbol{x})$, rather than on the underlying secret bit masked by ϕ.

To circumvent this obstacle, we revert our perspective to the base field. Specifically, we can canonically derive a sub-algorithm from each scheme's share algorithm. Instead of directly outputting the extension field shares, this sub-algorithm yields a single bit of the share, meaning that the required computation solely relies on $\mathbb{F}_2$-linearity to enable us to extend the sharing transformation technique to these sub-algorithms. Then we can securely output the bits at the same position across multiple schemes in parallel. After repeating this process m times, these bits can be reassembled to reconstruct the correct extension-field shares, thereby forming a random sharing instance for each required scheme.

Let $f_1, \ldots, f_{k\cdot\ell}$ be $k\cdot\ell$ $\mathbb{F}_2$-linear maps with $f_i : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$ and $\Pi_1, \ldots, \Pi_{k\cdot\ell}$ be $k\cdot\ell$ $\mathbb{F}_2$-ASSS that outputs $[\![\phi(f_i(\boldsymbol{x}))]\!]_t$ as a Π_i-sharing. We use the ideal Functionality $\mathcal{F}_{\mathsf{rand}}(\Pi)$ that distributes a random $[\![\phi(f(\boldsymbol{r}))]\!]_t$ with a $\mathbb{F}_2$-linear map f to each party. The Functionality is shown in Figure 3.

Concretely, we first prepare $n\cdot m$ random sharings as $[\![\phi(\boldsymbol{r})]\!]_t$ generated by $\mathcal{F}_{\mathsf{rand}}(f, d)$ and $n\cdot m$ random zero sharings generated by $\mathcal{F}_{\mathsf{zero}}$. Our goal is to generate a single Π_p-sharing for each $p \in [k\cdot\ell]$ in parallel. A key observation is that each bit of any share in the Π_p-sharing can be written as a $\mathbb{F}_2$-linear combination of all secrets in the p-th position of $n\cdot m$ normal random sharings. In addition, given a public vector $\boldsymbol{c} \in \mathbb{F}_2^{k\cdot\ell}$ and a PSS $[\![\phi(\boldsymbol{r})]\!]_t$, we can obtain $[\![\phi(\boldsymbol{c} * \boldsymbol{r})]\!]_t$ by encoding $\boldsymbol{c}$ into $[\![\phi(\boldsymbol{c})]\!]_{k-1}$ and perform calculations as

Protocol $\Pi_{\mathsf{specific\text{-}rand}}$

1. Let $\Pi_1, \ldots, \Pi_{k\cdot\ell}$ be $\mathbb{F}_2$-ASSS described by $\mathbb{F}_2$-linear functions $f_1, \ldots, f_{k\cdot\ell}$ mapping $\mathbb{F}_2^{k\cdot\ell}$ to $\mathbb{F}_2^{k\cdot\ell}$. For each Π_i, let $Z = \mathbb{F}_2^{k\cdot\ell}$ be the common secret space, and $\mathbf{share}_i : Z \times \mathbb{F}_2^{(t+1-k)\cdot m} \to C \subset \mathbb{F}_2^{n\cdot m}$. We have $k \cdot l + (t+1-k)\cdot m < (t+1)\cdot m < n \cdot m$.
2. All parties invoke $\mathcal{F}_{\mathsf{rand}}(f, d)$ $n \cdot m$ times to obtain random $\{[\![\phi(\boldsymbol{r}_i)]\!]_t\}_{i\in[n\cdot m]}$, where $\boldsymbol{r}_i = (r_{i,1}, \ldots, r_{i,k\cdot\ell}) \in \mathbb{F}_2^{k\cdot\ell}$. For each $p \in [k\cdot\ell]$, we denote $\boldsymbol{x}_p = (r_{1,p}, \ldots, r_{k\cdot\ell,p})$ as the input and $\boldsymbol{\rho}_p = (r_{k\cdot\ell+1,p}, \ldots, r_{k\cdot\ell+(t+1-k)\cdot m,p})$ as the random tape to Π_p.
3. All parties invoke $\mathcal{F}_{\mathsf{zero}}$ $n \cdot m$ times to obtain random $\{[\![\mathbf{0}^{j,1}]\!]_{t+k-1}, \ldots, [\![\mathbf{0}^{j,m}]\!]_{t+k-1}\}_{j\in[n]}$.
4. For all $p \in [k\cdot\ell]$, $j \in [n]$ and $w \in [m]$, let $\mathbf{share}_p^{j,w} : Z \times \mathbb{F}_2^{(t+1-k)\cdot m} \to \mathbb{F}_2$ be the $\mathbb{F}_2$-linear map induced by $\mathbf{share}_p$ that outputs the w-th bit of the j-th share in the Π_p-sharing $\mathbf{share}_p(\boldsymbol{x}_p, \boldsymbol{\rho}_p)$. Then there exists $c_{1,p}^{j,w}, \ldots, c_{n\cdot m,p}^{j,w} \in \mathbb{F}_2$ such that $\mathbf{share}_p^{j,w}(\boldsymbol{x}_p, \boldsymbol{\rho}_p) = \sum_{i=1}^{k\cdot\ell} c_{i,p}^{j,w} \cdot r_{i,p} + \sum_{i=k\cdot\ell+1}^{k\cdot\ell+(t+1-k)\cdot m} c_{i,p}^{j,w} \cdot r_{i,p}$. For all $i > k\cdot\ell + (t+1-k)\cdot m$, $c_{i,p}^{j,w} = 0$.
5. For all $j \in [n]$, $w \in [m]$ and $i \in [n\cdot m]$, we denote $\boldsymbol{c}_i^{j,w} = (c_{i,1}^{j,w}, \ldots, c_{i,k\cdot\ell}^{j,w}) \in \mathbb{F}_2^{k\cdot\ell}$. Let $u_p^{j,w} = \mathbf{share}_p^{j,w}(\boldsymbol{x}_p, \boldsymbol{\rho}_p)$ and $\boldsymbol{u}^{j,w} = (u_1^{j,w}, \ldots, u_{k\cdot\ell}^{j,w}) \in \mathbb{F}_2^{k\cdot\ell}$.
6. For all $j \in [n]$, $w \in [m]$ and $i \in [n\cdot m]$, all parties first encode $\boldsymbol{c}_i^{j,w}$ into $[\![\phi(\boldsymbol{c}_v^{j,w})]\!]_{k-1}$ and computes $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1} = \sum_{i=1}^{n\cdot m} [\![\phi(\boldsymbol{c}_i^{j,w})]\!]_{k-1} \cdot [\![\phi(\boldsymbol{r}_i)]\!]_t + [\![\mathbf{0}^{j,w}]\!]_{t+k-1}$. Parties send shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ to P_j.
7. For each $w \in [m]$, P_j reconstructs $\boldsymbol{u}'^{j,w}$ and computes $\boldsymbol{u}^{j,w} = \phi(\psi(\boldsymbol{u}'^{j,w}))$. Then for each $p \in [k\cdot\ell]$, P_j sets its share to the Π_p-sharing as $u_p^j = (u_p^{j,1}, \ldots, u_p^{j,m}) \cong \mathbb{F}_{2^m}$.

Fig. 4: **Protocol for generating specific random sharings**

$[\![\phi(\boldsymbol{c})]\!]_{k-1} \cdot [\![\phi(\boldsymbol{r})]\!]_t \to [\![\phi(\boldsymbol{c} * \boldsymbol{r})]\!]_t$. Based on the above discussion, we illustrate our construction in Protocol $\Pi_{\mathsf{specific\text{-}rand}}$ illustrated in Figure 4.

Communication Cost: To batch generate a single random sharing for $k \cdot \ell$ schemes, we require consuming $n \cdot m$ random sharings $[\![\phi(\boldsymbol{r})]\!]_t$ and $n \cdot m$ random zero sharings, together with m times sending shares to each party P_j. On average, generating one specific random sharing costs $\frac{O(n^2\cdot m^2)}{k\cdot l} = O(n \cdot m)$ bits.

Theorem 5. *For any $\mathbb{F}_2$-arithmetic secret sharing schemes $\Pi_1, \ldots, \Pi_{k\ell}$ described by $\mathbb{F}_2$-linear functions $f_1, \ldots, f_{k\cdot\ell}$ with each $f_i : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$, Protocol 4 securely realizes $\{\mathcal{F}_{\mathsf{rand}}(\Pi_p)\}_{p=1}^{k\cdot\ell}$ in the $(\mathcal{F}_{\mathsf{rand}}(f, d), \mathcal{F}_{\mathsf{zero}})$-hybrid model against a semi-honest adversary controlling $t - k + 1$ parties.*

Proof. (sketch) We only consider privacy in the semi-honest setting, where applying degree-t PSS guarantees is private against $t' = t-k+1$ corrupted parties. For correctness, each bit of the share held by P_j in the Π_p sharing can be written as a $\mathbb{F}_2$ linear combination of secrets at the p-th position. Therefore, given a

fixed $u \in [m]$, P_j can compute a single sharing that contains the u-th bit of its share to each Π_p sharing for $p \in [k \cdot \ell]$. Then P_j can collect each bit of the share in the Π_p sharing and set this share as required.

A formal proof is presented in Appendix E. □

5 Unconditionally Secure MPC Protocol against Semi-honest Adversaries

In this section, we present our unconditionally secure MPC protocol in the semi-honest setting. We first describe an efficient protocol for network routing over Boolean circuits in Section 5.1. Then, we show how to evaluate a general Boolean circuit with constant communication cost in Section 5.2.

5.1 Network Routing for Boolean Circuits

The adoption of PSS will encounter network routing problems, such as secrets in an input sharing being dispersed across multiple previous output sharings or arranged in disorder. In this work, integrating PSS with RMFE further compounds the issue due to our dual-nested secret-sharing scheme, as existing routing methods (e.g., [27,28]) either operate exclusively at the extension field level, rather than bit secrets inside ϕ, or suffer from linear complexity (e.g., [30,39]). Concurrently, a pair of RMFE mappings (ϕ, ψ) only inherently provides $\mathbb{F}_2$-linearity, resulting in an incompatibility for our main computation performed over $\mathbb{F}_{2^m}$.

To construct our Boolean routing protocol, our key observation is that operations addressing network routing, including selection, fan-out, and permutation, can be viewed as linear operations executed over a secret vector, if we abstract away their underlying computation (where selection can be considered as a linear operation over a higher-dimensional vector containing all computed output secrets). From this perspective, the underlying framework of [27,28] is fundamentally equivalent whether it handles a k-dimensional secret vector (as in extension-field cases) or a $k \cdot \ell$-dimensional secret vector (as in our setting). Therefore, the entire construction can be reduced to the bit-level realization of the three routing primitives, coupled with extensions of previous field-size-independent mechanisms.

On the other hand, RMFE inherently provides coordinate-wise multiplication-friendliness for secret vectors, thereby implementing single-bit secret operations, albeit requiring parallel execution. As for the components that rely on extension-field calculations, we can either employ RMFE as an alternative or use bit-decomposition techniques to reduce the computations to an aggregation of multiple Boolean operations.

Realizing Bit-Level Fan-Out and Permutation. For each type of fan-out or permutation operation, it can be translated to an $\mathbb{F}_2$-linear map $f : \mathbb{F}_2^{k \cdot \ell} \to \mathbb{F}_2^{k \cdot \ell}$. Then this operation is equivalent to converting a sharing $[\![\phi(\boldsymbol{x})]\!]_t$ to a sharing $[\![\phi(f(\boldsymbol{x}))]\!]_t$ and can be implemented with a pair of random $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$

Protocol $\Pi_{\mathsf{semi\text{-}convert}}$

Let $f : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$ be an $\mathbb{F}_2$-linear map that is determined by a permutation or a fan-out operation on the sharing $[\![\phi(\boldsymbol{x})]\!]_t$.

Protocols:

1. Denote Π to be an $\mathbb{F}_2$-ASSS that outputs a sharing in the form of $([\![\phi(\boldsymbol{y})]\!]_t), [\![\phi(f(\boldsymbol{y}))]\!]_t$.
2. All parties call $\mathcal{F}_{\mathsf{rand}}(\Pi)$ to obtain a pair of random $([\![\phi(\boldsymbol{r})]\!]_t, [\![\phi(f(\boldsymbol{r}))]\!]_t)$.
3. All parties locally compute $[\![\boldsymbol{z}]\!]_t = [\![\phi(\boldsymbol{x})]\!]_t + [\![\phi(\boldsymbol{r})]\!]_t$ and send shares of $[\![\boldsymbol{z}]\!]_t$ to P_1.
4. P_1 reconstructs the secret $\boldsymbol{z}$, computes $\boldsymbol{x} + \boldsymbol{r} = \psi(\boldsymbol{z} \cdot \phi(\mathbf{1}))$, and distributes $[\![\phi(f(\boldsymbol{x} + \boldsymbol{r}))]\!]_t$ to each party.
5. All parties locally set $[\![\phi(f(\boldsymbol{x}))]\!]_t = [\![\phi(f(\boldsymbol{x} + \boldsymbol{r}))]\!]_t - [\![\phi(f(\boldsymbol{r}))]\!]_t$.

Fig. 5: **Protocol for permutation and fan-out gates**

(which can be efficiently generated by $\mathcal{F}_{\mathsf{rand}}(\Pi)$). We instantiate this conversion in Protocol $\Pi_{\mathsf{semi\text{-}convert}}$ shown in Figure 5.

Realizing Bit-Level Selection. Following [27], we first consider an ideal case where the collected secrets to form an input sharing are precisely distributed at distinct positions across several previous output secret sharings. Consequently, the targeted input secret vector can be written as a combination of coordinate-wise products between each unit vector and each extracted secret vector. This allows us to encode each unit vector $\boldsymbol{e}_i \in \mathbb{F}_2^{k\cdot\ell}$ into $[\![\phi(\boldsymbol{e}_i)]\!]_{k-1}$ to facilitate such computation, with the process formalized as Protocol $\Pi_{\mathsf{semi\text{-}select}}$ and presented in Figure 6.

The next step is to ensure that whenever a selection operation is required, the extracted secrets are distributed across distinct positions, enabling us to efficiently extract them via $\Pi_{\mathsf{semi\text{-}select}}$. And this step is achieved in [27] by using fan-out and permutation operations in conjunction with proper circuit transformation. Notice that while the bit-level versions of the first two operations have already been realized in our work, the latter depends solely on the circuit topology and remains independent of the field size, thereby allowing it to be naturally extended to our setting.

Specifically, we use fan-out gates that copy each secret in a sharing multiple times, equal to the number of times it is used in later layers. Then, each output wire of the input layer and all intermediate layers is used only once as an input wire of some input layers. Then, by inserting additional gates of the same type with public inputs in each layer, the number of input wires and the number of output wires of each layer are multiples of $k \cdot \ell$, which enables us to partition the secrets of each layer into groups of size $k \cdot \ell$ and perfectly pack each group into a single PSS. This transformation will increase the circuit size by a limited constant, the details of which are referred to Appendix C.

Protocol $\Pi_{\mathsf{semi\text{-}select}}$

Let $[\![\phi(\boldsymbol{y})]\!]_t$ be the PSS that collects secrets from $\{[\![\phi(\boldsymbol{x}_i)]\!]_t\}_{i=1}^{k\cdot\ell}$, where the i-th bit secret in $\boldsymbol{y}$ equals to the i-th bit secret in $\boldsymbol{x}_i$ for all $i \in [k \cdot \ell]$.

Protocols:

1. All parties call $\mathcal{F}_{\mathsf{rand}}(f, d)$ to obtain a pair of random $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$.
2. For each unit vector $\boldsymbol{e}_i \in \mathbb{F}_2^{k\cdot\ell}$, all parties encode them to $[\![\phi(\boldsymbol{e}_i)]\!]_{k-1} = [\![\phi(\boldsymbol{e}_i^1), \ldots, \phi(\boldsymbol{e}_i^k)]\!]$, where $\boldsymbol{e}_i = \boldsymbol{e}_i^1 || \ldots || \boldsymbol{e}_i^k$ with each $\boldsymbol{e}_i^j \in \mathbb{F}_2^{\ell}$.
3. All parties locally compute $[\![\boldsymbol{y}']\!]_{t+k-1} = \sum_{i=1}^{k\cdot\ell} [\![\phi(\boldsymbol{e}_i)]\!]_{k-1} \cdot [\![\phi(\boldsymbol{x}_i)]\!]_t + [\![\boldsymbol{r}]\!]_{t+k-1}$ and send shares of $[\![\boldsymbol{y}']\!]_{t+k-1}$ to P_1.
4. P_1 reconstructs the secret vector $\boldsymbol{y}'$, computes $\phi(\psi(\boldsymbol{y}'))$, and then distributes $[\![\phi(\psi(\boldsymbol{y}'))]\!]_t$ to all other parties.
5. All parties locally set $\phi(\boldsymbol{y}) := [\![\phi(\psi(\boldsymbol{y}'))]\!]_t - [\![\phi(\psi(\boldsymbol{r}))]\!]_t$.

Fig. 6: **Protocol for selection**

The remaining part is to use combinatorial arguments to efficiently find permutations on output sharings. Denote u to be the number of output packed Shamir sharings of the input layer and all intermediate layers in the circuit. Then the number of input packed Shamir sharings of the output layer and all intermediate layers is also u. We label all input sharings as $1, 2, \ldots, u$ and do the same to output sharings. We consider a matrix $N = (N_{i,j})_{i\in[u], j\in[k\cdot\ell]}$ where $N_{i,j} \in [u]$ is the position of the input sharing that the j-th secret of the i-th output sharing will go to. For each $v \in [u]$, there exactly exist $k \cdot \ell$ entries $N_{i,j}$ that are equal to v. Now we present the following theorem.

Theorem 6 (Theorem 3, restated). *Let N be a $u \times k\ell$ matrix in $\{1, \ldots, u\}^{u\times k\ell}$ such that there exactly exists $k \cdot \ell$ entries $N_{i,j}$ that are equal to v for all $v \in [u]$. Then, we can find m permutations $p_1, \ldots, p_u$ over $\{1, \ldots, k \cdot \ell\}$ such that after operating p_i on the i-th row of N, each column in the new matrix N' forms a permutation over $\{1, \ldots, u\}$. Additionally, $p_1, \ldots, p_u$ can be found efficiently, i.e., within polynomial time.*

Owing to space limitations, we move the formal proof to Appendix B and refer the reader to Theorem 11. By Theorem 6, we can find u permutations $p_1, \ldots, p_u$ over $\{1, \ldots, k \cdot \ell\}$ where each p_i operates on the i-th output sharing immediately after its computation. Then, the secrets that form a new input sharing come from previous output sharings, which have been permuted properly. We also leave the Functionalities that formalize the three operations to Appendix E, which are denoted as $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ in Figure 28 and $\mathcal{F}_{\mathsf{semi\text{-}select}}$ in Figure 29. Finally, we summarize the whole protocol for network routing in Figure 7.

Protocol $\Pi_{\text{network-routing}}$

Circuit Transformation: Let C be a general Boolean circuit to be evaluated. All parties transform C to another circuit C' as shown in Appendix C.

Input Phase:

1. For each input sharing distributed by clients, all parties invoke $\Pi_{\text{semi-convert}}$ as a fan-out gate to copy secrets in this sharing multiple times as required.
2. For each output sharing in the input layer, all parties invoke $\Pi_{\text{semi-convert}}$ to permute secrets in this sharing with some associated permutation p determined from Theorem 6.

Evaluation Phase:

1. By Theorem 6, input sharings in this layer can be formed by selecting secrets at different positions from permuted output sharings. All parties invoke $\Pi_{\text{semi-select}}$ to obtain the input sharing for this layer.
2. For each output sharing of addition gates and multiplication gates, all parties invoke $\Pi_{\text{semi-convert}}$ as a fan-out gate to copy secrets in this sharing multiple times as required.
3. After the fan-out gate operation, all parties invoke $\Pi_{\text{semi-convert}}$ to permute secrets in this sharing with some associated permutation p'.

Output Phase: All parties invoke $\Pi_{\text{semi-select}}$ to obtain final output sharings.

Fig. 7: **Protocol for network routing**

5.2 The Main Boolean Construction

After presenting the protocol for network routing, the main computation in the semi-honest setting is a standard process with degree-t packed secret sharings. The only difference is that we need another encoding phase on each output sharing of multiplication gates. The multiplication protocol is illustrated in Figure 8. Finally, we present the semi-honest protocol $\Pi_{\text{semi-honest}}$ in Figure 9 and leave an ideal functionality $\mathcal{F}_{\text{semi-mult}}$ in Figure 27 for multiplication to Appendix C.

In summary, each operation and each sharing generated in $\Pi_{\text{semi-honest}}$ cost only $O(n \cdot m)$ bits according to some results proved in this paper. Therefore, the amortized cost during the whole evaluation per gate is $\frac{O(n \cdot m)}{k \cdot \ell} = O(1)$ bits as required.

Theorem 7 (Theorem 1, restated). *For any Boolean circuit C with width at least $O(n \log n)$ and constant $\epsilon \in (0, 1/2)$, there exists an unconditionally secure MPC protocol $\Pi_{\text{semi-honest}}$ that securely realizes C in the presence of a semi-honest adversary controlling up to $t = (1/2 - \epsilon)n$ parties. Moreover, the amortized communication cost of the protocol is $O(1)$ bits per gate.*

Protocol Π_{mult}

Let $[\![\phi(\boldsymbol{x})]\!]_t$ and $[\![\phi(\boldsymbol{y})]\!]_t$ be the input sharings of multiplication.

1. All parties invoke $\mathcal{F}_{\mathsf{rand}}(f, d)$ to obtain a pair of random $([\![r_1, \ldots, r_k]\!]_t, [\![r_1, \ldots, r_k]\!]_{2t})$.
2. All parties locally compute $[\![\boldsymbol{z}']\!]_{2t} = [\![\phi(\boldsymbol{x})]\!]_t \cdot [\![\phi(\boldsymbol{y})]\!]_t + [\![r_1, \ldots, r_k]\!]_{2t}$ and send shares of $[\![\boldsymbol{z}']\!]_{2t}$ to P_1.
3. P_1 reconstructs the secret $\boldsymbol{z}'$ and distributes $[\![\boldsymbol{z}']\!]_t$ to each party.
4. All parties locally set $[\![\boldsymbol{z}_0]\!]_t = [\![\boldsymbol{z}']\!]_t - [\![r_1, \ldots, r_k]\!]_t$.
5. All parties invoke $\mathcal{F}_{\mathsf{rand}}(f, d)$ to obtain a pair of random $([\![\boldsymbol{r}]\!]_t, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$.
6. All parties locally compute $[\![\boldsymbol{z}_0 + \boldsymbol{r}]\!]_t = [\![\boldsymbol{z}_0]\!]_t + [\![\boldsymbol{r}]\!]_t$ and send shares of $[\![\boldsymbol{z}_0 + \boldsymbol{r}]\!]_t$ to P_1.
7. P_1 reconstructs the secret $\boldsymbol{z}_0 + \boldsymbol{r}$, computes $\boldsymbol{u} = \phi(\psi(\boldsymbol{z}_0 + \boldsymbol{r}))$ and distributes $[\![\boldsymbol{u}]\!]_t$ to each party.
8. Finally, all parties locally set $[\![\phi(\boldsymbol{x}) * \phi(\boldsymbol{y})]\!]_t = [\![\boldsymbol{u}]\!]_t - [\![\phi(\psi(\boldsymbol{r}))]\!]_t$.

Fig. 8: **Protocol for multiplication**

6 Unconditionally Secure MPC Protocol against Malicious Adversaries

In this section, we construct a verification protocol, which compiles the unconditionally secure MPC protocol $\Pi_{\mathsf{semi\text{-}honest}}$ to guarantee security in the presence of malicious adversaries. At a high level, following [23,27], our main components are secure up against additive attacks in the malicious setting. Therefore, the deviation of fully malicious adversaries can be confined to introducing additive errors on each sharing or distributing an inconsistent degree-t PSS. To detect these attacks and minimize verification overhead, we reuse the Tensoring-up Lemma in [9] by grouping each m sharings into an m-fold interleave tuple, thereby allowing us to perform $\mathbb{F}_{2^m}$-linear check on our sharings.

To check the consistency of each sharing, we construct a protocol $\Pi_{\mathsf{random\text{-}check}}$ shown in Figure 10 to check for randomness generated by $\Pi_{\mathsf{extend\text{-}rand}}$ and another protocol $\Pi_{\mathsf{input\text{-}check}}$ illustrated in Figure 11 to check whether all secrets in the input sharings are the image of ϕ. Another component requiring verification is the randomness from $\Pi_{\mathsf{specific\text{-}rand}}$. Here, our key observation is that if all consumed $[\![\phi(\boldsymbol{r})]\!]_t$ are consistent (checked by $\Pi_{\mathsf{random\text{-}check}}$), then each party will obtain m consistent degree-$(t + k - 1)$ sharings containing $k \cdot \ell$ bit shares with the same position. As each sharing contains $(t + k)$ honest shares, each honest party will either reconstruct the correct shares or abort. Meanwhile, these correct shares are computed by deterministic share algorithms operating on random secrets and must maintain consistency for all honest parties. Therefore, each type of randomness generated by $\Pi_{\mathsf{specific\text{-}rand}}$ requires no additional checks, thereby preserving the communication overhead.

Protocol $\Pi_{\text{semi-honest}}$

Circuit Transformation: Let C be a general Boolean circuit to be evaluated. All parties transform C to another circuit C' as shown in Appendix C.

Input Phase:

1. The input gates to one client are divided into groups where each group contains $k \cdot \ell$ inputs. For each group, the client shares a $[\![\phi(\boldsymbol{x})]\!]_t = [\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t$, where $\boldsymbol{x} = (x_1, \ldots, x_{k \cdot \ell}) \in \mathbb{F}_2^{k \cdot \ell}$ and $\boldsymbol{x}^i = (x_{(i-1)\ell+1}, \ldots, x_{i\ell})$.
2. All parties run sub-protocol $\Pi_{\text{network-routing}}$ to perform network routing of these sharings.

Evaluation Phase:

1. From $\Pi_{\text{network-routing}}$, all parties obtain input sharings that evaluate this layer.
2. For addition gates, all parties do local calculation as $[\![\phi(\boldsymbol{z})]\!]_t = [\![\phi(\boldsymbol{x})]\!]_t + [\![\phi(\boldsymbol{y})]\!]_t$.
3. For multiplication gates, all parties invokes Π_{mult} on $[\![\phi(\boldsymbol{x})]\!]_t$ and $[\![\phi(\boldsymbol{y})]\!]_t$ to compute $[\![\phi(\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y})]\!]_t$.
4. All parties do network routing according to $\Pi_{\text{network-routing}}$ to these output sharings.

Output Phase: From $\Pi_{\text{network-routing}}$, all parties obtain final output sharings $[\![\phi(\boldsymbol{z})]\!]_t$. All parties send shares of $[\![\phi(\boldsymbol{z})]\!]_t$ to the client. The client recovers the output with the help of ψ.

Fig. 9: **The semi-honest MPC protocol with unconditional security by invoking sub-protocols $\Pi_{\text{network-routing}}$ and Π_{mult}**

For network routing, verifying consistency between pre- and post-operation values is sufficient, since it does not involve value changes. For each pair of secrets, we use the list $L(i, j, u, v), i, j \in [k], v, u \in [\ell]$ to recode a pair of sharings where the secret in the u-th bit secret of the i-th shares in the first sharing equals to the secret in the v-th bit secret of the j-th shares in the second sharing. For each list $L(i, j, u, v)$, the consistency of paired sharings in this list can be captured by $\Pi_{\text{routing-check}}$ shown in Figure 12.

These three protocols mainly rely on the following lemma.

Lemma 4. *If all parties pass the check in $\Pi_{\text{random-check}}$, $\Pi_{\text{input-check}}$, or $\Pi_{\text{routing-check}}$, then the probability that one of all checked sharings is inconsistent is within $\frac{h}{2^m \cdot m}$.*

Proof. Note that any checked sharing or paired sharing is incorrect or inconsistent, then definitely yields an incorrect m-fold interleave tuple. Supposing there are $N+1$ field elements $\alpha_1, \ldots, \alpha_N$ with $N = \frac{h}{m}$ such that random combinations executed by them can pass the check. Let $A = (a_{ij} = \alpha_{i-1}^{j-1})$ be a $N \times N$ Vandermonde matrix. Then there exists a one-to-one map between $(N+1)$ checked

Protocol $\Pi_{\mathsf{random\text{-}check}}$

Let $\{[\![f(\boldsymbol{r}_i)]\!]_d\}_{i=1}^h$ be all random sharings in one type generated by $\mathcal{F}_{\mathsf{rand}}(f,d)$.

1. All parties locally set $\frac{h}{m}$ m-fold interleave tuple as $[\![f(\boldsymbol{r}^j)]\!]_d^{\times m} = ([\![f(\boldsymbol{r}_{(j-1)m+1})]\!]_d, \ldots, [\![f(\boldsymbol{r}_{jm})]\!]_d)$ for each $j \in [\frac{h}{m}]$.
2. All parties call $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $\alpha \in \mathbb{F}_{2^m}$.
3. All parties locally compute $[\![f(\boldsymbol{r})]\!]_d^{\times m} = \sum_{j=1}^{\frac{h}{m}} \alpha^j \cdot [\![f(\boldsymbol{r}^j)]\!]_d^{\times m}$ and send shares of $[\![f(\boldsymbol{r})]\!]_d^{\times m}$ to each other.
4. Each party checks whether $[\![f(\boldsymbol{r})]\!]_d^{\times m}$ contains m valid PSS in the form $[\![f(\boldsymbol{r})]\!]_d$. If the check fails, honest parties send **abort** to all parties. If the check passes, all parties discard the first m random sharings and use the remaining for computations.

Fig. 10: **Protocol for checking randomness**

Protocol $\Pi_{\mathsf{input\text{-}check}}$

Let $\{[\![\phi(\boldsymbol{x}_i)]\!]_t\}_{i=1}^h$ be all the input sharings.

1. All parties locally set $\frac{h}{m}$ m-fold interleave tuple as $[\![\phi(\boldsymbol{x}^j)]\!]_t^{\times m} = ([\![\phi(\boldsymbol{x}_{(j-1)m+1})]\!]_t, \ldots, [\![\phi(\boldsymbol{x}_{jm})]\!]_t)$ for each $j \in [\frac{h}{m}]$.
2. All parties call $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $\alpha \in \mathbb{F}_{2^m}$ and call $\mathcal{F}_{\mathsf{rand}}(f,d)$ m times to obtain a random m-fold interleave tuple $[\![\phi(\boldsymbol{r})]\!]_t^{\times m}$.
3. All parties locally compute $[\![\phi(\boldsymbol{x})]\!]_t^{\times m} = [\![\phi(\boldsymbol{r})]\!]_t^{\times m} + \sum_{j=1}^{\frac{h}{m}} \alpha^j \cdot [\![\phi(\boldsymbol{x}^j)]\!]_t^{\times m}$ and send shares of $[\![\phi(\boldsymbol{x})]\!]_t^{\times m}$ to each other.
4. Each party checks whether $[\![\phi(\boldsymbol{x})]\!]_t^{\times m}$ contains m valid PSS in the form $[\![\phi(\boldsymbol{x})]\!]_t$. If a check fails, honest parties send **abort** to all parties.

Fig. 11: **Protocol for checking inputs**

sharings and $(N+1)$ passed sharings. However, at most $(N-1)$ sharings in the former part can pass the check, which leads to a contradiction. Therefore, the soundness error among the three protocols is bounded by $\frac{h}{2^m \cdot m}$. □

For the multiplications, we prove that the three core techniques for distributed zero-knowledge proofs are all compatible with Boolean-circuit realization. Owing to space constraints, we give an informal compilation here; full details appear in Appendix A.2. We first assume that all checked multiplication triples have already been grouped into m-fold interleave tuples. For simplicity, we use $[\![\boldsymbol{x}]\!]^{\times m}$ to denote $([\![\phi(\boldsymbol{x}_1)]\!]_t, \ldots, [\![\phi(\boldsymbol{x}_m)]\!]_t)$, and define $[\![\boldsymbol{y}]\!]^{\times m}$,$[\![\boldsymbol{z}]\!]^{\times m}$ similarly.

De-Linearization. Given h m-fold interleave multiplication tuples $([\![\boldsymbol{x}_i]\!]^{\times m}, [\![\boldsymbol{y}_i]\!]^{\times m}, [\![\boldsymbol{z}_i]\!]^{\times m})_{i\in[h]}$, all parties call $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $r \in \mathbb{F}_{2^m}$.

Protocol $\Pi_{\mathsf{routing\text{-}check}}$

For each list $L(i,j,u,v)$, we denote all paired sharings in this list as $(\llbracket\phi(\boldsymbol{x}_i)\rrbracket_t, \llbracket\phi(\boldsymbol{y}_i)\rrbracket_t)_{i\in[h]}$.

1. All parties locally set $\frac{2h}{m}$ m-fold interleave tuple as $\llbracket\phi(\boldsymbol{x}^j)\rrbracket_t^{\times m} = (\llbracket\phi(\boldsymbol{x}_{(j-1)m+1})\rrbracket_t, \ldots, \llbracket\phi(\boldsymbol{x}_{jm})\rrbracket_t)$ and $\llbracket\phi(\boldsymbol{y}^j)\rrbracket_t^{\times m} = (\llbracket\phi(\boldsymbol{y}_{(j-1)m+1})\rrbracket_t, \ldots, \llbracket\phi(\boldsymbol{y}_{jm})\rrbracket_t)$ for each $j \in [\frac{h}{m}]$.
2. All parties call $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $\alpha \in \mathbb{F}_{2^m}$ and call $\mathcal{F}_{\mathsf{rand}}(\Pi)$ m times to obtain a pair of random m-fold interleave tuple $(\llbracket\phi(\boldsymbol{r}')\rrbracket_t^{\times m}, \llbracket\phi(\boldsymbol{r}'')\rrbracket_t^{\times m})$ such that each paired of the tuple satisfies conditions for the list $L(i,j,u,v)$.
3. All parties locally compute two random linear combinations as $\llbracket\phi(\boldsymbol{x}')\rrbracket_t^{\times m} = \llbracket\phi(\boldsymbol{r}')\rrbracket_t^{\times m} + \sum_{j=1}^{\frac{h}{m}} \alpha^j \cdot \llbracket\phi(\boldsymbol{x}^j)\rrbracket_t^{\times m}$ and $\llbracket\phi(\boldsymbol{y}')\rrbracket_t^{\times m} = \llbracket\phi(\boldsymbol{r}'')\rrbracket_t^{\times m} + \sum_{j=1}^{\frac{h}{m}} \alpha^j \cdot \llbracket\phi(\boldsymbol{y}^j)\rrbracket_t^{\times m}$. All parties send shares of $\llbracket\phi(\boldsymbol{x}')\rrbracket_t^{\times m}$ and $\llbracket\phi(\boldsymbol{y}')\rrbracket_t^{\times m}$ to each other.
4. Each party reconstructs two m-fold interleave sharings and checks whether all of their m paired sharings are consistent with $L(i,j,u,v)$. If the check fails, then honest parties send **abort** to all other parties.

Fig. 12: **Protocol for checking routing**

All parties set as follows:

$$\begin{aligned}
\llbracket\boldsymbol{a}\rrbracket^{\times m} &= (\llbracket\boldsymbol{x}_1\rrbracket^{\times m}, \ldots, r^{h-1} \cdot \llbracket\boldsymbol{x}_h\rrbracket^{\times m}) \\
\llbracket\boldsymbol{b}\rrbracket^{\times m} &= (\llbracket\boldsymbol{y}_1\rrbracket^{\times m}, \ldots, \llbracket\boldsymbol{y}_h\rrbracket^{\times m}) \\
\llbracket\boldsymbol{c}\rrbracket^{\times m} &= \sum_{p\in[h]} r^{p-1} \cdot \llbracket\boldsymbol{z}_p\rrbracket^{\times m}
\end{aligned}$$

then outputs an inner product tuple $(\llbracket\boldsymbol{a}\rrbracket^{\times m}, \llbracket\boldsymbol{b}\rrbracket^{\times m}, \llbracket\boldsymbol{c}\rrbracket^{\times m})$ with dimension h.

To make the verification valid, we must ensure that the inner product between degree-t sharings is compatible with the extension-field operation $\lambda\cdot$ defined in Lemma 2. According to Lemma 2, for each $\lambda \in \mathbb{F}_{2^m}$, λ can be mapped into a $\mathbb{F}_2^{m\times m}$ matrix Φ^λ as $(\Phi_1, \ldots, \Phi_m)$. Note that each Φ_j is only $\mathbb{F}_2$-2 linear, Φ_j can be defined by m $\mathbb{F}_2$ coefficients $c_{j,1}, \ldots, c_{j,m}$, i.e., $\Phi_j(w_1, \ldots, w_m) = \sum_{p\in[m]} c_{j,p} \cdot w_p$.

Then we can compute as $(\lambda\cdot\llbracket\boldsymbol{x}_p\rrbracket^{\times m})\cdot\llbracket\boldsymbol{y}_p\rrbracket^{\times m} = (\sum_{u\in[m]} c_{1,u}\cdot\llbracket\boldsymbol{x}_{p,1}\rrbracket, \ldots, \sum_{u\in[m]} c_{m,u}\cdot \llbracket\boldsymbol{x}_{p,m}\rrbracket) \cdot (\llbracket\boldsymbol{y}_{p,1}\rrbracket, \ldots, \llbracket\boldsymbol{y}_{p,m}\rrbracket) = \sum_{u\in[m]}(\sum_{v\in[m]} c_{v,u}) \cdot (\llbracket\boldsymbol{x}_{p,u}\rrbracket \cdot \llbracket\boldsymbol{y}_{p,u}\rrbracket)$, whereas $\lambda \cdot \llbracket\boldsymbol{z}_p\rrbracket^{\times m} = \sum_{u\in[m]}(\sum_{v\in[m]} c_{v,u}) \cdot \llbracket\boldsymbol{z}_{p,u}\rrbracket$ and $\boldsymbol{z}_{p,u} = \boldsymbol{x}_{p,u} * \boldsymbol{y}_{p,u}$. Therefore, the inner-product computation can be realized by using $\phi \circ \psi$ to operate on $(\llbracket\boldsymbol{x}_{p,u}\rrbracket \cdot \llbracket\boldsymbol{y}_{p,u}\rrbracket)$ for each $u \in [m]$.

Dimension-Reduction. Let w be the compression factor and split $\llbracket\boldsymbol{a}\rrbracket^{\times m} = (\llbracket\boldsymbol{a}^1\rrbracket^{\times m}, \ldots, \llbracket\boldsymbol{a}^{\frac{h}{w}}\rrbracket^{\times m})$ and $\llbracket\boldsymbol{b}\rrbracket^{\times m} = (\llbracket\boldsymbol{b}^1\rrbracket^{\times m}, \ldots, \llbracket\boldsymbol{b}^{\frac{h}{w}}\rrbracket^{\times m})$, where $(\boldsymbol{a}^i, \boldsymbol{b}^i)_{i\in[\frac{h}{w}]}$ are vectors of dimension w (treating a m-fold interleave tuple as an unit). For each $i \in [\frac{h}{w} - 1]$, $\llbracket\boldsymbol{a}^i \odot \boldsymbol{b}^i\rrbracket^{\times m} = \sum_{j\in[w]} \llbracket\boldsymbol{a}^i_j\rrbracket^{\times m} \cdot \llbracket\boldsymbol{b}^i_j\rrbracket^{\times m} \to \llbracket\boldsymbol{c}^i\rrbracket^{\times m}$ via an exten-

sion of our multiplication protocol by preparing $([\![\boldsymbol{r}]\!]_t), ([\![\phi(\psi(\boldsymbol{r}))]\!]_t)$. Finally, all parties set $[\![\boldsymbol{c}^k]\!]^{\times m} = [\![\boldsymbol{c}]\!]^{\times m} - \sum_{i\in[\frac{h}{w}-1]} [\![\boldsymbol{c}^i]\!]^{\times m}$ and proceed to the next round.

Recursion and Randomization. After $O(\log_w h)$ times of compressions, we can reduce the dimension of inner product tuples no more than w. Then we first call $\mathcal{F}_{\mathsf{rand}}(f, d)$ $2m$ times to generate two random $[\![\boldsymbol{a}_0]\!]^{\times m}$ and $[\![\boldsymbol{b}_0]\!]^{\times m}$, and uses another multiplication to compute the inner-product of $[\![\boldsymbol{a}_0]\!]^{\times m}$ and $[\![\boldsymbol{b}_0]\!]^{\times m}$ to obtain $[\![\boldsymbol{c}_0]\!]^{\times m}$. Bringing this new random triple $([\![\boldsymbol{a}_0]\!]^{\times m}, [\![\boldsymbol{b}_0]\!]^{\times m}, [\![\boldsymbol{c}_0]\!]^{\times m})$ into the final compression, all parties obtain a final m-fold interleaved triple and check its correctness publicly.

Finally, we integrate the entire construction into the protocol Π_{verify}. If all checks pass, the whole protocol achieves statistical security in the presence of malicious adversaries controlling at most $t' = t - k + 1$ parties. In other words, we have the following lemma:

Lemma 5. *The verification phase is correct,i.e., if it accepts all sharings are computed correctly. In addition, it rejects except with negligible probability if there exists one sharing not correctly computed.*

Proof (sketch). The correctness of multiplication can be guaranteed by $\mathcal{F}_{\mathsf{verifymult}}$. And the remaining part is proved in Lemma 4.

Corollary 2. *For any Boolean circuit C with width at least $O(n \log n)$ and constant $\epsilon \in (0, 1/2)$, there exists an unconditionally secure MPC protocol that securely realizes C in the presence of a malicious adversary controlling up to $t = (1/2 - \epsilon)n$ parties. The overall additional communication cost to achieve active security is bounded by $O(\log |C|)$, where $|C|$ denotes the circuit size. Moreover, the amortized communication cost of the protocol remains $O(1)$ bits per gate, achieving the same level as the semi-honest settings.*

Communication overhead: From Appendix A.2, the protocol that securely realizes $\mathcal{F}_{\mathsf{verifymult}}$ only leads to a sublinear communication overhead, i.e., bounded by $O(\log |C|)$ bits. Taking a random linear combination requires a call to $\mathcal{F}_{\mathsf{coin}}$ and opening m sharings, which totals $O(n^2 \cdot m^2)$ communication bits. However, there are $k^2 \cdot \ell^2 = O(n^2 \cdot m^2)$ lists, leading to an overall additional overhead of $O(n^4 \cdot m^4)$ bits among $\Pi_{\mathsf{random\text{-}check}}$, $\Pi_{\mathsf{input\text{-}check}}$, and $\Pi_{\mathsf{routing\text{-}check}}$. Therefore, for sufficiently large circuits, our protocols remain at the same level as the semi-honest settings.

Acknowledgements

The authors would like to thank Yifan Song and the anonymous reviewers for their insightful comments and constructive suggestions. This work is supported by National Cyber Security-National Science and Technology Major Project (No.2026ZD1501000).

Protocol Π_{verify}

Input Phase:

1. After receiving all inputs from clients, all parties invoke $\Pi_{\mathsf{input\text{-}check}}$ to check input sharings.

Execution Phase:

1. After receiving randomness distributed by $\mathcal{F}_{\mathsf{rand}}(f, d)$, all parties invoke $\Pi_{\mathsf{random\text{-}check}}$ to check the consistency of these random sharings.
2. After computing final output sharings to the whole protocol, for checking multiplications, all parties first invoke $\mathcal{F}_{\mathsf{verifymult}}$ to check all multiplication tuples $([\![\phi(\boldsymbol{x}_i)]\!]_t, [\![\phi(\boldsymbol{y}_i)]\!]_t, [\![\phi(\boldsymbol{x}_i * \boldsymbol{y}_i)]\!]_t)_{i\in[h]}$.
3. For network routing, and for each list $L(i, j, u, v)$, all parties invoke $\Pi_{\mathsf{routing\text{-}check}}$ to check the consistency between the pre- and post-operation values of network routing.

Output Phase: If all checks pass, all parties send output sharing to the client.

Fig. 13: **Protocol for verification**

References

1. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Rambaud, M., Xing, C., Yuan, C.: Asymptotically Good Multiplicative LSSS over Galois Rings and Applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In: Advances in Cryptology—Asiacrypt 2020, Part III. pp. 151–180. LNCS(2020). https://doi.org/10.1007/978-3-030-64840-4_6
2. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois Rings. pp. 471–501. LNCS(2019). https://doi.org/10.1007/978-3-030-36030-6_19
3. Beaver, D.: Multiparty Protocols Tolerating Half Faulty Processors. In: Advances in Cryptology—Crypto 1989. pp. 560–572. LNCS(1990). https://doi.org/10.1007/0-387-34805-0_49
4. Beaver, D.: Efficient Multiparty Protocols Using Circuit Randomization. In: Advances in Cryptology—Crypto 1991. pp. 420–432. LNCS(1992). https://doi.org/10.1007/3-540-46766-1_34
5. Beck, G., Goel, A., Jain, A., Kaptchuk, G.: Order-C Secure Multiparty Computation for Highly Repetitive Circuits. In: Advances in Cryptology—Eurocrypt 2021, Part II. pp. 663–693. LNCS(2021). https://doi.org/10.1007/978-3-030-77886-6_23
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In: 20th Annual ACM Symposium on Theory of Computing (STOC). pp. 1–10. ACM Press (1988). https://doi.org/10.1145/62212.62213
7. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In: Advances in Cryptology—

Crypto 2019, Part III. LNCS, vol. 11694, pp. 67–97 (2019). https://doi.org/10.1007/978-3-030-26954-8_3
8. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. In: Advances in Cryptology—Asiacrypt 2020, Part III. pp. 244–276. LNCS(2020). https://doi.org/10.1007/978-3-030-64840-4_9
9. Cascudo, I., Cramer, R., Xing, C., Yuan, C.: Amortized Complexity of Information-Theoretically Secure MPC Revisited. In: Advances in Cryptology—Crypto 2018, Part III. LNCS, vol. 10993, pp. 395–426 (2018). https://doi.org/10.1007/978-3-319-96878-0_14
10. Cascudo, I., Gundersen, J.S.: A Secret-Sharing Based MPC Protocol for Boolean Circuits with Good Amortized Complexity. pp. 652–682. LNCS(2020). https://doi.org/10.1007/978-3-030-64378-2_23
11. Chaum, D., Crépeau, C., Damgård, I.: Multiparty Unconditionally Secure Protocols (Extended Abstract). In: 20th Annual ACM Symposium on Theory of Computing (STOC). pp. 11–19. ACM Press (1988). https://doi.org/10.1145/62212.62214
12. Chen, H., Cramer, R.: Algebraic Geometric Secret Sharing Schemes and Secure Multi-Party Computations over Small Fields. In: Advances in Cryptology—Crypto 2006. LNCS, vol. 4117, pp. 521–536 (2006). https://doi.org/10.1007/11818175_31
13. Chen, H., Cramer, R., Goldwasser, S., de Haan, R., Vaikuntanathan, V.: Secure Computation from Random Error Correcting Codes. In: Advances in Cryptology—Eurocrypt 2007. LNCS, vol. 4515, pp. 291–310 (2007). https://doi.org/10.1007/978-3-540-72540-4_17
14. Cramer, R., Rambaud, M., Xing, C.: Asymptotically-Good Arithmetic Secret Sharing over $\mathbb{Z}/p^{\ell}\mathbb{Z}$ with Strong Multiplication and Its Applications to Efficient MPC. In: Advances in Cryptology—Crypto 2021, Part III. pp. 656–686. LNCS(2021). https://doi.org/10.1007/978-3-030-84252-9_22
15. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In: Advances in Cryptology—Eurocrypt 2010. pp. 445–465. LNCS(2010). https://doi.org/10.1007/978-3-642-13190-5_23
16. Damgård, I., Larsen, K.G., Nielsen, J.B.: Communication Lower Bounds for Statistically Secure MPC, With or Without Preprocessing. In: Advances in Cryptology—Crypto 2019, Part II. LNCS, vol. 11693, pp. 61–84 (2019). https://doi.org/10.1007/978-3-030-26951-7_3
17. Damgård, I., Nielsen, J.B.: Scalable and Unconditionally Secure Multiparty Computation. In: Advances in Cryptology—Crypto 2007. LNCS, vol. 4622, pp. 572–590 (2007). https://doi.org/10.1007/978-3-540-74143-5_32
18. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: Advances in Cryptology—Crypto 2012. LNCS, vol. 7417, pp. 643–662 (2012). https://doi.org/10.1007/978-3-642-32009-5_38
19. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: TurboPack: Honest Majority MPC with Constant Online Communication. pp. 951–964. ACM Press (2022). https://doi.org/10.1145/3548606.3560633
20. Escudero, D., Hong, C., Liu, H., Xing, C., Yuan, C.: Degree-D Reverse Multiplication-Friendly Embeddings: Constructions and Applications. pp. 106–138. LNCS(2023). https://doi.org/10.1007/978-981-99-8721-4_4

21. Franklin, M.K., Yung, M.: Communication Complexity of Secure Computation (Extended Abstract). In: 24th Annual ACM Symposium on Theory of Computing (STOC). pp. 699–710. ACM Press (1992). https://doi.org/10.1145/129712.129780
22. Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits. In: Advances in Cryptology—Crypto 2015, Part II. LNCS, vol. 9216, pp. 721–741 (2015). https://doi.org/10.1007/978-3-662-48000-7_35
23. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: 46th Annual ACM Symposium on Theory of Computing (STOC). pp. 495–504. ACM Press (2014). https://doi.org/10.1145/2591796.2591861
24. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: 19th Annual ACM Symposium on Theory of Computing (STOC). pp. 218–229. ACM Press (1987). https://doi.org/10.1145/28395.28420
25. Gordon, S.D., Starin, D., Yerukhimovich, A.: The More the Merrier: Reducing the Cost of Large Scale MPC. In: Advances in Cryptology—Eurocrypt 2021, Part II. pp. 694–723. LNCS(2021). https://doi.org/10.1007/978-3-030-77886-6_24
26. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. In: Advances in Cryptology—Crypto 2021, Part II. pp. 244–274. LNCS(2021). https://doi.org/10.1007/978-3-030-84245-1_9
27. Goyal, V., Polychroniadou, A., Song, Y.: Unconditional Communication-Efficient MPC via Hall's Marriage Theorem. In: Advances in Cryptology—Crypto 2021, Part II. pp. 275–304. LNCS(2021). https://doi.org/10.1007/978-3-030-84245-1_10
28. Goyal, V., Polychroniadou, A., Song, Y.: Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing. pp. 3–32. LNCS(2022). https://doi.org/10.1007/978-3-031-15985-5_1
29. Goyal, V., Song, Y., Zhu, C.: Guaranteed Output Delivery Comes Free in Honest Majority MPC. In: Advances in Cryptology—Crypto 2020, Part II. pp. 618–646. LNCS(2020). https://doi.org/10.1007/978-3-030-56880-1_22
30. Hu, Z., Yang, K., Yu, Y.: Unconditionally Secure MPC for Boolean Circuits With Constant Online Communication. In: CSF 2024: IEEE 37th Computer Security Foundations Symposium. pp. 557–572. IEEE Computer Society Press, Enschede, The Netherlands (July 8–12, 2024). https://doi.org/10.1109/CSF61375.2024.00028
31. Li, Y., Escudero, D., Duan, Y., Huang, Z., Hong, C., Zhang, C., Song, Y.: Sublinear Distributed Product Checks on Replicated Secret-Shared Data over Z_{2^k} Without Ring Extensions. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024: 31st Conference on Computer and Communications Security. pp. 825–839. ACM Press, Salt Lake City, UT, USA (October 14–18, 2024). https://doi.org/10.1145/3658644.3690260
32. Polychroniadou, A., Song, Y.: Constant-Overhead Unconditionally Secure Multiparty Computation Over Binary Fields. In: Advances in Cryptology—Eurocrypt 2021, Part II. pp. 812–841. LNCS(2021). https://doi.org/10.1007/978-3-030-77886-6_28
33. Rabin, T., Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In: 21st Annual ACM Symposium on The-

ory of Computing (STOC). pp. 73–85. ACM Press (1989). https://doi.org/10.1145/73007.73014
34. Shamir, A.: How to Share a Secret. Communications of the ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176
35. Wang, X., Ranellucci, S., Katz, J.: Global-Scale Secure Multiparty Computation. In: ACM Conf. on Computer and Communications Security (CCS) 2017. pp. 39–56. ACM Press (2017). https://doi.org/10.1145/3133956.3133979
36. Yang, K., Wang, X.: Non-interactive Zero-Knowledge Proofs to Multiple Verifiers. pp. 517–546. LNCS(2022). https://doi.org/10.1007/978-3-031-22969-5_18
37. Yao, A.C.C.: Protocols for Secure Computations (Extended Abstract). In: 23rd Annual Symposium on Foundations of Computer Science (FOCS). pp. 160–164. IEEE (1982). https://doi.org/10.1109/SFCS.1982.38
38. Yao, A.C.C.: How to Generate and Exchange Secrets (Extended Abstract). In: 27th Annual Symposium on Foundations of Computer Science (FOCS). pp. 162–167. IEEE (1986). https://doi.org/10.1109/SFCS.1986.25
39. Zhao, M.: Honest Majority Multiparty Computation over Rings with Constant Online Communication. In: Zhou, J., Quek, T.Q.S., Gao, D., Cárdenas, A.A. (eds.) ASIACCS 24: 19th ACM Symposium on Information, Computer and Communications Security. ACM Press, Singapore (July 1–5, 2024). https://doi.org/10.1145/3634737.3661136

Supplementary Material

A Useful Building Blocks

A.1 Normal Functionalities

In this section, we present some normal Functionalities that are used in this paper, and also give a formal realization and proof of them.

Realizing $\mathcal{F}_{\mathsf{coin}}$: We denote $\mathcal{F}_{\mathsf{coin}}$ as the Functionality that outputs a random element to all parties, which is described in Figure 14. We present a protocol Π_{coin} that securely realizes $\mathcal{F}_{\mathsf{coin}}$ in Figure 15. The construction idea is simply opening a random sharing generated by $\mathcal{F}_{\mathsf{rand}}(f, d)$.

Functionality $\mathcal{F}_{\mathsf{coin}}$

1. $\mathcal{F}_{\mathsf{coin}}$ randomly samples a $r \in \mathbb{F}_{2^m}$.
2. $\mathcal{F}_{\mathsf{coin}}$ sends r to the adversary.
   - If the adversary returns **continue**, $\mathcal{F}_{\mathsf{coin}}$ sends r to the honest parties.
   - If the adversary returns **abort**, $\mathcal{F}_{\mathsf{coin}}$ sends **abort** to the honest parties.

Fig. 14: **Coin tossing**

Protocol Π_{coin}

1. All parties invoke $\mathcal{F}_{\mathsf{rand}}(f, d)$ to obtain a random $[\![r_1, \ldots, r_k]\!]_t$.
2. Each party sends its shares of $[\![r_1, \ldots, r_k]\!]_t$ to each other and checks whether all shares lie on a polynomial in $\mathbb{F}_{2^m}[X]$ with degree at most t. If the check passes for all parties, parties output $r_1, \ldots, r_k$ as random coins. Otherwise, parties send **abort** to each other.

Fig. 15: **Protocol for coin tossing**

Functionality $\mathcal{F}_{\mathsf{zero}}$

Given a positive integer $d \geq t$.

1. Upon receiving the shares of $(s_i)_{i\in[C]}$ from corrupted parties, $\mathcal{F}_{\mathsf{zero}}$ randomly samples $d - t$ elements $b_1, \ldots, b_{d-t} \in \mathbb{F}_{2^m}$ as shares of the first $d - t$ honest parties. Based on $d - k + 1$ shares and k secrets 0, $\mathcal{F}_{\mathsf{zero}}$ computes a packed secret sharing $[\![\mathbf{0}]\!]_d$.
2. $\mathcal{F}_{\mathsf{zero}}$ waits for the reply from the adversary.
   - If the adversary returns **continue**, $\mathcal{F}_{\mathsf{zero}}$ sends shares of $[\![\mathbf{0}]\!]_d$ to the honest parties.
   - If the adversary returns **abort**, $\mathcal{F}_{\mathsf{zero}}$ sends **abort** to the honest parties.

Fig. 16: **Random zero sharing**

Simulation for Π_{coin}: S first receives k random elements $r_1, \ldots, r_k$ from $\mathcal{F}_{\mathsf{coin}}$. When invoking $\mathcal{F}_{\mathsf{rand}}(f, d)$, S receives a set of shares $(s_i)_{i\in[C]}$ held by corrupted parties from the adversary. Based on these $t+1$ elements, S computes the whole sharing $[\![\boldsymbol{r}]\!]_t$, where $\boldsymbol{r} = (r_1, \ldots, r_k)$. Then, S can follow the protocol Π_{coin}. If an honest P_j sends **abort**, S sends **abort** to $\mathcal{F}_{\mathsf{coin}}$.

Note that the only difference between the real world execution and the ideal world execution is that, in the first condition, $r_1, \ldots, r_k$ is randomly sampled by $\mathcal{F}_{\mathsf{rand}}$, while in the second execution, $r_1, \ldots, r_k$ is randomly sampled by $\mathcal{F}_{\mathsf{coin}}$. Since $r_1, \ldots, r_k$ is uniformly random in both executions. Therefore, the distribution of the real world is identical to the distribution of the ideal world.

Communication & Application for $\mathcal{F}_{\mathsf{coin}}$: From Π_{coin}, generating k random elements requires one invocation of $\mathcal{F}_{\mathsf{rand}}$ and opening a random $[\![\boldsymbol{r}]\!]_t$ to all parties. Then, generating a random field element costs $\frac{\frac{n^2\cdot m}{n-t'}+n^2\cdot m}{k} = \frac{n^2\cdot m}{k} \cdot (1 + \frac{1}{n-t'}) = O(n \cdot m)$ bits. We will use $\mathcal{F}_{\mathsf{coin}}$ in our verification protocol.

Realizing $\mathcal{F}_{\mathsf{zero}}$: We denote $\mathcal{F}_{\mathsf{zero}}$ as the Functionality that generates a random packed zero sharing $[\![\mathbf{0}]\!]_d$, which is described in Figure 16. We present a protocol Π_{zero} that securely realizes $\mathcal{F}_{\mathsf{zero}}$ in Figure 17. The construction idea is similar to realizing $\mathcal{F}_{\mathsf{rand}}$.

Protocol Π_{zero}

Given a positive integer $d \geq t$.

Protocols:

1. Each party P_j distributes a random $[\![\mathbf{0}_j]\!]_d$ to other parties.
2. All parties compute $([\![\mathbf{0}^1]\!]_d, \ldots, [\![\mathbf{0}^{n-t'}]\!]_d)^T = M \cdot ([\![\mathbf{0}_1]\!]_d, \ldots, [\![\mathbf{0}_n]\!]_d)^T$ with a fixed $(n-t') \times n$ Vandermonde matrix M.
3. Parties set $[\![\mathbf{0}^1]\!]_d, \ldots, [\![\mathbf{0}^{n-t'}]\!]_d$ as $n-t'$ random degree-d packed zero sharings.

Fig. 17: **Protocol for generating zero sharing**

Simulation for Π_{zero}: When an honest party P_j wants to distribute a random $[\![\mathbf{0}_j]\!]_d$, the simulator S samples $t-k+1$ field elements as the shares of $[\![\mathbf{0}_j]\!]_d$ held by corrupted parties. For each corrupted party P_j, S receives the shares of honest parties from the adversary. S reconstructs a packed secret sharing on these $n-t'$ shares. If such a sharing does not exists or is not a packed zero sharing, S sends **abort** to $\mathcal{F}_{\mathsf{zero}}$. Otherwise, S reconstructs a packed zero sharing $[\![\mathbf{0}_j]\!]_d$. Then, S can compute the shares held by corrupted parties for each $[\![\mathbf{0}^i]\!]_d, i \in [n-t']$. S sends these shares to $\mathcal{F}_{\mathsf{zero}}$.

Now we claim that S perfectly simulates the behaviors of honest parties. Consider the following hybrid arguments:

$\mathbf{Hybrid}_0$: This is the real world execution.

$\mathbf{Hybrid}_1$: In this hybrid, S uses another methods to prepare a random sharing for each honest party P_j:

- S first randomly samples $t-k+1$ elements as the shares of $[\![\mathbf{0}_j]\!]_d$ held by corrupted parties and sends these shares to corrupted parties.
- S randomly samples $d-t$ elements as the first $d-t$ shares to honest parties. Based on these $d+1$ shares, S reconstructs the whole sharing $[\![\mathbf{0}_j]\!]_d$ and distribute it to honest parties.

For each corrupted P_j, S locally computes the shares to corrupted parties if the whole sharing can be determined. Note that S does not change the distribution of random zero sharings which are generated by honest parties. Therefore, the distribution of $\mathbf{Hybrid}_0$ is identical to $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_2$: In this hybrid, S first randomly samples $t-k+1$ elements as the shares of $[\![\mathbf{0}_j]\!]_d$ held by corrupted parties for each honest party P_j. S locally computes the shares to corrupted parties if the whole sharing can be determined for each corrupted P_j. Then, S locally computes the shares held by corrupted parties of each $[\![\mathbf{0}^i]\!]_d$ and sends these shares to $\mathcal{F}_{\mathsf{zero}}$.

Here, we prove that the distribution of $\mathbf{Hybrid}_2$ is identical to $\mathbf{Hybrid}_1$. Let M^H be the sub-matrix of M consisting of all columns of M with indices in H and M^C similarly. Then, we can rewrite as:

$$([\![\mathbf{0}^1]\!]_d, \ldots, [\![\mathbf{0}^{n-t'}]\!]_d)^T = M^H \cdot ([\![\mathbf{0}_j]\!]_d, j \in H)^T + M^C \cdot ([\![\mathbf{0}_j]\!]_d, j \in C)^T$$

M^H is an invertible $(n-t') \times (n-t')$ matrix according to the property of the Vandermonde matrix. Since $(\llbracket \mathbf{0}_j \rrbracket_d, j \in H)^T$ is uniformly random to the adversary, then we can construct a bijective between $(\llbracket \mathbf{0}^1 \rrbracket_d, \ldots, \llbracket \mathbf{0}^{n-t'} \rrbracket_d)^T$ and $(\llbracket \mathbf{0}_j \rrbracket_d, j \in H)^T$. Moreover, noticing that the only difference between $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ is that, in $\mathbf{Hybrid}_1$, $(\llbracket \mathbf{0}_j \rrbracket_d, j \in H)^T$ is randomly generated by S, while in $\mathbf{Hybrid}_2$, $\mathcal{F}_{\mathsf{zero}}$ computes a random $(\llbracket \mathbf{0}^1 \rrbracket_d, \ldots, \llbracket \mathbf{0}^{n-t'} \rrbracket_d)^T$ based on corrupted shares. Hence, the distribution of $\mathbf{Hybrid}_2$ is identical to $\mathbf{Hybrid}_1$. In summary, the distribution of the ideal world is $\mathbf{Hybrid}_2$, which is identical to $\mathbf{Hybrid}_0$ that is the real world execution.

Communication & Application for $\mathcal{F}_{\mathsf{zero}}$: From Π_{zero}, parties obtain $n-t'$ random packed secret sharings by distributing n packed secret sharings. Each random degree-d packed secret sharing costs $\frac{n^2 \cdot m}{n-t'} < 2mn = n \cdot m$ bits. In this paper, we will use $\mathcal{F}_{\mathsf{zero}}$ to generate random packed zero sharings to protect some packed secret sharings when necessary.

A.2 Realizing $\mathcal{F}_{\mathsf{verifymult}}$

In this section, we show how to realize the Functionality $\mathcal{F}_{\mathsf{verifymult}}$ that checks packed multiplication tuples. This functionality is formalized in Figure 18. Following the idea from [29, 27], we extend the *distributed zero knowledge proof* technique to Boolean settings. At a high level, we construct a recursive verification protocol that iteratively compresses the packed multiplication triples round by round. After each round of compression, the dimension of packed triples is reduced by a constant factor, and a new packed triple is generated at the end of each round. Therefore, after $O(\log h)$ rounds, we only need to verify a very limited set of packed triples if given h packed multiplication triples, for example, just one triple. To achieve our goal, we first need to realize the "compress" and "Inner-Product" operations.

We denote $H_1 \subset H$ be a fixed set of $(t+1)$ honest parties and $H_2 = H - H_1$. We denote $\llbracket \boldsymbol{x} \rrbracket_t$ to be a m-fold interleave tuple as $\llbracket \boldsymbol{x} \rrbracket_t = (\llbracket \phi(\boldsymbol{x}_i) \rrbracket_t, \ldots, \llbracket \phi(\boldsymbol{x}_m) \rrbracket_t)$. We also use $\llbracket \boldsymbol{x}^H \rrbracket_t$ to denote a packed secret sharing that is fully determined by honest shares of $\llbracket \boldsymbol{x} \rrbracket_t$ with index in H_1. We also denote $\boldsymbol{\Delta}_{\boldsymbol{x}} = \llbracket \boldsymbol{x} \rrbracket_t - \llbracket \boldsymbol{x}^H \rrbracket_t$ to be the additive errors belonging to H_2.

Moreover, we have proved that $\llbracket \boldsymbol{x} \rrbracket_t$ not only supports $\mathbb{F}_{2^m}$-linear operations but also is compatible with the inner product for secret vectors. At a high level, this sharing can be fully regarded as a degree-t packed secret sharing over $\mathbb{F}_{2^m}$, requiring only slight modifications when evaluating the multiplication gates. Consequently, distributed zero-knowledge proofs can be naturally used to verify our Boolean triples.

The description of Π_{compress} is shown in Figure 19. The "compress" operation uses another $(h-1)$ multiplications to compress h packed multiplication triples into one random packed triples. The communication complexity of Π_{compress} is $O(h \cdot n \cdot m + n^2 \cdot m)$ bits. Lemma 6 presents some properties in Π_{compress}. Later, we formalize the "Inner-Product" operation as the Functionality $\mathcal{F}_{\mathsf{inner\text{-}product}}$ and

Functionality $\mathcal{F}_{\mathsf{verifymult}}$

Given h packed multiplication triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i\in[h]}$ over $\mathbb{F}_{2^m}$

1. For each $i \in [h]$, $\mathcal{F}_{\mathsf{verifymult}}$ collects honest shares in $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)$. Based on these honest shares, $\mathcal{F}_{\mathsf{verifymult}}$ recovers three sharings as $(\llbracket \boldsymbol{x}_i^H \rrbracket_t, \llbracket \boldsymbol{y}_i^H \rrbracket_t, \llbracket \boldsymbol{z}_i^H \rrbracket_t)$ and computes $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \llbracket \boldsymbol{x}_i \rrbracket_t - \llbracket \boldsymbol{x}_i^H \rrbracket_t, \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \llbracket \boldsymbol{y}_i \rrbracket_t - \llbracket \boldsymbol{y}_i^H \rrbracket_t, \boldsymbol{\Delta}_i^{\boldsymbol{z}} = \llbracket \boldsymbol{z}_i \rrbracket_t - \llbracket \boldsymbol{z}_i^H \rrbracket_t$. $\mathcal{F}_{\mathsf{verifymult}}$ further reconstructs the secrets $\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i$ and computes $\boldsymbol{d}_i = \boldsymbol{z}_i - \boldsymbol{x}_i * \boldsymbol{y}_i$.
2. For each $i \in [h]$, $\mathcal{F}_{\mathsf{verifymult}}$ sends corrupted shares in $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)$ and the vectors $\boldsymbol{\Delta}_i^{\boldsymbol{x}}, \boldsymbol{\Delta}_i^{\boldsymbol{y}}, \boldsymbol{\Delta}_i^{\boldsymbol{z}}$ to the adversary. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$, $\mathcal{F}_{\mathsf{verifymult}}$ further sends $\boldsymbol{d}_i$ to the adversary.
3. If there exists one $i \in [h]$ such that at least one of $\boldsymbol{\Delta}_i^{\boldsymbol{x}}, \boldsymbol{\Delta}_i^{\boldsymbol{y}}, \boldsymbol{\Delta}_i^{\boldsymbol{z}}, \boldsymbol{d}_i$ is nonzero, $\mathcal{F}_{\mathsf{verifymult}}$ sends **abort** to the adversary and waits for the response from the adversary. Otherwise, $\mathcal{F}_{\mathsf{verifymult}}$ sends **continue** to the adversary and waits for the response from the adversary.
   - If the adversary returns **continue**, $\mathcal{F}_{\mathsf{verifymult}}$ sends to the honest parties whatever it sends to the adversary.
   - If the adversary returns **abort**, $\mathcal{F}_{\mathsf{verifymult}}$ sends **abort** to the honest parties.

Fig. 18: **Verifying multiplication tuples**

instantiate $\mathcal{F}_{\mathsf{inner\text{-}product}}$ in the protocol $\Pi_{\mathsf{inner\text{-}product}}$. These two protocols are illustrated in Figure 20 and Figure 21 respectively. The communication complexity of $\Pi_{\mathsf{inner\text{-}product}}$ is $O(n \cdot m)$ bits. The "Inner-Product" operation can be naturally extended by normal multiplications for packed secret sharings. Now we can extend the "compress" operation into a vector version, which is shown in Figure 22. And Lemma 7 is an extension of Lemma 6. The communication complexity of $\Pi_{\mathsf{extend\text{-}compress}}$ is $O(h \cdot n \cdot m + n^2 \cdot m)$ bits.

Lemma 6. *With the overwhelming probability at least* $1 - \frac{8h}{2^m}$, Π_{compress} *satisfies the following:*

- $r \notin [2h-2]$.
- *If there exists one inconsistent* $\llbracket \boldsymbol{x}_i \rrbracket_t$, *then* $\llbracket \boldsymbol{F}(r) \rrbracket_t$ *is also inconsistent.*
- *If there exists one inconsistent* $\llbracket \boldsymbol{y}_i \rrbracket_t$, *then* $\llbracket \boldsymbol{G}(r) \rrbracket_t$ *is also inconsistent.*
- *If there exists one inconsistent* $\llbracket \boldsymbol{z}_i \rrbracket_t$, *then* $\llbracket \boldsymbol{H}(r) \rrbracket_t$ *is also inconsistent.*
- *If there exists one* $i \in [h]$ *such that* $\boldsymbol{z}_i \neq \boldsymbol{x}_i * \boldsymbol{y}_i$, *then* $\boldsymbol{H}(r) \neq \boldsymbol{F}(r) * \boldsymbol{G}(r)$.

Proof. We count the number of r that breaks any of the above conditions.

- The number of $r \in [2h-2]$ is $2h-2$.
- If there are h elements $r_1, \ldots, r_h$ such that $\llbracket \boldsymbol{F}(r_i) \rrbracket_t$ is consistent, then each $\llbracket \boldsymbol{x}_i \rrbracket_t$ is also consistent since it can be written as a linear combination of $\llbracket \boldsymbol{F}(r_i) \rrbracket_t, i \in [h]$. Hence, there are at most $(h-1)$ random coins such that if one of $\llbracket \boldsymbol{x}_i \rrbracket_t$ is inconsistent, but $\llbracket \boldsymbol{F}(r) \rrbracket_t$ is consistent.

Protocol Π_{compress}

Given h packed multiplication triples $([\![\boldsymbol{x}_i]\!]_t, [\![\boldsymbol{y}_i]\!]_t, [\![\boldsymbol{z}_i]\!]_t)_{i\in[h]}$ over $\mathbb{F}_{2^m}$.

Protocols:

1. Let $\boldsymbol{F}(\cdot)$ and $\boldsymbol{G}(\cdot)$ be vectors of degree-$(h-1)$ polynomials such that $\boldsymbol{F}(i) = \boldsymbol{x}_i, \boldsymbol{G}(i) = \boldsymbol{y}_i$ for $i \in [h]$.
2. For $i = h+1, \ldots, 2h-1$, all parties can locally compute $[\![\boldsymbol{F}(i)]\!]_t, [\![\boldsymbol{G}(i)]\!]_t$ since each $[\![\boldsymbol{F}(i)]\!]_t$ and $[\![\boldsymbol{G}(i)]\!]_t$ can be written as a linear combination of $([\![\boldsymbol{F}(j)]\!]_t)_{j\in[h]}$ and $([\![\boldsymbol{G}(j)]\!]_t)_{j\in[h]}$. And parties use another multiplication to compute $[\![\boldsymbol{z}_i]\!]_t = [\![\boldsymbol{F}(i) * \boldsymbol{G}(i)]\!]_t$.
3. Let $\boldsymbol{H}(\cdot)$ be vectors of degree-$(2h-2)$ polynomials such that $\boldsymbol{H}(i) = \boldsymbol{z}_i$ for $i \in [2h-2]$.
4. All parties invoke $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $r \in \mathbb{F}_{2^m}$. If $r \in [2h-2]$, honest parties send **abort** to each other. Otherwise, parties output $([\![\boldsymbol{F}(r)]\!]_t, [\![\boldsymbol{G}(r)]\!]_t, [\![\boldsymbol{H}(r)]\!]_t)$.

Fig. 19: **Protocol for compressing sharing**

– Similar to the argument for the second condition, the number of r is at most $h-1$.
– Similar to the argument for the second condition, the number of r is at most $2h-2$.
– If $\boldsymbol{z}_i \neq \boldsymbol{x}_i * \boldsymbol{y}_i$ for some $i \in [h]$, then $\boldsymbol{H} - \boldsymbol{F} * \boldsymbol{G}$ is a nonzero vector of degree at most $(2h-2)$ polynomials. There are at most $2h-2$ zero points over $\mathbb{F}_{2^m}$.

Hence, there are at most $2h-2+h-1+h-1+2h-2+2h-2 = 8h-8 < 8h$ random r that breaks any of the above conditions. So we can conclude that the above properties are achieved in Π_{compress} with the probability at least $1 - \frac{8h}{2^m}$. □

Theorem 8. *Protocol $\Pi_{\mathsf{inner\text{-}product}}$ securely realizes $\mathcal{F}_{\mathsf{inner\text{-}product}}$ in the $\mathcal{F}_{\mathsf{rand}}$-hybrid model against malicious adversaries controlling up to $t' = t - k + 1$ parties.*

Proof. The proof of Theorem 8 is a natural extension of Theorem 13. □

Lemma 7. *With the overwhelming probability at least $1 - \frac{(2p+6)h}{2^m}$, $\Pi_{\mathsf{extend\text{-}compress}}$ satisfies the following:*

- *$r \notin [2h-2]$.*
- *For all $j \in [p]$, if there exists one inconsistent $[\![\boldsymbol{x}_i^j]\!]_t$, then $[\![\boldsymbol{F}_j(r)]\!]_t$ is also inconsistent.*
- *For all $j \in [p]$, if there exists one inconsistent $[\![\boldsymbol{y}_i^j]\!]_t$, then $[\![\boldsymbol{G}_j(r)]\!]_t$ is also inconsistent.*
- *If there exists one inconsistent $[[\![\boldsymbol{z}_i]\!]_t$, then $[\![\boldsymbol{H}(r)]\!]_t$ is also inconsistent.*

Functionality $\mathcal{F}_{\mathsf{inner\text{-}product}}$

Given two vectors of packed secret sharings $(\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_p \rrbracket_t)$ and $(\llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_p \rrbracket_t)$ over $\mathbb{F}_{2^m}$.

1. For each $i \in [p]$, $\mathcal{F}_{\mathsf{inner\text{-}product}}$ collects honest shares in $(\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_p \rrbracket_t)$ and $(\llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_p \rrbracket_t)$. Based on these honest shares, $\mathcal{F}_{\mathsf{inner\text{-}product}}$ recovers three sharings as $(\llbracket \boldsymbol{x}_1^H \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_p^H \rrbracket_t)$ and $(\llbracket \boldsymbol{y}_1^H \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_p^H \rrbracket_t)$ and computes $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \llbracket \boldsymbol{x}_i \rrbracket_t - \llbracket \boldsymbol{x}_i^H \rrbracket_t, \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \llbracket \boldsymbol{y}_i \rrbracket_t - \llbracket \boldsymbol{y}_i^H \rrbracket_t$.
2. For each $i \in [p]$, $\mathcal{F}_{\mathsf{inner\text{-}product}}$ sends corrupted shares in $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t)$ and the vectors $\boldsymbol{\Delta}_i^{\boldsymbol{x}}, \boldsymbol{\Delta}_i^{\boldsymbol{y}}$ to the adversary.
3. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ receives an additive error vector $\boldsymbol{d} \in \mathbb{F}_{2^m}^k$. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ computes $\llbracket \boldsymbol{z} \rrbracket_{2t} = \sum_{i \in [p]} \llbracket \boldsymbol{x}_i \rrbracket_t \cdot \llbracket \boldsymbol{y}_i \rrbracket_t$ and reconstructs $\boldsymbol{z}$ from $\llbracket \boldsymbol{z} \rrbracket_{2t}$. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ resets $\boldsymbol{z} = \boldsymbol{z} + \boldsymbol{d}$.
4. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ receives a set of corrupted shares $(s_j)_{j \in C}$ from the adversary. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ randomly samples a $\llbracket \boldsymbol{z}' \rrbracket_t$ such that $\pi_C(\llbracket \boldsymbol{z}' \rrbracket_t) = (s_j)_{j \in C}$.
5. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ receives another additive error vector $\boldsymbol{d}' \in \mathbb{F}_{2^m}^n$ such that $d'_j = 0$ for all $j \in C$. $\mathcal{F}_{\mathsf{inner\text{-}product}}$ computes $\llbracket \boldsymbol{z} \rrbracket_t = \llbracket \boldsymbol{z}' \rrbracket_t + \boldsymbol{d}'$. Finally, $\mathcal{F}_{\mathsf{inner\text{-}product}}$ outputs $\llbracket \boldsymbol{z} \rrbracket_t$ to honest parties.

Fig. 20: **Inner Product**

Protocol $\Pi_{\mathsf{inner\text{-}product}}$

Given two vectors of packed secret sharings $(\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_p \rrbracket_t)$ and $(\llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_p \rrbracket_t)$ over $\mathbb{F}_{2^m}$.

Protocols:

1. All parties call $\mathcal{F}_{\mathsf{rand}}$ to obtain a pair of m-fold random $(\llbracket \boldsymbol{r} \rrbracket_t, \llbracket \boldsymbol{r} \rrbracket_{2t})$.
2. All parties locally compute $\llbracket \boldsymbol{e} \rrbracket_{2t} = \sum_{i \in [p]} \llbracket \boldsymbol{x}_i \rrbracket_t \cdot \llbracket \boldsymbol{y}_i \rrbracket_t + \llbracket \boldsymbol{r} \rrbracket_{2t}$ and send shares of $\llbracket \boldsymbol{e} \rrbracket_{2t}$ to P_1.
3. P_1 reconstructs the secret $\boldsymbol{e}$ and distributes $\llbracket \boldsymbol{e} \rrbracket_t$ to other parties.
4. All parties locally set $\llbracket \boldsymbol{z}' \rrbracket_t = \llbracket \boldsymbol{e} \rrbracket_t - \llbracket \boldsymbol{r} \rrbracket_t$.
5. All parties call $\mathcal{F}_{\mathsf{rand}}(f, d)$ to obtain a pair of m-fold random $(\llbracket \boldsymbol{r}' \rrbracket_t, \llbracket \phi(\psi(\boldsymbol{r}')) \rrbracket_t)$.
6. All parties locally compute $\llbracket \boldsymbol{e}' \rrbracket_t = \llbracket \boldsymbol{z}' \rrbracket_t + \llbracket \boldsymbol{r}' \rrbracket_t$ and send shares of $\llbracket \boldsymbol{e}' \rrbracket_t$ to P_1.
7. P_1 reconstructs the secret $\boldsymbol{e}'$ and distributes $\llbracket \phi(\psi(\boldsymbol{e}')) \rrbracket_t$ to other parties.
8. All parties locally compute $\llbracket \boldsymbol{z} \rrbracket_t = \llbracket \phi(\psi(\boldsymbol{e}')) \rrbracket_t - \llbracket \phi(\psi(\boldsymbol{r}')) \rrbracket_t$.

Fig. 21: **Protocol for inner product**

- *If there exists one $i \in [h]$ such that $\boldsymbol{z}_i \neq \sum_{j=1}^{l} \boldsymbol{x}_i^j * \boldsymbol{y}_i^j$, then $\boldsymbol{H}(r) \neq \sum_{j=1}^{l} \boldsymbol{F}_j(r) * \boldsymbol{G}_j(r)$.*

Proof. We count the number of r that breaks any of the above conditions.

– The number of $r \in [2h-2]$ is $2h-2$.

Protocol $\Pi_{\text{extend-compress}}$

Let h be the number of inner-product tuples and p be the dimension of each inner-product tuple. For each $i \in [h]$, the i-th inner-product tuple denotes $((\llbracket \boldsymbol{x}_i^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_i^p \rrbracket_t), (\llbracket \boldsymbol{y}_i^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_i^p \rrbracket_t), \llbracket \boldsymbol{z}_i \rrbracket_t)$.

Protocols:

1. For each $j \in [p]$, let $\boldsymbol{F}_j(\cdot)$ and $\boldsymbol{G}_j(\cdot)$ be vectors of degree-$(h-1)$ polynomials such that $\boldsymbol{F}_j(i) = \boldsymbol{x}_i^j, \boldsymbol{G}_j(i) = \boldsymbol{y}_i^j$ for $i \in [h]$.
2. For $i = h+1, \ldots, 2h-1$ and each $j \in [p]$, all parties can locally compute $\llbracket \boldsymbol{F}_j(i) \rrbracket_t, \llbracket \boldsymbol{G}_j(i) \rrbracket_t$ since each $\llbracket \boldsymbol{F}_j(i) \rrbracket_t$ and $\llbracket \boldsymbol{G}_j(i) \rrbracket_t$ can be written as a linear combination of $(\llbracket \boldsymbol{F}_j(u) \rrbracket_t)_{u \in [h]}$ and $(\llbracket \boldsymbol{G}_j(u) \rrbracket_t)_{u \in [h]}$. And parties invoke $\mathcal{F}_{\text{inner-product}}$ to compute $\llbracket \boldsymbol{z}_i \rrbracket_t = \llbracket \sum_{j=1}^p \boldsymbol{F}_j(i) * \boldsymbol{G}_j(i) \rrbracket_t$.
3. Let $\boldsymbol{H}(\cdot)$ be vectors of degree-$(2h-2)$ polynomials such that $\boldsymbol{H}(i) = \boldsymbol{z}_i$ for $i \in [2h-2]$.
4. All parties invoke $\mathcal{F}_{\text{coin}}$ to obtain a random $r \in \mathbb{F}_{2^m}$. If $r \in [2h-2]$, honest parties send **abort** to each other. Otherwise, parties output $((\llbracket \boldsymbol{F}_1(r) \rrbracket_t, \ldots, \llbracket \boldsymbol{F}_p(r) \rrbracket_t), (\llbracket \boldsymbol{G}_1(r) \rrbracket_t, \ldots, \llbracket \boldsymbol{G}_p(r) \rrbracket_t), \llbracket \boldsymbol{H}(r) \rrbracket_t)$.

Fig. 22: **Protocol for extend-compress sharing**

- Similar to the argument in Lemma 6, for each fixed $j \in [p]$, the number of r that breaks the second condition is bounded by $h-1$. Then, the total number of r that breaks the second condition is $p(h-1)$.
- Similar to the argument for the second condition, the number of r is at most $p(h-1)$.
- Similar to the argument in Lemma 6, the number of r is at most $2h-2$.
- If $\boldsymbol{z}_i \neq \sum_{j=1}^p \boldsymbol{x}_i^j * \boldsymbol{y}_i^j$ for some $i \in [h]$, then $\boldsymbol{H} - \sum_{j=1}^p \boldsymbol{F}_j * \boldsymbol{G}_j$ is a nonzero vector of degree at most $(2h-2)$ polynomials. There are at most $2h-2$ zero points over $\mathbb{F}_{2^m}$.

Hence, there are at most $2h-2+p(h-1)+p(h-1)+2h-2+2h-2 = (2p+6)h-8 < (2p+6)h$ random r that breaks any of the above conditions. So we can conclude that the above properties is achieved in $\Pi_{\text{extend-compress}}$ with the probability at least $1 - \frac{(2p+6)h}{2^m}$. □

Now we show how to realize *distributed zero knowledge proof* technique on packed secret sharings. At a high level, the construction idea follows [29] similarly. There are three main steps:

- De-linearization: This step transforms h packed multiplication triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i \in [h]}$ into a single inner-product tuple of dimension h after proper randomization. This operation is formalized in $\Pi_{\text{de-linearization}}$ shown in Figure 23 with the communication complexity $O(n^2 \cdot m)$ bits.
- Reduction: This step is to reduce the dimension of the inner-product tuple in the first step by a constant factor. In this paper, we set m to be the

Protocol $\Pi_{\mathsf{de\text{-}linearization}}$

Given h packed multiplication triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i\in[h]}$ over $\mathbb{F}_{2^m}$.

Protocols:

1. All parties invoke $\mathcal{F}_{\mathsf{coin}}$ to obtain a random $r \in \mathbb{F}_{2^m}$.
2. For each $i \in [h]$, all parties reset $\llbracket \boldsymbol{a}_i \rrbracket_t = r^{i-1} \cdot \llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{b}_i \rrbracket_t = \llbracket \boldsymbol{y}_i \rrbracket_t$, and computes $\llbracket \boldsymbol{c} \rrbracket_t = \sum_{i=1}^{h} r^{i-1} \cdot \llbracket \boldsymbol{z}_i \rrbracket_t$.
3. All parties output a single inner-product tuple $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_h \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_h \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$.

Fig. 23: **Protocol for de-linearization**

compression parameter. This operation is formalized in $\Pi_{\mathsf{reduction}}$ shown in Figure 24 with the communication complexity $O(h \cdot n \cdot m + n^2 \cdot m)$ bits.

– Recursion and Randomization: We iteratively send outputs of the second step to the first step until the dimension becomes 1. This operation is formalized in $\Pi_{\mathsf{randomization}}$ shown in Figure 25 with the communication complexity $O(n^2 \cdot m^2)$ bits.

Lemma 8. *With the overwhelming probability at least* $1 - \frac{2h}{2^m}$, $\Pi_{\mathsf{de\text{-}linearization}}$ *satisfies the following:*

- *If there exists one inconsistent* $\llbracket \boldsymbol{z}_i \rrbracket_t$, *then* $\llbracket \boldsymbol{c} \rrbracket_t$ *is also inconsistent.*
- *If there exists one* $i \in [h]$ *such that* $\boldsymbol{z}_i \neq \boldsymbol{x}_i * \boldsymbol{y}_i$, *then* $\boldsymbol{c} \neq \sum_{j=1}^{h} \boldsymbol{a}_j * \boldsymbol{b}_j$.

Proof. We count the number of r that breaks any of the above conditions.

– Note that $\llbracket \boldsymbol{c} \rrbracket_t$ is a linear combination of $\llbracket \boldsymbol{z}_i \rrbracket_t, i \in [h]$ determined by r. Similar to the argument in Lemma 6, there are at most $h-1$ random coins that break the first condition.

– If $\boldsymbol{z}_i \neq \boldsymbol{x}_i * \boldsymbol{y}_i$ for some $i \in [h]$, we consider two vectors of degree-$(h-1)$ polynomials $\boldsymbol{F}(X) = (\boldsymbol{x}_1 * \boldsymbol{y}_1) + (\boldsymbol{x}_2 * \boldsymbol{y}_2) \cdot X + \cdots + (\boldsymbol{x}_h * \boldsymbol{y}_h) \cdot X^{h-1}$ and $\boldsymbol{G}(X) = \boldsymbol{z}_1 + \boldsymbol{z}_2 \cdot X + \cdots + \boldsymbol{z}_h \cdot X^{h-1}$ determined by inner products of $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i\in[h]}$ and $(\boldsymbol{z}_i)_{i\in[h]}$ respectively. Then $\boldsymbol{\delta}(\cdot) = \boldsymbol{F}(\cdot) - \boldsymbol{G}(\cdot)$ is a nonzero vector of degree at most $(h-1)$ polynomial. Moreover, there are at most $h-1$ zero points over $\mathbb{F}_{2^m}$.

Hence, there are at most $h-1+h-1 = 2h-2 < 2h$ random r that breaks any of the above conditions. So we can conclude that the above properties is achieved in $\Pi_{\mathsf{de\text{-}linearization}}$ with the probability at least $1 - \frac{2h}{2^m}$. □

Lemma 9. *With the overwhelming probability at least* $1 - \frac{(2p+4)h}{2^m}$, $\Pi_{\mathsf{reduction}}$ *satisfies the following:*

- *If there exists one inconsistent* $\llbracket \boldsymbol{x}_i \rrbracket_t$ *for* $i \in [h]$, *then there exists one* $\llbracket \boldsymbol{a}_v \rrbracket_t$ *is also inconsistent for some* $v \in [p]$.

Protocol $\Pi_{\mathsf{reduction}}$

Given one single inner-product tuple $((\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_h \rrbracket_t), (\llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_h \rrbracket_t), \llbracket \boldsymbol{z} \rrbracket_t)$ over $\mathbb{F}_{2^m}$ with dimension h. Let $p = \frac{h}{m}$.

Protocols:

1. For each $u \in [m]$ and each $v \in [p]$, all parties set $\llbracket \boldsymbol{a}_u^v \rrbracket_t = \llbracket \boldsymbol{x}_{(u-1)p+v} \rrbracket_t$ and $\llbracket \boldsymbol{b}_u^v \rrbracket_t = \llbracket \boldsymbol{y}_{(u-1)p+v} \rrbracket_t$.
2. For $u = 1, \ldots, m-1$, all parties invoke $\mathcal{F}_{\mathsf{inner\text{-}product}}$ on $(\llbracket \boldsymbol{a}_u^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_u^p \rrbracket_t)$ and $(\llbracket \boldsymbol{b}_u^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_u^p \rrbracket_t)$ to compute $\llbracket \boldsymbol{c}_u \rrbracket_t = \llbracket \sum_{v=1}^p \boldsymbol{a}_u^v * \boldsymbol{b}_u^v \rrbracket_t$. And parties compute $\llbracket \boldsymbol{c}_m \rrbracket_t = \llbracket \boldsymbol{z} \rrbracket_t - \sum_{u=1}^{m-1} \llbracket \boldsymbol{c}_u \rrbracket_t$.
3. All parties invoke $\Pi_{\mathsf{extend\text{-}compress}}$ on m inner-product tuples $((\llbracket \boldsymbol{a}_u^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_u^p \rrbracket_t), (\llbracket \boldsymbol{b}_u^1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_u^p \rrbracket_t), \llbracket \boldsymbol{c}_u \rrbracket_t)_{u \in [m]}$ to outputs a new inner-product tuple as $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_p \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_p \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$.

Fig. 24: **Protocol for reduction**

- *If there exists one inconsistent $\llbracket \boldsymbol{y}_i \rrbracket_t$ for $i \in [h]$, then there exists one $\llbracket \boldsymbol{b}_v \rrbracket_t$ is also inconsistent for some $v \in [p]$.*
- *If $\llbracket \boldsymbol{z} \rrbracket_t$ is inconsistent, then $\llbracket \boldsymbol{c} \rrbracket_t$ is inconsistent.*
- *If $\boldsymbol{z} \neq \sum_{i=1}^h \boldsymbol{x}_i * \boldsymbol{y}_i$, then $\boldsymbol{c} \neq \sum_{v=1}^p \boldsymbol{a}_v * \boldsymbol{b}_v$.*

Proof. – Similar to the argument in Lemma 7 for the second condition, the probability that breaks the first condition is at most $p(h-1)$.
– Similar to the argument in Lemma 7 for the third condition, the probability that breaks the first condition is at most $p(h-1)$.
– If $\llbracket \boldsymbol{z} \rrbracket_t$ is inconsistent, then there exists one inconsistent $\llbracket \boldsymbol{c}_u \rrbracket_t$ for some $u \in [m]$ since $\llbracket \boldsymbol{z} \rrbracket_t = \sum_{u=1}^m \llbracket \boldsymbol{c}_u \rrbracket_t$. Then we can use the result of Lemma 7 for the third condition, the probability that breaks the first condition is at most $2h-2$.
– If $\boldsymbol{z} \neq \sum_{i=1}^h \boldsymbol{x}_i * \boldsymbol{y}_i$, then there exists one $u \in [m]$ such that $\boldsymbol{c}_u \neq \sum_{v=1}^p \boldsymbol{a}_u^v * \boldsymbol{b}_u^v$. Otherwise, $\boldsymbol{z} = \sum_{u=1}^m \boldsymbol{c}_u = \sum_{u=1}^m \sum_{v=1}^p \boldsymbol{a}_u^v * \boldsymbol{b}_u^v = \sum_{i=1}^h \boldsymbol{x}_i * \boldsymbol{y}_i$, which leads to a contradiction. If $\boldsymbol{c}_u \neq \sum_{v=1}^p \boldsymbol{a}_u^v * \boldsymbol{b}_u^v$ for some $u \in [m]$, similar to the argument in Lemma 7 for the fifth condition, the probability that breaks the fourth condition is at most $2h-2$.

Hence, the probability that breaks any of the conditions is at most $p(h-1) + p(h-1) + 2h - 2 + 2h - 2 = (2p+4)h - (2l+4) < (2p+4)h$. So we can conclude that the above properties is achieved in $\Pi_{\mathsf{reduction}}$ with the probability at least $1 - \frac{(2p+4)h}{2^m}$. □

Lemma 10. *With the overwhelming probability at least $1 - \frac{7h}{2^m}$, at least one honest party sends **abort** in $\Pi_{\mathsf{randomization}}$ if the inner-product tuple is incorrect or any sharing is inconsistent.*

Proof. The conditions in Lemma 10 is totally contained in Lemma 6. By Lemma 6, with the probability at least $1 - \frac{7h}{2^m}$, the output multiplication tuple $(\llbracket \boldsymbol{a} \rrbracket_t, \llbracket \boldsymbol{b} \rrbracket_t, \llbracket \boldsymbol{c} \rrbracket_t)$ is either incorrect or at least one of $(\llbracket \boldsymbol{a} \rrbracket_t, \llbracket \boldsymbol{b} \rrbracket_t, \llbracket \boldsymbol{c} \rrbracket_t)$ is inconsistent. □

Protocol $\Pi_{\mathsf{randomization}}$

Given one single inner-product tuple $((\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_m \rrbracket_t), (\llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_m \rrbracket_t), \llbracket \boldsymbol{z} \rrbracket_t)$ over $\mathbb{F}_{2^m}$ with dimension m.

Protocols:

1. All parties invoke $\mathcal{F}_{\mathsf{rand}}$ to prepare two random sharings $(\llbracket \boldsymbol{x}_0 \rrbracket_t, \llbracket \boldsymbol{y}_0 \rrbracket_t)$.
2. All parties use another multiplication on $(\llbracket \boldsymbol{x}_0 \rrbracket_t, \llbracket \boldsymbol{y}_0 \rrbracket_t)$ to compute $\llbracket \boldsymbol{z}_0 \rrbracket_t = \llbracket \boldsymbol{x}_0 * \boldsymbol{y}_0 \rrbracket_t$.
3. For each $i \in [m-1]$, parties use multiplication on $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t)$ to compute $\llbracket \boldsymbol{z}_i \rrbracket_t = \llbracket \boldsymbol{x}_i * \boldsymbol{y}_i \rrbracket_t$. Then parties set $\llbracket \boldsymbol{z}_m \rrbracket_t = \llbracket \boldsymbol{z} \rrbracket_t - \sum_{i=1}^{m-1} \llbracket \boldsymbol{z}_i \rrbracket_t$.
4. All parties invoke Π_{compress} on $(m+1)$ packed triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i=0}^{m}$. The final output of Π_{compress} is $(\llbracket \boldsymbol{a} \rrbracket_t, \llbracket \boldsymbol{b} \rrbracket_t, \llbracket \boldsymbol{c} \rrbracket_t)$.
5. All parties send shares of $(\llbracket \boldsymbol{a} \rrbracket_t, \llbracket \boldsymbol{b} \rrbracket_t)$ to each other. Each party P_j checks whether $(\llbracket \boldsymbol{a} \rrbracket_t, \llbracket \boldsymbol{b} \rrbracket_t)$ are valid degree-t packed secret sharings. If not, P_j sends **abort** to other parties.
6. All parties send shares of $\llbracket \boldsymbol{c} \rrbracket_t$ to each other. Each party P_j checks whether $\llbracket \boldsymbol{c} \rrbracket_t$ are valid degree-t packed secret sharings and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. If not, P_j sends **abort** to other parties.

Fig. 25: **Protocol for randomization**

Protocol $\Pi_{\mathsf{multcheck}}$

Given h packed multiplication triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i \in [h]}$ over $\mathbb{F}_{2^m}$.

Protocols:

1. All parties invoke $\Pi_{\mathsf{de\text{-}linearization}}$ on these h packed multiplication triples to obtain a single inner-product tuple $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_h \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_h \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$.
2. Denote p to be the dimension of the inner-product tuple. Initially $p = h$. While $p \geq m$, all parties operate as follows:
   - All parties invoke $\Pi_{\mathsf{reduction}}$ on the current inner-product tuple, and then obtain a new inner-product tuple denoted as $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_{\frac{p}{m}} \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_{\frac{p}{m}} \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$.
   - All parties set $p = \frac{p}{m}$.
3. Let $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_m \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_m \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$ be the outputs in the second step. All parties invoke $\Pi_{\mathsf{randomization}}$ to check the final inner-product tuple $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_m \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_m \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$.

Fig. 26: **Protocol for verifying multiplication**

Now we integrate the above protocols into our final protocol $\Pi_{\mathsf{multcheck}}$ that securely realizes $\mathcal{F}_{\mathsf{verifymult}}$, which is shown in Figure 26.

Theorem 9. *Protocol $\Pi_{\mathsf{multcheck}}$ shown in Figure 26 securely realizes $\mathcal{F}_{\mathsf{verifymult}}$ in the $(\mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{inner\text{-}product}})$-hybrid model against malicious adversaries controlling up to $t' = t - k + 1$ parties.*

Proof. We simulate for $\Pi_{\mathsf{multcheck}}$ as: For each $i \in [h]$, S receives corrupted shares in $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)$ and three additive errors $\boldsymbol{\Delta}_i^{\boldsymbol{x}}, \boldsymbol{\Delta}_i^{\boldsymbol{y}}, \boldsymbol{\Delta}_i^{\boldsymbol{z}}$ to the honest shares in H_2 from the adversary. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$, S also receives $\boldsymbol{d}_i$ from the adversary. S receives the command $b \in \{\textbf{abort, continue}\}$ depending on whether all packed multiplication triples are computed correctly.

- Simulation for $\Pi_{\mathsf{de\text{-}linearization}}$: When all parties invoke $\mathcal{F}_{\mathsf{coin}}$ in the first step, S emulates $\mathcal{F}_{\mathsf{coin}}$ and generate a random $r \in \mathbb{F}_{2^m}$. We denote $((\llbracket \boldsymbol{a}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{a}_h \rrbracket_t), (\llbracket \boldsymbol{b}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{b}_h \rrbracket_t), \llbracket \boldsymbol{c} \rrbracket_t)$ to be the output of $\Pi_{\mathsf{de\text{-}linearization}}$. For each sharing in this inner-product tuple, S computes its corrupted shares and additive errors related to H_2.
  Specifically, for each $i \in [h]$, since $\llbracket \boldsymbol{a}_i \rrbracket_t = r^{i-1} \cdot \llbracket \boldsymbol{x}_i \rrbracket_t$, the corrupted shares of $\llbracket \boldsymbol{a}_i \rrbracket_t$ can be computed by the corrupted shares of $\llbracket \boldsymbol{x}_i \rrbracket_t$. Also, S can compute $\boldsymbol{\Delta}_i^{\boldsymbol{a}} = r^{i-1}\boldsymbol{\Delta}_i^{\boldsymbol{x}}$ as the additive errors of $\llbracket \boldsymbol{a}_i \rrbracket_t$ in H_2. Similarly, since $\llbracket \boldsymbol{b}_i \rrbracket_t = \llbracket \boldsymbol{y}_i \rrbracket_t$, then the corrupted shares of $\llbracket \boldsymbol{b}_i \rrbracket_t$ are the same as the corrupted shares of $\llbracket \boldsymbol{y}_i \rrbracket_t$, so does $\boldsymbol{\Delta}_i^{\boldsymbol{b}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}}$. Since $\llbracket \boldsymbol{c} \rrbracket_t = \sum_{i=1}^{h} r^{i-1} \cdot \llbracket \boldsymbol{z}_i \rrbracket_t$, the corrupted shares of $\llbracket \boldsymbol{a}_i \rrbracket_t$ can be written as a linear combination of the corrupted shares of $(\llbracket \boldsymbol{z}_i \rrbracket_t)_{i \in [h]}$. Also S can compute $\boldsymbol{\Delta}_i^{\boldsymbol{c}} = \sum_{i=1}^{h} r^{i-1} \cdot \boldsymbol{\Delta}_i^{\boldsymbol{z}}$ as the additive errors of $\llbracket \boldsymbol{a}_i \rrbracket_t$ in H_2. And S can compute the additive errors of this inner-product tuple as $\boldsymbol{d} = \sum_{i=1}^{h} r^{i-1} \cdot \boldsymbol{d}_i$.
  If $\llbracket \boldsymbol{x}_i \rrbracket_t$ or $\llbracket \boldsymbol{y}_i \rrbracket_t$ is inconsistent, then $\llbracket \boldsymbol{a}_i \rrbracket_t$ or $\llbracket \boldsymbol{b}_i \rrbracket_t$ is inconsistent. S will send **abort** to $\mathcal{F}_{\mathsf{verifymult}}$ if either $\boldsymbol{\Delta}_i^{\boldsymbol{c}} = \boldsymbol{0}$ but $\boldsymbol{\Delta}_i^{\boldsymbol{z}} \neq \boldsymbol{0}$ for some $i \in [h]$, or $\boldsymbol{d} \neq \boldsymbol{0}$ but $\boldsymbol{d}_i \neq \boldsymbol{0}$. By Lemma 8, these two conditions happen with negligible probability.
- Simulation for $\Pi_{\mathsf{reduction}}$: Given the input inner-product tuple $((\llbracket \boldsymbol{x}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{x}_h \rrbracket_t), \llbracket \boldsymbol{y}_1 \rrbracket_t, \ldots, \llbracket \boldsymbol{y}_h \rrbracket_t), \llbracket \boldsymbol{z} \rrbracket_t)$, S receives corrupted shares of each sharing and additive errors to the shares of parties in H_2 denoted as $((\boldsymbol{\Delta}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{\Delta}_h^{\boldsymbol{x}}), (\boldsymbol{\Delta}_1^{\boldsymbol{y}}, \ldots, \boldsymbol{\Delta}_h^{\boldsymbol{y}}), \boldsymbol{\Delta}_i^{\boldsymbol{z}})$. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$ for all $i \in [h]$, S learns the additive errors to the inner-product results as $\boldsymbol{d} = \boldsymbol{z} - \sum_{i=1}^{h} \boldsymbol{x}_i * \boldsymbol{y}_i$. If $\Pi_{\mathsf{de\text{-}linearization}}$ is executed correctly, i.e., not aborted, then this value is the real additive errors.
  In the first step, S faithfully follows the protocol. For each $u \in [m]$ and each $v \in [p]$, S learns corrupted shares of $\llbracket \boldsymbol{a}_u^v \rrbracket_t$ and $\llbracket \boldsymbol{b}_u^v \rrbracket_t$ and sets $\boldsymbol{\Delta}_u^{\boldsymbol{a},v}$ and $\boldsymbol{\Delta}_u^{\boldsymbol{b},v}$ to be additive errors of $\llbracket \boldsymbol{a}_u^v \rrbracket_t$ and $\llbracket \boldsymbol{b}_u^v \rrbracket_t$ to the shares of parties in H_2 respectively.
  In the second step, for $u = 1, \ldots, m-1$, S emulates $\mathcal{F}_{\mathsf{inner\text{-}product}}$ when computing $\llbracket \boldsymbol{c}_u \rrbracket_t$. For each $v \in [p]$, S first sends corrupted shares of $\llbracket \boldsymbol{a}_u^v \rrbracket_t$ and $\llbracket \boldsymbol{b}_u^v \rrbracket_t$ and additive errors $\boldsymbol{\Delta}_u^{\boldsymbol{a},v}$ and $\boldsymbol{\Delta}_u^{\boldsymbol{b},v}$ to the adversary. S receives corrupted shares in $\llbracket \boldsymbol{c}_u \rrbracket_t$ and additive errors $\boldsymbol{\Delta}_u^{\boldsymbol{c}}$ to the shares of parties in H_2 from the adversary. S also receives additive errors of $\llbracket \boldsymbol{c}_u \rrbracket_t$ as $\boldsymbol{d}_u$ from the adversary. If $\boldsymbol{\Delta}_u^{\boldsymbol{a},v} = \boldsymbol{\Delta}_u^{\boldsymbol{b},v} = \boldsymbol{0}$ for all $v \in [p]$, then $\boldsymbol{d}_u = \boldsymbol{c}_u - \sum_{v=1}^{p} \boldsymbol{a}_u^v * \boldsymbol{b}_u^v$. Moreover, S computes corrupted shares of $\llbracket \boldsymbol{c}_m \rrbracket_t$ and additive errors $\boldsymbol{\Delta}_m^{\boldsymbol{c}} = \boldsymbol{\Delta}^{\boldsymbol{z}} - \sum_{u=1}^{m-1} \boldsymbol{\Delta}_u^{\boldsymbol{c}}$ by following the calculation $\llbracket \boldsymbol{c}_m \rrbracket_t = \llbracket \boldsymbol{z} \rrbracket_t - \sum_{u=1}^{m-1} \llbracket \boldsymbol{c}_u \rrbracket_t$. If $\boldsymbol{\Delta}^{\boldsymbol{z}} \neq \boldsymbol{0}$, then $\boldsymbol{\Delta}_u^{\boldsymbol{c}} \neq \boldsymbol{0}$ for some $u \in [m]$ since $\llbracket \boldsymbol{z} \rrbracket_t = \sum_{u=1}^{m} \llbracket \boldsymbol{c}_u \rrbracket_t$, which implies that $\llbracket \boldsymbol{c}_u \rrbracket_t$ is inconsistent. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$ for all $i \in [h]$, then S computes $\boldsymbol{d}_m = \boldsymbol{d} - \sum_{u=1}^{m-1} \boldsymbol{d}_u$, where $\boldsymbol{d}_u = \boldsymbol{c}_u - \sum_{v=1}^{p} \boldsymbol{a}_u^v * \boldsymbol{b}_u^v$ in this case.

In the last step, S needs to simulate $\Pi_{\mathsf{extend\text{-}compress}}$. Note that $\Pi_{\mathsf{extend\text{-}compress}}$ is formed by local computation and invocations of $\mathcal{F}_{\mathsf{inner\text{-}product}}$ and $\mathcal{F}_{\mathsf{coin}}$. For simulating $\mathcal{F}_{\mathsf{inner\text{-}product}}$, the input sharings are linear combinations of $[\![\boldsymbol{a}_u^v]\!]_t$ and $[\![\boldsymbol{b}_u^v]\!]_t$, then S can locally compute corrupted shares in the input sharings and additive errors of the input sharings to the shares of parties in H_2. Later, S receives both corrupted shares and additive errors of the input sharings to the shares of parties in H_2 from the adversary. S also receives the additive errors to the secrets of the output sharing. For simulating $\mathcal{F}_{\mathsf{coin}}$, S faithfully samples a random $r \in \mathbb{F}_{2^m}$. Denote $(([\![\boldsymbol{a}_1]\!]_t, \ldots, [\![\boldsymbol{a}_p]\!]_t), ([\![\boldsymbol{b}_1]\!]_t, \ldots, [\![\boldsymbol{b}_p]\!]_t), [\![\boldsymbol{c}]\!]_t)$ to be the output after invoking $\Pi_{\mathsf{reduction}}$. S can compute corrupted shares in this tuple and additive errors of sharings in this tuple to the shares of parties in H_2 by simulating $\mathcal{F}_{\mathsf{inner\text{-}product}}$ and $\mathcal{F}_{\mathsf{coin}}$ together with local calculations. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$ for all $i \in [h]$, S learns the additive errors to the inner-product results of each inner-product tuple from $\mathcal{F}_{\mathsf{inner\text{-}product}}$. S further computes the additive errors $\boldsymbol{c} - \sum_{v=1}^{p} \boldsymbol{a}_v * \boldsymbol{b}_v$.

Note that for each $v \in [p]$, $[\![\boldsymbol{a}_v]\!]_t, [\![\boldsymbol{b}_v]\!]_t$ are linear combinations of $[\![\boldsymbol{x}_i]\!]_t, [\![\boldsymbol{y}_i]\!], i \in [h]$. Therefore, if $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$ for all $i \in [h]$, then $[\![\boldsymbol{a}_v]\!]_t, [\![\boldsymbol{b}_v]\!]_t$ are consistent. In this case, S also computes the additive errors $\boldsymbol{c} - \sum_{v=1}^{p} \boldsymbol{a}_v * \boldsymbol{b}_v$, which preserves the invariant of invoking $\Pi_{\mathsf{reduction}}$ in next round. Additionally, S will send **abort** to $\mathcal{F}_{\mathsf{verifymult}}$ if any of the following conditions holds,

- For all $v \in [p]$, $[\![\boldsymbol{a}_v]\!]_t$ is consistent, but $\boldsymbol{\Delta}_i^{\boldsymbol{x}} \neq \boldsymbol{0}$ for some $i \in [h]$.
- For all $v \in [p]$, $[\![\boldsymbol{b}_v]\!]_t$ is consistent, but $\boldsymbol{\Delta}_i^{\boldsymbol{y}} \neq \boldsymbol{0}$ for some $i \in [h]$.
- $[\![\boldsymbol{c}]\!]_t$ is consistent, but $\boldsymbol{\Delta}^{\boldsymbol{z}} \neq \boldsymbol{0}$.
- $\boldsymbol{c} - \sum_{v=1}^{p} \boldsymbol{a}_v * \boldsymbol{b}_v = \boldsymbol{0}$, but $\boldsymbol{z} - \sum_{i=1}^{h} \boldsymbol{x}_i * \boldsymbol{y}_i \neq \boldsymbol{0}$.

By Lemma 9, any of these conditions happens with negligible probability.

– Simulation for $\Pi_{\mathsf{randomization}}$: Given the final inner-product tuple $(([\![\boldsymbol{x}_1]\!]_t, \ldots, [\![\boldsymbol{x}_m]\!]_t), ([\![\boldsymbol{y}_1]\!]_t, \ldots, [\![\boldsymbol{y}_m]\!]_t), [\![\boldsymbol{z}]\!]_t)$. S learns corrupted shares of each sharing and additive errors to the shares of parties in H_2 denoted as $((\boldsymbol{\Delta}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{\Delta}_m^{\boldsymbol{x}}), (\boldsymbol{\Delta}_1^{\boldsymbol{y}}, \ldots, \boldsymbol{\Delta}_m^{\boldsymbol{y}}), \boldsymbol{\Delta}^{\boldsymbol{z}})$. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \boldsymbol{0}$ for all $i \in [m]$, then S learns the additive errors to the inner-product results as $\boldsymbol{d} = \boldsymbol{z} - \sum_{i=1}^{m} \boldsymbol{x}_i * \boldsymbol{y}_i$, which maintains the invariant when simulating $\Pi_{\mathsf{reduction}}$.

In the first step, S simulates $\mathcal{F}_{\mathsf{rand}}$ to prepare $([\![\boldsymbol{x}_0]\!]_t, [\![\boldsymbol{y}_0]\!]_t)$. S receives corrupted shares of $([\![\boldsymbol{x}_0]\!]_t, [\![\boldsymbol{y}_0]\!]_t)$ and additive errors of them to the shares of parties in H_2. Note that $\boldsymbol{\Delta}_0^{\boldsymbol{x}} = \boldsymbol{\Delta}_0^{\boldsymbol{y}} = \boldsymbol{0}$ since $\mathcal{F}_{\mathsf{rand}}$ ensures the consistency of $([\![\boldsymbol{x}_0]\!]_t, [\![\boldsymbol{y}_0]\!]_t)$.

In the second step, S simulates multiplication when computing $[\![\boldsymbol{z}_0]\!]_t = [\![\boldsymbol{x}_0 * \boldsymbol{y}_0]\!]_t$. S first sends corrupted shares of $([\![\boldsymbol{x}_0]\!]_t, [\![\boldsymbol{y}_0]\!]_t)$ and additive errors $\boldsymbol{\Delta}_0^{\boldsymbol{x}}, \boldsymbol{\Delta}_0^{\boldsymbol{y}}$ to the adversary. S receives corrupted shares in $[\![\boldsymbol{z}_0]\!]_t$ and additive errors $\boldsymbol{\Delta}_0^{\boldsymbol{z}}$ to the shares of parties in H_2 from the adversary. S also receives the additive errors $\boldsymbol{d}_0$ to $[\![\boldsymbol{z}_0]\!]_t$, where $\boldsymbol{d}_0 = \boldsymbol{z}_0 - \boldsymbol{x}_0 * \boldsymbol{y}_0$ since $\boldsymbol{\Delta}_0^{\boldsymbol{x}} = \boldsymbol{\Delta}_0^{\boldsymbol{y}} = \boldsymbol{0}$.

The third and fourth steps are the same operations in $\Pi_{\mathsf{reduction}}$, which can be simulated in the same way. Let $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t, [\![\boldsymbol{c}]\!]_t)$ be the output of $\Pi_{\mathsf{reduction}}$ in the fourth step. S learns corrupted shares of each sharing and additive errors in each sharing to the shares of parties in H_2 denoted as $(\boldsymbol{\Delta}_{\boldsymbol{a}}, \boldsymbol{\Delta}_{\boldsymbol{b}}, \boldsymbol{\Delta}_{\boldsymbol{c}})$

respectively. If $\boldsymbol{\Delta}_i^{\boldsymbol{x}} = \boldsymbol{\Delta}_i^{\boldsymbol{y}} = \mathbf{0}$ for all $i \in [m]$, S also computes the additive errors $\boldsymbol{c} - \boldsymbol{a} * \boldsymbol{b}$. Additionally, S will send **abort** to $\mathcal{F}_{\mathsf{verifymult}}$ if any of the following conditions holds,

- $[\![\boldsymbol{a}]\!]_t$ is consistent, but $\boldsymbol{\Delta}_i^{\boldsymbol{x}} \neq \mathbf{0}$ for some $i \in [m]$.
- $[\![\boldsymbol{b}]\!]_t$ is consistent, but $\boldsymbol{\Delta}_i^{\boldsymbol{y}} \neq \mathbf{0}$ for some $i \in [m]$.
- $[\![\boldsymbol{c}]\!]_t$ is consistent, but $\boldsymbol{\Delta}^{\boldsymbol{z}} \neq \mathbf{0}$.
- $\boldsymbol{c} - \boldsymbol{a} * \boldsymbol{b} = \mathbf{0}$, but $\boldsymbol{z} - \sum_{i=1}^{m} \boldsymbol{x}_i * \boldsymbol{y}_i \neq \mathbf{0}$.

By Lemma 6, any of these conditions happens with negligible probability.
In the fifth step, S first randomly samples two vectors $\boldsymbol{a}, \boldsymbol{b}$. Then S reconstructs two sharings $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t)$ based on secret vectors, corrupted shares and additive errors $(\boldsymbol{\Delta}_{\boldsymbol{a}}, \boldsymbol{\Delta}_{\boldsymbol{b}})$ in H_2. S sends the shares of $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t)$ of honest parties to corrupted parties and then honestly checks the consistency of $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t)$.
In the last step, S computes $\boldsymbol{c}$ based on $\boldsymbol{a} * \boldsymbol{b}$ and the additive error $\boldsymbol{c} - \boldsymbol{a} * \boldsymbol{b}$. S reconstructs the whole sharing $[\![\boldsymbol{c}]\!]_t$ based on the secret vector $\boldsymbol{c}$, corrupted shares and additive errors to the shares of parties in H_2. Then S sends the shares of $[\![\boldsymbol{c}]\!]_t$ of honest parties to corrupted parties and then honestly checks the correctness of the multiplication tuple $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t, [\![\boldsymbol{c}]\!]_t)$. If S receives **continue** from $\mathcal{F}_{\mathsf{verifymult}}$ but one honest party aborts, S sends **abort** to $\mathcal{F}_{\mathsf{verifymult}}$. Otherwise, S sends **continue** to $\mathcal{F}_{\mathsf{verifymult}}$.

We complete the simulation for $\Pi_{\mathsf{multcheck}}$ as above. Now we claim that S perfectly simulates the behaviors of honest parties with overwhelming probability. Consider the following hybrid arguments:

Hybrid$_0$: This is the real world execution.

Hybrid$_1$: In this hybrid, for each sharing, S computes the corrupted shares and additive errors to the shares of parties in H_2 as mentioned above. For each packed multiplication triple or inner-product tuple, S first computes the additive errors to the multiplication results or the inner-product results as mentioned above, then verifies them together with the additive errors to the shares of parties in H_2. We do not affect any behaviors of honest parties in this hybrid. By Lemma 6, Lemma 9, and Lemma 10, the probability that S sends **abort** during the verification is negligible. Therefore, the distribution of **Hybrid**$_0$ is statistically close to the distribution of **Hybrid**$_1$.

Hybrid$_2$: In this hybrid, S constructs the final triple $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t, [\![\boldsymbol{c}]\!]_t)$ as mentioned above instead of using real triples. Note that $[\![\boldsymbol{a}]\!]_t$ and $[\![\boldsymbol{b}]\!]_t$ are linear combinations of $([\![\boldsymbol{x}_i]\!]_t)_{i\in[m]}$ and $([\![\boldsymbol{y}_i]\!]_t)_{i\in[m]}$ respectively with nonzero coefficients in $\Pi_{\mathsf{randomization}}$. Both $[\![\boldsymbol{a}]\!]_t$ and $[\![\boldsymbol{b}]\!]_t$ are masked by a random pair of $[\![\boldsymbol{a}]\!]_t$ and $[\![\boldsymbol{b}]\!]_t$, which indicates that $\boldsymbol{a}, \boldsymbol{b}$ are uniformly random. The only difference between **Hybrid**$_1$ and **Hybrid**$_2$ is that, $\boldsymbol{a}, \boldsymbol{b}$ are randomly masked in the first hybrid, whereas $\boldsymbol{a}, \boldsymbol{b}$ are randomly sampled by S in the second hybrid. Hence, the distribution of **Hybrid**$_1$ is identical to the distribution of **Hybrid**$_2$.

Hybrid$_3$: In this hybrid, S totally follows the complete simulation. Note that honest parties are required to communicate with corrupted parties in the last two steps for checking the correctness of packed triples. However, preparing $([\![\boldsymbol{a}]\!]_t, [\![\boldsymbol{b}]\!]_t, [\![\boldsymbol{c}]\!]_t)$ can be performed by S without holding honest shares. Moreover,

the shares of corrupted parties and the additive errors to the shares of parties in H_2 that require S to send to the adversary have been computed in $\mathbf{Hybrid}_1$ when simulating multiplication and $\mathcal{F}_{\mathsf{inner\text{-}product}}$ by S. Therefore, S also does not need to know the shares of honest parties during the whole simulation. Thus, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_3$.

In summary, we conclude that $\mathbf{Hybrid}_3$, which is the execution in the ideal world is statistically close to $\mathbf{Hybrid}_0$, which is the execution in the real world. □

Communication Cost: Given h packed multiplication triples $(\llbracket \boldsymbol{x}_i \rrbracket_t, \llbracket \boldsymbol{y}_i \rrbracket_t, \llbracket \boldsymbol{z}_i \rrbracket_t)_{i\in[h]}$ over $\mathbb{F}_{2^m}$, $\Pi_{\mathsf{multcheck}}$ needs to invoke one invocation of $\Pi_{\mathsf{de\text{-}linearization}}$, $O(\log h)$ invocations of $\Pi_{\mathsf{reduction}}$ and one invocation of $\Pi_{\mathsf{randomization}}$. Therefore, checking h packed multiplication triples leads to a sublinear communication complexity as $O(\log h \cdot n^2 \cdot m^2)$ bits.

A.3 Efficiently Sampling

To use $\mathcal{F}_{\mathsf{rand}}(f, d)$ to generate random sharings, we need to prove that our generating scheme satisfies the following property:

- Given a set $A \subset I$ and a set of shares $(a_i)_{i\in A}$ held by parties in A, we denote

$$\Sigma(A, (a_i)_{i\in A}) = \{[x] | [x] \in \Sigma, \pi_A([x]) = (a_i)_{i\in A}\}$$

  Then there exists an efficient algorithm that outputs either a random $[r]$ in $\Sigma(A, (a_i)_{i\in A})$ or $\Sigma(A, (a_i)_{i\in A}) = \emptyset$.

Generating Random $(\llbracket \boldsymbol{r} \rrbracket_t, \llbracket \boldsymbol{r} \rrbracket_{2t})$. We consider a scheme Σ to generate $(\llbracket \boldsymbol{r} \rrbracket_t, \llbracket \boldsymbol{r} \rrbracket_{2t})$ and summarize as follows:

- Randomly sample k field elements $r_1, \ldots, r_k \in \mathbb{F}_{2^m}$.
- Randomly sample $(t + 1 - k)$ and another $(2t + 1 - k)$ field elements as the shares of the first and the second sharings.
- Distribute two packed secret sharings $\llbracket r_1, \ldots, r_k \rrbracket_t$ and $\llbracket \boldsymbol{r} \rrbracket_{2t}$.

It is convenient to check that Σ is an $\mathbb{F}_2$-GLSSS since main computations are operated over $\mathbb{F}_{2^m}$. Now we prove that Σ satisfies the above property. Depending on the size of A, there are three cases:

– If $|A| \geq 2t + 1 - k$, the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharing $\llbracket r_1, \ldots, r_k \rrbracket_t$. Here, $a_i = (a_i^1, a_i^2)$ denotes two shares of two sharings respectively. Then the algorithm checks whether the shares of $(a_i^1)_{i\in A}$ lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm first reconstructs k secrets $\boldsymbol{r}$. Later, the algorithm checks whether k secrets $\boldsymbol{r}$ and the shares of $(a_i^1)_{i\in A}$ lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most $2t$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm reconstructs the whole sharing $\llbracket \boldsymbol{r} \rrbracket_{2t}$ and outputs $(\llbracket \boldsymbol{r} \rrbracket_t, \llbracket \boldsymbol{r} \rrbracket_{2t})$.

– If $t+1 \leq |A| < 2t+1-k$, the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharing $[\![r_1,\ldots,r_k]\!]_t$. Here, $a_i = (a_i^1, a_i^2)$ denotes two shares of two sharings respectively. Then the algorithm checks whether the shares of $(a_i^1)_{i\in A}$ lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm first reconstructs k secrets $\boldsymbol{r}$. Later, the algorithm randomly samples $(2t+1-k-|A|)$ field elements in $\mathbb{F}_{2^m}$ and reconstructs the whole sharing $[\![\boldsymbol{r}]\!]_{2t}$ based on k secrets $\boldsymbol{r}$, the shares of $(a_i^2)_{i\in A}$ and new $(2t+1-k-|A|)$ field elements. The algorithm finally outputs $([\![\boldsymbol{r}]\!]_t, [\![\boldsymbol{r}]\!]_{2t})$.
– If $|A| < t+1$, the shares of $(a_i)_{i\in A}$ are independent of both k secrets. Here, $a_i = (a_i^1, a_i^2)$ denotes two shares of two sharings respectively. Then the algorithm first randomly samples $(t+1-k-|A|)$ field elements in $\mathbb{F}_{2^m}$ and reconstructs the whole sharing $[\![\boldsymbol{r}]\!]_t$ based on the shares of $(a_i^1)_{i\in A}$ and new $(t+1-k-|A|)$ field elements. Later, the algorithm continues to randomly sample $(2t+1-k-|A|)$ field elements in $\mathbb{F}_{2^m}$ and reconstructs the whole sharing $[\![\boldsymbol{r}]\!]_{2t}$ based on k secrets $\boldsymbol{r}$, the shares of $(a_i^2)_{i\in A}$ and new $(2t+1-k-|A|)$ field elements. The algorithm finally outputs $([\![\boldsymbol{r}]\!]_t, [\![\boldsymbol{r}]\!]_{2t})$.

Generating Random $([\![\boldsymbol{r}]\!]_t, [\![f(\boldsymbol{r})]\!]_t)$. We consider a scheme Σ to generate $([\![\boldsymbol{r}]\!]_t, [\![f(\boldsymbol{r})]\!]_t)$ and summarize as follows:

- Randomly sample k field elements $r_1,\ldots,r_k \in \mathbb{F}_{2^m}$.
- Compute $R_i = f(r_i) \in \mathbb{F}_{2^m}$.
- Randomly sample $(t+1-k)$ and another $(t+1-k)$ field elements as the shares of the first and the second sharings.
- Distribute two packed secret sharings $[\![r_1,\ldots,r_k]\!]_t$ and $[\![R_1,\ldots,R_k]\!]_t$.

It is convenient to check that Σ is an $\mathbb{F}_2$-GLSSS since main computations are operated over $\mathbb{F}_{2^m}$ and f is $\mathbb{F}_2$-linear. Now we prove that Σ satisfies the above property. Depending on the size of A, there are three cases:

– If $|A| \geq t+1$, the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharings. Here, $a_i = (a_i^1, a_i^2)$ denotes two shares of two sharings respectively. The algorithm first checks whether these shares lie on two polynomials $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm recovers k secrets $r_1,\ldots,r_k$ and secrets $f(r_1),\ldots,f(r_k)$. The algorithm can check whether $f(r_i) = f(r_i)$ for each $i \in [k]$. If not, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs two sharings determined by the shares of $(a_i)_{i\in A}$.
– If $|A| \leq t-k+1$, the shares of $(a_i)_{i\in A}$ are independent of both k secrets. The algorithm randomly samples $t+1-|A|$ field elements $s_1,\ldots,s_{t+1-|A|} \in \mathbb{F}_{2^m}$. Together with $|A|$ shares $(a_i^1)_{i\in A}$, the algorithm can reconstruct $[\![r_1,\ldots,r_k]\!]_t$ based on these $t+1$ shares. Then the algorithm locally computes $R_i = f(r_i)$ for each $i \in [k]$ and randomly samples another $t+1-k-|A|$ field elements $u_1,\ldots,u_{t+1-k-|A|} \in \mathbb{F}_{2^m}$. Based on $|A|$ shares $(a_i^2)_{i\in A}$, k secrets $(f(r_i))_{i\in[k]}$ and $t+1-k-|A|$ field elements $u_1,\ldots,u_{t+1-k-|A|}$, the algorithm outputs a packed secret sharing $[\![f(r_1),\ldots,f(r_k)]\!]_t$. Finally, the algorithm outputs two packed secret sharings as $[\![r_1,\ldots,r_k]\!]_t$ and $[\![f(r_1),\ldots,f(r_k)]\!]_t$.

– If $t-k+1 < |A| \leq t$, the shares of $(a_i)_{i\in A}$ do not determine the whole sharing but are correlated with secrets. Even worse, the shares to each sharing are also correlated with each other. In this case, we create $2t+2-2|A|$ variables $(b_i^1, b_i^2)_{i\in[t+1-|A|]}$. Each secret r_p can be written as a linear combination of $(a_i^1)_{i\in A} \cup (b_i^1)_{i\in[t+1-|A|]}$. Similarly, each secret $f(r_p)$ can be written as a linear combination of $(a_i^2)_{i\in A} \cup (b_i^2)_{i\in[t+1-|A|]}$. Now we decompose each share (a_i^1, a_i^2) and each variable (b_i^1, b_i^2) into bit representations. From checking $f(r_p) = f(r_p)$, we can obtain an $\mathbb{F}_2$-linear equation system on these bit shares and bit variables. It is well known that linear equations can be solved within polynomial time, and it is possible to efficiently compute a random solution if it exists. The algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$ if no solution exists. Otherwise, the algorithm outputs a pair of $([\![\boldsymbol{r}]\!]_t, [\![f(\boldsymbol{r})]\!]_t)$.

Generating Random $[\![\phi(\boldsymbol{r})]\!]_t$.

We consider a secret sharing scheme Σ to generate $[\![\phi(\boldsymbol{r})]\!]_t$ and summarize as follows:

- Randomly sample k vectors $\boldsymbol{r}^1, \ldots, \boldsymbol{r}^k \in \mathbb{F}_2^\ell$.
- Compute $R^i = \phi(\boldsymbol{r}^i) \in \mathbb{F}_{2^m}$.
- Randomly sample another $t-k+1$ field elements over $\mathbb{F}_{2^m}$ and generate $[\![R^1, \ldots, R^k]\!]_d$ based on k secrets R^i and $t-k+1$ random elements.

It is convenient to check that Σ is an $\mathbb{F}_2$-GLSSS since main computations are operated over $\mathbb{F}_{2^m}$ and f is $\mathbb{F}_2$-linear. Now we prove that Σ satisfies the above property. Depending on the size of A, there are three cases:

– If $|A| \geq t+1$, the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharing. The algorithm first checks whether these shares lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm recovers k secrets $x_1, \ldots, x_k$ and checks whether each secret is the image of ϕ. The algorithm can check whether $\phi(\phi^{-1}(x_i)) = x_i$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs the PSS determined by the shares of $(a_i)_{i\in A}$.
– If $|A| \leq t-k+1$, the shares of $(a_i)_{i\in A}$ are independent of k secrets. The algorithm randomly samples k secrets in $\mathbb{F}_2^\ell$ and encodes them to $\phi(\boldsymbol{r}^1), \cdots, \phi(\boldsymbol{r}^k)$. The algorithm also randomly samples $t+1-k-|A|$ field elements over $\mathbb{F}_{2^m}$, and computes the whole sharings based on the shares of $(a_i)_{i\in A}$, k secrets together with $t+1-k-|A|$ random elements. Finally, the algorithm outputs $[\![\phi(\boldsymbol{r}^1), \ldots, \phi(\boldsymbol{r}^k)]\!]_t$.
– If $t-k+1 < |A| \leq t$, the shares of $(a_i)_{i\in A}$ do not determine the whole sharing but are correlated with secrets. Even worse, these shares can not guarantee that each secret is the image of ϕ. Let B be a subset of I such that $|B| = t+1-|A|$ and $B \cap A = \emptyset$. The algorithm takes another $t+1-|A|$ field elements $(b_j)_{j\in B}$ over $\mathbb{F}_{2^m}$. The shares of $(a_i)_{i\in A}$ and $(b_j)_{j\in B}$ can fully determine the whole sharing, i.e., each secret can be written as a linearization of these $t+1$ shares. The algorithm first considers $(b_j)_{j\in B}$ as variables and constructs

equations over $\mathbb{F}_2$ rather than $\mathbb{F}_{2^m}$. Concretely, let $x_1, \ldots, x_k$ be k secrets determined by the shares of $(a_i)_{i\in A}$ and $(b_j)_{j\in B}$ and $x_i \cong (x_i^1, \ldots, x_i^m) \in \mathbb{F}_2^m$ for each $i \in [k]$, the constraint $\phi(\phi^{-1}(x_i)) = x_i$ can be converted to a $\mathbb{F}_2$-linear equation system involving $x_i^1, \ldots, x_i^m$ and each bit of all elements in $(a_i)_{i\in A}$ and $(b_j)_{j\in B}$ for all $i \in [k]$ since ϕ is $\mathbb{F}_2$-linear. It is well known that linear equations can be solved within polynomial time. The algorithm checks whether there exists a solution. If it exists, the algorithm outputs the final sharing. Otherwise, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$.

Generating Random $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$. We denote $[\![f(\boldsymbol{r})]\!]_t = [\![\phi(\psi(\boldsymbol{r}))]\!]_t$ for simplicity. We consider a scheme Σ to generate $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![f(\boldsymbol{r})]\!]_t)$ and summarize as follows:

- Randomly sample k field elements $r_1, \ldots, r_k \in \mathbb{F}_{2^m}$.
- Compute $R_i = f(r_i) \in \mathbb{F}_{2^m}$ and samples another $t - k + 1$ field elements as the shares of $[\![\boldsymbol{r}]\!]_{t+k-1}$.
- Distribute two packed secret sharings $[\![\boldsymbol{r}]\!]_{t+k-1}$ and $[\![R_1, \ldots, R_k]\!]_t$.

It is convenient to check that Σ is an $\mathbb{F}_2$-GLSSS since main computations are operated over $\mathbb{F}_{2^m}$ and f is $\mathbb{F}_2$-linear. Now we prove that Σ satisfies the above property. Depending on the size of A, there are four cases:

- If $|A| \geq t + k$, the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharing $[\![r_1, \ldots, r_k]\!]_{t+k-1}$. The algorithm first checks whether these shares lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm recovers k secrets $r_1, \ldots, r_k$ and checks whether $R_i = f(r_i) \in \mathbb{F}_{2^m}$ for all $i \in [k]$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs the PSS determined by the shares of $(a_i)_{i\in A}$.
- If $|A| \leq t - k + 1$, the shares of $(a_i)_{i\in A}$ are independent of secrets. The algorithm randomly samples k secrets $r_1, \ldots, r_k \in \mathbb{F}_{2^m}$ and encodes them to $R_i = f(r_i) \in \mathbb{F}_{2^m}$. The algorithm further samples $t+1-k-|A|$ random field elements as shares of the second sharing and another $t - |A|$ random field elements as the shares of the first sharing. Finally, the algorithm outputs a pair of random $([\![\boldsymbol{r}]\!]_{t+k-1}, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$.
- If $t - k + 1 < |A| \leq t$, the algorithm introduces $t + 1 - |A|$ free variables to the second sharing and $t + k - |A|$ free variables to the second sharing. Then, a $\mathbb{F}_2$-2 linear system can be induced by solving $R_i = f(r_i) \in \mathbb{F}_{2^m}$. If the system exists solutions, then the algorithm allocates free variables to random elements and output final sharings. If the system has no solutions, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$.
- If $t + 1 \leq |A| < t + k$, then the shares of $(a_i)_{i\in A}$ can fully determine the whole packed secret sharing $[\![\phi(\psi(\boldsymbol{r}))]\!]_t$. The algorithm first checks whether these shares lie on a polynomial $f(\cdot) \in \mathbb{F}_{2^m}[X]$ of degree at most t. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm recovers k secrets $R_1, \ldots, R_k$. The algorithm introduces $t + k - |A|$ free variables to solve a $\mathbb{F}_2$-2 linear system induced by $R_i = f(r_i) \in \mathbb{F}_{2^m}$. If the system exists

solutions, then the algorithm allocates free variables to random elements and output final sharings. If the system has no solutions, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$.

B Bipartite Graph and Hall's Marriage Theorem

Here, we present some combinatorial tools for a better explanation and proof of the network routing.

Definition 7. *A graph $G = (V, E)$ is a bipartite graph if there exists a partition of $V = (V_1, V_2)$ such that for all edges $(v_i, v_j) \in E$, $v_i \in V_1$ and $v_j \in V_2$.*

We usually denote (V_1, V_2, E) to be a bipartite graph. We say that a bipartite graph is a k-regular bipartite graph if for each vertex $v \in V$, the degree of v is exactly k.

Definition 8. *For a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$, a perfect matching is a subset of G as $G' = (V_1, V_2, E') \subset G$ which satisfies that each vertex in $V_1 \cup V_2$ is of degree 1.*

Theorem 10 (Hall's Marriage Theorem). *For a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$, there exists a perfect matching in G if and only if for all subsets $V_1' \subset V_1$, we have $|N(V_1')| \geq |V_1'|$, where $N(V_1')$ denotes the neighbor of vertices in V_2.*

Proof. If G has a perfect matching G', for each subset $V_1' \in V_1$, any vertex v in V_1' is connected to a different vertex $u_v \in V_2$ such that (v, u_v) is an edge of G'. Then we have $|N(V_1')| \geq |\{u_v | v \in V_1'\}| = |V_1'|$.

In the opposite direction, we give an induction proof on the size of $|V_1|$. If $|V_1| = 1$, the case is trivial. Suppose $|V_1| > 1$. Pick any vertex $v \in V_1$. v is connected to some $u \in V_2$ since $|N(v)| \geq |v| = 1$. Now we remove the edge (v, u) and its two vertices from G. If the left graph also satisfies the required condition, by induction hypothesis, there is a perfect matching in this graph. Plus the edge (v, u), we obtain a perfect matching of G. Otherwise, the only case where this process would fail is that there exists a subset $S \subset V_1 - v$ such that $N(S) \subset V_2 - u, |N(S)| < |S|$. Back to G, S must have $|S|$ neighbors in V_2. By induction hypothesis, there exists a perfect matching between S and $N(S)$. For each subset $S' \subset V_1 - S$, $S \cup S'$ has at least $|S| + |S'| = |N(S)| + |S'|$ neighbors in V_2. Then S' has at least $|S'|$ neighbors in $V_2 - N(S)$. By induction hypothesis, there exists a perfect matching between $V_1 - S$ and $V_2 - N(S)$. Finally, we combine two perfect matchings to one perfect matching of G as required. □

Lemma 11. *A regular bipartite graph has a perfect matching. And the perfect matching can be found within polynomial time.*

Proof. Let $G = (V_1, V_2, E)$ be a k-regular bipartite graph. For each subset $V_1' \subset V_1$, the number of edges connected to one vertex in V_1' is exactly $k \cdot |V_1'|$. And each vertex in $N(V_1')$ has k edges in G. Therefore, $k \cdot |N(V_1')| \geq k \cdot |V_1'| \Rightarrow |N(V_1')| \geq |V_1'|$. By Hall's Marriage Theorem, there exists a perfect matching in G. □

Moreover, there exist known efficient polynomial-time algorithms to find a perfect matching in a bipartite graph, e.g., the Hopcroft-Karp algorithm. Hence, a perfect matching can be found within polynomial time in a regular bipartite graph.

Theorem 11 (Theorem 6, restated). *Let N be a $u \times k\ell$ matrix in $\{1, \ldots, u\}^{u \times k\ell}$ such that there exactly exists $k \cdot \ell$ entries $N_{i,j}$ that are equal to v for all $v \in [u]$. Then, we can find m permutations $p_1, \ldots, p_u$ over $\{1, \ldots, k \cdot \ell\}$ such that after operating p_i on the i-th row of N, each column in the new matrix N' forms a permutation over $\{1, \ldots, u\}$. Additionally, $p_1, \ldots, p_u$ can be found efficiently, i.e., within polynomial time.*

Proof. Let $p = k \cdot \ell$ be a positive integer. We prove this theorem by induction on p. If $p = 1$, N is an $u \times 1$ matrix and contains only one column (itself) where all elements of it form a permutation over $\{1, \ldots, u\}$. We assume that Theorem 6 holds for p and consider the case for $p+1$. We will construct a bipartite graph $G = (V_1, V_2, E)$, where $V_1 = V_2 = \{1, \ldots, u\}$. If $N_{i,j} = v \in [u]$, we add an edge $(i, v) \in V_1 \times V_2$ for any $i \in [u], j \in [p]$.

Next, we show that G is a p-regular bipartite graph. For each $i \in V_1$, the i-th column contains p elements $N_{i,j}, j \in [p]$. Then there exists p edges connected with i. For each $v \in V_2$, there exists exactly p entries such that $N_{i,j} = v$. Then there exists p edges connected with v. Hence, G is a p-regular bipartite.

By Lemma 11, there exists a perfect matching $(1, v_1), \ldots, (u, v_u)$ in G. We find the first index j_i such that $N_{i,j_i} = v_i$. For $i \in [u]$, we switch the p-th entry and the j_i-th entry in the i-th row. Then all entries in the last column form a permutation over $\{1, \ldots, u\}$ after swapping. We denote N' to be the new matrix. Consider a sub-matrix M that contains the first $p-1$ columns of N'. We note that M exactly satisfies the induction hypothesis for the case $p - 1$. By the induction hypothesis, there exists u permutations $p'_1, \ldots, p'_u$ over $\{1, \ldots, p-1\}$ such that after operating p'_i on the i-th row of M, each column in the new matrix M' forms a permutation over $\{1, \ldots, u\}$.

Now we present u permutations $p_1, \ldots, p_u$ over $\{1, \ldots, p\}$ as follows: For all $i \in [u]$,

- For each $j \in [p-1], j \neq j_i$, $p_i(j) = p'_i(j)$
- For each j_i, $p_i(j_i) = p$
- For p, $p_i(p) = p'_i(j_i)$

Supposing that each p_i operates on the i-th row of N. For $j \in [p-1]$, the j-th column is the same as the j-th column in M'. For $j = p$, the p-th column is the same as the j-th column in N'. Therefore, each column in the new matrix after operations from each p_i forms a permutation over $\{1, \ldots, u\}$.

In addition, by the induction hypothesis and Lemma 11, all permutations above can be found within polynomial time. □

C Circuit Transformation

In this part, we present a circuit transformation phase that efficiently complies a circuit C into a circuit C' satisfying the following property:

- For all input $\boldsymbol{x}$, $C'(x) = C(x)$.
- In the input layer and the output layer, the number of input gates belonging to each client and the number of output gates belonging to each client are multiples of $k \cdot \ell$. In each intermediate layer, the number of addition gates and the number of multiplication gates are multiples of $k \cdot \ell$.
- C' will group $k \cdot \ell$ gates with the same type in each layer such that the number of times that the output wires of each group are used later is a multiple of $k \cdot l$.
- Circuit size: $|C'| = O(|C| + k \cdot \ell \cdot (c + d(C))$, where c denotes the number of clients and $d(C)$ denotes the number of the depth of C.

This is achieved by creating a virtual party C_0 which handles with extra gates that we create in each layer. Let $C_1, \ldots, C_c$ be all the clients to this protocol. For each layer, we offers two type of gates: One is the input gate to C_0, another is the output gate to C_0. C_0 will share public values to input gates and receive outputs which are not sent to any clients except C_0. Without loss of generality, we may assume that each input of C_0 is used once in the circuit. If one input is used multiple times, we can add multiple input gates to C_0 such that since all inputs of C_0 are known to all clients.

In the i-th layer, we denote M_i and A_i to be the number of multiplication and addition gates respectively. For these two gates, we add $\frac{\lceil M_i \rceil}{k \cdot \ell} \cdot k \cdot \ell - M_i$ multiplication gates and $\frac{\lceil A_i \rceil}{k \cdot \ell} \cdot k \cdot \ell - A_i$ addition gates in the i-th layer. C_0 inputs 0 to two input wires to all gates and receives outputs from all gates. After inserting gates, the number of addition gates and the number of multiplication gates are multiples of $k \cdot \ell$ in the i-th layer. This operation increase the circuit size with $O(k \cdot \ell \cdot d(C))$.

In the i-th layer, then we can group each $k \cdot \ell$ gates with the same type. For each group of gates, we use $(w_1, \ldots, w_{k \cdot \ell})$ to denote output wires to this group and $(n_1, \ldots, n_{k \cdot \ell})$ to denote the number of times that each output wire is used in later layers. We add $n^i = \frac{\lceil n_1 + \cdots + n_{k \cdot \ell} \rceil}{k \cdot \ell} \cdot k \cdot \ell - (n_1 + \cdots + n_{k \cdot \ell})$ output gates in this layer. Specifically, we replicate the gate that outputs $w_{k \cdot \ell}$ n^i times but require these n^i gates to output $w_{k \cdot \ell}$ to C_0 only. This operation can be achieved by a fan-out gate that copies n_1 times of w_1, $\ldots$, $n_{k \cdot \ell - 1}$ times of $w_{k \cdot \ell - 1}$ and $n_{k \cdot \ell} + n^i$ times of $w_{k \cdot \ell}$. Note that $n^i < k \cdot \ell \leq (n_1 + \cdots + n_{k \cdot \ell})$, this operation in this step increases the circuit size by at most a factor of 2.

For each client C_i, we denote c_i and d_i to be the number of input gates and output gates belonging to C_i respectively. We add $\frac{\lceil c_i \rceil}{k \cdot \ell} \cdot k \cdot \ell - c_i$ input gates belonging to C_i. And we add the same number of output gates belonging to C_0 that take these inputs as output. Similarly, we add $\frac{\lceil d_i \rceil}{k \cdot \ell} \cdot k \cdot \ell - d_i$ output gates belonging to C_0 that takes 0 as inputs. We also add the same number of output gates belonging to C_i that take these inputs as output. For the virtual client C_0,

we add $\frac{\lceil c_0 \rceil}{k \cdot \ell} \cdot k \cdot \ell - c_0$ input gates belonging to C_0 that take 0 as inputs, where c_0 is the number of input gates belonging to C_0. We also add the same number of output gates belonging to C_0 that take these inputs as output. After inserting gates, the number of input gates and output gates belonging to each client C_i are multiples of $k \cdot \ell$. This operation increase the circuit size by $O(k \cdot \ell \cdot c)$.

For each client C_i, it can group each $k \cdot \ell$ input gates and each $k \cdot \ell$ output gates. The remaining part is to ensure the sum of output wires to each group to be a multiple of $k \cdot \ell$. We can achieve this by following the above operation similarly.

Now we claim that the presented properties is satisfies after the above operations. First, in the input layer, we add some input gates to each client C_i that directly outputs 0 to the output gates belonging to C_0. In the output layer, we add some output gates to each client C_i that take inputs 0 from input gates belonging to C_0. Therefore, these new gates do not affect the whole computation, which indicates that correctness is guaranteed after the operation.

For the second property, by adding input gates to each client, including C_0, the number of input gates belonging to each client is a multiple of $k \cdot \ell$. For the intermediate layer, by adding multiplication and addition gates belonging to C_0, the number of addition gates and the number of multiplication gates are multiples of $k \cdot \ell$. By adding output gates to each client, including C_0 shown in the third step, the number of output gates belonging to each client is a multiple of $k \cdot \ell$ to each client except C_0. As for C_0, we notice that the total number of output wires of the input layer and all intermediate layers is a multiple of $k \cdot \ell$, and the number of input wires of each intermediate layer is also a multiple of $k \cdot \ell$. Hence, the input wires of the output layer are a multiple of $k \cdot \ell$, which implies that the number of output gates is a multiple of $k \cdot \ell$. And the number of output gates belonging to each client $C_1, \ldots, C_c$ is a multiple of $k \cdot \ell$ as proved above. Then, the number of output gates belonging to C_0 is a multiple of $k \cdot \ell$.

For the third property, by adding proper gates belonging to C_0 shown in the second operation, the output wires of each group are used later is a multiple of $k \cdot \ell$.

For the last property, we sum together the costs of each operation to compute as $|C'| = O(|C| + k \cdot \ell \cdot (c + d(C))$.

Application: With the above transformation, all clients and all parties can distribute or use the sharing $[\![\phi(\boldsymbol{x})]\!]_t = [\![\phi(\boldsymbol{x}^1), \ldots, \phi(\boldsymbol{x}^k)]\!]_t$ that contains each $k \cdot \ell$ secrets as a group to carry out computation. Moreover, if the number of output wires to each group is a multiple of $k \cdot \ell$, then we can accurately copy each secret in $[\![\phi(\boldsymbol{x})]\!]_t$ multiple times that it is used in later layers by using a fan-out gate. Then, we satisfy the property that every output wire is only used once as an input wire in a later layer. Together with the achieved property in this section, the number of the input wires and the number of the output wires of each layer are multiples of $k \cdot \ell$. We can use Theorem 6 to evaluate circuits in this paper.

D Proof of Lemma 3

Lemma 12 (Lemma 3, restated). *If Σ is a multiplication-friendly threshold-t $\mathbb{F}$-ASSS, and Σ' being the $\mathbb{F}$-ASSS defined in Definition 6, then Σ' holds threshold t.*

Proof. We are required to prove

$$Pr_{\boldsymbol{\rho}'}(\pi_A(\mathbf{share}'(\boldsymbol{x}_1, \boldsymbol{\rho}')) = \boldsymbol{u}') = Pr_{\boldsymbol{\rho}'}(\pi_A(\mathbf{share}'(\boldsymbol{x}_2, \boldsymbol{\rho}')) = \boldsymbol{u}').$$

for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in Z$ and $\boldsymbol{u}' \in (U')^A$. Since Σ holds threshold t, then we have

$$Pr_{\boldsymbol{\rho}}(\pi_A(\mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho})) = \boldsymbol{u}) = Pr_{\boldsymbol{\rho}}(\pi_A(\mathbf{share}(\boldsymbol{x}_2, \boldsymbol{\rho})) = \boldsymbol{u}).$$

We denote $\mathbf{1}$ to be the vector $(1, 1, \ldots, 1) \in Z$ and $\mathbf{0}$ similarly. Then for all $\boldsymbol{u}' \in (U')^A$, we can obtain

$$\begin{aligned} &Pr_{\boldsymbol{\rho},\boldsymbol{\rho}'}(\pi_A(\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho}) + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}')) = \boldsymbol{u}') = \\ &Pr_{\boldsymbol{\rho},\boldsymbol{\rho}'}(\pi_A(\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_2, \boldsymbol{\rho}) + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}')) = \boldsymbol{u}'). \end{aligned}$$

Given fixed $\boldsymbol{\rho}'$ and $\boldsymbol{u}'$, the equation $\pi_A(\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho}) + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}')) = \boldsymbol{u}'$ can be reduced to $\pi_A(\mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho})) = \boldsymbol{u}$ for some $\boldsymbol{u} \in U^A$ or a case that exists no solution. Therefore, the above two probabilities are the same. Now we only need to prove that $\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho}) + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}')$ is a random Σ'-sharing. Actually, $\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho})$ is a Σ'-sharing of $\boldsymbol{x}_1$, then there exists a $\boldsymbol{\rho}''$ such that $\mathbf{share}'(\boldsymbol{x}_1, \boldsymbol{\rho}'') = \mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho})$. Then, $\mathbf{1} \cdot \mathbf{share}(\boldsymbol{x}_1, \boldsymbol{\rho}) + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}') = \mathbf{share}'(\boldsymbol{x}_1, \boldsymbol{\rho}'') + \mathbf{share}'(\mathbf{0}, \boldsymbol{\rho}') = \mathbf{share}'(\boldsymbol{x}_1, \boldsymbol{\rho}' + \boldsymbol{\rho}'')$ is a random Σ'-sharing of $\boldsymbol{x}_1$ since $\boldsymbol{\rho}' + \boldsymbol{\rho}''$ is uniformly random. Hence, the last equation is equivalent to the first equation that we wish to prove. □

Lemma 13 (Lemma 2, restated). *Let L be an extension field of $\mathbb{K}$ of degree m. Let V be a $\mathbb{K}$-vector space. Then the following holds:*

1. *Let $\mathbb{K}^{m,m}$ denote the matrix algebra over $\mathbb{K}$ consisting of all $m \times m$ matrices with entries in $\mathbb{K}$, with the usual addition and $\mathbb{K}$-scalar multiplication. Then there is a (non-unique) injective $\mathbb{K}$-algebra morphism*

$$\Phi : Ł \to \mathbb{K}^{m,m}; \lambda \to \Phi^{(\lambda)}.$$

   , where a $\mathbb{K}$-algebra is a ring having the field $\mathbb{K}$ as a subring and a $\mathbb{K}$-algebra morphism $R \to S$ is a ring morphism that fixes $\mathbb{K}$. In particular, the image of L is a field isomorphic to it.
2. *Each such Φ induces an Ł-vector space structure on V^m by defining L-scalar multiplication, for each $\lambda \in Ł$, as*

$$\lambda \cdot V^m \to V^m; w \to \Phi^{(\lambda)}(w),$$

   where the action of $\mathbb{K}^{m,m}$ on V^m is natural one, i.e., multiplication of an $m \times m$ matrix with an m-column vector.

If $\lambda \in \mathbb{K}$, it restricts to

$$\lambda \cdot V^m \to V^m; w \to \lambda \cdot w,$$

Since Φ fixes $\mathbb{K}$. Hence, this structure is compatible with the standard $\mathbb{K}$-vector space structure on V^m, i.e., as a direct sum over V.

Proof. As to the first claim, for each $\lambda \in L$, the multiplication-by-λ map on L is a $\mathbb{K}$-vector space endomorphism on L (i.e., a morphism from L to L). The map Φ that sends $\lambda \in L$ to this associated morphism is clearly a K-algebra morphism from L to the $\mathbb{K}$-algebra $\mathrm{End}_{\mathbb{K}}(L)$ of $\mathbb{K}$-vector space endomorphisms of L. Note that Φ is injective as its domain is a field; the only possibility for its kernel is the trivial ideal (0) of L. Since L is a vector space of dimension m over $\mathbb{K}$, it is clear that, once a basis is fixed, $\mathrm{End}_{\mathbb{K}}(L)$ may be given as $\mathbb{K}^{m\times m}$.

As to the second claim, an R-module M consists of an abelian group M, a ring R (with 1), together with a ring morphism mapping R to the ring of group endomorphisms of M. It is called a vector space if R is a field. In the present case, M is the direct sum V^m and R is L. By construction, (L, V^m, Φ) satisfies this condition. Finally, note that Φ maps $\lambda \in \mathbb{K}$ to $\lambda \cdot I$, where I is the identity matrix.

As the lemma reflects one of the basic merits of tensor products, we verify it below in such terms and in more generality. By the very definition of the tensor product, if M is an R-module (where M and R take the role of V and K, resp.), where R is a commutative ring with 1, and if S is an extension ring (S takes the role of L), then the tensor product $S \otimes_R M$ is an R-module. By base change, we may, in fact, naturally view $S \otimes_R M$ as an S-module, compatible with the R-module structure already mentioned. Namely, for each $s \in S$, for each $r \in R$ and for each $m \in M$, define $s \cdot (r \otimes m) = (sr \otimes m)$ and extend this linearly to all of $S \otimes_R M$. If, in addition, S is free of rank r over R, then, as an R-module, the tensor product $S \otimes_R M$ is isomorphic to M^r. Since S-multiplication by a constant as defined above is, in particular, an endomorphism of R-modules, it is clear that such a map can be represented by an element of $\mathbb{K}^{m\times m}$.

E Proof in the semi-honest setting

E.1 Proof for randomness

Theorem 12 (Theorem 5, restated). *For any $\mathbb{F}_2$-arithmetic secret sharing schemes $\Pi_1, \ldots, \Pi_{kl}$ described by $\mathbb{F}_2$-linear functions $f_1, \ldots, f_{k\cdot\ell}$ with each $f_i : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$, Protocol 4 securely realizes $\{\mathcal{F}_{\mathsf{rand}}(\Pi_i)\}_{i=1}^{k\cdot\ell}$ in the $(\mathcal{F}_{\mathsf{rand}}(f, d), \mathcal{F}_{\mathsf{zero}})$-hybrid model against a semi-honest adversary controlling $t - k + 1$ parties.*

Proof. Let A be the adversary in the real world. We will construct a simulator S in the ideal world to simulate the behaviors of honest parties. (We will omit this part in later proofs.)

Simulation for Protocol $\Pi_{\mathsf{specific\text{-}rand}}$: The simulator works as follows:

- In Step 2, S receives the shares of $\{[\![\phi(\boldsymbol{r}_i)]\!]_t\}_{i\in[n\cdot m]}$ held by corrupted parties.
- In Step 3, S receives the shares of $\{[\![\mathbf{0}^{j,1}]\!]_{t+k-1},\ldots,[\![\mathbf{0}^{j,m}]\!]_{t+k-1}\}_{j\in[n]}$ held by corrupted parties.
- In Step 4, S obtains the coefficients $c^{j,w}_{1,p},\ldots,c^{j,w}_{n\cdot m,p}\in\mathbb{F}_2$ from $\mathbf{share}_p$ and $\mathbf{share}^{j,w}_p$ for all $p\in[k\cdot\ell]$, $j\in[n]$ and $w\in[m]$.
- In Step 6, S first computes the shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by corrupted parties. Later, S must simulate the shares that are sent from honest parties to corrupted parties. S executes as follows:
  - For each $p\in[k\cdot\ell]$, S randomly samples a Π_p-sharing and obtains the shares of corrupted parties $\{\boldsymbol{u}^j_p\}_{j\in C}$, where $\boldsymbol{u}^j_p=(u^{j,1}_p,\ldots,u^{j,m}_p)$.
  - For each $j\in[n]$ and $w\in[m]$, S can compute the secret vector $\boldsymbol{u}^{j,w}$ based on secret vectors $\boldsymbol{u}^j_p$.
  - A packed secret sharing with degree $t+k-1$ has threshold t, whereas A only controls $t+1-k$ corrupted parties. S constructs a random $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ based on k secrets calculated by $\boldsymbol{u}^{j,w}$, $t+1-k$ corrupted shares computed by S and together with another random $k-1$ field elements over $\mathbb{F}_{2^m}$ sampled by S.
  - S sends the shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by honest parties to A.
- In Step 7, S sends the shares of $\{\boldsymbol{u}^j_p\}_{j\in C}$ to $\{\mathcal{F}_{\mathsf{rand}}(\Pi_i)\}^{k\cdot\ell}_{i=1}$.

.

Now we claim that S perfectly simulates the behaviors of honest parties. For each secret in $\{\boldsymbol{u}^j_p\}_{j\in C}$, in the ideal world, S obtains them by randomly sampling a Π_p-sharing. In the real world, the shares of corrupted parties are determined by $\mathbf{share}_p(\boldsymbol{x}_p,\boldsymbol{\rho}_p)$. The input $\boldsymbol{x}_p$ and the random tape $\boldsymbol{\rho}_p$ are hidden by many degree-t PSS, which are exactly private against $t+1-k$ corrupted parties. Therefore, $(\boldsymbol{x}_p,\boldsymbol{\rho}_p)$ are random to A. Hence, the distributions of $\{\boldsymbol{u}^j_p\}_{j\in C}$ are the same between two worlds.

Here, we prove that the shares that are sent from honest parties to corrupted parties have the same distribution between the real and the ideal world. In the real world, each $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ is masked by a random zero sharing $[\![\mathbf{0}^{j,w}]\!]_{t+k-1}$. Then $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ is random to A. Moreover, a degree-$t+k-1$ PSS is private against t shares, which implies that $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ is also random given k secrets and $t+1-k$ corrupted shares. $\boldsymbol{u}'^{j,w}$ is determined by $\{\boldsymbol{x}_p,\boldsymbol{\rho}_p\}_{p\in[k\cdot\ell]}$, which are independent of the corrupted shares of $\{[\![\phi(\boldsymbol{r}_i)]\!]_t\}_{i\in[n\cdot m]}$ and $\{[\![\mathbf{0}^1_j]\!]_t,\ldots,[\![\mathbf{0}^m_j]\!]_t\}_{j\in[n]}$. Hence, the secret $\boldsymbol{u}'^{j,w}$ is independent of the shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by corrupted parties.

In the ideal world, S can compute the shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by corrupted parties by $\mathbb{F}_2$-linear calculations over the shares of $\{[\![\phi(\boldsymbol{r}_i)]\!]_t\}_{i\in[n\cdot m]}$ and the shares of $\{[\![\mathbf{0}^1_j]\!]_t,\ldots,[\![\mathbf{0}^m_j]\!]_t\}_{j\in[n]}$ held by corrupted parties. We have proved that the secret $\boldsymbol{u}'^{j,w}$ shows the same distribution between two worlds. Noting that $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ is determined by $\boldsymbol{u}'^{j,w}$, $t+1-k$ shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by corrupted parties and another $k-1$ random field elements. Hence, the shares of honest parties simulated by S have the same distribution in the real world.

Finally, we show that the output distribution of honest parties has the same distribution as that in the real world, given the joint view of corrupted parties in

Functionality $\mathcal{F}_{\mathsf{semi\text{-}mult}}$

Let $[\![\phi(\boldsymbol{x})]\!]_t$ and $[\![\phi(\boldsymbol{y})]\!]_t$ be the input packed Shamir sharings of the multiplication gate.

1. $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ receives the shares of $[\![\phi(\boldsymbol{x})]\!]_t$ and $[\![\phi(\boldsymbol{y})]\!]_t$ from honest parties.
2. $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ reconstructs the secrets $(\phi(\boldsymbol{x}), \phi(\boldsymbol{y}))$, and computes $(\boldsymbol{x} = \phi^{-1}(\phi(\boldsymbol{x})), \boldsymbol{y} = \phi^{-1}(\phi(\boldsymbol{y})))$.
3. $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ further computes $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$.
4. $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ receives a set of shares $(s_j)_{j\in C}$ from the adversary. Based on k secrets from $\boldsymbol{z}$ and $t-k+1$ shares from the adversary, $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ reconstructs the whole sharing $[\![\phi(\boldsymbol{z})]\!]_t$.
5. $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ send shares of $[\![\phi(\boldsymbol{z})]\!]_t$ to honest parties.

Fig. 27: **Semi-mult**

the ideal world. The joint view of corrupted parties is determined by the shares of $\{[\![\phi(\boldsymbol{r}_i)]\!]_t\}_{i\in[n\cdot m]}$ and the shares of $\{[\![\mathbf{0}_j^1]\!]_t, \ldots, [\![\mathbf{0}_j^m]\!]_t\}_{j\in[n]}$ held by corrupted parties, and the shares of $[\![\boldsymbol{u}'^{j,w}]\!]_{t+k-1}$ held by honest parties. The former is independent of secrets guaranteed by the threshold; the latter has been proved in the above. Hence, in two worlds, the output sharing $\mathbf{share}_p(\boldsymbol{x}_p, \boldsymbol{\rho}_p)$ is truly random with the same distributions. □

E.2 Proof for semi-honest protocols

In this section, we give a formal proof for realizing $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ shown in 27 based on our constructed protocols.

Theorem 13. *Protocol Π_{mult} shown in Figure 8 securely realizes $\mathcal{F}_{\mathsf{semi\text{-}mult}}$ in the $(\mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{rand}}(f,d))$-hybrid model against a semi-honest adversary controlling $t-k+1$ parties.*

Proof. The simulation is divided into two parts: simulating calculations from $[\![\phi(\boldsymbol{x})]\!]_t \cdot [\![\phi(\boldsymbol{y})]\!]_t \to [\![\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})]\!]_t$ and simulating encoding $[\![\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})]\!]_t$ to $[\![\phi(\boldsymbol{x} * \boldsymbol{y})]\!]_t$.

– For the first part, we first prove that the honest shares of $[\![r_1, \ldots, r_k]\!]_{2t}$ are uniformly random given the shares of $([\![r_1, \ldots, r_k]\!]_t, [\![r_1, \ldots, r_k]\!]_{2t})$ held by corrupted parties. Each degree-t and each degree-$2t$ PSS guarantees the privacy against $t' = t-k+1$ corrupted parties.
To generate $([\![r_1, \ldots, r_k]\!]_t, [\![r_1, \ldots, r_k]\!]_{2t})$, $\mathcal{F}_{\mathsf{rand}}$ receives a set of shares $(s_i^1, s_i^2)_{i\in C}$ from the adversary. Then $\mathcal{F}_{\mathsf{rand}}$ randomly samples one element to each honest party to reconstruct $[\![r_1, \ldots, r_k]\!]_{2t}$ together with $t-k+1$ shares $(s_i^1)_{i\in C}$. $\mathcal{F}_{\mathsf{rand}}$ recovers the secret $r_1, \ldots, r_k$, and reconstructs $[\![r_1, \ldots, r_k]\!]_t$ based on k secrets and $t-k+1$ corrupted shares. Therefore, the honest shares of $[\![r_1, \ldots, r_k]\!]_{2t}$ are uniform given the shares $(s_i^1, s_i^2)_{i\in C}$ in this generation.

Functionality $\mathcal{F}_{\mathsf{semi\text{-}convert}}$

Let $f : \mathbb{F}_2^{k\cdot\ell} \to \mathbb{F}_2^{k\cdot\ell}$ be an $\mathbb{F}_2$-linear map. This Functionality converts $[\![\phi(\boldsymbol{x})]\!]_t$ to $[\![\phi(f(\boldsymbol{x}))]\!]_t$.

1. $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ receives the honest shares of $[\![\phi(\boldsymbol{x})]\!]_t$.
2. $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ reconstructs the secrets $\phi(\boldsymbol{x})$, and computes $\boldsymbol{x} = \phi^{-1}(\phi(\boldsymbol{x}))$.
3. $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ further computes $\boldsymbol{y} = f(\boldsymbol{x})$.
4. $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ receives a set of shares $(s_j)_{j\in C}$ from the adversary. Based on k secrets from $\boldsymbol{y}$ and $t-k+1$ shares from the adversary, $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ reconstructs the whole sharing $[\![\phi(\boldsymbol{y})]\!]_t$.
5. $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ send shares of $[\![\phi(\boldsymbol{y})]\!]_t$ to honest parties.

Fig. 28: **Semi-convert**

Here, we describe the simulation of S. In Step 1, S emulates $\mathcal{F}_{\mathsf{rand}}$ to receive $(s_i^1, s_i^2)_{i\in C}$ from the adversary. In Step 2, S samples a random element to each honest party as its share of $[\![r_1, \ldots, r_k]\!]_{2t}$. Since the honest shares of $[\![r_1, \ldots, r_k]\!]_{2t}$ is uniformly random in the real world, then the distribution of honest shares of $[\![\boldsymbol{z}']\!]_{2t}$ generated by S is identical to the distribution in the real world. S honestly follows Step 2 and Step 3. At the end of Step 3, S receives the shares of $[\![\boldsymbol{z}']\!]_t$ held by honest parties, then S can compute the corrupted shares of $[\![\boldsymbol{z}']\!]_t$ based on these honest shares. In Step 4, S computes the corrupted shares of $[\![\boldsymbol{z}_0]\!]_t$ by simply following $[\![\boldsymbol{z}_0]\!]_t = [\![\boldsymbol{z}']\!]_t - [\![r_1, \ldots, r_k]\!]_t$ since S holds the corrupted shares of $[\![r_1, \ldots, r_k]\!]_t$ when emulating $\mathcal{F}_{\mathsf{rand}}$.
Finally, S holds the corrupted shares of $[\![\boldsymbol{z}_0]\!]_t$. And the whole sharing $[\![\boldsymbol{z}_0]\!]_t$ is fully determined by the secrets and corrupted shares of it, which are identical in both worlds. Hence, S perfectly simulates the behaviors of honest parties in the first part.

– For the second part, in Step 5, S receives $(r_i^1, r_i^2)_{i\in C}$ as the corrupted shares of $([\![\boldsymbol{r}]\!]_t, [\![\phi(\psi(\boldsymbol{r}))]\!]_t)$. In Step 6, S computes the corrupted shares of $[\![\boldsymbol{z}_0 + \boldsymbol{r}]\!]_t$ by adding the corrupted shares of $[\![\boldsymbol{z}_0]\!]_t$ and $[\![\boldsymbol{r}]\!]_t$, sends them to the adversary. In Step 7, S receives the honest shares of $[\![\boldsymbol{u}]\!]_t$, and then computes the shares of $[\![\boldsymbol{u}]\!]_t$ held by corrupted parties based on these honest shares. In Step 8, S computes the corrupted shares of $[\![\phi(\boldsymbol{x} * \boldsymbol{y})]\!]_t$ and sends them to $\mathcal{F}_{\mathsf{semi\text{-}mult}}$. Note that $[\![\phi(\boldsymbol{x} * \boldsymbol{y})]\!]_t$ is fully determined by the secrets and the corrupted shares of it, which are identical in both worlds. Hence, S perfectly simulates the behaviors of honest parties in the second part.

□

Theorem 14. *Protocol $\Pi_{\mathsf{semi\text{-}convert}}$ shown in Figure 5 securely realizes $\mathcal{F}_{\mathsf{semi\text{-}convert}}$ in the $\mathcal{F}_{\mathsf{rand}}(\Pi)$-hybrid model against a semi-honest adversary controlling $t-k+1$ parties.*

Functionality $\mathcal{F}_{\mathsf{semi\text{-}select}}$

This Functionality collects secrets from $\{[\![\phi(\boldsymbol{x}_i)]\!]_t\}_{i=1}^{k\cdot\ell}$.

1. $\mathcal{F}_{\mathsf{semi\text{-}select}}$ receives the honest shares of $\{[\![\phi(\boldsymbol{x}_i)]\!]_t\}_{i=1}^{k\cdot\ell}$.
2. $\mathcal{F}_{\mathsf{semi\text{-}select}}$ reconstructs the secrets $\phi(\boldsymbol{x})_i$, and computes $\boldsymbol{x}_i = \phi^{-1}(\phi(\boldsymbol{x}_i))$ for each $i \in [k \cdot \ell]$.
3. $\mathcal{F}_{\mathsf{semi\text{-}select}}$ further sets $\boldsymbol{y} = (y_1, \ldots, y_{k\cdot\ell})$ such that $y_i = (\boldsymbol{x}_i)_i$ for $i \in [k \cdot \ell]$.
4. $\mathcal{F}_{\mathsf{semi\text{-}select}}$ receives a set of shares $(s_j)_{j\in C}$ from the adversary. Based on k secrets from $\boldsymbol{y}$ and $t-k+1$ shares from the adversary, $\mathcal{F}_{\mathsf{semi\text{-}select}}$ reconstructs the whole sharing $[\![\phi(\boldsymbol{y})]\!]_t$.
5. $\mathcal{F}_{\mathsf{semi\text{-}select}}$ send shares of $[\![\phi(\boldsymbol{y})]\!]_t$ to honest parties.

Fig. 29: **Semi-select**

Proof. This proof can almost inherit the proof of the second part in Theorem 13. The only difference is that S emulates $\mathcal{F}_{\mathsf{rand}}(\Sigma)$ in Theorem 13, whereas S emulates $\mathcal{F}_{\mathsf{rand}}(\Pi)$ in this Theorem. □

Theorem 15. *Protocol $\Pi_{\mathsf{semi\text{-}select}}$ shown in Figure 6 securely realizes $\mathcal{F}_{\mathsf{semi\text{-}select}}$ in the $\mathcal{F}_{\mathsf{rand}}(f,d)$-hybrid model against a semi-honest adversary controlling $t-k+1$ parties.*

Proof. This proof can almost inherit the proof of the second part in Theorem 13. The only difference is that S emulates $\mathcal{F}_{\mathsf{rand}}(f,d)$ in Theorem 13, whereas S emulates $\mathcal{F}_{\mathsf{rand}}(f,d)$ in this Theorem. □