Unconditionally secure MPC for Boolean circuits with constant online communication

Zhenkai Hu
Shanghai Jiao Tong University
Shanghai, China
zhenkaihu@sjtu.edu.cn

Kang Yang
State Key Laboratory of Cryptology
Beijing, China
yangk@sklc.org

Yu Yu
Shanghai Jiao Tong University
Shanghai Qi Zhi Institute
Shanghai, China
yyuu@sjtu.edu.cn

***Abstract*—Through tremendous efforts, the communication cost of secure multi-party computation (MPC) in the honest-majority setting has been significantly improved. In particular, the state-of-the-art honest-majority MPC protocol by Escudero et al. (CCS'22) takes 12 field elements in total per multiplication gate for arithmetic circuits in the online phase. However, it still requires $12\log(5n/4)$ bits of online communication per AND gate for Boolean circuits. That is, for Boolean circuits, no MPC protocol with constant online communication is known.**

In this paper, we present an unconditionally secure MPC protocol for Boolean circuits in the honest-majority setting, which has constant online communication complexity and the offline communication complexity linear to the number n of parties. We first describe the semi-honest MPC protocol and then show how to extend it to achieve malicious security, where the maliciously secure protocol has the same communication cost as the semi-honest protocol. In particular, our protocol achieves the amortized communication cost 36 bits per AND gate in the online phase, $18n + 24$ bits per AND gate in the offline phase. For those circuit wires that require routing, each wire incurs a communication overhead of $15n$ bits in the offline phase.

***Index Terms*—Secure multi-party computation, honest majority, information-theoretic security**

I. Introduction

Secure multi-party computation (MPC) [1]–[4] allows a number of parties to jointly compute a function over their inputs while keeping their inputs private and only the output of the function is disclosed after the execution of the protocol. MPC integrates cryptography and distributed computing technology, which is an important research direction in the field of information security. It has a very wide application prospect, such as privacy preserving machine learning (PPML), privacy preserving data mining, etc.

We focus on honest majority setting, where the number of corrupted parties t, satisfies $t \leq (n-1)/2$, n is the total number of parties. Communication complexity serves as a critical metric in evaluating the performance of secure multiparty computation protocols. In this work, we present a constant honest majority multiparty computation protocol for Boolean circuits with information theoretic security in the preprocessing model that achieve the constant online communication complexity. Here, the term 'constant' signifies that the communication complexity per AND gate is independent of the number of parties involved. Several works [5]–[10] have achieved the overall communication complexity $O(|C| \cdot n)$ elements in the evaluation of arithmetic circuit C, where $|C|$ is the number of multiplication gates in the circuit. The protocol [9] achieves $4n$ elements per multiplication gate, which includes $2n$ elements in offline phase and $2n$ elements in online phase. And [11] achieves $1.5n$ elements per multiplication gate in online phase and $4n$ elements in the offline phase.

There are also several works achieve constant online communication complexity for arithmetic circuits. For instance, TURBOPACK [12] achieves 12 elements in online phase and $10n + 32$ elements in offline phase (includes $10n + 24$ elements in circuit-independent phase and 8 elements in circuit-dependent phase), which is roughly about $12\log(1.25n)$ bits in online phase and $(10n + 32)\log(1.25n)$ bits in offline phase per multiplication gate. Note for Shamir secret scheme, the field size must be larger than n, and for packed secret sharing scheme, the field size must be larger than $n + k$, where n is the number of parties, k is parameter of packed secret sharing scheme and $k \approx n/4$ in the honest majority setting. The study of [13] achieves $14n/k$ elements per multiplication gate and addition gate for dishonest majority over arithmetic circuits. The state-of-the-art dishonest majority protocol SUPERPACK [14] achieves $6/\epsilon$ elements per multiplication gate over arithmetic circuits, which is about $6 \cdot (\log(n + k))/\epsilon$ bits and ϵ is the percentage of honest parties among all parties.

Inspired by the works described above, we proposed a constant online communication MPC protocol for Boolean circuits. This protocol achieves amortized 36 bits per AND gate in the online phase and $18n + 24$ bits per AND gate in the offline phase, in both semi-honest and malicious settings. The potential application scenarios for our protocol are numerous, such as multi-party PPML, privacy-preserving database queries, and more. In PPML, when dealing with non-linear operations such as ReLU, max-pooling, etc., elements in the field are usually decomposed into bit representations, which leads to significant communication overhead. Our constant online MPC protocol, based on Boolean circuits, offers a potential solution for reducing the communication overhead associated with such operations. For instance, consider a ReLU operation based on packed secret sharing (PSS). This operation incurs a communication complexity of at least $O(l^2)$ in the online phase, where l denotes the size of an element. When taking $l = 63$, the complexity equates to an estimated 30,000 bits for a singular ReLU operation. In contrast, our protocol

slashes this number down to an estimated 2300 bits. This significant reduction stems from the Boolean circuit's ReLU operation, which strategically uses the most significant bit to perform an AND operation.

In some application scenarios, the primary burden is communication overhead, a large portion of which is independent of the input. Consequently, this part of the communication can be conducted during the offline phase to enhance the efficiency of the protocol during the online phase. These approaches of dividing the protocol into online and offline phases are also commonly used in dishonest majority multi-party computation protocols, such as [15]–[17], and all other subsequent works which based on these works. Thus we also divide our protocol into an offline phase and an online phase.

A. Our Contribution

In this work, we propose an honest majority MPC protocol that is the first constant online communication complexity protocol for Boolean circuits, has an online phase whose total communication complexity per AND gate is *constant* and independent of the number of parties. Previous related work [12]–[14], [18] was focused on arithmetic circuits not for Boolean circuits, and did not achieve constant online communication complexity in terms of bits. Our approach, which is based on [12], achieves the first constant bits online communication protocol for Boolean circuits. We also optimize the offline communication complexity of [12], reducing the communication overhead of the whole protocol. We split our protocol into two phases:

1) **Offline Phase:** Requires $18n+24$ bits per AND gate for both semi-honest security and malicious security in honest majority setting. For those circuit wires that require routing, each wire incurs a communication overhead of $15n$ bits in the offline phase.
2) **Online Phase:** Requires 36 bits per AND gate for both semi-honest security and malicious security in honest majority setting.

In the offline phase, parties generate two types of randomness: circuit-dependent randomness and circuit-independent randomness. These random values are unrelated to the circuit inputs. The generation of correlated randomness for each AND gate in the offline phase incurs a total communication complexity of $18n + 24$ bits.

In the online phase, the parties evaluate the AND gates and XOR gates layer by layer. The communication complexity for each AND gate in the online phase is 36 bits, while the communication for XOR gates is free. It is important to note that the communication complexity mentioned above is in the average sense. In the case of our online protocol with malicious adversaries, the parties also need to verify secrets and ensure the correctness of computation results during the online phase.

Recalling our protocol's achievement of both semi-honest and malicious security, our focus in this section will be primarily on the semi-honest protocol. Due to space constraints, the primary concept of our malicious security protocol is described in Section V, while the detailed protocols are presented in Appendix IX. To achieve a constant communication complexity in the online phase for Boolean circuits, we leverage two key techniques: packed secret sharing [19] and reverse multiplication friendly embeddings [20]:

– Packed secret sharing is a technique that enables the distribution and operation on multiple secrets simultaneously, while incurring the cost of a single secret. It allows us to efficiently handle multiple inputs within a single secret sharing scheme.
– Reverse multiplication friendly embeddings are employed to map pairs of vectors from the binary field to its extended field. This technique facilitates the mapping of multiple inputs of gates in the binary field to a single element in the extended field, reducing the overall complexity.

By utilizing reverse multiplication friendly embeddings and packed secret sharing, we can distribute and operate on multiple elements in the extended field simultaneously, further optimizing the communication complexity. A more detailed explanation of packed secret sharing and reverse multiplication friendly embeddings will be provided later in the paper.

The comparison of the communication cost per AND gate between our protocol and the state-of-the-art protocols appears in Table I. For protocol in arithmetic circuits, we denote the field size is $\log(n + k) \approx \log(1.25n)$ bits when the protocol uses packed secret sharing scheme, and the field size is $\log(n)$ bits when the protocol uses Shamir secret sharing scheme. When the number of parties $n = 50$, our communication complexity of offline phase per AND gate is from $1.89\times$ to $3.43\times$ better than [12], from $1.43\times$ to $2.59\times$ better than [21], depending on the number of gates requiring network routing. The communication complexity of online phase is $1.98\times$ better than [12], $4.16\times$ better than [21]. When the number of parties $n = 100$, our communication complexity of offline phase per AND gate is from $2.16\times$ to $3.94\times$ better than [12], from $1.44\times$ to $2.63\times$ better than [21], depending on the number of gates requiring network routing. The communication complexity of online phase is $8.3\times$ better than [21], $2.32\times$ better than [12].

B. Related Work

We focus on the honest majority information-theoretic MPC protocol in both semi-honest and malicious settings. The first protocol which achieves linear communication complexity is DN07 [7], commonly known as the DN protocol. Then there are many works have utilized the DN protocol to achieve security-with-abort [6], [8], [10], [11], [22] or guaranteed output delivery [11], [23]. Some works even achieved better performance, such as [11] achieves 5.5n elements per multiplication gate and [9] achieved 4n elements per multiplication gate.

The Order-C protocol, as outlined in [18], proposed a constant communication complexity for highly repetitive circuits, based on packed secret sharing scheme. However, it is not optimized for common arithmetic or Boolean circuits.

TABLE I
COMPARISON OF THE COMMUNICATION COST PER AND GATE BETWEEN OUR PROTOCOL AND THE STATE-OF-THE-ART PROTOCOLS.

Honest-majority MPC	Offline communication (bits)	Online communication (bits)	Corruption threshold	Security
[12]	$(10n+32)\log(5n/4)$	$12\log(5n/4)$	$(n-1)/2$	Information Theoretic and Active
[21]	$48n$	$3n$	$(n-1)/2$	Information Theoretic and Active
Our protocol	$18n+24$	36	$(n-1)/2$	Information Theoretic and Active

While both our approach and Order-C employ packed secret sharing schemes, the methodologies diverge significantly. In Order-C, packed secret sharing is directly applied to secret operations, necessitating a rearrangement step after each layer of circuit operations. In contrast, our approach obfuscates the secret with random values. This allows us to minimize communication overhead primarily when executing multiplication via packed secret sharing. Moreover, addition operations are executed locally by participating parties, and any necessary rearrangement is performed on the masked plaintext rather than on the shared secret. As a result, our scheme offers greater versatility and is adaptable to a wider range of circuit types compared to Order-C. Then [12] proposed a constant online communication complexity MPC over arithmetic circuit with the communication complexity of $12\log(1.25n)$ bits per multiplication gate.

The work in [20] revisited the amortized complexity of unconditional MPC and proposed the concept of Reverse Multiplication Friendly Embeddings (RMFE), which can embed several instances of the computation of a Boolean circuit into a single computation of an arithmetic circuit over an extension field. By using RMFE, the work achieves communication complexity of $O(n)$ bits per gate for SIMD circuits. Subsequently, [21] employed RMFE to propose a communication complexity of $O(n)$ bits per gate MPC protocol over binary fields.

II. TECHNICAL OVERVIEW

In this section, we aim to provide a high-level overview of our techniques, illustrating how we achieve a constant online communication complexity for Boolean circuits. Throughout this explanation, we will use bold letters to denote vectors, while n represents the number of parties and t represents the number of corrupted parties. In the setting of the honest majority, we have $n = 2t+1$. The packing parameter of packed secret sharing is represented by k, which specifies the number of secrets that can be stored within a single secret sharing scheme, and d represents the degree of packed secret sharing. By utilizing packed secret sharing with appropriate values of k and d, we are able to achieve efficient and secure operations on multiple secrets simultaneously, thereby reducing overall communication complexity.

Throughout the subsequent sections, we will delve into the details of our techniques in order to provide a comprehensive understanding of our approach.

A. Starting Idea: Constant Online Protocol for Boolean circuits Based on [12], [13], [24]

TURBOPACK [12] serves as a constant online multi-party computation protocol focused on arithmetic circuits, employing a method to aggregate multiple secrets into a single polynomial over a large field. This innovation motivated us to design a constant online protocol tailored for Boolean circuits. One key challenge is efficiently packing multiple bits into a single secret. While Packed Secret Sharing offers a straightforward method, its reliance on using a field element to represent each bit is both space-inefficient and non-constant in terms of communication complexity, at $O(\log(n))$ per AND gate.

Our solution leverages Reverse Multiplication Friendly Embeddings (RMFE) [20], perfectly suited for our needs. RMFE allows us to fold multiple instances of Boolean circuit computations into a singular arithmetic circuit operation over an extension field. To be more precise, RMFE enables the conversion of several bits into a single field element. This conversion process is repeated to form multiple field elements, which are then packed and shared via a packed secret sharing scheme.

However, the use of RMFE introduces a new challenge: each RMFE-mapped element necessitates an inverse transformation following each multiplication operation. Fortuitously, the multiplication process in TURBOPACK inherently requires masking the multiplication result and sending it to a designated leader for repackaging. This step conveniently allows us to perform the required RMFE inverse transformation without adding extra communication overhead. Having addressed these challenges, we successfully integrated the TURBOPACK to realize a constant online MPC protocol tailored for Boolean circuits.

Subsequently, we will delineate the particular architecture of our protocol. To accomplish this, we will employ two types of secret sharing schemes:

- The variant Shamir Secret Sharing scheme [25]: To represent this scheme, we will use $[x|s_i]_t$ to denote a degree-t Shamir secret sharing scheme that corresponds to a degree-t polynomial f such that $f(1), ..., f(n)$ are the shares and $f(s_i)$ as the secret.
- Packed Secret Sharing scheme [19]: For this scheme, we will use the notation $[\![\boldsymbol{x}]\!]_d$ to denote a degree-d Packed Secret Sharing scheme that corresponds to a degree-d polynomial f such that $f(1), ..., f(n)$ are the shares and $f(s_1), ..., f(s_k)$ are the secrets $x_1, ..., x_k$ where $\boldsymbol{x} = (x_1, ..., x_k)$ and $(s_1, ..., s_k)$ are the default positions to store the secrets $(x_1, ..., x_k)$.

Packed secret sharing, proposed by [19], is a generic approach for reducing the communication complexity which based on Shamir secret sharing. To clarify, we employ the variable k as the packing parameter, signifying that the Packed

Secret Sharing scheme can encapsulate k secrets within a single instance of secret sharing. For a packed secret sharing scheme of degree-d capable of accommodating k secrets, it is imperative to secure the scheme by ensuring that $d \geq k+t-1$. Furthermore, when performing multiplication operations between two Packed Secret Sharing instances, one with a of degree-d and another with d', attentiveness to degree constraints is vital. Since $d' \geq k-1$, and the combined degree $d+d'$ must not exceed $n-1$. This dictates that d should be less than or equal to $n-k$, culminating in the inequality $n-k \geq d \geq t+k-1$.

To increase the storage capacity for secrets, we set $n-k = d = t+k-1$, which allows us to store more secrets. By solving for k in terms of n and t, we derive $k = (n-t+1)/2$. This selection maximizes the packing parameter, thereby enhancing the efficiency of both storage and manipulation of multiple secrets within our Packed Secret Sharing scheme.

Reverse Multiplication Friendly Embeddings (RMFE), introduced by [20], are a pair of specialized mapping techniques that enable the transformation of a pair of vectors from a binary field to its corresponding extended field while preserving certain operational properties.

Review of the Reverse Multiplication-Friendly Embeddings [20]. Let $\mathbb{F}_2^l$ denote a vector space of $\mathbb{F}_2$ with dimension k, and let $\mathbb{F}_{2^m}$ denote the extension field of $\mathbb{F}_2$ with degree m. A reverse multiplication-friendly embeddings consists of a pair of $\mathbb{F}_2$-linear maps (ϕ, ψ), where $\phi : \mathbb{F}_2^l \to \mathbb{F}_{2^m}$ and $\psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^l$. For all vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^l$, these maps satisfy the condition:

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

Here, $*$ denotes the coordinate-wise product. The work [20] proved that a family of RMFEs exists such that $m = \Theta(k)$. Consider the superscript (2) to indicate a binary value, and let vectors $\boldsymbol{a} = (a_1^{(2)}, ..., a_l^{(2)}), \boldsymbol{b} = (b_1^{(2)}, ..., b_l^{(2)})$ satisfy $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^l$ and $a_i^{(2)}, b_i^{(2)} \in \mathbb{F}_2$ for all $i \in \{1, ..., l\}$. A pair of mappings (ϕ, ψ) forms RMFE that satisfies $\boldsymbol{a} * \boldsymbol{b} = \psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}))$, where $\phi(\boldsymbol{a}), \phi(\boldsymbol{b}) \in \mathbb{F}_{2^m}$. Moreover, [20] proved that for all $r < 33$, there exists a $(3r, 10r-5)_2$-RMFE. In this work, we set $m \approx 3l$.

We utilize the combination of packed secret sharing and reverse multiplication friendly embeddings to construct an MPC protocol with constant online communication complexity. In our approach, we adopt the superscript (2) to denote binary values.

To begin, we group each $l \cdot k$ values that need to be computed by the same type of circuit gate on a given layer, denoted as $\{v_{i,1}^{(2)}, ..., v_{i,l}^{(2)}\}_{i=1}^{k}$. Next, employing the RMFE mapping $\phi(\cdot)$, we convert a set of l binary values to an element v_i within its corresponding extended field $\mathbb{F}_{2^m}$. Specifically, for all $i \in \{1, ..., k\}$, we have $v_i = \phi(v_{i,1}^{(2)}, ..., v_{i,l}^{(2)}) \in \mathbb{F}_{2^m}$. Then we use a degree-$(n-k)$ packed secret sharing scheme to pack the values and then secretly share $\boldsymbol{v} = (v_1, ..., v_k)$, obtaining the sharing $[\![\boldsymbol{v}]\!]_{n-k}$.

We use $\mathcal{K}$ to denote the set $\mathcal{K} = \{1, ...k\}$ and use $\mathcal{L}$ to denote the set $\mathcal{L} = \{1, ..., l\}$. Let $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i \in \mathcal{K}, j \in \mathcal{L}}$, $\boldsymbol{\beta} = \{\beta_{i,j}\}_{i \in \mathcal{K}, j \in \mathcal{L}}$ denote the input wires of $k \cdot l$ XOR or AND gates for Boolean circuit. Let $v_{\alpha_{i,j}}^{(2)}$ be the value of the wire $\alpha_{i,j}$, $v_{\beta_{i,j}}^{(2)}$ be the value of the wire $\beta_{i,j}$ for all $i \in \mathcal{K}$, $j \in \mathcal{L}$. As described above, we can compute the packed sharing $[\![\boldsymbol{v_\alpha}]\!]_{n-k}, [\![\boldsymbol{v_\beta}]\!]_{n-k}$ such that $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$, $\boldsymbol{v_\beta} = (v_{\beta_1}, ..., v_{\beta_k})$ and $v_{\alpha_i} = \phi(v_{\alpha_{i,1}}^{(2)}, ..., v_{\alpha_{i,l}}^{(2)})$, $v_{\beta_i} = \phi(v_{\beta_{i,1}}^{(2)}, ..., v_{\beta_{i,l}}^{(2)})$ for all $i \in \mathcal{K}$. This enables us to perform batch computation of $k \cdot l$ gates at a time.

We have adopted the approach of TURBOPACK [12] as a basis for achieving constant online communication complexity. However, TURBOPACK was originally designed for arithmetic circuits and cannot be directly applied to Boolean circuits. Therefore, we have made significant modifications to TURBOPACK in order to achieve constant online communication complexity specifically for Boolean circuits. Furthermore, we have optimized the communication in the offline phase of TURBOPACK to enhance its efficiency. In the following sections, we will provide a detailed introduction to TURBOPACK and the modified version that we have developed.

Review of TURBOPACK [12]. In the offline phase of TURBOPACK, every wire α that is not the output of an addition gate is assigned to a uniformly random value λ_α. If a wire γ is the output of an addition gate with input wire α, β, λ_γ is defined as $\lambda_\alpha + \lambda_\beta$. Assume a group of multiplication (or addition) gates in a given circuit level with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$. Let $\boldsymbol{\lambda_\alpha}$, $\boldsymbol{\lambda_\beta}$, $\boldsymbol{\lambda_\gamma}$ represent the random values associated with the wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, respectively. The random sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$ and Beaver triple $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-1})$ are distributed to all parties in the offline phase. Define $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ and $\boldsymbol{\mu_\beta} = \boldsymbol{v_\beta} - \boldsymbol{\lambda_\beta}$, where $\boldsymbol{v_\alpha}, \boldsymbol{v_\beta}$ is the values associated with corresponding wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ respectively.

For input gates with the output wires $\boldsymbol{\alpha}$, parties will group and send the masks $\boldsymbol{\lambda_\alpha}$ of input wires to the client who owns the gates. Then client can compute and send $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ to P_1 where $\boldsymbol{v_\alpha}$ is the input values of client.

For a group of addition gates with the input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, note that P_1 holds $\boldsymbol{\mu_\alpha}$, $\boldsymbol{\mu_\beta}$. Thus P_1 can locally compute $\boldsymbol{\mu_\gamma} = \boldsymbol{\mu_\alpha} + \boldsymbol{\mu_\beta}$.

Recall that random sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$ and Beaver triple $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-1})$ have been distributed to all parties in the offline phase, where $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. For a group of multiplication gates with the input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$:

1. All parties locally compute $[\![\boldsymbol{\lambda_\alpha} - \boldsymbol{a}]\!]_{n-k} = [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} - [\![\boldsymbol{a}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\alpha} - \boldsymbol{b}]\!]_{n-k} = [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} - [\![\boldsymbol{b}]\!]_{n-k}$ and send $[\![\boldsymbol{\lambda_\alpha} - \boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta} - \boldsymbol{b}]\!]_{n-k}$ to P_1.
2. P_1 reconstructs $\boldsymbol{\lambda_\alpha} - \boldsymbol{a}, \boldsymbol{\lambda_\alpha} - \boldsymbol{b}$ and computes $\boldsymbol{v_\alpha} - \boldsymbol{a} = \boldsymbol{\mu_\alpha} + (\boldsymbol{\lambda_\alpha} - \boldsymbol{a}), \boldsymbol{v_\beta} - \boldsymbol{b} = \boldsymbol{\mu_\beta} + (\boldsymbol{\lambda_\beta} - \boldsymbol{b})$. Then P_1 distributes shares $[\![\boldsymbol{v_\alpha} - \boldsymbol{a}]\!]_{k-1}$ and $[\![\boldsymbol{v_\beta} - \boldsymbol{b}]\!]_{k-1}$ to the parties.
3. Using the $[\![\boldsymbol{v_\alpha} - \boldsymbol{a}]\!]_{k-1}, [\![\boldsymbol{v_\beta} - \boldsymbol{b}]\!]_{k-1}$ and

$([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-1})$, parties can locally compute:

$$\begin{aligned}[\![\boldsymbol{\mu_\gamma}]\!]_{n-1} = &[\![\boldsymbol{v_\alpha} - \boldsymbol{a}]\!]_{k-1} * [\![\boldsymbol{v_\beta} - \boldsymbol{b}]\!]_{k-1} \\ &+ [\![\boldsymbol{v_\alpha} - \boldsymbol{a}]\!]_{k-1} * [\![\boldsymbol{b}]\!]_{n-k} \\ &+ [\![\boldsymbol{v_\beta} - \boldsymbol{b}]\!]_{k-1} * [\![\boldsymbol{a}]\!]_{n-k} \\ &+ [\![\boldsymbol{c}]\!]_{n-1} - [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}.\end{aligned}$$

4. Parties send $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ to P_1 and P_1 reconstructs $\boldsymbol{\mu_\gamma}$.

There are two methods to compute multiplication gates in TURBOPACK. One is as described above, known as improved multiplication. The other method, termed original multiplication, requires all parties use a Beaver triple to compute $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_{n-1}$ in the offline phase. In the online phase, P_1 sends $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}, [\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to the parties. Subsequently, the parties can compute $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ locally, and the rest of the process is the same as above. In either method, parties need to generate $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$ and a packed Beaver triple in the offline phase. We noticed that we can generate a packed Beaver triple $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$ and set $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = [\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = [\![\boldsymbol{b}]\!]_{n-k}$ and $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} = [\![\boldsymbol{c}]\!]_{n-k}$ in the offline phase. And because the random value $\boldsymbol{a}$, $\boldsymbol{b}$ are bound to the wires, the values of $\boldsymbol{a}$ and $\boldsymbol{b}$ originate from the output wires of the circuit gates in the previous layer that are connected to this wire. This approach eliminates the communication of assigning random values to the input wires of each multiplication gate.

Note that TURBOPACK is designed for arithmetic circuits and does not directly apply to binary circuits. However, we have observed that the RMFE technique can convert a pair of vectors from a binary field to its corresponding extended field while maintaining specific operational properties. Therefore, we can utilize the RMFE transformation to convert a vector of inputs from a binary field to its extended field representation. Subsequently, we can effectively perform computations on these values using arithmetic circuits and achieve batch AND gate computations by performing multiplications over the extended field.

Network Routing. It should be noted, in TURBOPACK, each random value of wires is Shamir sharing, thus parties can locally compute packed secret sharing from the Shamir secret sharing according the circuit structure. But in our protocol, each random value of wires are bits, Shamir sharing pack l random values via RMFE, the group of output wires may connect different group of input wires of next circuit layer gates. Therefore, we need to perform network routing to ensure that the random values for each wire are correctly packaged into Shamir secret sharing by the RMFE, in accordance with the circuit structure.

The method for conducting network routing involves assigning a random bit value to each wire that requires routing, each party will hold an XOR sharing $\langle r \rangle^B$ of this random bit, where $\langle r \rangle^B$ denote every party holds $r_i \in \mathbb{F}_2$ and $r = \oplus_{i=1}^n r_i \in \mathbb{F}_2$. Parties can use these XOR shares to mask the random values of output wires. All parties can locally rearrange the XOR shares $\langle r \rangle^B$ according to the circuit structure. For the detailed implementation of the protocol, please refer to Protocol 1. The amortized communication complexity of preparing one random XOR sharing $\langle r \rangle^B$ is $n^2/(n-t) \approx 2n$ bits. The communication complexity of network routing is $(2n \cdot l + 2n \cdot m + l \cdot n + n \cdot m + n \cdot m)/l \approx 15n$ bits per gate that requiring routing.

Protocol 1: $\Pi_{routing}$

1. All parties invoke Π_{random} to prepare random XOR sharing $\langle r_{w_i} \rangle^B$, where $r_{w_i} \in \mathbb{F}_2$, for every wire w_i requiring routing.
2. Note all parties hold $[\phi(\boldsymbol{\lambda_w})|s]_t \in \mathbb{F}_{2^m}$ of the output wires for each group of wires $\boldsymbol{w} = (w_1, ..., w_l)$.
3. For each group of wires requiring routing, let $\boldsymbol{r_w} = (r_{w_1}, ..., r_{w_l})$, and $\langle \boldsymbol{r_w} \rangle = (\langle r_{w_1} \rangle^B, ..., \langle r_{w_l} \rangle^B)$.
5. All parties locally transform $[\phi(\boldsymbol{\lambda_w})|s]_t \in \mathbb{F}_{2^m}$ to additive sharing, denoted by $\langle \phi(\boldsymbol{\lambda_w}) \rangle$, and compute $\langle \boldsymbol{\lambda_w} \rangle = \psi(\langle \phi(\boldsymbol{\lambda_w}) \rangle \cdot \phi(\mathbf{1}))$. Note that every bit of additive secret sharing in $\mathbb{F}_{2^m}$ is also a XOR sharing.
6. We use $(\langle r_{w_1} \rangle, ..., \langle r_{w_N} \rangle)$ denote the wires $(w_1, ..., w_N)$ in current layer assigned random XOR sharing in Step 1. For each group of input wires of next circuit layer requiring routing, all parties locally rearrange XOR sharing $(\langle r_{w_1} \rangle, ..., \langle r_{w_N} \rangle)$ according to the next layer's circuit structure to get $(\langle r_{w'_1} \rangle, ..., \langle r_{w'_N} \rangle)$ where wire w_i will route to wire w'_j.
7. All parties locally compute $\langle \phi(\boldsymbol{r_{w'}}) \rangle = \phi(\langle r_{w'_1} \rangle^B, ..., \langle r_{w'_l} \rangle^B) \in \mathbb{F}_{2^m}$, and invoke Π_{random} to prepare random degree-t Shamir sharing $[u_{\boldsymbol{w'}}|s']_t \in \mathbb{F}_{2^m}$ for each group of l wires $\boldsymbol{w'} = (w'_1, ..., w'_l)$, transform $[u_{\boldsymbol{w'}}|s']_t$ to additive secret sharing $\langle u_{\boldsymbol{w'}} \rangle c$.
8. All parties locally compute all groups of $\langle \boldsymbol{\lambda_w} \rangle + \langle \boldsymbol{r_w} \rangle$, $\langle u_{w'} \rangle - \langle \phi(\boldsymbol{r_{w'}}) \rangle$ and send to P_1.
9. P_1 reconstructs all groups of $\boldsymbol{\lambda_w} + \boldsymbol{r_w}$, $u_{w'} - \phi(\boldsymbol{r_{w'}})$.
10. P_1 rearranges all groups of the $\boldsymbol{\lambda_w} \oplus \boldsymbol{r_w}$ according to the circuit structure to get new groups of $\boldsymbol{\lambda_{w'}} \oplus \boldsymbol{r_{w'}}$, computes $\phi(\boldsymbol{\lambda_{w'}}) + u_{w'} = \phi(\boldsymbol{\lambda_{w'}} \oplus \boldsymbol{r_{w'}}) + u_{w'} - \phi(\boldsymbol{r_{w'}})$, shares $[\phi(\boldsymbol{\lambda_{w'}}) + u_{w'}|s']_t$ to all other parties.
11. All parties locally compute $[\phi(\boldsymbol{\lambda_{w'}})|s']_t = [\phi(\boldsymbol{\lambda_{w'}}) + u_{w'}|s']_t - [u_{\boldsymbol{w'}}|s']_t$.

Fig. 1. Protocol for preparing network routing random values in offline phase.

Because the random value of the next layer of circuit wires is pre-generated and redistributed according to the circuit structure in the offline phase, it is only necessary for P_1 to locally rearrange the bit vectors according to the circuit structure in the online phase. P_1 needs to compute $\{\psi(\mu_{\gamma_i})\}_{i=1}^k$ to get the masked bit vectors, then P_1 rearranges the bits according to the next layer of circuits and use $\phi(\cdot)$ to repack the bit vectors

It should be noted that the properties of RMFE are not preserved in all cases. RMFE can only guarantee $\boldsymbol{a} * \boldsymbol{b} = \psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}))$ where $\boldsymbol{a}, \boldsymbol{b}$ are two binary vectors and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^l$. This does not imply any two elements in $\mathbb{F}_{2^m}$ will still hold the properties. For instance, it may not be possible that $\boldsymbol{a} * \boldsymbol{b} * \boldsymbol{c} = \psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}) \cdot \phi(\boldsymbol{c}))$. Therefore, when using RMFE for multiplication operations, such as computing $c' = \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})$, it is necessary to re-encode c' and compute $c = \phi(\psi(c'))$ after each multiplication operation on $\mathbb{F}_{2^m}$.

Summary of our protocol for each group of AND gates:

Recall that we use $\mathcal{K}$ to denote the set $\mathcal{K} = \{1, ...k\}$ and use $\mathcal{L}$ to denote the set $\mathcal{L} = \{1, ..., l\}$. For each group of AND gates with input wires $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$, $\boldsymbol{\beta} = \{\beta_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ and output wires $\boldsymbol{\gamma}$, let $v^{(2)}_{\alpha_{i,j}}$ be the value of the wire $\alpha_{i,j}$ and $v^{(2)}_{\beta_{i,j}}$ be the value of the wire $\beta_{i,j}$ for all $i \in \mathcal{K}$, $j \in \mathcal{L}$. We can compute the packed sharings $[\![\boldsymbol{v_\alpha}]\!]_{n-k}, [\![\boldsymbol{v_\beta}]\!]_{n-k}$ where $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$ and $\boldsymbol{v_\beta} = (v_{\beta_1}, ..., v_{\beta_k})$. Here, $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}})$, $v_{\beta_i} = \phi(v^{(2)}_{\beta_{i,1}}, ..., v^{(2)}_{\beta_{i,l}})$ for all $i \in \mathcal{K}$. Then we can generate the random sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ and compute $[\![\boldsymbol{\mu_\alpha}]\!]_{n-k} = [\![\boldsymbol{v_\alpha}]\!]_{n-k} - [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\mu_\beta}]\!]_{n-k} = [\![\boldsymbol{v_\beta}]\!]_{n-k} - [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$.

For each group of AND gates:

1. Note that P_1 holds $\boldsymbol{\mu_\alpha}, \boldsymbol{\mu_\beta}$. P_1 computes and sends $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}, [\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to the parties.
2. Parties receive $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}, [\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ from P_1. Note that the parties hold $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$, thus they can locally compute:
$$[\![\boldsymbol{\mu_\gamma}]\!]_{n-1} = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} + [\![\boldsymbol{\mu_\beta}]\!]_{k-1} * [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} - [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$$
3. Parties send $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ to P_1. P_1 reconstructs $\boldsymbol{\mu_\gamma} = (\mu_{\gamma_1}, ..., \mu_{\gamma_k}) \in \mathbb{F}^k_{2^m}$
4. P_1 computes $\{\psi(\mu_{\gamma_i})\}_{i=1}^k$ and rearranges the bits according to the circuit's next layer to get $\boldsymbol{\mu'^{(2)}_{\gamma_i}} = (\mu'^{(2)}_{\gamma_{i,1}}, ..., \mu'^{(2)}_{\gamma_{i,l}})$ for $i \in \mathcal{K}$. Then P_1 computes $\mu_{\gamma_i} = \phi(\boldsymbol{\mu'^{(2)}_{\gamma_i}})$ for $i \in \mathcal{K}$.

Correctness: $\boldsymbol{\mu_\gamma} = \boldsymbol{v_\alpha} * \boldsymbol{v_\beta} - \boldsymbol{\lambda_\gamma} = (\boldsymbol{\mu_\alpha} + \boldsymbol{\lambda_\alpha}) * (\boldsymbol{\mu_\beta} + \boldsymbol{\lambda_\beta}) - \boldsymbol{\lambda_\gamma} = \boldsymbol{\mu_\alpha} * \boldsymbol{\mu_\beta} + \boldsymbol{\mu_\alpha} * \boldsymbol{\lambda_\beta} + \boldsymbol{\mu_\beta} * \boldsymbol{\lambda_\alpha} + \boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}$.

For each group of AND gates, P_1 needs to distribute two degree-$(k-1)$ packed Shamir sharings $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}, [\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to all parties, and all parties need to compute $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ and send it to P_1. As a result, the total communication complexity is $3n$ field elements per group of $k \cdot l$ AND gates. The amortized communication complexity per AND gate is $3 \cdot n \cdot m/(k \cdot l) \approx 36$ bits (note m is roughly equal to $3 \cdot l$ [20], and k is roughly equal to $n/4$). .

B. Instantiating the Offline Phase

In the offline phase, our goal is to prepare packed Beaver triples $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$ for each group of AND gates and assign random values $\boldsymbol{\lambda_\alpha}$ for each group of wires $\boldsymbol{\alpha}$:

- For each group of AND gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$, all parties prepare a packed Beaver triple $([\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{c}]\!]_{n-k})$ and let $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = [\![\boldsymbol{a}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = [\![\boldsymbol{b}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} = [\![\boldsymbol{c}]\!]_{n-k}$.
- For each group of AND gates with the output wires $\boldsymbol{\gamma}$, all parties prepare $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$.
- For each group of XOR gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties prepare $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ and let $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-k} = [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$.
- For each group of input gates with the output wires $\boldsymbol{\alpha}$ or output gates with the input wires $\boldsymbol{\alpha}$, all parties prepare $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$.

There are potential solutions from [12], [13], [24] that could prepare random packed Shamir sharings. The approach in [24] is not in the IT setting and need constant corrupted parties and the approach in [13] requires interaction for addition gates. The approach in [12] can potentially generate the random packed Shamir sharings, but the protocol in [12] is for arithmetic circuit, and the random sharings generated by [12] may not satisfy the RMFE properties. Our approach first prepares random Shamir sharings over $\mathbb{F}_2^l$ and then use ϕ to map to $\mathbb{F}_{2^m}$.

Preparing Random Sharings. We make use of the following functionality $\mathcal{F}_{random}$ from [21] to let all parties prepare random sharings in the form of $[\phi(\boldsymbol{r})]_t$. Recall that (ϕ, ψ) is an $(l, m)_2$-RMFE. Here each $[\phi(\boldsymbol{r})]_t$ is a degree-t Shamir sharing of the secret $\phi(\boldsymbol{r})$ where $\boldsymbol{r}$ is a random vector in $\mathbb{F}_2^l$. Let Σ be a linear secret sharing scheme in $\mathbb{F}$. The description of the functionality $\mathcal{F}_{random}$ appears in Functionality 1. At a high level, this approach is similar to [7]:

1. Each party P_i generates and distributes random Σ sharings, denoted by $\boldsymbol{S}^{(i)}$.
2. Let $\boldsymbol{M}^T$ be a Vandermonde matrix of size $n \times (t+1)$ in $\mathbb{F}$. All parties use $\boldsymbol{M}$ as a random extractor to extract $n - t = t + 1$ random sharings. Parties compute $(\boldsymbol{R}^{(1)}, ..., \boldsymbol{R}^{(t+1)})^T = \boldsymbol{M}(\boldsymbol{S}^{(1)}, ..., \boldsymbol{S}^{(n)})^T$.

Because each $\boldsymbol{R}^{(i)}$ is a linear combination of $\{\boldsymbol{S}^{(j)}\}_{j=1}^n$ and any sub-matrix of size $(t+1) \times (t+1)$ of $n \times (t+1)$ Vandermonde matrix is invertible. Thus, for each random sharing $\boldsymbol{R}^{(i)}$ there is a one-to-one map between the random sharings prepared by honest parties and the output sharings. Therefore, the output sharings are still random.

It is important to emphasize here that the random sharings we use must take the form of $[\phi(\boldsymbol{r})]_t$ and cannot be generated directly on the field $\mathbb{F}_{2^m}$. This is because a random element over $\mathbb{F}_{2^m}$ may not lie within the domain of the mapping $\psi(\cdot)$. Mapping $\psi(\cdot)$ only guarantees $\psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})) = \boldsymbol{a} * \boldsymbol{b}$, meaning that only an element c resulting from $c = \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})$ is guaranteed to be in the domain of $\psi(\cdot)$.

Preparing Packed Beaver Triples. Let $\boldsymbol{e_i}$ denote the unit vector where the length is k and i-th entry is 1, and all the other entries are 0. Note that parties can convert $\boldsymbol{e_i}$ to a degree-$(k-1)$ Packed Secret Sharings locally. Then, the parties can compute degree-$(n-k)$ packed secret sharing by degree-t Shamir secret sharing. For Beaver triple $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$, the first two sharings come from the random values of wires $\{[\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t\}_{i=1}^k$in the previous layer of the circuit that have been correctly routed. Then:

- To compute $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$, we need to calculate $[c_i|i]_t$ where $c_i = \lambda_{\alpha_i} \cdot \lambda_{\beta_i}$. Note that there is no need to compute $\phi(\psi(\lambda_{\alpha_i} \cdot \lambda_{\beta_i}))$, because in the last step of evaluation of AND gate, P_1 will always re-encode $\boldsymbol{\mu_\gamma}$. We rely on the state-of-the-art multiplication protocol [9] in the standard honest majority setting to compute $[c_i|i]_t$, which costs $4n$ elements, the functionality appears in $\mathcal{F}_{single\text{-}mult}$. After obtained $\{[c_i|i]_t\}_{i=1}^k$ from [9], parties

can compute $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!]_{k-1} * [c_i|i]_t$ locally.

The communication complexity for preparing $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$ involves $4n$ elements over $\mathbb{F}_{2^m}$ per l AND gates and $12n$ bits per AND gate. if $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ correspond to wires that require routing, then an additional communication cost of $30n$ bits per triple is necessary.

With the above approach, we can instantiate the full offline phase, for each layer, first invoke $\Pi_{routing}$ to perform network routing to get $[\lambda_\alpha|s]_t, [\lambda_\beta|s]_t$:

- For each group of $k \cdot l$ XOR gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties locally compute $[\lambda_{\gamma_i}|] = [\lambda_{\alpha_i}|i]_t + [\lambda_{\beta_i}|i]_t$ and associate $\{[\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [\lambda_{\gamma_i}|]\}_{i=1}^{k}$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$,
- For each group of $k \cdot l$ AND gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, parties use the above approach to get triple $\{([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t)\}_{i=1}^{k}$ and associate $\{[\lambda_{\alpha_i}|i]_t\}_{i=1}^{k}$, $\{[\lambda_{\beta_i}|i]_t\}_{i=1}^{k}$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Parties use the above approach to generate random values $\{[\lambda_{\gamma_i}|i]_t\}_{i=1}^{k}$ and associate these random values with the wires $\boldsymbol{\gamma}$, which involves $2n$ elements over $\mathbb{F}_{2^m}$ per l AND gates.

In addition to generating random sharings as described above, we also need to prepare three sets of random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}_{2^m}^k$. These sets correspond to the each group of output wires of the AND gates, the input wires of the output gates, and the output wires of the input gates. These random sharings of $\mathbf{0}$ serve to re-randomize the sharings and protect the shares of honest parties. Utilizing the above approach for preparation, the communication complexity of generating random zero sharing for per AND gate is $2 \cdot n \cdot m/(k \cdot l) \approx 24$ bits.

In summary, our offline phase has communication complexity of $6n + 12n + 24 = 18n + 24$ bits per AND gate, and an additional communication cost of $30n$ bits if correspond wires that require routing.

C. Achieving Malicious Security

In this section, we will discuss how to attain malicious security without compromising the practical efficiency of the protocol. For degree-t Shamir secret sharing in honest majority setting, the whole sharing is fully determined by the shares of honest parties. But for the degree-$(n-k)$ packed secret sharing, malicious parties can modify the secret without attracting the attention of honest parties, and this makes the verification protocols in the recent IT MPC protocols with honest majority setting [5], [9], [26] not work. We will explain in more detail how malicious parties achieve this process in Section V.

The approach proposed in [12] can achieve malicious security in a degree-$(n-k)$ packed secret sharing scheme. However, the Beaver triples and random values generated using this approach are not guaranteed to satisfy the properties of RFFEs, making it unsuitable for direct use. On the other hand, the approach presented in [21] can generate shares of random values and Beaver triples that satisfy the RMFE properties while ensuring malicious security. However, this approach is based on a degree-t Shamir secret sharing scheme.

To address this, we combine the ideas from both [12] and [21]. We utilize the approach of [21] to generate the random values and Beaver triples with malicious security in a degree-t Shamir secret sharing scheme, ensuring that they satisfy the RMFE properties. We then locally convert these shares from the degree-t Shamir secret sharing to a degree-$(n-k)$ packed secret sharing. Subsequently, we can leverage the approach proposed in [12], which is based on [22], to achieve malicious security in the online phase. Due to space limitations, we only show the main body of our malicious protocols in Section V, the security proof and details of our malicious security protocol can be found in the Appendix IX.

III. PRELIMINARIES

A. The Model and Notation

In this work, we use the $client\text{-}server$ model. In this model, clients only provide inputs to the functionality and receive outputs, and servers can participate in the computation but do not have inputs or get outputs. Each party may have a different role in the computation, perhaps as both a client and a server. We only focus on functions which can be represent as a Boolean circuit with input, XOR, AND, and output gates. We use C to denote the circuit, $|C|$ to denote the size of the circuit.

For clarity, we will use superscript (2) to denote a binary value. If there is no superscript, the value defaults to $\mathbb{F}_{2^m}$. For example, $v^{(2)}$ represents a binary value and v represents a value on $\mathbb{F}_{2^m}$.

We use $[\![\boldsymbol{x}]\!]_d$ to represent a degree-d packed secret sharing of $\boldsymbol{x}$, where $\boldsymbol{x}$ is a vector of $(x_1, ..., x_k) \in \mathbb{F}_{2^m}^k$. We use $[x]_d$ to denote a standard degree-d Shamir Secret Sharing of x. We use n to denote the number of parties, use $\{P_1, ..., P_n\}$ to denote the set of parties. We use $[x|s_i]_d$ to denote a degree-d Shamir sharing of x such that the secret x is stored at the evaluation point s_i, which represents the secret sharing polynomial $f(s_i) = x$. We use $\mathcal{A}$ to denote the Adversary and t to denote the size of corrupted parties by $\mathcal{A}$. We use $\mathcal{F}$ to denote the secure function evaluation functionality, $\mathcal{C}$ to denote the set of all corrupted parties and $\mathcal{H}$ to denote the set of all honest parties. We will use m, l to denote the parameters of $(l, m)_2$-RMFE, use $\mathcal{L}$ to denote the set $\{1, ..., l\}$, use $\mathcal{K}$ to denote the set $\{1, ..., k\}$ where k is the packing parameter of the packed secret sharing scheme.

We use κ to denote the security parameter and we will use an extension field of $\mathbb{F}_2$ denoted by $\mathbb{F}_{2^m}$ (of size 2^m). We always assume that $|\mathbb{F}_{2^m}| = 2^m \geq 2^\kappa$

B. Security Definition

We construct an ideal model that satisfies our security requirements. If our real model is no different from our constructed ideal model in a certain sense, it proves that the real protocol we designed has the same security as the ideal

model. Our security definition and model are same as [8], [21], reader can find more details from [8], [21]. $\mathcal{A}$ can corrupt at most c clients and t servers. $\mathcal{A}$ can receive all messages sent to corrupted parties.

- If $\mathcal{A}$ is semi-honest, then corrupted parties honestly follow the protocol.
- If $\mathcal{A}$ is fully malicious, then corrupted parties can deviate from the protocol arbitrarily, and $\mathcal{A}$ can provide inputs to corrupted clients.

Due to space limitations, we will provide a more detailed description of our security model and security definitions in Appendix VII.

C. Secret Sharing Scheme

1) Shamir Secret Sharing. : Shamir Secret Sharing [25] is a linear secret sharing scheme and Shamir secret shares are homomorphic in a special way. Let n be the number of parties and $\mathbb{G}$ be a finite field of size $|\mathbb{G}| \geq n+1$. Let $\alpha_1, ..., \alpha_n$ be n distinct non-zero elements in $\mathbb{G}$. A degree-d Shamir Secret Sharing of $x \in \mathbb{G}$ is a vector $(y_1, ..., y_n)$ which satisfies that there exists a polynomial $f(\cdot) \in \mathbb{G}[X]$ of degree at most d such that $f(0) = x$ and $f(\alpha_i) = y_i$ for $i \in \{1, ..., n\}$. Each party P_i holds a share y_i and the whole sharing is denoted by $[x]_d$.

2) Packed Secret Sharing Scheme. : Franklin and Yung proposed a vectorized version of Shamir Secret Sharing called Packed Secret Sharing (PSS) [19] which can implement batch computing based on Shamir Secret Sharing. Let n be the number of parties and $\mathbb{G}$ be a finite field of size $|\mathbb{G}| \geq n+1$. Let $\alpha_1, ..., \alpha_n$ be n distinct non-zero elements in $\mathbb{G}$. Let $\boldsymbol{x} = (x_1, ..., x_k)$,$(s_1, ..., s_k)$ are two vectors which satisfy $x_i \in \mathbb{G}, e_i \in \mathbb{G}$ for $i \in \{1, ..., k\}$. A degree-d Packed Secret Sharing of $\boldsymbol{x} = (x_1, ..., x_k)$ is a vector $(y_1, ..., y_n)$ which satisfies that there exists a polynomial $f(\cdot) \in \mathbb{G}[X]$ of degree at most d such that $f(s_i) = x_i$ and $f(\alpha_i) = y_i$ for $i \in \{1, ..., n\}$. Each party P_i holds a share y_i and the whole sharing is denoted by $[\![\boldsymbol{x}]\!]_d$.

D. Reverse Multiplication Friendly Embeddings

Reverse Multiplication Friendly Embeddings can embed several instances of the computation of a binary circuit into a single computation of an arithmetic circuit over an extension field, as proposed by [20].

Definition III.1 ([20])**.** Let q be a power of a prime and $\mathbb{F}_{q}$ a field of q elements, let $l, m \geq 1$ be integers. A pair (ϕ, ψ) is called an $(l, m)_q$-reverse multiplication friendly embeddings (RMFE for short) if $\phi : \mathbb{F}_q^l \to \mathbb{F}_{q^m}$ and $\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^l$ are two $\mathbb{F}_q$-linear maps satisfying

$$\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})) \tag{1}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^l$, where $*$ denotes coordinate-wise product.

Lemma III.1 ([21])**.** *Let l, m be integers and $\mathbb{F}_q$ be a finite field. Suppose (ϕ, ψ) is an $(l, m)_q$-reverse multiplication friendly embedding. Then there exists an $\mathbb{F}_q$-linear map $\hat{\phi}^{-1}$: $\mathbb{F}_{q^m} \to \mathbb{F}_q^l$ such that for all $\boldsymbol{x} \in \mathbb{F}_q^l$,*

$$\hat{\phi}^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}. \tag{2}$$

Functionality 1: $\mathcal{F}_{random}$

1. $\mathcal{F}_{random}$ receives the set of corrupted parties, denoted by $Corr$. And $\mathcal{F}_{random}$ receives from the adversary a set of shares $\{u_i\}_{\{i \in Corr\}}$. Then $\mathcal{F}_{random}$ randomly samples $\boldsymbol{r} \in \mathbb{F}_2^k$ and generates a degree-t Shamir sharing $[\phi(\boldsymbol{r})]_t$ such that the share of $P_i \in Corr$ is u_i.
2. $\mathcal{F}_{random}$ distributes the shares of $[\phi(\boldsymbol{r})]_t$ to honest parties.

Fig. 2. **Functionality for generating random sharings**

E. Building Blocks

In this part, we will introduce some functionalities that will be used in our main construction.

- Functionality $\mathcal{F}_{random}$ allows all parties to generate a random element which is a degree-t Shamir secret sharing and satisfies RMFE properties (for instance, parties generate $[r]_t$ such that $r \in \mathbb{F}_{2^m}$ and $\psi(r \cdot \phi(\mathbf{1})) \in \mathbb{F}_2^l$).

The instantiation of $\mathcal{F}_{random}$ for the malicious security version is described in [21].

Let $\langle b \rangle^B$ represent random XOR bit sharing, where the superscript B indicates $b \in \{0, 1\}$. Let $\langle b \rangle_i^B$ present the sharing of $\langle b \rangle^B$ that P_i holds, with $b = \oplus_{i=1}^n \langle b \rangle_i^B$. Let $\langle r \rangle$ represent a vector of random XOR sharing bits, where $r = \oplus_{i=1}^n \langle r \rangle_i \in \mathbb{F}_2^l$, and P_i holds the sharing $\langle r \rangle_i$. Note that each bit of $\langle r \rangle$ also forms a $\langle \cdot \rangle^B$ sharing.

We will use Π_{random} to generate random bit sharing in the form of $\langle b \rangle^B$. Let Σ be the secret sharing scheme corresponding to $\langle r \rangle$ sharing, where each party P_i shares $\langle r^{(i)} \rangle \in \mathbb{F}_2^l$ with other parties. Let the Vandermonde matrix M of Π_{random} be over $\mathbb{F}_{2^l}$, and $\boldsymbol{S}^{(i)}$ be $\langle r^{(i)} \rangle$ for all $i \in \{1, ...n\}$. Then all parties will get $t+1$ random XOR sharing vectors, denoted by $\langle r_i \rangle$, and the j-th bit of $\langle r_i \rangle$ forms a random XOR bit sharing $\langle r_{i,j} \rangle^B$. Using this method, the communication complexity to generate $\langle b \rangle^B$ is $n \cdot n \cdot l/(l \cdot (t+1)) \approx 2n$ bits.

IV. Efficient MPC for Boolean Circuits with Semi-honest Security via RMFE and PSS

Recall that we use c to denote the number of clients and n to denote the number of parties. Recall the corruption threshold $t = (n-1)/2$ and the packing parameter $k = (n - t + 1)/2 = (n+3)/4$. We will use $(\phi, \psi), \phi : \mathbb{F}_q^l \to \mathbb{F}_{q^m}$ and $\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^l$ to denote an $(l, m)_2$-reverse multiplication friendly embedding (RMFE), and $m \approx 3l$. For $\boldsymbol{r} = (r_1, ..., r_k) \in \mathbb{F}_{2^m}^k$, we will simplify the notation to use $\psi(\boldsymbol{r})$ to represent $\psi(r_i)$ for all $i \in \mathcal{K}$.

Functionality 2: $\mathcal{F}_{offline}$

1. **Prepare Random Zero Shaings:** For each group of AND gates, input gates, output gates:
   a. $\mathcal{F}_{offline}$ receives from the adversary a set of shares $\{u_j\}_{j\in Corr}$. $\mathcal{F}_{offline}$ sets $\mathbf{0} \in \mathbb{F}_{2^m}^k$, samples a random degree-$(n-1)$ packed Shamir sharings $[\![\mathbf{0}]\!]_{n-1}$ such that for all $P_j \in Corr$, the j-th share of $[\![\mathbf{0}]\!]_{n-1}$ is u_j.
   b. $\mathcal{F}_{offline}$ distributes the shares of $[\![\mathbf{0}]\!]_{n-1}$ to honest parties.
2. **Assigning Random Values and Distribute Shares for each Wires:** $\mathcal{F}_{offline}$ receives the circuit from all parties.
   a. For each group of $k \cdot l$ output wires $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ of input gates and AND gates, $\mathcal{F}_{offline}$ samples $k \cdot l$ uniform values $\{r_{i,j}^{(2)}\}_{i\in\mathcal{K}}^{j\in\mathcal{L}}$ and sets $\lambda_{\alpha_i} = \phi(\boldsymbol{r_i})$, associates these random values with the wires $\boldsymbol{\alpha}$. The same step is done for the input wires of each group of output gates.
   b. Starting from the first layer of C to the last layer, for XOR gates, the random values assigned to the input wires $\alpha_{i,j}$, $\beta_{i,j}$, denoted as $\lambda_{\alpha_{i,j}}^{(2)}$ and $\lambda_{\beta_{i,j}}^{(2)}$, are inherited from the output wires of gates in the preceding layer of the circuit. Each input wire of an XOR gate is effectively linked to an output wire of a gate from the previous layer. As such, the random value λ associated with an output wire of a gate in the prior layer, which is subsequently connected to an XOR gate, is designated as the random value $\lambda_{\alpha_{i,j}}^{(2)}$ or $\lambda_{\beta_{i,j}}^{(2)}$ for the corresponding input wire of the XOR gate. Furthermore, $\mathcal{F}_{offline}$ sets the random value for the output wire of the XOR gate $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$.
   c. Starting from the first layer of C to the last layer, for AND gates, the random values of input wires are inherited from the output wires of gates in the preceding layer as Step 3.b.
3. **Preparing Packed Shamir Sharings:** $\mathcal{F}_{offline}$ receives the set of corrupted parties, denoted by $Corr$. For each intermediate layer in C:
   a. For each group of $l \cdot k$ output wires$\{\alpha_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ of input gates and AND gates, $\mathcal{F}_{offline}$ receives from the adversary a set of shares $\{u_j\}_{j\in Corr}$. $\mathcal{F}_{offline}$ samples a random degree-$(n-1)$ packed Shamir sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ such that for all $P_j \in Corr$, the j-th share of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ is u_j. $\mathcal{F}_{offline}$ distributes the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ to honest parties. The same step is done for the input wires of each group of output gates.
   b. For each group of AND gates with input wires $\{(\alpha_{i,j},\beta_{i,j})\}_{i\in\mathcal{K},j\in\mathcal{L}}$ and output wires $\{\gamma_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$, $\mathcal{F}_{offline}$ receives from the adversary a set of shares $\{u_{1,j},u_{2,j}\}_{j\in Corr}$. $\mathcal{F}_{offline}$ computes degree-$(n-k)$ packed Shamir sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ such that for all $P_j \in Corr$, the j-th share of $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k})$ is $(u_{1,j},u_{2,j})$. $\mathcal{F}_{offline}$ distributes the shares of $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k})$ to honest parties.
   c. For each group of AND gates with input wires $\{(\alpha_{i,j},\beta_{i,j})\}_{i\in\{1,\ldots,k\},j\in\{1,\ldots,l\}}$ and output wires $\{\gamma_{i,j}\}_{i\in\{1,\ldots,k\},j\in\{1,\ldots,l\}}$, $\mathcal{F}_{offline}$ receives from the adversary a set of shares $\{u_{1,j},u_{2,j},u_{3,j}\}_{j\in Corr}$. $\mathcal{F}_{offline}$ computes degree-$(n-k)$ packed Shamir sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$ such that for all $P_j \in Corr$, the j-th share of $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k})$, $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$ is $(u_{1,j},u_{2,j},u_{3,j})$. $\mathcal{F}_{offline}$ distributes the shares of $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$ to honest parties.

Fig. 3. **Functionality for offline phase**

A. Ideal Functionality for Offline Phase

We present the ideal functionality $\mathcal{F}_{offline}$, as given in Functionality 2, which prepares correlated randomness for the online phase and associates random values with wires. We will use superscript (2) to denote a binary value, use $\mathcal{K}$ to denote the set $\{1,...,k\}$, use $\mathcal{L}$ to denote the set $\{1,...,l\}$.

B. Online Protocol

In the online phase, P_1 will learn $\mu_\alpha = v_\alpha - \lambda_\alpha$, where $v_\alpha = \phi(v_{\alpha_1}^{(2)},...,v_{\alpha_l}^{(2)}) \in \mathbb{F}_{2^m}$ and $v_{\alpha_1}^{(2)},...,v_{\alpha_l}^{(2)}$ are the real values associated with the wire $\alpha_1,...,\alpha_l$. At the end of the online protocol, for each group of gates that belong to some clients, all parties will send their shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ to clients, where $\boldsymbol{\alpha}$ are the input wires associated with these output gates. P_1 will send $\boldsymbol{\mu_\alpha}$ to clients, allowing them to reconstruct the output $\boldsymbol{v_\alpha} = \boldsymbol{\mu_\alpha} + \boldsymbol{\lambda_\alpha}$. Assumed $\boldsymbol{v_\alpha} = (v_{\alpha_1},...,v_{\alpha_k}) \in \mathbb{F}_{2^m}^l$, $\mathbf{1} = (1,1,...,1) \in \mathbb{F}_2^l$, clients can compute $(v_{\alpha_{j,1}}^{(2)},...,v_{\alpha_{j,l}}^{(2)}) = \psi(v_{\alpha_j} \cdot \phi(\mathbf{1}))$ for $j \in \mathcal{K}$. Note that we denote a group of $l \cdot k$ wires by $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$.

Input Phase. Recall that in the offline phase, for each group of input gates that belong to some clients, $\mathcal{F}_{offline}$ distributes a degree-$(n-1)$ packed Shamir sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ to all parties, where $\boldsymbol{\alpha}$ refers to the output wires associated with these input gates. To allow P_1 to learn $\boldsymbol{\mu_\alpha}$, clients collect the whole sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ from all parties, then they reconstruct the secret $\boldsymbol{\lambda_\alpha}$, and finally, they compute and send $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ to P_1, where $\boldsymbol{v_\alpha} = (v_{\alpha_1},...,v_{\alpha_k})$ and $v_{\alpha_i} = \phi(v_{\alpha_{i,1}}^{(2)},...,v_{\alpha_{i,l}}^{(2)})$ for all $i \in \mathcal{K}$, $\{v_{\alpha_{i,j}}^{(2)}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ are the inputs of the client. The description of the protocol Π_{input} appears in Protocol 1. The communication complexity per batch of $l \cdot k$ input gates is $(n+k) \cdot m$ bits, and the communication complexity per input gate is approximately $(n+k) \cdot m/(k \cdot l) \approx 15$ bits.

Computation Phase. Now we will introduce how P_1 can learn μ_α for every wire α in the circuit C. This follows the idea in [12], [13], [24] with the change that we use the technique of RMFEs introduced in [21] for converting AND gate to multiplication gate.

Note that P_1 has already learned μ_α for the first layer (the input layer) and the circuit is evaluated layer by layer. Assuming that P_1 learns μ_α in previous layers for every group of input wires $\boldsymbol{\alpha}$ of the current layer since $\boldsymbol{\alpha}$ serves as an output wires in previous layers.

For a group of l XOR gates with input wires $\{\alpha_i,\beta_i\}_{i=1}^l$ and output wire $\{\gamma_i\}_{i=1}^l$, the goal is to compute $\mu_\gamma =$

Protocol 2: Π_{input}

1. For each group of input gates that belong to Client, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates. All parties receive the sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ from $\mathcal{F}_{offline}$ and clients hold inputs $\{v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}}\}_{i=1}^{k}$.
2. All parties send to Client their shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$.
3. Client reconstructs the secret $\boldsymbol{\lambda_\alpha}$ and computes $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}})$ for $i \in \{1, ..., k\}$, $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ where $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$.
4. Client sends $\boldsymbol{\mu_\alpha}$ to P_1.

Fig. 4. Protocol for Client input

$v_\gamma - \lambda_\gamma$. Recall that $v_\gamma = v_\alpha + v_\beta$,$\lambda_\gamma = \lambda_\alpha + \lambda_\beta$ and $v_\alpha = \phi(v^{(2)}_{\alpha_1}, ..., v^{(2)}_{\alpha_l})$,$v_\beta = \phi(v^{(2)}_{\beta_1}, ..., v^{(2)}_{\beta_l})$, where $\phi(\cdot)$ is an $\mathbb{F}_2$-linear map. Therefore P_1 can locally compute

$$\mu_\gamma = v_\gamma - \lambda_\gamma = \mu_\alpha + \mu_\beta.$$

We handle AND gates by converting them to multiplication gates using packed secret sharing and RMFE. The description of the protocol Π_{mult} appears in Protocol 3. The communication complexity per AND gate is $3 \cdot n \cdot m/(k \cdot l) \approx 36$ bits.

Furthermore, it is imperative to rearrange the computation results' bits for the next layer. Here, P_1 possesses $\boldsymbol{\mu_\gamma}$, which is masked by a random value vector $\boldsymbol{\lambda_\gamma}$. Random values, such as $\boldsymbol{\lambda_\alpha}$, intend for the next layer of circuit wires, have been pre-generated and redistributed in accordance with the circuit structure during the offline phase. P_1 needs to compute $\{\psi(\mu_{\gamma_i})\}_{i=1}^{k}$ to get the masked bit vectors. Subsequently, P_1 rearranges these bits $\{\psi(\mu_{\gamma_i})\}_{i=1}^{k}$ according to the structure of the subsequent circuit layer. Following this, P_1 will use $\phi(\cdot)$ to repack the bit vectors, resulting in a new $\boldsymbol{\lambda_\gamma}$.

Protocol 3: Π_{mult}

1. For each group of AND gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, P_1 holds $\boldsymbol{\mu_\alpha}$ an $\boldsymbol{\mu_\beta}$, and all parties have received packed Shamir sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$ from $\mathcal{F}_{offline}$.
2. P_1 computes $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$,$[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ and distributes the shares to all parties
3. All parties locally compute:

$$\begin{aligned}[\![\boldsymbol{\mu_\gamma}]\!]_{n-1} = {} & [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} \\ & + [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\![\boldsymbol{\lambda_\alpha} \cdot \boldsymbol{\lambda_\beta}]\!]_{n-k} \\ & - [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}\end{aligned}$$

   and send $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ to P_1.
4. P_1 collects the whole sharing $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ from all parties and reconstructs $\boldsymbol{\mu_\gamma}$.
5. P_1 computes $\psi(\boldsymbol{\mu_\gamma})$ and rearranges the bits according to the next layer of circuits to get $\boldsymbol{\mu_\gamma^{(2)}}$, then sets $\boldsymbol{\mu_\gamma} = \phi(\boldsymbol{\mu_\gamma^{(2)}})$.

Fig. 5. Protocol for evaluating AND gates by multiplication

Output Phase. In the output layer, for each group of $k \cdot l$ output gates that belong to some Client, we use $\boldsymbol{\alpha}$ to denote the input wires of these output gates. Recall that all parties have received a degree-$(n-1)$ packed Shamir sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ from $\mathcal{F}_{offline}$ in the offline phase and P_1 holds $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$. We use $\{v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}}\}_{i=1}^{k}$ to denote the output values of Client, denote $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$ and $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}})$ for $i \in \mathcal{K}$. The description of the protocol Π_{output} appears in Protocol 4.

Protocol 4: Π_{output}

1. For each goup of $k \cdot l$ output gates that belong to some Client, let $\boldsymbol{\alpha}$ denote the input wires of these gates. All parties send their shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ to Client.
2. P_1 sends $\boldsymbol{\mu_\alpha}$ to Client.
3. Client reconstructs $\boldsymbol{\lambda_\alpha}$ and computes $\boldsymbol{v_\alpha} = \boldsymbol{\lambda_\alpha} + \boldsymbol{\mu_\alpha}$, $(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}}) = \psi(v_{\alpha_i} \cdot \phi(\mathbf{1}))$ for $i \in \{1, ..., k\}$, where $\mathbf{1} = (1, 1, ..., 1) \in \mathbb{F}_2^l$.

Fig. 6. Protocol for output

Online Protocol. Once we have established the preceding protocols, we can now provide a comprehensive description of our online protocol, denoted as Π_{online}. The detailed description of our online protocol can be found in Protocol 5.

Protocol 5: Π_{online}

1. **Offline Phase:** All parties invoke $\mathcal{F}_{offline}$ to receive correlated randomness that will be used in the online phase.
2. **Input Phase:** In the input layer, for each group of $k \cdot l$ input gates that belong to some Client, let $\boldsymbol{\alpha}$ denote the output wires of these input gates. All parties and Client invoke Π_{input}. At the end of the protocol, P_1 learns $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$, where $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$ and $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}})$ for $i \in \mathcal{K}$, $\{v^{(2)}_{\alpha_{i,j}}\}_{i \in \mathcal{K}, j \in \mathcal{L}}$ are the input values of Client. And $\boldsymbol{\lambda_\alpha}$ are the random values associated with the batch of wires $\boldsymbol{\alpha}$ generated by $\mathcal{F}_{offline}$.
3. **Computation Phase:**Note for each computation layer, P_1 holds $\mu_\alpha = v_\alpha - \lambda_\alpha$. The circuit is evaluated layer by layer. Assume that the invariant holds for wires in previous layers. Consider gates in the current layer:
   - For each group of l XOR gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, P_1 locally compute $\mu_\gamma = \mu_\alpha + \mu_\beta$.
   - For each group of $k \cdot l$ AND gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties invoke Π_{mult}. At the end of the protocol, P_1 learns $\boldsymbol{\mu_\gamma}$.
4. **Output Phase:** For each group of $k \cdot l$ output gates, all parties and Client invoke Π_{output}. At the end of the protocol, Client learns $\{v^{(2)}_{\alpha_{i,j}}\}_{i \in \mathcal{K}, j \in \mathcal{L}}$.

Fig. 7. Protocol for online phase

The online communication complexity of Π_{online} is $3n/(k \cdot l) \approx 36$ bits per AND gate. Recall that $k = (n+3)/4$ and $m = 3 \cdot l$.

Functionality 3: $\mathcal{F}_{main}$

1. $\mathcal{F}_{main}$ receives the input from all clients. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{main}$ computes $C(x)$ and distributes the output to all clients.

Fig. 8. Functionality for main protocol

Protocol 6: $\Pi_{random}(\Sigma)$

1. All parties agree on a Vandermonde matrix $\boldsymbol{M}^T$ of size $n \times (t+1)$ in $\mathbb{F}$.
2. Each party P_i randomly samples a random-Σ sharing $\boldsymbol{S}^{(i)}$ and distributes the shares to other parties.
3. All parties locally compute $(\boldsymbol{R}^{(1)}, ..., \boldsymbol{R}^{(t+1)}) = \boldsymbol{M}(\boldsymbol{S}^{(1)}, ..., \boldsymbol{S}^{(n)})^T$ and output $(\boldsymbol{R}^{(1)}, ..., \boldsymbol{R}^{(t+1)})$.

Fig. 9. Protocol for generating random sharing

Lemma IV.1. *Protocol Π_{online} securely computes $\mathcal{F}_{main}$ in the $\mathcal{F}_{offline}$-hybrid model against a semi-honest adversary who controls t out of $n = 2t+1$ parties and corrupts up to c of the clients.*

The proof of Lemma IV.1 is given in the Appendix VIII-A.

C. Offline Protocol

In this section, we will show how to implement $\mathcal{F}_{offline}$. Recall that in $\mathcal{F}_{offline}$, we need to prepare packed Shamir sharings for the random values and Beaver triples, and assign random value for each wire.

To prepare random sharings that satisfy RMFE properties, we follow the technique in [21] as described in Functionality 1. The instantiation of $\mathcal{F}_{random}$ in the semi-honest and honest majority setting is Protocol 6.

For preparing Beaver triples $\{([a_i|i]_t, [b_i|i]_t, [c_i|i]_t)\}_{i=1}^k$, we rely on the state-of-the-art multiplication protocol ATLAS [9] in the standard honest majority setting to compute $[c_i|i]_t$, which costs $4n$ elements in $\mathbb{F}_{2^m}$. In ATLAS [9], the communication complexity of preparing packed Beaver triple is $8n$ elements. We use the ideal functionality $\mathcal{F}_{single\text{-}mult}$ from [12] to perform the multiplication of degree-t Shamir sharings. This functionality is described in Functionality 4. The description of our offline protocol $\Pi_{offline}$ appears in Protocol 7.

We analyze the communication complexity of $\Pi_{offline}$:

- For each gate whose input wires require network routing, the communication complexity is $15n$ each input wire.
- For each AND gate, the total communication complexity is $(4n + 2n + 2n/k) \cdot m/l = 18n + 24$ bits.
- For each input or output gate, the total communication complexity is $(2n + 2n/k) \cdot m/l \approx 6n + 24$ bits.

Lemma IV.2. *Protocol $\Pi_{offline}$ securely computes $\mathcal{F}_{offline}$ in the $\mathcal{F}_{random}$-hybrid model against a semi-honest adversary who controls t out of $n = 2t+1$ parties.*

The proof of Lemma IV.2 is given in the Appendix VIII-B.

Recall the communication complexity of our AND gates:

- In the offline phase, the communication complexity for each group of $l \cdot k$ AND gates consists of generating the random values $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-k}$, the random zero sharing $[\![\boldsymbol{0}]\!]_{n-1}$, and $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k}$. The total communication complexity per AND gate is $(2n+(2n/k)+4n) \cdot m/l \approx 18n + 24$ bits.
- In the online phase, the communication complexity for each group of $l \cdot k$ AND gates consists of P_1 sending $\boldsymbol{\mu_\alpha}$, $\boldsymbol{\mu_\beta}$ to parties, and P_1 collecting $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$, which is $3n$ bits. The total communication complexity per AND gate is $3n \cdot m/(k \cdot l) \approx 36$ bits.

V. Achieving Malicious Security

In this section, we will show how to achieve malicious security in the honest majority setting. Our main idea follows the approach of [12], compiling our semi-honest protocol to achieve malicious security. Due to space limitations, we mainly introduce the main ideas of our malicious protocol and the differences between it and the semi-honest protocol here. The complete protocol and security proof will be shown in Appendix IX.

Recall that for degree-t Shamir secret sharing of honest majority setting, the whole sharing is fully determined by the shares of honest parties, if malicious parties add a linear error to the secret, it will be discovered by honest parties. But for the degree-$(n-k)$ packed secret sharing, malicious parties can modify the secret without attracting the notice of honest parties. Malicious parties can let the honest parties hold the secret shares of 0 to generate a linear attack of any error $\boldsymbol{\delta} = (\delta_1, ..., \delta_{k-1})$. For example, assume the packed secret sharing is $[\![\boldsymbol{x}]\!]_{n-k}$, and the secret is store in the default positions $s_1, ..., s_k$ in the corresponding packed secret sharing polynomial $f(\cdot)$ where $f(s_i) = x_i$. Let P_i hold the packed share $f(p_i)$, and let $\mathcal{C}$ denote the set of corrupted parties, $\mathcal{H}$ denote the set of honest parties. Adversary can generate a degree-$(n-k)$ polynomial $g(\cdot)$ satisfying $g(p_j) = 0$ for all $j \in \mathcal{H}$, and $g(s_i) = \delta_i$ for all $i \in \{1, ..., k-1\}$. Thus, depending on the $g(\cdot)$, the adversary can let the shares held by malicious parties be $g(i) + f(i)$ for all $i \in Corr$, then honest parties can't notice the error.

The previous works [13], [16] use information-theoretic MACs, this method can detect attacks in the dishonest majority setting, but increases the communication complexity at least twice. The MPC protocols [5]–[12] in the honest majority setting, compute degree-t Shamir sharings to detect attacks. And [12] uses different evaluation points to store the secret, computes k degree-t Shamir sharings and converts such Shamir sharing into a packed secret sharing. We follow the approach of [12], the difference is that we don't assign a random degree-t Shamir sharing for a single wire, but assign a random degree-t Shamir sharing for a group of l wires.

For each group of l wires, we will assign a random degree-t Shamir sharing. The degree-t random sharings used in what

Protocol 7: $\Pi_{offline}$

1. **Preparing Degree-$(n-1)$ Zero Sharings:** Let Σ be the secret sharing scheme corresponding to $[\![\mathbf{0}]\!]_{n-1}$, where $\mathbf{0} \in \mathbb{F}^k$. For each goup of AND gates, input gates, and output gates, all parties invoke $\mathcal{F}_{random}$ to prepare random sharings in the form of $[\![\mathbf{0}]\!]_{n-1}$.
2. **Assigning Random Sharing for Each Wire:**
   a. For each group of $k \cdot l$ output wires $\boldsymbol{\alpha}$ of input gates and AND gates, or input wires of output gates, all parties invoke $\mathcal{F}_{random}$ to prepare random sharings in the form of $[\phi(\boldsymbol{r_i})|i]_t$ for all $i \in \{1, ..., k\}$. All parties associate values $[\phi(\boldsymbol{r_i})|i]_t$ with each l wires as $[\lambda_{\alpha_i}|i]_t$. Let $e_i \in \mathbb{F}^k$ be the i-th unit vector, all parties locally transform e_i to the degree-$(k-1)$ packed Shamir sharing $[\![\boldsymbol{e_i}]\!]_{k-1}$ and use the $[\![\mathbf{0}]\!]_{n-1}$ which generated by Step 2 to compute:

$$[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!]_{k-1} * [\![\phi(\boldsymbol{r_i})|i]\!]_t + [\![\mathbf{0}]\!]_{n-1}$$

   b. Starting from the first layer of C to the last layer, for all group of gates requiring routing, all parties invoke $\Pi_{routing}$ with the input of corresponding random values $[\phi(\boldsymbol{\lambda_w})|s]_t$ from previous layer's output wires. All parties get the random wire values $[\phi(\boldsymbol{\lambda'_w})|s']_t$ of this layer corresponding wires $\boldsymbol{w'}$.
      i. For each group of $l \cdot k$ XOR gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties set the random values of output wires $[\lambda_\gamma|i]_t = [\lambda_{\alpha_i}|i]_t + [\lambda_{\beta_i}|i]_t$ and set $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\alpha_i}|i]_t$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\beta_i}|i]_t$, and let $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-k} = [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$. Parties associate $\{[\lambda_\alpha|i], [\lambda_\beta|i], \lambda_\gamma|i]\}_{i=1}^{k}$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
      ii. For each group of $l \cdot k$ AND gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties invoke $\mathcal{F}_{single\text{-}mult}$ with the input of $[\lambda_{\alpha_i}|i]$, $[\lambda_{\beta_i}|i]_t$ to compute $[\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$ for all $i \in \mathcal{K}$. Parties set $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\alpha_i}|i]_t$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\beta_i}|i]_t$, $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$. Parties associate $\{[\lambda_\alpha|i], [\lambda_\alpha|i]\}_{i=1}^{k}$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$.

Fig. 10. Protocol for offline phase

Functionality 4: $\mathcal{F}_{single\text{-}mult}$

1. $\mathcal{F}_{single\text{-}mult}$ receives the secret position i from all parties. Let $[x|i]_t$ denote the input sharings. $\mathcal{F}_{single\text{-}mult}$ receives from honest parties their shares of $[x|i]_t$,$[y|i]_t$. Then $\mathcal{F}_{single\text{-}mult}$ reconstructs the secret x, y and computes the shares of $[x|i]_t, [y|i]_t$ held by corrupted parties, and sends these shares to the adversary.
2. $\mathcal{F}_{single\text{-}mult}$ receives from the adversary a set of shares $\{z_i\}_{i \in Corr}$.
3. $\mathcal{F}_{single\text{-}mult}$ computes $x \cdot y$. Based on the secret $z = x \cdot y$ and t shares $\{z_i\}_{i \in Corr}$, $\mathcal{F}_{single\text{-}mult}$ reconstruct the whole sharing $[z|i]_t$ and distributes the shares of $[z|i]_t$ to honest parties which satisfied the shares of $[z|i]_t$ of corrupted parties are $\{z_i\}_{i \in Corr}$.

Fig. 11. Functionality for single elements multiplication

follows have the form of $[\phi(\boldsymbol{r})]_t$, which we will abbreviate as $[r]_t$. For each group of $k \cdot l$ AND gates:

1. First, we will generate k Beaver triples in the form of $\{([a_i|i]_t, [b_i|i]_t, [c_i|i]_t)\}_{i=1}^{k}$ such that $c_i = a_i \cdot b_i$.
2. Let $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ be the input wires, where $\boldsymbol{\alpha} = \{\alpha_{i,j}\}$, $\boldsymbol{\beta} = \{\beta_{i,j}\}$ for $i \in \mathcal{K}$ and $j \in \mathcal{L}$. For each l wires of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, let $[\lambda_{\alpha_i}|i]_t = [a_i|i]_t$, $[\lambda_{\beta_i}|i]_t = [b_i|i]_t$, associate $[\lambda_{\alpha_i}|i]_t$, $[\lambda_{\beta_i}|i]_t$ with the wires $\{\alpha_{i,1}, ..., \alpha_{i,l}\}$, $\{\beta_{i,1}, ..., \beta_{i,l}\}$.
3. In the online phase, P_1 sends $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to parties, then parties can locally compute $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$. And all parties can locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} + [\lambda_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t = [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\lambda_{\beta_i}|i]_t$ for all $i \in \mathcal{K}$.

If we view $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$ as a degree-t Shamir sharing, we can compute the degree-t Shamir sharings of input values $[v_{\beta_i}|i]_t$ locally. Recall that for degree-t Shamir secret sharing of honest majority setting, the whole sharing is fully determined by the shares of the honest parties. In the following, we will show that this is sufficient to verify the correctness of the computation.

A. Offline Phase

In the offline phase, which diverges from its semi-honest counterpart, parties will receive for each packed triple $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$ and the output $\{([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t)\}_{i=1}^{k}$. For each group of input gates or output gates, all parties will prepare a set of random degree-t Shamir sharing $\{[r_i|i]_t\}_{i=1}^{k}$. During the output phase, these degree-t Shamir sharings can help Client to detect attacks launched by corrupted parties. The functionality $\mathcal{F}_{offline\text{-}mal}$ and protocol $\Pi_{offline\text{-}mal}$ for the offline phase with malicious security are given in Appendix IX Protocol 9.

The amortized communication complexity of preparing a random degree-t sharing for l binary values with malicious security is $2n$ elements.

The communication complexity of $\Pi_{offline\text{-}mal}$ is as follows:

- For each gate whose input wires require network routing, the communication complexity is $15n$ each input wire.
- For each AND gate, the amortized communication complexity is $(2n + 4n + 2n/k) * m/l \approx 18n + 24$ bits.
- For each input gate, the amortized communication complexity is $(2n+2n+3n/k)*m/l \approx 12n+36$ bits. For each output gate, the amortized communication complexity is $12n + 24$ bits.

B. Online Phase

In the online phase, we compute degree-t Shamir sharings for each l input wires of AND gates. Recall that in honest majority setting, the degree-t Shamir sharing is totally decided by honest parties. By using these degree-t Shamir sharing, parties can verify the correctness of the secret.

Input Phase. Note that in $\mathcal{F}_{offline\text{-}mal}$, for each group of input gates with input wires $\boldsymbol{\alpha}$ that belong to some Client, parties hold $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$, $\{[r_i|i]_t\}_{i=1}^k$. We use $\{v^{(2)}_{\alpha_{i,j}}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ to denote the input values of Client, let $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,l}})$ and $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$.

1. First, all parties send their shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$, $\{[r_i|i]_t\}_{i=1}^k$ to Client.
2. Client reconstructs $\boldsymbol{r} = (r_1, ..., r_k)$, computes and distributes $[\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t$ to all parties.
3. Parties locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t - [r_i|i]_t$ for $i \in \{1, ..., k\}$.
4. Client also computes $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ and sends to P_1.

Because the degree-t Shamir sharing $[\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_t$ is determined by $t+1$ parties, the honest parties can reconstruct these shares by themselves, malicious parties are unable to alter these values without being detected.

The description of $\Pi_{input\text{-}mal}$ appears in Appendix IX Protocol 10. The communication complexity of $\Pi_{input\text{-}mal}$ is $(n \cdot (k+1) + n + k) \cdot m/(k \cdot l) \approx 3n + 27$ bits per input gate.

Computation Phase. For computing AND gates, we follow the approach of semi-honest protocol of Π_{mult} in general, but the difference is that parties need to locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} + [\lambda_{\alpha_i}|i]_t$ and $[v_{\beta_i}|i]_t = [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\lambda_{\beta_i}|i]_t$, and $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\alpha_i}|i]_t$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\beta_i}|i]_t$, $[\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$.

The correctness of the protocol follows the same argument as the semi-honest version. The description of $\Pi_{mult\text{-}mal}$ appears in Appendix IX, Protocol 11. The communication complexity of $\Pi_{mult\text{-}mal}$ is $12 \cdot m/l \approx 36$ bits per AND gate.

Output Phase and Validity Check. We use $\Pi_{consistency}$ to check the degree-$(k-1)$ packed sharings lie on a degree-$(k-1)$ polynomial. This is described in Appendix IX, protocol 12. We describe the functionality $\mathcal{F}_{evaluate}$ in Appendix IX, Functionality 9. The protocol $\Pi_{evaluate}$ is an instantiation of $\mathcal{F}_{evaluate}$ and appears in Appendix IX, Protocol 13.

We also follow the approach of [12], [24] to verify the correctness of the secret, the functionality $\mathcal{F}_{verify}$ to check the correctness of the computation, the instantiation of $\mathcal{F}_{verify}$ can be found in [12]. The description of $\mathcal{F}_{verify}$ appears in Appendix IX Functionality 11. we refer the readers to Appendix IX for more details.

VI. Conclusion

Building upon previous research, we developed a constant online communication MPC protocol for Boolean circuits using RMFE and PSS. This protocol achieves amortized 36 bits per AND gate in the online phase and $30n + 24$ bits per AND gate in the offline phase, in both semi-honest and malicious settings.

In multi-party PPML applications, non-linear operations often lead to significant communication overheads. Our protocol, by leveraging the strengths of Boolean circuits, stands out in situations like machine learning inference and training that hinge on activation function computations. Moving forward, our goal is to delve deeper into the practical application of this protocol in the realm of PPML, capitalizing on the unique benefits Boolean circuits offer in processing nonlinear functions.

Acknowledgements

We sincerely thank Yifan Song for his valuable comments and suggestions, which helps us improve this work. We also thank anonymous reviewers for their helpful comments.

Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 92270201 and 62125204). Yu Yu also acknowledges the support from the XPLORER PRIZE.

References

[1] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (abstract) (informal contribution)," in *Advances in Cryptology – CRYPTO'87*, ser. Lecture Notes in Computer Science, C. Pomerance, Ed., vol. 293. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 16–20, 1988, p. 462.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 2–4, 1988, pp. 1–10.

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *19th Annual ACM Symposium on Theory of Computing*, A. Aho, Ed. New York City, NY, USA: ACM Press, May 25–27, 1987, pp. 218–229.

[4] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th Annual Symposium on Foundations of Computer Science*. Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 27–29, 1986, pp. 162–167.

[5] E. Boyle, N. Gilboa, Y. Ishai, and A. Nof, "Efficient fully secure computation via distributed zero-knowledge proofs," in *Advances in Cryptology – ASIACRYPT 2020, Part III*, ser. Lecture Notes in Computer Science, S. Moriai and H. Wang, Eds., vol. 12493. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 7–11, 2020, pp. 244–276.

[6] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, "Fast large-scale honest-majority MPC for malicious adversaries," in *Advances in Cryptology – CRYPTO 2018, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2018, pp. 34–64.

[7] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Advances in Cryptology – CRYPTO 2007*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 4622. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2007, pp. 572–590.

[8] D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer, "Circuits resilient to additive attacks with applications to secure computation," in *46th Annual ACM Symposium on Theory of Computing*, D. B. Shmoys, Ed. New York, NY, USA: ACM Press, May 31 – Jun. 3, 2014, pp. 495–504.

[9] V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song, "ATLAS: Efficient and scalable MPC in the honest majority setting," in *Advances in Cryptology – CRYPTO 2021, Part II*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Virtual Event: Springer, Heidelberg, Germany, Aug. 16–20, 2021, pp. 244–274.

[10] P. S. Nordholt and M. Veeningen, "Minimising communication in honest-majority MPC by batchwise multiplication verification," in *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, B. Preneel and F. Vercauteren, Eds., vol. 10892. Leuven, Belgium: Springer, Heidelberg, Germany, Jul. 2–4, 2018, pp. 321–339.

[11] V. Goyal, Y. Song, and C. Zhu, "Guaranteed output delivery comes free in honest majority MPC," in *Advances in Cryptology – CRYPTO 2020, Part II*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2020, pp. 618–646.

[12] D. Escudero, V. Goyal, A. Polychroniadou, and Y. Song, "TurboPack: Honest majority MPC with constant online communication," in *ACM CCS 2022: 29th Conference on Computer and Communications Security*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. Los Angeles, CA, USA: ACM Press, Nov. 7–11, 2022, pp. 951–964.

[13] V. Goyal, A. Polychroniadou, and Y. Song, "Sharing transformation and dishonest majority MPC with packed secret sharing," in *Advances in Cryptology – CRYPTO 2022, Part IV*, ser. Lecture Notes in Computer Science, Y. Dodis and T. Shrimpton, Eds., vol. 13510. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 15–18, 2022, pp. 3–32.

[14] D. Escudero, V. Goyal, A. Polychroniadou, Y. Song, and C. Weng, "SuperPack: Dishonest majority MPC with constant online communication," in *Advances in Cryptology – EUROCRYPT 2023, Part II*, ser. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, Jun. 2023, pp. 220–250.

[15] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Advances in Cryptology – EUROCRYPT 2011*, ser. Lecture Notes in Computer Science, K. G. Paterson, Ed., vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, May 15–19, 2011, pp. 169–188.

[16] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2012, pp. 643–662.

[17] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *ESORICS 2013: 18th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, J. Crampton, S. Jajodia, and K. Mayes, Eds., vol. 8134. Egham, UK: Springer, Heidelberg, Germany, Sep. 9–13, 2013, pp. 1–18.

[18] G. Beck, A. Goel, A. Jain, and G. Kaptchuk, "Order-C secure multiparty computation for highly repetitive circuits," in *Advances in Cryptology – EUROCRYPT 2021, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and F.-X. Standaert, Eds., vol. 12697. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 17–21, 2021, pp. 663–693.

[19] M. K. Franklin and M. Yung, "Communication complexity of secure computation (extended abstract)," in *24th Annual ACM Symposium on Theory of Computing*. Victoria, BC, Canada: ACM Press, May 4–6, 1992, pp. 699–710.

[20] I. Cascudo, R. Cramer, C. Xing, and C. Yuan, "Amortized complexity of information-theoretically secure MPC revisited," in *Advances in Cryptology – CRYPTO 2018, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2018, pp. 395–426.

[21] A. Polychroniadou and Y. Song, "Constant-overhead unconditionally secure multiparty computation over binary fields," in *Advances in Cryptology – EUROCRYPT 2021, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and F.-X. Standaert, Eds., vol. 12697. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 17–21, 2021, pp. 812–841.

[22] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear PCPs," in *Advances in Cryptology – CRYPTO 2019, Part III*, ser. Lecture Notes in Computer Science, A. Boldyreva and D. Micciancio, Eds., vol. 11694. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 18–22, 2019, pp. 67–97.

[23] E. Ben-Sasson, S. Fehr, and R. Ostrovsky, "Near-linear unconditionally-secure multiparty computation with a dishonest minority," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2012, pp. 663–680.

[24] F. Benhamouda, E. Boyle, N. Gilboa, S. Halevi, Y. Ishai, and A. Nof, "Generalized pseudorandom secret sharing and efficient straggler-resilient secure computation," in *TCC 2021: 19th Theory of Cryptography Conference, Part II*, ser. Lecture Notes in Computer Science, K. Nissim and B. Waters, Eds., vol. 13043. Raleigh, NC, USA: Springer, Heidelberg, Germany, Nov. 8–11, 2021, pp. 129–161.

[25] A. Shamir, "How to share a secret," *Communications of the Association for Computing Machinery*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[26] V. Goyal and Y. Song, "Malicious security comes free in honest-majority MPC," Cryptology ePrint Archive, Report 2020/134, 2020, https://eprint.iacr.org/2020/134.

[27] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, Jan. 2000.

[28] V. Goyal, A. Polychroniadou, and Y. Song, "Unconditional communication-efficient MPC via hall's marriage theorem," in *Advances in Cryptology – CRYPTO 2021, Part II*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Virtual Event: Springer, Heidelberg, Germany, Aug. 16–20, 2021, pp. 275–304.

APPENDIX

VII. THE MODEL

A. Security Definition

In the work, we focus on the honest majority setting. Let $t = (n-1)/2$ be an integer. Let $\mathcal{F}$ be a secure function evaluation functionality. An adversary $\mathcal{A}$ can corrupt at most t parties, provide inputs to corrupted parties, and receive all messages sent to corrupted parties. In this work, we consider both semi-honest adversaries and malicious adversaries.

- If $\mathcal{A}$ is semi-honest, then corrupted parties honestly follow the protocol.
- If $\mathcal{A}$ is fully malicious, then corrupted parties can deviate from the protocol arbitrarily.

Real-World Execution. In the real world, the adversary $\mathcal{A}$ controlling corrupted parties interacts with honest parties. At the end of the protocol, the output of the real-world execution includes the inputs and outputs of honest parties and the view of the adversary.

Ideal-World Execution. In the ideal world, a simulator $\mathcal{S}$ simulates honest parties and interacts with the adversary $\mathcal{A}$. Furthermore, $\mathcal{S}$ has one-time access to $\mathcal{F}$, which includes providing inputs of corrupted parties to $\mathcal{F}$, receiving the outputs of corrupted parties, and sending instructions specified in $\mathcal{F}$ (e.g., asking $\mathcal{F}$ to abort). The output of the ideal-world execution includes the inputs and outputs of honest parties and the view of the adversary.

Semi-honest Security. We say that a protocol Π computes $\mathcal{F}$ with perfect security if for all semi-honest adversary $\mathcal{A}$, there exists a simulator S such that the distribution of the output of the real-world execution is identical to the distribution in the ideal-world execution.

Security-with-abort. We say that a protocol Π securely computes $\mathcal{F}$ with abort if for all adversary $\mathcal{A}$, there exists

a simulator $\mathcal{S}$, which is allowed to abort the protocol, such that the distribution of the output of the real-world execution is statistically close to the distribution in the ideal-world execution.

B. Hybrid Model

The hybrid model is used in [27] to prove security. In the hybrid model, all parties are given access to a trusted party (or alternatively, an ideal functionality) which computes a particular function for them. The modular sequential composition theorem from [27] shows that it is possible to replace the ideal functionality used in the construction by a secure protocol computing this function. When the ideal functionality is denoted by g, we say the construction works in the g-hybrid model.

C. Client-server Model

To simplify the security proofs, we consider the client-server model. In the client-server model, clients provide inputs to the functionality and receive outputs, and servers can participate in the computation but do not have inputs or get outputs. Each party may have different roles in the computation. Note that, if every party plays a single client and a single server, this corresponds to a protocol in the standard MPC model. Let c denote the number of clients and n denote the number of servers. For all clients and servers, we assume that every two of them are connected via a secure (private and authentic) synchronous channel so that they can directly send messages to each other. The communication complexity is measured in the same way as that in the standard MPC model.

Security in the Client-server Model. In the client-server model, an adversary $\mathcal{A}$ can corrupt at most c clients and t servers, provide inputs to corrupted clients, and receive all messages sent to corrupted clients and servers. The security is defined similarly to the standard MPC model.

We say that a protocol Π securely computes $\mathcal{F}$ if there exists a simulator Sim, such that for all adversaries $\mathcal{A}$, the distribution of the output of the real world execution is statistically close to the distribution of the output of the ideal world execution.

VIII. Security Proofs of Our Semi-honest Protocol

A. Proof of Lemma IV.1

Lemma IV.1. Protocol Π_{online} securely computes $\mathcal{F}_{main}$ in the $\mathcal{F}_{offline}$-hybrid model against a semi-honest adversary who controls t out of $n = 2t + 1$ parties and corrupts up to c of the clients.

Proof. We use $\mathcal{A}$ to denote the adversary, use $Corr$ to denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties.

The correctness of Π_{online} follows from the description.

Now we will describe the construction of the simulator $\mathcal{S}$:

1. In the offline phase, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{offline}$ and receives the shares of corrupted parties for each packed Shamir sharing. Recall that $\mathcal{F}_{offline}$ does not need to send any message to corrupted parties.
2. In the input phase, for each group of $k \cdot l$ input gates that belong to some Client, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates.
   - If Client is honest, after receiving the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ from all parties. $\mathcal{S}$ samples random values and converts them to $\boldsymbol{\mu_\alpha}$ by RMFE and sends them to P_1.
   - If Client is corrupted, $\mathcal{S}$ samples random values and converts them to $\boldsymbol{\lambda_\alpha}$ by RMFE. Then based on the secrets $\boldsymbol{\lambda_\alpha}$ and the shares of corrupted parties, $\mathcal{S}$ samples the whole sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ and sends the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ held by honest parties to Client. From the inputs $\boldsymbol{v_\alpha}$ of Client, $\mathcal{S}$ computes $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$.
3. Note that in the computation phase, $\mathcal{S}$ will always know $\boldsymbol{\mu_\alpha}$ for each group of wires. In the input layer $\mathcal{S}$ has known $\boldsymbol{\mu_\alpha}$. For each group of XOR gate with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wire $\boldsymbol{\gamma}$, $\mathcal{S}$ honestly compute $\boldsymbol{\mu_\gamma} = \boldsymbol{\mu_\alpha} + \boldsymbol{\mu_\beta}$. For each group of AND gates with input wires $\boldsymbol{\alpha}$,$\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{S}$ simulates Π_{mult} as follows:
   - If P_1 is honest, $\mathcal{S}$ computes degree-$(k-1)$ packed Shamir sharings $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$ and $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ based on $\boldsymbol{\mu_\alpha}$ and $\boldsymbol{\mu_\beta}$. Then $\mathcal{S}$ computes the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of corrupted parties. $\mathcal{S}$ samples $2 \cdot k \cdot l$ random values $\{(a_{i,j}^{(2)}, b_{i,j}^{(2)})\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ over $\mathbb{F}_2$ and uses RMFE to convert them to $2 \cdot k$ random values as $\boldsymbol{a}, \boldsymbol{b}$ over $\mathbb{F}_{2^m}$, compute $\boldsymbol{\mu_\gamma} = \boldsymbol{a} * \boldsymbol{b}$. Finally, based on the random values $\boldsymbol{\mu_\gamma}$ and the shares of corrupted parties, $\mathcal{S}$ computes the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of honest parties.
   - If P_1 is corrupted, $\mathcal{S}$ receives from P_1 the shares of $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$ and $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ of honest parties. Then $\mathcal{S}$ recovers the whole sharings $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$, and learns the shares of corrputed parties. Now $\mathcal{S}$ can compute the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of corrupted parties. $\mathcal{S}$ samples $2 \cdot k \cdot l$ random values $\{(a_{i,j}^{(2)}, b_{i,j}^{(2)})\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ over $\mathbb{F}_2$ and uses RMFE to convert them to $2 \cdot k$ random values as $\boldsymbol{a}, \boldsymbol{b}$ over $\mathbb{F}_{2^m}$, compute $\boldsymbol{\mu_\gamma} = \boldsymbol{a} * \boldsymbol{b}$. Based on the random values $\boldsymbol{\mu_\gamma}$ and the shares of corrupted parties, $\mathcal{S}$ computes the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of honest parties and sends them to P_1.
4. In the output phase, for each group of $k \cdot l$ output gates that belong to some Client, let $\boldsymbol{\alpha}$ denote the batch of input wires of these output gates.
   - If Client is honest, $\mathcal{S}$ does nothing.
   - If Client is corrupted, $\mathcal{S}$ sends the inputs of corrupted clients to $\mathcal{F}_{main}$ (note $\mathcal{S}$ can access to the inputs and random tapes of corrupted clients and corrupted parties). Then $\mathcal{S}$ receives the output $\boldsymbol{v_\alpha}$ of Client from $\mathcal{F}_{main}$. $\mathcal{S}$ computes $\boldsymbol{\lambda_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\mu_\alpha}$. Based on

the secrets $\boldsymbol{\lambda_\alpha}$ and the shares of corrupted parties, $\mathcal{S}$ randomly samples the whole sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$. $\mathcal{S}$ sends to Client the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ of honest parties. If P_1 is honest, $\mathcal{S}$ also sends $\boldsymbol{\mu_\alpha}$ to Client.

This completes the description of $\mathcal{S}$.

Now we will show that $\mathcal{S}$ can perfectly simulate the behaviors of honest parties. It is sufficient to focus on the places where honest parties and clients need to communicate with corrupted parties and clients:

1. In the input phase, for each group of $k \cdot l$ input gates of honest parties that belong to some Client, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates. If Client is honest, then $\mathcal{S}$ needs to simulate the values $\boldsymbol{\mu_\alpha}$ sent from Client to P_1. In the ideal world, $\mathcal{S}$ samples $l \cdot k$ random values and converts them to $\boldsymbol{\mu_\alpha}$ by RMFE. Since $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ and $\boldsymbol{\lambda_\alpha}$ are uniformly random, both $\boldsymbol{v_\alpha}$ and $\boldsymbol{\lambda_\alpha}$ satisfy the properties of RMFE. Therefore $\boldsymbol{\mu_\alpha}$ are uniformly random and satisfy the properties of RMFE, the distribution of $\boldsymbol{\mu_\alpha}$ simulated by $\mathcal{S}$ has the same distribution as that in the real world.
   If Client is corrupted, then $\mathcal{S}$ needs to simulated the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ of honest parties. In real world, parties need to send the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ to Client. In the ideal world, $\mathcal{S}$ randomly samples $\boldsymbol{\lambda_\alpha}$ which satisfies the properties of RMFE. Note the degree of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ is $n-1$, at least n values can totally decide the polynomial. Then $\mathcal{S}$ randomly samples the shares of honest parties based on the secrets $\boldsymbol{\lambda_\alpha}$ and the shares of corrupted parties. Therefore, the distribution of the shares of $\boldsymbol{\lambda_\alpha}$ of honest parties is identical to that in the real world. And from the inputs of Client, $\mathcal{S}$ can also compute $\boldsymbol{\mu_\alpha}$. Therefore, $\mathcal{S}$ perfectly simulates the behaviors of honest parties and clients in the input phase.
2. In the computation phase, we will show that $\mathcal{S}$ can always learn $\boldsymbol{\mu_\alpha}$ for each group of wires $\boldsymbol{\alpha}$ in the circuit, and $\boldsymbol{\mu_\alpha}$ has the same distribution with $\boldsymbol{\mu_\alpha}$ in real world. Recall that $\mathcal{S}$ already learns $\boldsymbol{\mu_\alpha}$ in input layer. For each group of XOR gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{S}$ can compute $\boldsymbol{\mu_\gamma} = \boldsymbol{\mu_\alpha} + \boldsymbol{\mu_\beta}$. For each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, we will divide the case into whether P_1 is honest or not:
   – If P_1 is honest, $\mathcal{S}$ honestly computes and distributes $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$,$[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$. Therefore the distribution of the $\boldsymbol{\mu_\alpha}, \boldsymbol{\mu_\beta}$ and the distribution of sharings $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$,$[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ is the same in both worlds. Then for $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$, in real word parties compute $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ as follow:

$$\begin{aligned}[\![\boldsymbol{\mu_\gamma}]\!]_{n-1} = &[\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} \\ &+ [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} + [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} \\ &+ [\![\boldsymbol{\lambda_\alpha} \cdot \boldsymbol{\lambda_\beta}]\!]_{n-1} - [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}\end{aligned}$$

   send to P_1, P_1 reconstructs $\boldsymbol{\mu_\gamma}$ and rearranges the bits according the next layer of circuit, computes $\boldsymbol{\mu_\gamma} = \phi(\psi(\boldsymbol{\mu_\gamma}))$. Because $\boldsymbol{\lambda_\gamma}$ are uniformly random and generated by $\mathcal{F}_{offline}$, the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of honest parties are also uniformly random. In the ideal world, $\mathcal{S}$ samples $2 \cdot k \cdot l$ random values $\{(a_{i,j}^{(2)}, b_{i,j}^{(2)})\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ over $\mathbb{F}_2$ and uses RMFE to convert them to $2 \cdot k$ random values as $\boldsymbol{a}, \boldsymbol{b}$ over $\mathbb{F}_{2^m}$, compute $\boldsymbol{\mu_\gamma} = \boldsymbol{a} * \boldsymbol{b}$, and based on the shares of corrupted parties compute $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$. Thus, the values of $\boldsymbol{\mu_\gamma}$ and the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ all have the same distribution in both worlds.
   – If P_1 is corrupted, $\mathcal{S}$ receives the shares of $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$,$[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$, and computes the shares of honest parties. Then $\mathcal{S}$ computes the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of corrupted parties. With the same argument as above, the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of honest parties are uniformly random. In the idea world, $\mathcal{S}$ samples $2 \cdot k \cdot l$ random values $\{(a_{i,j}^{(2)}, b_{i,j}^{(2)})\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ over $\mathbb{F}_2$ and uses RMFE to convert them to $2 \cdot k$ random values as $\boldsymbol{a}, \boldsymbol{b}$ over $\mathbb{F}_{2^m}$, computes $\boldsymbol{\mu_\gamma} = \boldsymbol{a} * \boldsymbol{b}$, and based on the shares of corrupted parties computes the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ of honest parties and send to P_1. Thus, the values of $\boldsymbol{\mu_\gamma}$ and the shares of $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ all have the same distribution in both worlds.
3. In the output phase, for each group of $k \cdot l$ output gates that belong to some Client, let $\boldsymbol{\alpha}$ denote the batch of input wires of these output wires. If Client is honest, honest parties and clients do not need to send any messages to corrupted parties and clients. If Client is corrupted, $\mathcal{S}$ can learn the output of Client from $\mathcal{F}_{main}$. Since $\mathcal{S}$ learns $\boldsymbol{\mu_\alpha}$, $\mathcal{S}$ can compute $\boldsymbol{\lambda_\alpha}$. In both worlds, $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the secrets $\boldsymbol{\lambda_\alpha}$ and the shares of corrupted parties. Thus, the shares of honest parties generated by $\mathcal{S}$ have the same distribution as that in the real world.

B. Proof of Lemma IV.2

Lemma IV.2. Protocol $\Pi_{offline}$ securely computes $\mathcal{F}_{offline}$ in the $\mathcal{F}_{random}$-hybrid model against a semi-honest adversary who controls t out of $n = 2t + 1$ parties.

Proof. We will prove that the secrets of the output sharings of $\Pi_{offline}$ have the same distribution of these of $\mathcal{F}_{offline}$. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. First, recall the structure of Protocol $\Pi_{offline}$:

1. Preparing Degree-$(n-1)$ Zero Sharings.
2. Computing Random Sharing for Each Group of Wires.

Now, we will construct the simulator $\mathcal{S}$ to simulate the behaviors of honest parties:

1. In the phase of preparing random degree-$(n-1)$ zero sharings, $\mathcal{S}$ simulates the invocation of $\mathcal{F}_{random}$ to generate random sharings $[\![\mathbf{0}]\!]_{n-1}$, where $\mathbf{0} \in \mathbb{F}^k$.
2. In the phase of computing random sharings for each group of wires,
   a. For each group of output wires of input gates and AND gates, and input wires of output gates, $\mathcal{S}$

emulates the functionality of $\mathcal{F}_{random}$ to prepare $[\phi(\boldsymbol{r_i})|i]_t$ and receives the shares of corrupted parties. $\mathcal{S}$ follow the protocol compute $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$.

b. For each group of XOR gates, $\mathcal{S}$ computes $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-k} = [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$.

c. For each group of AND gates, $\mathcal{S}$ emulates the ideal functionality of $\mathcal{F}_{single\text{-}mult}$ to obtain $\phi(\boldsymbol{c_i})$, such that $\boldsymbol{c_i} = \boldsymbol{\lambda_{\alpha_i}} * \boldsymbol{\lambda_{\beta_i}}$, receives the shares of $\phi(\boldsymbol{c_i})$ of corrupted parties. Finally, $\mathcal{S}$ computes the packed Beaver triples $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$ following the same local computations as in the protocol.

Next, we will show $\mathcal{S}$ perfectly simulate the behaviors of honest parties.

– In the phase of computing random sharing for each group of wires:
  – For each group of output wires of input gates and AND gates, and input wires of output gates, in the real world, $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the secrets $\boldsymbol{\lambda_\alpha}$ and the shares of corrupted parties. In the ideal world, $\mathcal{F}_{offline}$ generates a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{\lambda_\alpha}$ given the shares of corrupted parties. Therefore, the sharing $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ has the same distribution in both worlds.
  – For each group of XOR gates with the input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, in the real world, the random sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$,$[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ come from previous layer, which has the same distrbution in both worlds, thus $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$,$[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$,$[\![\boldsymbol{\lambda_\gamma}]\!]_{n-k} = [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$ has the same ditribution in both worlds.
  – For each group of AND gates with the input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, the invocation of $\mathcal{F}_{single\text{-}mult}$ generated the shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, which have the same distribution in both worlds.

Thus, we can conclude that Protocol $\Pi_{offline}$ securely computes $\mathcal{F}_{offline}$ in the $\mathcal{F}_{random}$-hybrid model against a semi-honest adversary who controls t out of $n = 2t + 1$ parties.

IX. Potocol and Proof of Malicious Security

In this section, we will show how to achieve malicious security in the honest majority setting. Our main idea follows the approach of [12], compiling our semi-honest protocol to achieve malicious security.

Recall that for degree-t Shamir secret sharing of honest majority setting, the whole sharing is fully determined by the shares of honest, if malicious parties add a linear error to the secret, it will be discovered by honest parties. But for the degree-$(n-k)$ packed secret sharing, malicious parties can modify the secret without attracting the notice of honest parties. Because for a degree-$(n-k)$ packed secret sharing, malicious parties can let the honest parties hold the secret shares of 0 to generate a linear attack of any error $\boldsymbol{\delta} = (\delta_1, ..., \delta_{k-1})$. For example, assumed the packed secret sharing is $[\![\boldsymbol{x}]\!]_{n-k}$, and the secret is store in default positions $s_1, ..., s_k$ in the corresponding packed secret sharing polynomial $f(\cdot)$ where $f(s_i) = x_i$, and P_i hold the packed share $f(p_i)$, let $\mathcal{C}$ denote the set of corrupted parties, $\mathcal{H}$ denote the set of honest parties. Adversary can generate a degree-$(n-k)$ polynomial $g(\cdot)$ satisfied $g(p_j) = 0$, for all $j \in \mathcal{H}$ and $g(s_i) = \delta_i$, for all$i \in \{1, ..., k-1\}$. Thus depend on the $g(\cdot)$, the adversary can let the shares held by malicious parties be $g(i) + f(i)$ for all $i \in Corr$, then honest party can't notice the error.

The previous work [13], [16] use information-theoretic MACs, this method can detect attack in the dishonest majority setting, but increases the communication complexity at least twice. The state-of-the-art MPC protocol [5]–[12] in the honest majority setting, compute degree-t Shamir sharing. And [12] uses different evaluation points to store the secret, computes k degree-t Shamir sharings and convert such Shamir sharing into a packed secret sharing. We follow the approach of [12], the difference is we don't assign a random degree-t Shamir sharing for a single wire, but assign a random degree-t Shamir sharing for a group of l wires.

For each l wires, we will assign a random degree-t Shamir sharing. Note in honest majority setting, the degree-t Shamir sharing is totally decided by honest parties. For each group of $k \cdot l$ AND gates:

1. First we will generate k triples in the form of $\{([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t)\}_{i=1}^k$.
2. Let $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ be the input wires, where $\boldsymbol{\alpha} = \{\alpha_{i,j}\}$, $\boldsymbol{\beta} = \{\beta_{i,j}\}$ for $i \in \mathcal{K}$ and $j \in \mathcal{L}$. For each l wires of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, associates $[\lambda_{\alpha_i}|i]_t$, $[\lambda_{\beta_i}|i]_t$ with the wires $\{\alpha_{i,1}, ..., \alpha_{i,l}\}$, $\{\beta_{i,1}, ..., \beta_{i,l}\}$.
3. In online phase, P_1 sends $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to parties, then parties can locally compute $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k}$, $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$. And all parties can locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} + [\lambda_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t = [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\lambda_{\beta_i}|i]_t$ for all $i \in \{1, ..., k\}$.

If we view $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$ as a degree-t Shamir sharing, we can compute the degree-t Shamir sharings of input values $[v_{\beta_i}|i]_t$. And recall that for degree-t Shamir secret sharing of honest majority setting, the whole sharing is fully determined by the shares of honest. In the following, we will show this is sufficient to verify the correctness of the computation.

A. Offline Phase

In the offline phase, different from the semi-honest offline phase, for each packed Beaver triple $([\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}, [\![\boldsymbol{\lambda_\beta}]\!]_{n-k}, [\![\boldsymbol{\lambda_\alpha} * \boldsymbol{\lambda_\beta}]\!]_{n-k})$, parties will also take $\{([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t)\}_{i=1}^k$ as output. And for each group of input gates or output gates, all parties will prepare a set of random degree-t Shamir sharing $\{[r_i|i]_t\}_{i=1}^k$. In the output phase, these degree-t Shamir sharings can help Client to detect attacks launched by corrupted parties. The functionality for the offline phase with malicious security appears in Functionality 6.

In the case of malicious behavior, it is also necessary to ensure that errors caused by malicious parties during network routing can be detected. We need check whether the λ_i in

the i-slot of $[\phi(\boldsymbol{\lambda})]_t$ is equal to λ_j in the j-th slot of $[\phi(\boldsymbol{\lambda}')]_t$ (assume λ_i in i-th slot of $\boldsymbol{\lambda}$ and λ_i is routed to j-th slot of $\boldsymbol{\lambda}'$). We follow the idea of [28] to verify the correctness of network routing. We utilizing the degree-t Shamir secret sharing over $\mathbb{F}_{2^l}$. A degree-t Shamir secret sharing $[\phi(\boldsymbol{\lambda})]_t$ over $\mathbb{F}_{2^l}$ can be locally converted into a threshold-t additive secret sharing $\langle\phi(\boldsymbol{\lambda})\rangle$. Each bit of $\langle\phi(\boldsymbol{\lambda})\rangle$ forms a threshold-t additive secret sharing, employing XOR operations over $\mathbb{F}_2$. Additionally, any group of $t+1$ parties can consistently reconstruct the same result. Thus, the presence of $t+1$ honest parties ensures complete reconstruction of these sharings, allowing them to detect any errors introduced by malicious parties of corresponding slot. Any group of $t+1$ parties can reconstruct the secret. If the secret reconstructed by a group of $t+1$ honest parties differs from that reconstructed by a group mixed with honest and malicious parties, or if the pair of secrets being checked are not equal, then an attack introduced by the malicious party is detected. Note that due to the properties of RMFE, we have $\psi(\langle\phi(\boldsymbol{\lambda})\rangle \cdot \phi(\mathbf{1})) = \langle\psi(\phi(\boldsymbol{\lambda}) \cdot \phi(\mathbf{1}))\rangle = \langle\boldsymbol{\lambda}\rangle$. To check multiple pairs of tuples at once, all parties construct a list $L(i,j)$ for all $i,j \in \{1,...,l\}$ that containing all wire tuple whose last two entries are i,j. To check the correctness of the wire tuples in $L(i,j)$, all parties compute a random linear combination of the tuples in $L(i,j)$ and obtain two sharings $([x]_t,[y]_t)$. If all the wire tuples are correct, we should have $\psi(x \cdot \phi((1)))_i = \psi(y \cdot \phi((1)))_i$ where the subscripts i and j to denote the i-th and j-th components of the corresponding vector. Let $\mathbb{K}$ is a large enough extension field and $|\mathbb{K}| \geq 2^\kappa$, the description of the functionality $\mathcal{F}_{verify\text{-}routing}$ appears in Functionality 5, the protocol $\Pi_{verify\text{-}routing}$ that realizes $\mathcal{F}_{verify\text{-}routing}$ appears in Protocol 8. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties, $\mathcal{A}$ denote the adversary, $[x]_t^{\mathcal{H}}$ be the shares that fully determined by $\mathcal{H}$, which is parties should correctly hold.

Lemma IX.1. *Protocol $\Pi_{verify\text{-}routing}$ securely computes $\mathcal{F}_{verify\text{-}routing}$ in the $(\mathcal{F}_{rand},\mathcal{F}_{coin})$-hybrid model against a fully malicious adversary who controls $t=(n-1)/2$ parties with overwhelming probability.*

Proof. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties, $\mathcal{A}$ denote the adversary.

Simulation for Network. In the beginning, for all $\eta \in \{1,...,N\}$, $\mathcal{S}$ receives from $\mathcal{F}_{verify\text{-}routing}$ the shares of $[x^{(\eta)}]_t^{\mathcal{H}}$, $[y^{(\eta)}]^{\mathcal{H}}$ held by corrupted parties, the additive errors $\boldsymbol{\Delta}_x^{(\eta)}$, $\boldsymbol{\Delta}_y^{(\eta)}$ to the shares of $[x^{(\eta)}]_t$, $[y^{(\eta)}]_t$ of parties in $Corr$, and the additive error $d^{(\eta)} = v_{j_\eta}^{(\eta)} - u_{i_\eta}^{(\eta)}$ where $u_{j_\eta}^{(\eta)} = \psi(x^{(\eta)} \cdot \phi(\mathbf{1}))_{i_\eta}$ and $v_{j_\eta}^{(\eta)} = \psi(y^{(\eta)} \cdot \phi(\mathbf{1}))_{j_\eta}$.

1. In Step 1, for all $\eta \in \{1,...,N\}$, $\mathcal{S}$ learns the shares of $[a^{(\eta)}]_t^{\mathcal{H}}$, $[b^{(\eta)}]_t^{\mathcal{H}}$, held by corrupted parties, the additive errors $\boldsymbol{\Delta}_a^{(\eta)}$, $\boldsymbol{\Delta}_b^{(\eta)}$ to the shares of $[a^{(\eta)}]_t$, $[b^{(\eta)}]_t$ of parties in $Corr$, and the additive error $\tilde{d}^{(\eta)} = b_{j_\eta}^{(\eta)} - a_{i_\eta}^{(\eta)}$.
2. In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{rand}$ when preparing a random wire tuple $([a^{(0)}]_t,[b^{(0)}]_t,i_0,j_0)$. $\mathcal{S}$ receives the shares of $[a^{(0)}]_t,[b^{(0)}]_t$ held by corrupted parties. Note that $[a^{(0)}]_t,[b^{(0)}]_t$ are guaranteed to be consistent, and $a_{i_0}^{(0)} = b_{j_0}^{(0)}$. Therefore, $\boldsymbol{\Delta}_a^{(0)} = \boldsymbol{\Delta}_b^{(0)} = 0$ and $\tilde{d}^{(0)} = 0$.
3. In Step 3, $\mathcal{S}$ emulates $\mathcal{F}_{coin}$ and faithfully samples $r \in \mathbb{K}$.
4. Note $\boldsymbol{\Delta}_a^{\eta}$, $\boldsymbol{\Delta}_b^{\eta}$ is a Shamir secret sharing, in Step 4, $\mathcal{S}$ transform $\boldsymbol{\Delta}_a^{\eta}$, $\boldsymbol{\Delta}_b^{\eta}$ to additive secret sharing $\langle\Delta_a^{\eta}\rangle$, $\langle\Delta_b^{\eta}\rangle$, and computes $\langle\Delta_a\rangle = \sum_{\eta=0}^{N} r^\eta \cdot \psi(\langle\Delta_a^{(\eta)}\rangle \cdot \phi(\mathbf{1}))_{i_\eta}$, $\langle\Delta_b\rangle = \sum_{\eta=0}^{N} r^\eta \cdot \psi(\langle\Delta_b^{(\eta)}\rangle \cdot \phi(\mathbf{1}))_{j_\eta}$, and $\tilde{d} = \sum_{i=0}^{N} r^i \cdot \tilde{d}^{(i)}$. $\mathcal{S}$ also computes the shares of $\langle\alpha\rangle$, $\langle\beta\rangle$ held by the corrupted parties within all groups of $t+1$ parties. Then, $\mathcal{S}$ samples a random element $\mathbb{K}$ as α. Based on the secret α, the shares of $\langle\alpha\rangle$ held by corrupted parties, and the additive errors $\langle\Delta_a\rangle$ to the shares of parties in $Corr$, $\mathcal{S}$, reconstructs the whole sharing $\langle\alpha\rangle$. For $\langle\beta\rangle$, $\mathcal{S}$ sets $\beta = \alpha + \tilde{d}$. Based on the secrets β, the shares of $\langle\beta\rangle$ held by corrupted parties, and the additive errors $\langle\Delta_b\rangle$ to the shares of parties in $Corr$, $\mathcal{S}$ reconstructs the whole sharing $\langle\beta\rangle$.
5. In Step 5, $\mathcal{S}$ honestly follows the protocol.
6. Finally, if an honest party aborts, $\mathcal{S}$ sends abort to $\mathcal{F}_{verify\text{-}routing}$. Otherwise, $\mathcal{S}$ sends continue to $\mathcal{F}_{verify\text{-}routing}$.

We first show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. Note that honest parties need to send messages to corrupted parties only in step 5. Relying on the values received form $\mathcal{F}_{verify\text{-}routing}$ in the beginning and the values received from the adversary when emulating $\mathcal{F}_{rand}$, $\mathcal{S}$ can compute the shares of $\langle\alpha\rangle$, $\langle\beta\rangle$ held by corrupted parties and the additive errors to the shares of parties in $Corr$. Furthermore, $\mathcal{S}$ can compute the additive error $\tilde{d} = \beta - \alpha$.

Functionality 5: $\mathcal{F}_{verify\text{-}routing}$

1. Let N denote the number of wire tuples all parties want to verify. For all $\eta \in \{1,...,N\}$, $\mathcal{F}_{verify\text{-}routing}$ receives from honest parties their shares of $[a^{(\eta)}]_t,[b^{(\eta)}]_t$ and two positions $i_\eta,j_\eta \in \{1,...,l\}$. The η-th tuple is stored $([a^{(\eta)}]_t,[b^{(\eta)}]_t,i_\eta,j_\eta)$.
2. For all $\eta \in \{1,...,N\}$, $\mathcal{F}_{verify\text{-}routing}$ reconstructs the whole sharings $[a^{(\eta)}]_t^H,[a^{(\eta)}]_t,[b^{(\eta)}]_t^H,[b^{(\eta)}]_t$ using the shares of honest parties and computes $\boldsymbol{\Delta}_a^{(\eta)} = [a^{(\eta)}]_t - [a^{(\eta)}]_t^H$ and $\boldsymbol{\Delta}_b^{(\eta)} = [b^{(\eta)}]_t - [b^{(\eta)}]_t^H$. Then $\mathcal{F}_{verify\text{-}routing}$ sends to the adversary the shares of $[a^{(\eta)}]_t^H$, $[b^{(\eta)}]_t^H$ of corrupted parties and the vectors $\boldsymbol{\Delta}_a^{(\eta)}$, $\boldsymbol{\Delta}_b^{(\eta)}$.
3. For all $\eta \in \{1,...,N\}$, $\mathcal{F}_{verify\text{-}routing}$ reconstructs $\boldsymbol{u}^{(\eta)} = \psi(a^{(\eta)} \cdot \phi(\mathbf{1}))$, $\boldsymbol{v}^{(\eta)} = \psi(b^{(\eta)} \cdot \phi(\mathbf{1}))$ and computes $d^{(\eta)} = v_{j_\eta}^{(\eta)} - u_{i_\eta}^{(\eta)}$. Then $\mathcal{F}_{verify\text{-}routing}$ sends $d^{(\eta)}$ to the adversary.
4. Let $b \in \{abort, accept\}$ denote whether there exists $\eta \in \{1,...,N\}$ such that $d^{(\eta)} \neq 0$. $\mathcal{F}_{verify\text{-}routing}$ sends b to the adversary and waits for its response.
   - If the adversary replies continue, $\mathcal{F}_{verify\text{-}routing}$ sends b to honest parties.
   - If the adversary replies abort, $\mathcal{F}_{verify\text{-}routing}$ sends abort to honest parties.

Protocol 8: $\Pi_{verify\text{-}routing}$

1. Let N denote the number of wire tuples all parties want to verify. The wire tuples are denoted by
$$([a^{(1)}]_t, [b^{(1)}]_t, i_1, j_1), ([a^{(2)}]_t, [b^{(2)}]_t, i_2, j_2), ..., ([a^{(N)}]_t, [b^{(N)}]_t, i_N, j_N)$$
2. All parties invoke $\mathcal{F}_{rand}$ to prepare a random wire tuple $([a^{(0)}]_t, [b^{(0)}]_t, i_0, j_0)$ and and it satisfies the RMFE mapping requirements.
3. All parties invoke $\mathcal{F}_{coin}$ to generate a random element $r \in \mathbb{K}$.
4. All parties locally transform $[a^{(z)}]_t$, $[b^{(z)}]_t$ to additive secret sharing $\langle a^{(z)} \rangle$, $\langle b^{(z)} \rangle$ for all $z \in \{1, ..., N\}$ and compute the following two sharings:
$$\langle \alpha \rangle = \sum_{\eta=0}^{N} r^{\eta} \cdot \psi(\langle a^{(\eta)} \rangle \cdot \phi(\mathbf{1}))_{i_\eta}$$
$$\langle \beta \rangle = \sum_{\eta=0}^{N} r^{\eta} \cdot \psi(\langle b^{(\eta)} \rangle \cdot \phi(\mathbf{1}))_{j_\eta}$$
5. All parties send their shares of $\langle \alpha \rangle$, $\langle \beta \rangle$ to other parties. Then each party checks whether $\langle \alpha \rangle$, $\langle \beta \rangle$ are consistent and $\alpha = \beta$. If not, this party aborts.

To determine the whole sharings $\langle \alpha \rangle$, $\langle \beta \rangle$, $\mathcal{S}$ only needs to know of α, β. In the real world, α, β are masked by $a^{(0)}$, $b^{(0)}$, which are random subject to $a^{(0)}_{i_0} = b^{(0)}_{j_0}$ guaranteed by $\mathcal{F}_{rand}$. Therefore, α is a uniform element, $\beta = \alpha + \tilde{d}$. In the ideal world, $\mathcal{S}$ randomly samples α and sets $\beta = \alpha + \tilde{d}$, which have the same distribution as that in the real world. Therefore, S perfectly simulates the behaviors of honest parties.

Consider the distributing of the output, the only difference is when there exists a wire tuple that $([a^{(\eta)}]_t, [b^{(\eta)}]_t, i_{(\eta)}, j_{(\eta)})$ such that $\psi(a^{(\eta)} \cdot \phi(\mathbf{1}))_{(i_\eta)} \neq \psi(b^{(\eta)} \cdot \phi(\mathbf{1}))_{(j_\eta)}$ but $\alpha = \beta$. In the ideal world, all parties will finally abort, while in the real world, all parties will continue. We show that this happens with negligible probability. Note if there exists $\eta \in \{1, ..., N\}$ such that $\tilde{d}^{(\eta)} \neq 0$, then with overwhelming probability $\tilde{d} \neq 0$. Since we will get a degree-N polynomial about r, $\delta(r) = \tilde{d}^{(0)} + \tilde{d}^{(1)} \cdot r + ... + \tilde{d}^{(N)} \cdot r^N$, r is randomly sampled by $\mathcal{F}_{coin}$ in $\mathbb{K}$, the probability that this polynomial equal to zero is bounded by $N/|\mathbb{K}| \leq N/2^{\kappa}$, which is negligible. Therefore, with overwhelming probability, $\tilde{d} \neq 0$.

Now we will describe the protocol $\Pi_{offline\text{-}mal}$ that realizes $\mathcal{F}_{offline\text{-}mal}$. It follows the semi-honest protocol $\Pi_{offline}$ and we use the ideal functionality $\mathcal{F}_{single\text{-}mult\text{-}mal}$ from [9], [12], use the ideal functionality $\mathcal{F}_{random}$ from [21]. The ideal functionality $\mathcal{F}_{single\text{-}mult\text{-}mal}$ allows an additive attack towards the multiplication result, and can be instantiated by the protocol in [9]. we refer the readers to [9], [21] for more details. The amortized communication complexity of preparing a random degree-t sharing for l binary values with malicious security is $2n$ elements. The protocol $\Pi_{offline\text{-}mal}$ appears in Protocol 9.

The communication complexity of $\Pi_{offline\text{-}mal}$:

- For each gate whose input wires require network routing, the communication complexity is $15n$ each input wire.
- For each AND gate, the amortized communication complexity is $(2n + 8n + 2n/k) * m/l \approx 30n + 24$ bits.
- For each input gate, the amortized communication complexity is $(2n+2n+3n/k)*m/l \approx 12n+36$ bits, for each output gate, the amortized communication complexity is $12n + 24$ bits.

Lemma IX.2. *Protocol $\Pi_{offline\text{-}mal}$ securely computes $\mathcal{F}_{offline\text{-}mal}$ in the $\mathcal{F}_{single\text{-}mult\text{-}mal}$-hybrid model against a fully malicious adversary who controls $t = (n-1)/2$ parties.*

Proof. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The simulator $\mathcal{S}$ works as follows.

1. In step 1, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{random}$ to prepare the random sharing in the form of $[\![\mathbf{0}]\!]_{n-1}$ and receives the shares of $[\![\mathbf{0}]\!]_{n-1}$ of corrupted parties. $\mathcal{S}$ sends the shares of corrupted parties to $\mathcal{F}_{offline\text{-}mal}$.
2. In Step 2, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{random}$ to prepare the random sharings in the form of $[\phi(\boldsymbol{r_i})|i]_t$ and receives the shares of $[\phi(\boldsymbol{r_i})|i]_t$ of corrupted parties. $\mathcal{S}$ sends the shares of corrupted parties to $\mathcal{F}_{offline\text{-}mal}$.
3. In Step 3,
   a. For each group of $k \cdot l$ output wires $\boldsymbol{\alpha}$ of input gates and AND gates, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{random}$ to prepare the random sharings in the form of $[\phi(\boldsymbol{r_i})|i]_t$ as described above, and receives the shares of $[\phi(\boldsymbol{r_i})|i]_t$ of corrupted parties. Then $\mathcal{S}$ uses the zero sharing generated in Step 2 and follows the same step as the protocol to compute $[\boldsymbol{\lambda_\gamma}]_{n-1}$.
   b. For each group of $k \cdot l$ XOR gate with the input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{S}$ follows the same step as the protocol to get $[\lambda_{\alpha_i}|i]_t$, $[\lambda_{\beta_i}|i]_t$, $[\lambda_{\gamma_i}|i]_t$.
   c. For each group of $k \cdot l$ AND gates, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{single\text{-}mult\text{-}mal}$ and receives from the adversary the shares of $[\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$ of corrupted parties and the additive error δ_i. Then for each pair of $([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t)$, $\mathcal{S}$ sends the shares of $[[\lambda_{\alpha_i} \cdot$

Functionality 6: $\mathcal{F}_{offline\text{-}mal}$

1. **Preparing Degree-$(n-1)$ Zero Sharings:** For each group of $l \cdot k$ AND gates, input gates, and output gates:
   a. $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_j\}_{j\in Corr}$. Then $\mathcal{F}_{offline\text{-}mal}$ prepares a random degree-$(n-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}_{2^m}^k$, denote by $[\![\mathbf{0}]\!]_{n-1}$ such that for all $P_j \in Corr$, the j-th share of $[\![\mathbf{0}]\!]_{n-1}$ is u_j.
   b. $\mathcal{F}_{offline\text{-}mal}$ distributes the shares of $[\![\mathbf{0}]\!]_{n-1}$ to honest parties.
2. **Preparing Shamir sharings for Input and Output Gates:** For each group of $k \cdot l$ input gates or output gates:
   a. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_{i,j}\}_{j\in Corr}$. $\mathcal{F}_{offline\text{-}mal}$ samples a random value in the form of $\phi(\boldsymbol{r_i})$. Then $\mathcal{F}_{offline\text{-}mal}$ computes a degree-t Shamir shaings $[\phi(\boldsymbol{r_i})|i]_t$ such that for all $P_j \in Corr$, the j-th shares of $[\phi(\boldsymbol{r_i})|i]_t$ is $u_{i,j}$.
   b. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{offline\text{-}mal}$ distributes the shares of $[\phi(\boldsymbol{r_i})|i]_t$ to honest parties.
3. **Assigning Random Sharings for Each Group of Wires:**
   a. For each group of $k \cdot l$ output wires $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ of input gates and AND gates, $\mathcal{F}_{offline\text{-}mal}$ samples a random values $\{r_{i,j}^{(2)}\}_{i\in\mathcal{K}}^{j\in\mathcal{L}}$, associates these random values with the wires $\boldsymbol{\alpha}$ as $\lambda_{\alpha_{i,j}}$. Let $\lambda_{\alpha_i} = \phi(\boldsymbol{r_i})$ for all $i \in \mathcal{K}$, and $\boldsymbol{\lambda_\alpha} = (\lambda_{\alpha_1}, ..., \lambda_{\alpha_k})$. $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_j\}_{j\in Corr}$ and a vector $\boldsymbol{\delta}$. $\mathcal{F}_{offline\text{-}mal}$ samples a random degree-$(n-1)$ packed Shamir sharings $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{\delta}]\!]_{n-1}$ such that for all $P_j \in Corr$, the j-th share of $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{\delta}]\!]_{n-1}$ is u_j. Then, $\mathcal{F}_{offline\text{-}mal}$ distributes the shares of $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{\delta}]\!]_{n-1}$ to honest parties.
   b. Starting from the first layer of C to the last layer, for XOR gates, for each group of $k \cdot l$ XOR gates with $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, the random values assigned to the input wires $\alpha_{i,j}$, $\beta_{i,j}$, denoted as $\lambda_{\alpha_{i,j}}^{(2)}$ and $\lambda_{\beta_{i,j}}^{(2)}$, are inherited from the output wires of gates in the preceding layer of the circuit. Let $\lambda_{\alpha_i} = \phi(\lambda_{\alpha_{i,1}}^{(2)}, ..., \lambda_{\alpha_{i,1}}^{(2)})$ and so as λ_{β_i}, $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_{1,j}^i, u_{2,j}^i\}_{j\in Corr}$. Based on the $\{u_{1,j}^i, u_{2,j}^i\}_{j\in Corr}$ and $m\lambda_{\alpha_i}$, λ_{β_i}, $\mathcal{F}_{offline\text{-}mal}$ computes $[\lambda_{\alpha_i}|i]_t$, $[\lambda_{\beta_i}|i]_t$ such that the j-th shares of $[\lambda_{\alpha_i}|i]_t$, $[\boldsymbol{\lambda_{\beta_i}}|i]_t$ are $u_{1,j}^i, u_{2,j}^i$. $\mathcal{F}_{offline\text{-}mal}$ sets $[\lambda_{\gamma_i}|i]_t = [\lambda_{\alpha_i}|i]_t + [\boldsymbol{\lambda_{\beta_i}}|i]_t$.
   c. Starting from the first layer of C to the last layer, for AND gates, for each group of $k \cdot l$ XOR gates with $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, the random values assigned to the input wires $\alpha_{i,j}$, $\beta_{i,j}$, denoted as $\lambda_{\alpha_{i,j}}^{(2)}$ and $\lambda_{\beta_{i,j}}^{(2)}$, are inherited from the output wires of gates in the preceding layer of the circuit. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_{1,j}^i, u_{2,j}^i, u_{3,j}^i\}_{j\in Corr}$ and an additive δ_i. $\mathcal{F}_{offline\text{-}mal}$ computes $c_i = \lambda_{\alpha_i} \cdot \lambda_{\beta_i} + \delta_i$. Then $\mathcal{F}_{offline\text{-}mal}$ computes three degree-t Shamir sharings $[\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [c_i|i]_t$, such that for all $P_j \in Corr$, the j-th share of $[\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [c_i|i]_t$ is $\{u_{1,j}^i, u_{2,j}^i, u_{3,j}^i\}_{j\in Corr}$. For all $i \in \{1, .., k\}$, $\mathcal{F}_{offline\text{-}mal}$ dsitrbutes the shares of $[\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t, [c_i|i]_t$ to honest parties.
   d. For each group of output gates with the input wires $\boldsymbol{\alpha}$, $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a set of shares $\{u_{i,j}\}_{j\in Corr}$. $\mathcal{F}_{offline\text{-}mal}$ samples k random elements in the form of $\{\phi(\boldsymbol{r_i'})\}_{i=1}^k$. Based on the $\{u_{i,j}\}_{j\in Corr}$ and $\{\phi(\boldsymbol{r_i'})\}_{i=1}^k$, $\mathcal{F}_{offline\text{-}mal}$ computes the degree-t sharings $[\phi(\boldsymbol{r_i'})|i]_t$ where j-share of $[\phi(\boldsymbol{r_i'})|i]_t$ is $u_{i,j}$ for all $i \in \{1, ..., k\}$. Let $[\lambda_{\alpha_i}|i]_t = [\phi(\boldsymbol{r_i})|i]_t$ and $\boldsymbol{\lambda_\alpha} = (\lambda_{\alpha_i}, ..., \lambda_{\alpha_k})$, $\{[\phi(\boldsymbol{r_i})|i]_t\}_{i=1}^k$ are the random sharings that prepared in Step 3. $\mathcal{F}_{offline\text{-}mal}$ receives from the adversary a vector $\boldsymbol{\delta} = (\delta_1, ..., \delta_k)$. Then $\mathcal{F}_{offline\text{-}mal}$ computes $\boldsymbol{\lambda_\alpha} + \boldsymbol{r} + \boldsymbol{\delta}$ and sends these values to P_1.

Fig. 12. Functionality for offline phase with malicious security

Functionality 7: $\mathcal{F}_{single\text{-}mult\text{-}mal}$

1. $\mathcal{F}_{single\text{-}mult\text{-}mal}$ receives the secret position i from all parties. Let $[x|i]_t$,$[y|i]_t$ denote the input sharings. $\mathcal{F}_{single\text{-}mult\text{-}mal}$ receives from honest parties their shares of $[x|i]_t$,$[y|i]_t$. Then $\mathcal{F}_{single\text{-}mult\text{-}mal}$ reconstructs the secrets x, y. $\mathcal{F}_{single\text{-}mult\text{-}mal}$ further computes the shares of $[x|i]_t, [y|i]_t$ held by corrupted parties, and sends these shares to the adversary.
2. $\mathcal{F}_{single\text{-}mult\text{-}mal}$ receives from the adversary a set of shares $\{z_i\}_{i\in Corr}$ and an additive error δ.
3. $\mathcal{F}_{single\text{-}mult\text{-}mal}$ computes $x \cdot y + \delta$. Based on the secret $z = x \cdot y + \delta$ and the t shares $\{z_i\}_{i\in Corr}$, $\mathcal{F}_{single\text{-}mult\text{-}mal}$ reconstructs the whole sharing $[z|i]_t$ and distributes the shares of $[z|i]_t$ to honest parties.

Fig. 13. Functionality for single element multiplication with malicious security

$\lambda_{\beta_i}|i]_t$ of corrupted parties and the additive error δ_i to $\mathcal{F}_{offline\text{-}mal}$.

d. For each group of $k \cdot l$ output gates with the input wires $\boldsymbol{\alpha}$, $\mathcal{S}$ follows the protocol and computes $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}+]\!]_{n-1}$ held by corrupted parties.

– If P_1 is honest, $\mathcal{S}$ receives from the adversary the shares of $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1}$ of corrupted parties, which can be different from those computed by $\mathcal{S}$. Let $[\![\overline{\boldsymbol{\lambda_\alpha} + \boldsymbol{r}}]\!]_{n-1}$ denote the degree-$(n-1)$ packed Shamir sharing where the shares of corrupted parties are those computed by $\mathcal{S}$. $\mathcal{S}$ locally compute the shares of

$$[\![\boldsymbol{\delta}]\!]_{n-1} = [\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1} - [\![\overline{\boldsymbol{\lambda_\alpha} + \boldsymbol{r}}]\!]_{n-1}$$

of corrupted parties and sets the shares of honest parties to be 0. Then, $\mathcal{S}$ reconstructs the secrets $\boldsymbol{\delta}$ and sends to $\mathcal{F}_{offline\text{-}mal}$.

– If P_1 is corrupted, $\mathcal{S}$ sets $\boldsymbol{\delta} = \mathbf{0} \in \mathbb{F}_{2^m}^k$. Then, $\mathcal{S}$ sends to $\mathcal{F}_{offline\text{-}mal}$ the vector of additive errors and receives $\boldsymbol{\lambda_\alpha} + \boldsymbol{r}$. $\mathcal{S}$ generates a random degree-$(n-1)$ packed secret sharing $[\overline{\boldsymbol{\lambda_\alpha} + \boldsymbol{r}}]_{n-1}$ based on the $\boldsymbol{\lambda_\alpha} + \boldsymbol{r}$ and the shares of corrupted parties computed by $\mathcal{S}$. Finally, $\mathcal{S}$ sends the shares of $[\![\overline{\boldsymbol{\lambda_\alpha} + \boldsymbol{r}}]\!]_{n-1}$ of honest parties to P_1.

This completes the description of $\mathcal{S}$.

Protocol 9: $\Pi_{offline\text{-}mal}$

1. **Preparing Degree-$(n-1)$ Zero Sharings:**
   a. For each group of AND gates, all parties invoke $\mathcal{F}_{random}$ to prepare random sharings in the form of $[\![\mathbf{0}]\!]_{n-1}$ where $\mathbf{0} = (0, 0, ..., 0) \in \mathbb{F}_{2^m}^k$.
   b. For each group of $k \cdot l$ input gates or output gates, all parties invoke $\mathcal{F}_{random}$ to prepare random sharings in the form of $[\![\mathbf{0}]\!]_{n-1}$.
2. **Preparing Random Sharing for Input and Output Gates:** For each group of $k \cdot l$ input gates or output gates, parties invoke $\mathcal{F}_{random}$ to prepare k random sharings in the form of $[\phi(\boldsymbol{r_i})|i]_t$ for all $i \in \{1, ..., k\}$.
3. **Assigning Random Sharing for Each Wires:**
   a. For each group of $k \cdot l$ output wires $\boldsymbol{\alpha}$ of input gates and AND gates, all parties invoke $\mathcal{F}_{random}$ to prepare random sharings in the form of $[\phi(\boldsymbol{r_i})|i]_t$ for all $i \in \{1, ..., k\}$. Let $e_i \in \mathbb{F}^k$ be the i-th unit vector, all parties locally transform e_i to the degree-$(k-1)$ packed Shamir sharing $[\![\boldsymbol{e_i}]\!]_{k-1}$ and use the $[\![\mathbf{0}]\!]_{n-1}$ which generated by Step 2 to compute:

$$[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!]_{k-1} * [\![\phi(\boldsymbol{r_i})|i]\!]_t + [\![\mathbf{0}]\!]_{n-1}$$

   b. Starting from the first layer of C to the last layer, for all group of gates requiring routing, all parties invoke $\Pi_{routing}$ with the input of corresponding random values $[\phi(\boldsymbol{\lambda_w})|s]_t$ from previous layer's output wires. All parties get the random wire values $[\phi(\boldsymbol{\lambda'_w})|s']_t$ of this layer corresponding wires $\boldsymbol{w'}$.
      i. For each group of $l \cdot k$ XOR gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties set the random values of output wires $[\lambda_\gamma|i]_t = [\lambda_{\alpha_i}|i]_t + [\lambda_{\beta_i}|i]_t$ and set $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\alpha_i}|i]_t$, $[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!] * [\lambda_{\beta_i}|i]_t$. Parties associate $\{[\lambda_\alpha|i], [\lambda_\alpha|i]\}_{i=1}^k$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
      ii. For each group of $l \cdot k$ AND gates with the input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties invoke $\mathcal{F}_{single\text{-}mult\text{-}mal}$ with the input of $[\lambda_{\alpha_i}|i]$, $[\lambda_{\beta_i}|i]_t$ to compute $[\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$ for all $i \in \mathcal{K}$. Parties associate $\{[\lambda_\alpha|i], [\lambda_\alpha|i]\}_{i=1}^k$ with the wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$.
   c. For each group of output gates with input wires $\boldsymbol{\alpha}$, all parties invoke $\mathcal{F}_{random}$ to generate the random sharings in the form of $[\phi(\boldsymbol{r'_i})|i]_t$ and let $[\lambda_{\alpha_i}|i]_t = [\phi(\boldsymbol{r'_i})|i]_t$ for all $i \in \mathcal{K}$. Then parties use the random sharings $\{[\phi(\boldsymbol{r_i})|i]_t\}_{i=1}^k$ that generated by Step 3 and random zero sharing that generated in Step 2, locally compute $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1} = \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!]_{k-1} * [\phi(\boldsymbol{r_i})|i]_t + \sum_{i=1}^{k} [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\alpha_i}|i]_t + [\![\mathbf{0}]\!]_{n-1}$ and send the shares to P_1. P_1 reconstructs $\boldsymbol{\lambda_\alpha} + \boldsymbol{r}$.

Fig. 14. Protocol for offline phase with malicious security

Now we will show that $\mathcal{S}$ perfectly simulate the behaviors of honest parties.

1. In Step 1, $\mathcal{S}$ emulates $\mathcal{F}_{random}$ to generate the random sharings in the form of $[\![\mathbf{0}]\!]_{n-1}$. Note a degree-$(n-1)$ packed secret sharing requires n values to reconstruct the whole sharing, and given the shares of corrupted parties, the sharings are still random. As same described in Step 1, the sharings have the same distribution in both worlds.
2. In Step 2, as described above, the sharings have the same distribution in both worlds.
3. In Step 3.a, $\mathcal{S}$ emulates $\mathcal{F}_{random}$ to generate degree-t random sharings and uses a random degree-$(n-1)$ zero sharing, as described above, the sharings have the same distribution in both worlds. In Step 3.b, the degree-t Shamir sharing has the same distribution in both worlds as described above. In Step 3.c, $\mathcal{S}$ emulates $\mathcal{F}_{single\text{-}mult\text{-}mal}$ to generate $[\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$. The inputs of $\mathcal{S}$ is random elements as these in real world. Therefore, the distribution of $([\lambda_{\alpha_i}|i]_t, [\lambda_{\beta_i}|i]_t), [\lambda_{\alpha_i} \cdot \lambda_{\beta_i}|i]_t$ is identical in both worlds. In Step 3.d, if P_1 is honest, because P_1 collects and reconstruct shares, adversary will not get any new information from P_1, and the shares of corrupted parties are degree-t Shamir sharing and random zero sharing, as described above, the distribution of shares of corrupted parties is same in both world. If P_1 is corrupted, in the real world, the degree-$(n-1)$ sharings which P_1 received are masked by degree-$(n-1)$ random zero sharing, thus the $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1}$ is uniformly random in real world. And in the ideal world, the shares of $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1}$ are also uniformly. Thus the distribution of the shares of $[\![\boldsymbol{\lambda_\alpha} + \boldsymbol{r}]\!]_{n-1}$ is same in both worlds.

B. Online Phase

In the online phase, we will compute degree-t Shamir sharings for each l input wires of AND gates. Recall that in honest majority setting, the degree-t Shamir sharing is totally decided by honest parties. By using these degree-t Shamir sharings, parties can verify the correctness of the secret. Recall that in our semi-honest online protocol, for each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, P_1 distributes $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to all parties, and parties can compute $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ by using these two sharings.

Input Phase. Note in $\mathcal{F}_{offline\text{-}mal}$, for each group of input gates with the input wires $\boldsymbol{\alpha}$ that belong to some Client, parties hold $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$, $\{[r_i|i]_t\}_{i=1}^k$. We use $\{v_{\alpha_{i,j}}^{(2)}\}_{i \in \mathcal{K}, j \in \mathcal{L}}$ to denote the input values of Client, let $v_{\alpha_i} = \phi(v_{\alpha_{i,1}}^{(2)}, ..., v_{\alpha_{i,l}}^{(2)})$ and $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$. First, All parties send their shares of $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$, $\{[r_i|i]_t\}_{i=1}^k$ to Client. Client reconstruct $\boldsymbol{r} = (r_1, ..., r_k)$, computes and distributes $[\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t$ to all parties. Because $[\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t$ is a degree-t packed secret sharing, honest

Protocol 10: $\Pi_{input\text{-}mal}$

1. For each group of input gates that belong to Client, let $\boldsymbol{\alpha} = \{\alpha_{i,j}\}_{i\in\mathcal{K},j\in\mathcal{L}}$ denote the batch of output wires of these input gates. All parties receive $\{[r_i|i]_t\}_{i=1}^k$ and $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$ from $\mathcal{F}_{offline}$ and Client holds $\boldsymbol{v_\alpha} = (v_{\alpha_1},...,v_{\alpha_k})$ where $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}},...,v^{(2)}_{\alpha_{i,l}})$ for each $i \in \{1,...,k\}$ and $v^{(2)}_{\alpha_{i,j}}$ is the input value of wire $\alpha_{i,j}$.
2. All parties send to Client their shares of $\{[r_i|i]_t\}_{i=1}^k$ and $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-1}$.
3. For all $i \in \{1,...,k\}$, Client checks whether the shares of $\{[r_i|i]_t\}_{i=1}^k$ lie on a same degree-t polynomial. If not, Client aborts.
4. Client reconstruct the secrets $\boldsymbol{r} = (r_1,...,r_k)$ and $\boldsymbol{\lambda_\alpha}$, then samples a random degree-t packed Shamir sharing $[\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_t$ and computes $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$.
5. Client distributes the shares of $[\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_t$ to all parties and sends $\boldsymbol{\mu_\alpha}$ to P_1.
6. For all $i \in \{1,...,k\}$, all parties locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_t - [r_i|i]_t$.

Fig. 15. Protocol for input phase with malicious security

parties can totally decide the secret, parties can locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_t - [r_i|i]_t$ for $i \in \{1,...,k\}$. Client also computes and sends $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ to P_1.

The description of $\Pi_{input\text{-}mal}$ appears in Protocol 10, the communication complexity of $\Pi_{input\text{-}mal}$ is $(n\cdot(k+1)+n+k)\cdot m/(k\cdot l) \approx 3n+27$ bits per input gate.

Computation Phase. For computing AND gates, We follow the approach of semi-honest protocol of Π_{mult} in general, but the difference is parties will locally compute $[v_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t$, and $[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$,$[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k}$. The correctness of the protocol follows the same argument as the semi-honest version. The description of $\Pi_{mult\text{-}mal}$ appears in Protocol 11. The communication complexity of $\Pi_{mult\text{-}mal}$ is $12\cdot m/l \approx 36$ bits per AND gate.

Output Phase and Validity Check. The main goal of output phase is send the output of the circuit to Client and verify the correctness of the computation. Recall that all parties have received $\{[r_i|i]_t\}_{i=1}^k$ in offline phase. And P_1 has received $\boldsymbol{\lambda_\alpha}+\boldsymbol{r}$ in offline phase, P_1 also holds $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$.

Now we will show how to achieve these goals:

P_1 locally compute $\boldsymbol{v_\alpha}+\boldsymbol{r} = \boldsymbol{\mu_\alpha}+(\boldsymbol{\lambda_\alpha}+\boldsymbol{r})$ and distributes the degree-$(k-1)$ packed sharing $[\![\boldsymbol{v_\alpha}+\boldsymbol{r}]\!]_{k-1}$ to all parties. In this way, all parties can locally compute the degree-t Shamir sharings for input wires of output gates. But before this, parties need to verify the correctness of the computation. The verification contains two parts:

- Parties need to verify the degree-$(k-1)$ packed Shamir sharings distributed by P_1 are valid. Parties need to check each degree-$(k-1)$ packed Shamir sharing which distributed by P_1 lie on a degree-$(k-1)$ polynomial.
- Then parties need to check that for each group of input wires of AND gate or output gate, the secret of the corresponding degree-t Shamir sharing is the correct wire value.

Then we will discuss how to check the shares lie on a degree-$(k-1)$ polynomial. We follow the approach of [12], use the ideal functionality $\mathcal{F}_{coin}$ to samples random field element to all parties, the instantiation of $\mathcal{F}_{coin}$ can be found in [26]. We also use the $\Pi_{consistency}$ from [12] to check the degree-$(k-1)$ packed sharings lie on a degree-$(k-1)$ polynomial. We refer the readers to [12], [26] for more details. The description of $\mathcal{F}_{coin}$ appears in Functionality 8, and the description of $\Pi_{consistency}$ appears in Protocol 12.

Let $\mathbb{K}$ be an extension field of $\mathbb{F}$ such that $|\mathbb{K}| \geq 2^\kappa$ where κ is the security. All parties use $\mathcal{F}_{coin}$ to generate a random field element $r \in \mathbb{K}$. Let $\{[\![\boldsymbol{w_i}]\!]_{k-1}\}_{i=1}^N$ denote all degree-$(k-1)$ packed Shamir sharings distributed by P_1. All parties locally compute: $[\![\boldsymbol{w}]\!]_{k-1} = \sum_{i=1}^N r^{i-1}\cdot[\![\boldsymbol{w_i}]\!]_{k-1}$.

Then each party collects the whole sharing $[\![\boldsymbol{w}]\!]_{k-1}$ and checks whether the shares form a valid degree-$(k-1)$ packed Shamir sharing. The description of $\Pi_{consistency}$ appears in Protocol 12. The communication complexity of $\Pi_{consistency}$ is $O(n^2)$ elements in $\mathbb{K}$, which is independent of the number of sharings.

Lemma IX.3. *If there exists $i \in \{1,...,N\}$ such that $[\![\boldsymbol{w_i}]\!]_{k-1}$ is not a vaild degree-$(k-1)$ packed Shamir sharing, then all honest parties abort in $\Pi_{consistency}$ with overwhelming probability. ($[\![\boldsymbol{w_i}]\!]_{k-1}$ is vaild means the shares of $[\![\boldsymbol{w_i}]\!]_{k-1}$ of*

Protocol 11: $\Pi_{mult\text{-}mal}$

1. For each group of AND gates with input wires $\boldsymbol{\alpha},\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties receive from $\mathcal{F}_{offline\text{-}mal}$
   - $\{([\lambda_{\alpha_i}|i]_t,[\lambda_{\beta_i}|i]_t,[\lambda_{\beta_i}\cdot\lambda_{\alpha_i}|i]_t)\}_{i=1}^k$
   - A random degree-$(n-1)$ packed Shamir sharing $[\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$
2. Note P_1 learns $\boldsymbol{\mu_\alpha},\boldsymbol{\mu_\beta}$ during the online phase. P_1 locally computes $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ distributes the shares of and $[\![\boldsymbol{\mu_\alpha}]\!]_{k-1}$, $[\![\boldsymbol{\mu_\beta}]\!]_{k-1}$ to all parties.
3. All parties locally compute

$$[\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\alpha_i}|i]_t$$

$$[\![\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\beta_i}|i]_t$$

$$[\![\boldsymbol{\lambda_\alpha}*\boldsymbol{\lambda_\beta}]\!]_{n-k} = \sum_{i=1}^k [\![\boldsymbol{e_i}]\!]_{k-1} * [\lambda_{\alpha_i}\cdot\lambda_{\beta_i}|i]_t$$

$$[\![\boldsymbol{\mu_\gamma}]\!]_{n-1} = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} * [\![\boldsymbol{\lambda_\beta}]\!]_{n-k} + [\![\boldsymbol{\mu_\beta}]\!]_{k-1} * [\![\boldsymbol{\lambda_\alpha}]\!]_{n-k} + [\![\boldsymbol{\lambda_\alpha}*\boldsymbol{\lambda_\beta}]\!]_{n-k} - [\![\boldsymbol{\lambda_\gamma}]\!]_{n-1}$$

4. P_1 collects the whole sharing $[\![\boldsymbol{\mu_\gamma}]\!]_{n-1}$ from all parties and reconstructs $\boldsymbol{\mu_\gamma}$ and computes $\psi(\boldsymbol{\mu_\gamma})$, rearranges the bits according to the next layer of circuits to get $\boldsymbol{\mu_\gamma^{(2)}}$, then sets $\boldsymbol{\mu_\gamma} = \phi(\boldsymbol{\mu_\gamma^{(2)}})$.
5. For all $i \in \{1,...,k\}$, all parties locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{\mu_\alpha}]\!]_{k-1} + [\lambda_{\alpha_i}|i]_t$ and $[v_{\beta_i}|i]_t = [\![\boldsymbol{\mu_\beta}]\!]_{k-1} + [\lambda_{\beta_i}|i]_t$.

Fig. 16. Protocol for evaluating AND gate by multiplication with malicious security

Functionality 8: $\mathcal{F}_{coin}$

1. $\mathcal{F}_{coin}$ samples a random field element r.
2. $\mathcal{F}_{coin}$ sends r to the adversary.
   - If the adversary replies continue, $\mathcal{F}_{coin}$ sends r to honest parties.
   - If the adversary replies abort, $\mathcal{F}_{coin}$ sends abort to honest parties.

Fig. 17. Functionality for generating random elements to all parties

Protocol 12: $\Pi_{consistency}$

1. Let $\{[\![\boldsymbol{w_i}]\!]_{k-1}\}_{i=1}^{N}$ denote all degree-$(k-1)$ packed secret sharings distributed by P_1.
2. All parties invoke $\mathcal{F}_{coin}$ to generate a random element $r \in \mathbb{F}$.
3. All parties locally compute
$$[\![\boldsymbol{w}]\!]_{k-1} = \sum_{i=1}^{N} r^{i-1} \cdot [\![\boldsymbol{w_i}]\!]_{k-1}.$$
4. Each party P_i sends its share of $[\![\boldsymbol{w}]\!]_{k-1}$ to all other parties. Then each party P_j checks whether the shares of $[\![\boldsymbol{w}]\!]_{k-1}$ lie on a degree-$(k-1)$ polynomial. If true, P_j acceptes the check. Otherwise, P_j aborts.

Fig. 18. Protocol for check the consistency

honest parties lie on a degree-$(k-1)$ polynomial) [12].

After check the validity of degree-$(k-1)$ packed secret sharing that distributed by P_1, all parties can locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_{k-1} - [r_i|i]_t$ for all $i \in \{1, ..., k\}$.

We describe the functionality $\mathcal{F}_{evaluate}$ in Functionality 9 from [12] for the evaluation of the circuit in the online phase. The protocol $\Pi_{evaluate}$ is the instantiation of $\mathcal{F}_{evaluate}$, appears in Protocol 13.

Lemma IX.4. *Protocol $\Pi_{evaluate}$ securely computes $\mathcal{F}_{evaluate}$ in the $\mathcal{F}_{offline\text{-}mal}$-hybrid model against a fully malicious adversary who controls t parties and up to c clients. [12]*

We will follow the approach of [12], [24] to verify the correctness of the secret, the functionality $\mathcal{F}_{verify}$ to check the correctness of the computation, the instantiation of $\mathcal{F}_{verify}$ can be found in [12]. The description of $\mathcal{F}_{verify}$ appears in Functionality 11.

C. Main Protocol with Malicious Security

Now we will introduce our main protocol $\Pi_{main\text{-}mal}$ with malicious seurity in Protocol 14. The ideal functionality $\mathcal{F}_{main\text{-}mal}$ appears in Functionality 10.

Functionality 9: $\mathcal{F}_{evaluate}$

1. $\mathcal{F}_{evaluate}$ receives the input from all clients. Let C denote the circuit.
2. $\mathcal{F}_{evaluate}$ receives the set of corrupted parties, denoted by $Corr$. For each group of input gates with output wire $\boldsymbol{\alpha}$, let $\boldsymbol{v_\alpha}$ denote the input values associated with $\boldsymbol{\alpha}$. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{evaluate}$ receives from the adversary a set of shares $\{u_{i,j}\}_{j \in Corr}$. Then $\mathcal{F}_{evaluate}$ computes a degree-t Shamir sharing $[v_{\alpha_i}|i]_t$ such that for all $P_j \in Corr$, the j-th share of $[v_{\alpha_i}|i]_t$ is $u_{i,j}$. Finally, $\mathcal{F}_{evaluate}$ distributes the shares of $[v_{\alpha_i}|i]_t$ to honest parties.
3. $\mathcal{F}_{evaluate}$ evaluates the circuit C layer by layer. For each group of addition gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{F}_{evaluate}$ computes $\boldsymbol{v_\gamma} = \boldsymbol{v_\alpha} + \boldsymbol{v_\beta}$. For each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$:
   a. $\mathcal{F}_{evaluate}$ receives two vectors of additive errors $\boldsymbol{\delta}(\boldsymbol{v_\alpha})$, $\boldsymbol{\delta}(\boldsymbol{v_\beta})$ from the adversary. Then $\mathcal{F}_{evaluate}$ sets $\boldsymbol{v_\alpha} = \boldsymbol{v_\alpha} + \boldsymbol{\delta}(\boldsymbol{v_\alpha})$ and $\boldsymbol{v_\beta} = \boldsymbol{v_\beta} + \boldsymbol{\delta}(\boldsymbol{v_\beta})$.
   b. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{evaluate}$ receives from the adversary a set of shares $\{(u^i_{1,j}, u^i_{2,j})\}_{j \in Corr}$. Then $\mathcal{F}_{evaluate}$ computes degree-t Shamir sharings $[v_{\alpha_i}|i]_t$ and $[v_{\beta_i}|i]_t$ such that for all $P_j \in Corr$, the j-th share of $[v_{\alpha_i}|i]_t$ is $u^i_{1,j}$ and the j-th share of $[v_{\beta_i}|i]_t$ is $u^i_{2,j}$. Finally, $\mathcal{F}_{evaluate}$ distributes the shares of $[v_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t$ to honest parties.
   c. $\mathcal{F}_{evaluate}$ computes $\boldsymbol{v_\gamma} = \phi(\psi(\boldsymbol{v_\alpha} * \boldsymbol{v_\beta}))$.
4. For each group of output gates with input wires $\boldsymbol{\alpha}$:
   a. $\mathcal{F}_{evaluate}$ receives a vector of additive errors $\boldsymbol{\delta}(\boldsymbol{v_\alpha})$ from the adversary. Then, $\mathcal{F}_{evaluate}$ sets $\boldsymbol{v_\alpha} = \boldsymbol{v_\alpha} + \boldsymbol{\delta}(\boldsymbol{v_\alpha})$
   b. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{evaluate}$ receives from the adversary a set of shares $\{u_{i,j}\}_{j \in Corr}$. Then $\mathcal{F}_{evaluate}$ computes a degree-t Shamir sharing $[v_{\alpha_i}|i]_t$ such that for all $P_j \in Corr$, the j-th share of $[v_{\alpha_i}|i]_t$ is $u_{i,j}$. Finally, $\mathcal{F}_{evaluate}$ distributes the shares of $[v_{\alpha_i}|i]_t$ to honest parties.
5. On receiving abort, $\mathcal{F}_{evaluate}$ sends abort to all parties.

Fig. 19. Functionality for evaluating Boolean circuits

Protocol 13: $\Pi_{evaluate}$

1. **Offline Phase:** All parties invoke $\mathcal{F}_{offline\text{-}mal}$ to receive correlated randomness that will be used in the online phase.
2. **Input Phase:** In the input layer, for each group of $k \cdot l$ input gates belong to some Client, let $\boldsymbol{\alpha}$ denote the output wires of these input gates. All parties and Client invoke $\Pi_{input\text{-}mal}$. At the end of the protocol, all parties hold $\{[v_{\alpha_i}|i]_t\}_{i=1}^k$. And P_1 learns $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ where $\boldsymbol{v_\alpha} = (v_{\alpha_1}, ..., v_{\alpha_k})$ and $v_{\alpha_i} = \phi(v^{(2)}_{\alpha_{i,1}}, ..., v^{(2)}_{\alpha_{i,1}})$ for $i \in \{1, ..., k\}$, $\{v^{(2)}_{\alpha_{i,j}}\}_{i\in\mathcal{K}, j\in\mathcal{L}}$ is the input values of Client.
3. **Computation Phase:** Note that P_1 holds $\boldsymbol{\mu_\alpha}$.
   a. For each group of XOR gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, P_1 locally compute $\boldsymbol{\mu_\gamma} = \boldsymbol{\mu_\alpha} + \boldsymbol{\mu_\beta}$
   b. For each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties invoke $\Pi_{mult\text{-}mal}$. At the end of the protocol, all parties hold $\{[v_{\alpha_i}|i]_t, [v_{\beta_i}|i]_t\}_{i=1}^k$, and P_1 learns $\boldsymbol{\mu_\gamma}$.
4. **Output Phase and Validity Check:** For each group of k output gates with input wires $\boldsymbol{\alpha}$, recall that all parties have received $\{[r_i|i]_t\}_{i=1}^k$ from $\mathcal{F}_{offline\text{-}mal}$. P_1 has received $\boldsymbol{\lambda_\alpha} + \boldsymbol{r}$ from $\mathcal{F}_{offline\text{-}mal}$ and P_1 has been learned $\boldsymbol{\mu_\alpha}$.
   a. P_1 computes $\boldsymbol{v_\alpha} + \boldsymbol{r} = \boldsymbol{\mu_\alpha} + (\boldsymbol{\lambda_\alpha} + \boldsymbol{r})$, distributes the degree-$(k-1)$ packed Shamir sharing $[\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t$ to all parties.
   b. For all $i \in \{1, ..., k\}$, parties locally compute $[v_{\alpha_i}|i]_t = [\![\boldsymbol{v_\alpha} + \boldsymbol{r}]\!]_t - [r_i|i]_t$.
   c. Let $\{[\![\boldsymbol{u}]\!]_{k-1}\}^N$ denote all degree-$(k-1)$ packed Shamir sharings distributed by P_1, all parties invoke $\mathcal{F}_{consistency}$ to check the validity of these sharings.

Fig. 20. Protocol for evaluating Boolean circuits

Functionality 10: $\mathcal{F}_{main\text{-}mal}$

1. $\mathcal{F}_{main\text{-}mal}$ receives the input from all clients. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{main\text{-}mal}$ computes $C(x)$. $\mathcal{F}_{main\text{-}mal}$ first distributes the output of corrupted clients to the adversary.
   – If the adversary replies continue, $\mathcal{F}_{main\text{-}mal}$ distributes the output to all clients.
   – If the adversary replies abort, $\mathcal{F}_{main\text{-}mal}$ sends abort to all clients.

Fig. 21. Functionality for main protocol with malicious security

Functionality 11: $\mathcal{F}_{verify}$

1. Let C denote the Circuit.
   a. For each group of input gates with output wires $\boldsymbol{\alpha}$, $\mathcal{F}_{verify}$ receives from honest parties their shares of $\{[v_{\alpha_i}|i]_t\}_{i=1}^k$. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{verify}$ recovers the whole sharing $[v_{\alpha_i}|i]_t$ and reconstructs the secret v_{α_i}. Then $\mathcal{F}_{verify}$ sends the shares of $[v_{\alpha_i}|i]_t$ of corrupted parties to the adversary.
   b. For each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathcal{F}_{verify}$ receives from honest parties their shares of $\{[v_{\alpha_i}|i]_t, [v_{\beta_i}|i]_t\}_{i=1}^k$. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{verify}$ recovers the whole sharings $[v_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t$ and reconstructs the secrets $\widetilde{v_{\alpha_i}}$, $\widetilde{v_{\beta_i}}$. Then $\mathcal{F}_{verify}$ sends the shares of $[v_{\alpha_i}|i]_t$, $[v_{\beta_i}|i]_t$ of corrupted parties to the adversary.
   c. For each group of output gates with input wires $\boldsymbol{\alpha}$, $\mathcal{F}_{verify}$ receives from honest parties their shares of $\{[v_{\alpha_i}|i]_t\}_{i=1}^k$. For all $i \in \{1, ..., k\}$, $\mathcal{F}_{verify}$ recovers the whole sharings $[v_{\alpha_i}|i]_t$ and reconstructs the secrets $\widetilde{v_{\alpha_i}}$. Then $\mathcal{F}_{verify}$ sends the shares of $[v_{\alpha_i}|i]_t$ of corrupted parties to the adversary.
2. $\mathcal{F}_{verify}$ evaluates the circuit C by using the secrets of the degree-t Shamir sharings associated with input gates.
   a. For each group of XOR gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{F}_{verify}$ computes $\boldsymbol{v_\gamma} = \boldsymbol{v_\alpha} + \boldsymbol{v_\beta}$.
   b. For each group of AND gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{F}_{verify}$ computes $\delta(\boldsymbol{v_\alpha}) = \widetilde{\boldsymbol{v_\alpha}} - \boldsymbol{v_\alpha}$ and $\delta(\boldsymbol{v_\beta}) = \widetilde{\boldsymbol{v_\beta}} - \boldsymbol{v_\beta}$. Then, $\mathcal{F}_{verify}$ sends $\delta(\boldsymbol{v_\alpha})$, $\delta(\boldsymbol{v_\beta})$ to the adversary. Finally, $\mathcal{F}_{verify}$ computes $\boldsymbol{v_\gamma} = \widetilde{\boldsymbol{v_\alpha}} * \widetilde{\boldsymbol{v_\beta}}$.
   c. For each group of output gates with input wires $\boldsymbol{\alpha}$, $\mathcal{F}_{verify}$ computes $\delta(\boldsymbol{v_\alpha}) = \widetilde{\boldsymbol{v_\alpha}} - \boldsymbol{v_\alpha}$. Then, $\mathcal{F}_{verify}$ sends $\delta(\boldsymbol{v_\alpha})$ to the adversary.
3. $\mathcal{F}_{verify}$ checks whether there exists an input wire $\boldsymbol{\alpha}$ of AND gates and output gates such that $\delta(\boldsymbol{v_\alpha}) \neq 0$. If true, $\mathcal{F}_{verify}$ sends abort to all parties. Otherwise, $\mathcal{F}_{verify}$ sends accept to all parties.
4. On receiving abort, $\mathcal{F}_{verify}$ sends abort to all parties.

Fig. 22. Functionality for verify the result

Protocol 14: $\Pi_{main\text{-}mal}$

1. All parties and clients invoke $\mathcal{F}_{evaluate}$ to compute a degree-t Shamir sharing for each group of output wires of input gates, and for each group of input wires of AND gates and output gates.
2. All parties invoke $\mathcal{F}_{verify}$ to check the correctness of the computation.
3. For each group of output gates that belongs to some Client, all parties hold a set of degree-t Shamir sharing $\{[v_{\alpha_i}|i]_t\}_{i=1}^k$ that is associated with these gates.
   a. All parties send their shares of $[v_{\alpha_i}|i]_t$ to Client.
   b. Client checks whether the shares of $[v_{\alpha_i}|i]_t$ lie on a degree-t polynomial. If true, Client reconstructs the secret v_{α_i} and computes $\psi(v_{\alpha_i} \cdot \phi(\mathbf{1}))$ and take it as the output of this gate. Otherwise, Client aborts.

Fig. 23. Our main malicious security protocol