Constant-Overhead Unconditionally Secure Multiparty Computation over Binary Fields

Antigoni Polychroniadou[1] and Yifan Song[2]

[1] J.P. Morgan AI Research, New York, USA
antigonipoly@gmail.com
[2] Carnegie Mellon University, Pittsburgh, USA
yifans2@andrew.cmu.edu

Abstract. We study the communication complexity of unconditionally secure multiparty computation (MPC) protocols in the honest majority setting. Despite tremendous efforts in achieving efficient protocols for binary fields under computational assumptions, there are no efficient unconditional MPC protocols in this setting. In particular, there are no n-party protocols with constant overhead admitting communication complexity of $O(n)$ bits per gate. Cascudo, Cramer, Xing and Yuan (CRYPTO 2018) were the first ones to achieve such an overhead in the amortized setting by evaluating $O(\log n)$ copies of the same circuit in the binary field in parallel. In this work, we construct the first unconditional MPC protocol secure against a malicious adversary in the honest majority setting evaluating just a *single* boolean circuit with amortized communication complexity of $O(n)$ bits per gate.

1 Introduction

Secure multiparty computation (MPC) [Yao82,GMW87,CCD88,BOGW88] allows n parties to compute any function of their local inputs while guaranteeing the privacy of the the inputs and the correctness of the outputs even if t of the parties are corrupted by an adversary.

Given that point-to-point secure channels are established across the parties, any function can be computed with unconditional (perfect) security, against a semi-honest adversary if $n \geq 2t + 1$ and against a malicious adversary if $n \geq 3t + 1$ [BOGW88,CCD88]. If we accept small error probability, $n \geq 2t + 1$ is sufficient to get malicious security [RBO89,Bea89].

The methods used in unconditional secure protocols tend to be computationally much more efficient than the cryptographic machinery required for computational security. So unconditionally secure protocols are very attractive from a computational point of view, but they seem to require a lot of interaction. In fact, such protocols require communication complexity proportional to the size of the (arithmetic) circuit computing the function. In this work we focus on the communication complexity per multiplication of unconditional MPC protocols in the honest majority setting.

Known unconditional secure MPC protocols represent the inputs as elements of a finite field F_q and represent the function as an arithmetic circuit over that finite field. Moreover, protocols that are efficient in the circuit size of the evaluated function process the circuit gate-by-gate using Shamir secret sharing [Sha79].

Y. Song—Work done in part while at J.P. Morgan AI Research. Supported in part by the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

This approach usually allows non-interactive processing of addition gates but requires communication for every multiplication gate. However, secret-sharing-based protocols require that the size of the underlying finite field is larger than the number of parties, i.e., $q > n$. The work of [BTH08] based on hyper-invertible matrices requires the underlying finite field to be $q \geq 2n$.[3] Other types of protocols with unconditional online phase based on message authentication codes, such as the SPDZ-based protocol [DPSZ12], require the size of the underlying finite field to be large, i.e., $q > 2^{\kappa}$, where κ is the security parameter. This is based on the fact that the cheating probability of the adversary needs to be inverse proportional to the size of the field.

In this paper, we ask a very natural question for unconditionally secure protocols which, to the best of our knowledge, has not been studied in detail before:

Is it possible to construct unconditional MPC protocols for $t < n/2$ for computing an arithmetic circuit over a small field F_q (such as $q = 2$) with amortized communication complexity $O(n)$ field elements (bits) per gate?

Note that the standard solution of applying the existing protocols to functions which are already represented as binary circuits requires to lift the circuit to a large enough extension field. That said, in such a scenario the communication complexity incurs a multiplicative overhead of $\log n$.

Recently, Cascudo, et al. [CCXY18] revisited the amortized complexity of unconditional MPC. At a high level, the authors leverage the large extension field to evaluate more than one instance of the same binary circuit in parallel. In particular, the authors compile an MPC protocol for a circuit over an extension field to a parallel MPC protocol of the same circuit but with inputs defined over its base field. That said, their protocol can evaluate $O(\log n)$ copies of the same circuit in the binary field in parallel and achieve communication complexity of $O(Cn)$ bits where C is the size of the circuit. However, such an overhead cannot be achieved for a single copy of the circuit. The works of [DZ13,CG20] also allow efficient parallel computation of several evaluations of the same binary circuits with a special focus on the dishonest majority. Note that these works are based on packed secret sharing for SIMD circuits, however this induces an extra overhead of $\log C$ in the circuit size when using for a single binary circuit.

Our Results. We answer the above question in the affirmative, obtaining an unconditional MPC protocol in the honest majority setting for calculations over $\mathbb{F}_2$. Informally, we prove the following:

Theorem 1 (informal). *There exists an unconditional MPC protocol for n parties secure against $t < n/2$ corruptions in the presence of a malicious adversary evaluating a single boolean circuit with an amortized communication complexity of $O(n)$ bits per gate.*

We formally state our results and communication overhead in Theorem 5. To establish our result, we propose an online phase based on additive sharings where we are able to authenticate the shares with $O(Cn)$ communication overhead as opposed to prior works which achieve an overhead of $O(Cn\kappa)$ for a single boolean circuit, where κ is the security parameter.

We are aware that the works of Hazay et al. [HVW20] and Boyle et al. [BGIN20] (building on Boneh et al. [BBCG$^+$19]) provide general compilers from semi-honest security to malicious security in the honest-majority setting, with at most a constant communication overhead. We leave the possibility of an alternative approach to achieve malicious security by applying these compilers to a semi-honest protocol which communicates $O(n)$ field elements per gate, such as our semi-honest protocol, to future work.

2 Technical Overview

In the following, we will use n to denote the number of parties and t to denote the number of corrupted parties. In the setting of the honest majority, we have $n = 2t + 1$.

Our construction will utilize two kinds of secret sharing schemes:

[3] In [CCXY18], Cascudo, et al. show that the requirement $q \geq 2n$ of using hyper-invertible matrices can be relaxed to any field size. However, $q > n$ is still necessary to use Shamir secret sharing in [BTH08].

– The standard Shamir secret sharing scheme [Sha79]: We will use $[x]_t$ to denote a degree-t Shamir sharing, or a $(t+1)$-out-of-n Shamir sharing. It requires at least $t+1$ shares to reconstruct the secret and any t shares do not leak any information about the secret.
– An additive sharing among the first $t+1$ parties: We will use $\langle x \rangle$ to denote an additive sharing, which satisfies that the summation of the shares held by the first $t+1$ parties is the secret x, and the shares of the rest of parties are 0.

Jumping ahead, the additive sharing is used to perform the secret sharing reconstructions, and the Shamir secret sharing scheme is used to verify the correctness of the reconstructions post execution.

In this paper, we are interested in the information-theoretic setting. Our goal is to construct a secure-with-abort MPC protocol for a *single* arithmetic circuit over the binary field $\mathbb{F}_2$, such that the communication complexity is $O(Cn)$ bits (ignoring terms which are sub-linear in the circuit size), where C is the circuit size and n is the number of parties. The structure of our overview is as follows:

1. We first provide an overview of related works and discuss why their protocols cannot achieve $O(Cn)$ bits for a single binary circuit.
2. Then we introduce a high-level structure of our construction. Very informally, our protocol uses additive sharings to achieve high efficiency in the online phase. However, using additive sharings requires authentications of the secrets to detect malicious behaviors. Based on the prior works, directly generating an authentication for each sharing already requires the communication of $O(Cn\kappa)$ bits, where κ is the security parameter. The main difficulty is how to efficiently authenticate the secrets of additive sharings.
3. Next we review the notion of reverse multiplication-friendly embeddings (RMFE) introduced in [CCXY18], which is an important building block of our protocol.
4. Finally, we introduce our main technique. Our idea stems from a new way to authenticate the secret of an additive sharing. Combining with RMFEs, we can authenticate the secret of a single additive sharing with the communication of $O(n)$ bits. Relying on this new technique, we can obtain a secure-with-abort MPC protocol for a single binary circuit with the communication complexity of $O(Cn)$ bits.

How Previous Constructions Work. In the honest majority setting, the best-known semi-honest protocol is introduced in the work of Damgård and Nielsen [DN07] in 2007 (hereafter referred to as the DN protocol). The communication complexity of the DN protocol is $O(Cn\phi)$ bits, where ϕ is the size of a field element. A beautiful line of works [GIP+14,LN17,CGH+18,NV18,GSZ20] have shown how to compile the DN protocol to achieve security-with-abort. In particular, the recent work [GSZ20] gives the first construction where the communication complexity matches the DN protocol. At a high-level, these protocols follow the idea of computing a degree-t Shamir sharing for each wire, and making use of the properties of the Shamir secret sharing scheme to evaluate addition gates and multiplication gates. However, the Shamir secret sharing scheme requires the field size to be at least $n+1$. It means that the size of a field element $\phi \geq \log n$. When we want to evaluate a binary circuit by using these protocols, we need to use a large enough extension field so that the Shamir secret sharing scheme is well-defined, which results in $O(Cn \log n)$ bits in the communication complexity.

Recently, a beautiful work [CCXY18] revisited the amortized complexity of information-theoretically secure MPC. Their idea is to compile an MPC for a circuit over an extension field to a parallel MPC of the same circuit but with inputs defined over its base field. In this way, we can evaluate $O(\log n)$ copies of the same circuit in the binary field at the same time and achieve $O(Cn)$ bits per circuit. The main technique is the notion of reverse multiplication-friendly embeddings (RMFE) introduced in this work [CCXY18]. At a high-level, RMFE allows us to perform a coordinate-wise product between two vectors of bits by multiplying two elements in the extension field. When evaluating $O(\log n)$ copies of the same circuit in the binary field, each multiplication is just a coordinate-wise product between the vectors of bits associated with the input wires. Relying on RMFE, all parties can transform the computation to one multiplication between two elements in the extension field, which can be handled by the DN protocol. This is the first paper which sheds light on the possibility of evaluating a binary circuit with communication complexity of $O(Cn)$ bits. However, it is unclear how to use this technique to evaluate a *single* binary circuit.

In the setting of the dishonest majority, the well-known work SPDZ [DPSZ12] shows that, with necessary correlated randomness prepared in the preprocessing phase, we can use an information-theoretic protocol in the online phase to achieve high efficiency. The high-level idea of the online phase protocol is to use the notion of Beaver tuples to transform a multiplication operation to two reconstructions. We will elaborate this technique at a later point. In the online phase, all parties will compute an additive sharing for each wire. One benefit of the additive secret sharing scheme is that it is well-defined in the binary field and each party holds a single bit as its share. As a result, the communication complexity in the online phase is just $O(Cn)$ bits. However, unlike the honest majority setting where the shares of honest parties can determine the secret of a degree-t Shamir sharing, the secret of an additive sharing can be easily altered by a corrupted party changing its own share. Therefore, a secure MAC is required to authenticate the secret of each additive sharing. To make the MAC effective, the MAC size should be proportional to the security parameter κ. Although it does not necessarily affect the sharing space, e.g., the work TinyOT [NNOB12] uses an additive sharing in the binary field with a secure MAC in the extension field, generating a secure MAC for each sharing in the preprocessing phase brings in an overhead of κ, which results in $O(Cn\kappa)$ bits in the overall communication complexity. We however note that, this protocol achieves a highly efficient online phase, which is $O(Cn)$ bits. Our starting idea is the online phase protocol in [DPSZ12]. In the honest majority setting, the preprocessing phase can also be done by an information-theoretic protocol. In fact, the idea of using Beaver tuples has been used in several previous works [BTH08,BSFO12,CCXY18] in the honest majority setting. We first describe a prototype protocol of using Beaver tuples in this setting.

A Prototype Protocol of Using Beaver Tuples. This protocol follows the same structure as the protocol in [DPSZ12], but in the honest majority setting. Recall that we use $\langle x \rangle$ to denote an additive sharing among the first $t+1$ parties. We use $\mathsf{MAC}(x)$ to denote an abstract MAC for x. It satisfies that all parties can use $\mathsf{MAC}(x)$ to check the correctness of x. We further require that $\mathsf{MAC}(\cdot)$ is linear homomorphic, i.e., $\mathsf{MAC}(x) + \mathsf{MAC}(y) = \mathsf{MAC}(x+y)$. Let $[\![x]\!] := (\langle x \rangle, \mathsf{MAC}(x))$.

In the preprocessing phase, all parties prepare a batch of Beaver tuples in the form of $([\![a]\!], [\![b]\!], [\![c]\!])$, where a, b are random bits and $c := a \cdot b$. These tuples will be used in the online phase to evaluate multiplication gates.

In the online phase, all parties start with holding $[\![x]\!]$ for each input wire. Addition gates and multiplication gates are evaluated in a predetermined topological order.

- For an addition gate with input sharings $[\![x]\!]$ and $[\![y]\!]$, all parties can locally compute
$$[\![z]\!] := (\langle z \rangle, \mathsf{MAC}(z)) = (\langle x \rangle, \mathsf{MAC}(x)) + (\langle y \rangle, \mathsf{MAC}(y)) = [\![x]\!] + [\![y]\!].$$
- For a multiplication gate with input sharings $[\![x]\!]$ and $[\![y]\!]$, let $([\![a]\!], [\![b]\!], [\![c]\!])$ be the first unused Beaver tuple. Note that:
$$\begin{aligned} z = x \cdot y &= (x + a - a) \cdot (y + b - b) \\ &= (x+a) \cdot (y+b) - (x+a) \cdot b - (y+b) \cdot a + a \cdot b \end{aligned}$$
Therefore, if all parties know $x + a$ and $y + b$, $[\![z]\!]$ can be locally computed by
$$[\![z]\!] := (x+a) \cdot (y+b) - (x+a) \cdot [\![b]\!] - (y+b) \cdot [\![a]\!] + [\![c]\!].$$
The task of computing $[\![z]\!]$ becomes to reconstruct $[\![x]\!] + [\![a]\!]$ and $[\![y]\!] + [\![b]\!]$. We will use $\langle x+a \rangle$ and $\langle y+b \rangle$ to do the reconstructions. All parties send their shares of $\langle x+a \rangle, \langle y+b \rangle$ to the first party. Then, the first party reconstructs the $x+a, y+b$, and sends the result back to other parties.

To check the correctness of the computation, it is sufficient to verify the reconstructions. For each $x+a$, all parties use $[\![x]\!], [\![a]\!]$ to compute $\mathsf{MAC}(x+a)$, which can be used to verify the reconstruction.

Note that we only need to communicate $O(n)$ bits per multiplication gates. Therefore, the communication complexity is $O(Cn)$ bits in the online phase. The main bottleneck of this approach is how to generate Beaver tuples efficiently. Our protocol relies on the notion of reverse multiplication-friendly embeddings and a novel MAC to achieve high efficiency in generating Beaver tuples.

Review of the Reverse Multiplication-Friendly Embeddings [CCXY18]. We note that a Beaver tuple can be prepared by the following two steps: (1) prepare two random sharings $[\![a]\!], [\![b]\!]$, and (2) compute $[\![c]\!]$ such that $c := a \cdot b$. Note that a, b are random bits. It naturally connects to the idea of RMFE, which allows us to perform a coordinate-wise product between two vector of bits by multiplying two elements in the extension field. We first give a quick review of this notion.

Let $\mathbb{F}_2^k$ denote a vector space of $\mathbb{F}_2$ of dimension k, and $\mathbb{F}_{2^m}$ denote the extension field of $\mathbb{F}_2$ of degree m. A reverse multiplication-friendly embedding is a pair of $\mathbb{F}_2$-linear maps (ϕ, ψ), where $\phi : \mathbb{F}_2^k \to \mathbb{F}_{2^m}$ and $\psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^k$, such that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^k$,

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

where $*$ denotes the coordinate-wise product. In [CCXY18], it has been shown that there exists a family of RMFEs such that $m = \Theta(k)$.

In [CCXY18], recall that $k = O(\log n)$ copies of the same circuit are evaluated together. For each wire, there is a vector of k bits associated with this wire, where the i-th bit is the wire value of the i-th copy of the circuit. Thus, an addition gate corresponds to a coordinate-wise addition, and a multiplication gate corresponds to a coordinate-wise product. In the construction of [CCXY18], for each wire, the vector $\boldsymbol{x}$ associated with this wire is encoded to $\phi(\boldsymbol{x}) \in \mathbb{F}_{2^m}$. All parties hold a degree-t Shamir sharing $[\phi(\boldsymbol{x})]_t$. Since $\phi(\cdot)$ is an $\mathbb{F}_2$-linear map, addition gates can be computed locally. The main task is to evaluate multiplication gates:

- For a multiplication gate with input sharings $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$, the goal is to compute a degree-t Shamir sharing $[\phi(\boldsymbol{z})]_t$ such that $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$.
- Relying on the DN protocol [DN07], all parties can compute a degree-t Shamir sharing $[w]_t := [\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})]_t$. By the property of the RMFE, we have $\boldsymbol{z} = \psi(w)$. Therefore, all parties need to transform $[w]_t$ to $[\phi(\psi(w))]_t$.
- In [CCXY18], this is done by using a pair of random sharings $([r]_t, [\phi(\psi(r))]_t)$. All parties reconstruct $[w + r]_t$ and compute $[\phi(\psi(w))]_t := \phi(\psi(w + r)) - [\phi(\psi(r))]_t$. The correctness follows from the fact that ϕ and ψ are $\mathbb{F}_2$-linear maps.
- Finally, all parties set $[\phi(\boldsymbol{z})]_t := [\phi(\psi(w))]_t$.

As analyzed in [CCXY18], the communication complexity per multiplication gate is $O(m \cdot n)$ bits. Since each multiplication gate corresponds to k multiplications in the binary field, the amortized communication complexity per multiplication is $O(m/k \cdot n) = O(n)$ bits.

Following the idea in [CCXY18], we can prepare a random tuple of sharings $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_2^k$, and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. In particular, the communication complexity per tuple is $O(m \cdot n)$ bits. Suppose that $\boldsymbol{a} = (a_1, a_2, \ldots, a_k), \boldsymbol{b} = (b_1, b_2, \ldots, b_k)$, and $\boldsymbol{c} = (c_1, c_2, \ldots, c_k)$. If we can transform a random tuple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ to k Beaver tuples:

$$([\![a_1]\!], [\![b_1]\!], [\![c_1]\!]), ([\![a_2]\!], [\![b_2]\!], [\![c_2]\!]), \ldots, ([\![a_k]\!], [\![b_k]\!], [\![c_k]\!]),$$

then the communication complexity per Beaver tuple is $O(m/k \cdot n) = O(n)$ bits! More concretely, our goal is to efficiently separate a degree-t Shamir sharing $[\phi(\boldsymbol{a})]_t$ to k sharings $[\![a_1]\!], [\![a_2]\!], \ldots, [\![a_k]\!]$. For all $i \in [k]$, recall that $[\![a_i]\!] = (\langle a_i \rangle, \mathsf{MAC}(a_i))$. Therefore, we need to efficiently obtain *an additive sharing* $\langle a_i \rangle$ and *a secure* $\mathsf{MAC}(a_i)$ from a degree-t Shamir sharing $[\phi(\boldsymbol{a})]_t$.

Establish a Connection between $[\phi(\boldsymbol{x})]_t$ *and* $\{[\![x_i]\!]\}_{i=1}^k$. We first consider the following question: Given $\phi(\boldsymbol{x})$, how can we obtain the i-th bit x_i from $\phi(\boldsymbol{x})$? Let $\boldsymbol{e}^{(i)}$ be a vector in $\mathbb{F}_2^k$ such that all entries are 0 except that the i-th entry is 1. Then $\boldsymbol{e}^{(i)} * \boldsymbol{x}$ is a vector in $\mathbb{F}_2^k$ such that all entries are 0 except that the i-th entry is x_i. According to the definition of RMFEs, we have

$$\boldsymbol{e}^{(i)} * \boldsymbol{x} = \psi(\phi(\boldsymbol{e}^{(i)}) \cdot \phi(\boldsymbol{x})).$$

To obtain x_i from $\boldsymbol{e}^{(i)} * \boldsymbol{x}$, we can compute the summation of all entries in $\boldsymbol{e}^{(i)} * \boldsymbol{x}$. We define an $\mathbb{F}_2$-linear map $\mathsf{val}(\cdot) : \mathbb{F}_{2^m} \to \mathbb{F}_2$ as follows:

– For an input element $y \in \mathbb{F}_{2^m}$, suppose $\psi(y) = (y_1, y_2, \ldots, y_k)$.
– $\mathsf{val}(y)$ is defined to be $\sum_{i=1}^{k} y_i$.

Therefore, we have

$$x_i := \mathsf{val}(\phi(\boldsymbol{e}^{(i)}) \cdot \phi(\boldsymbol{x})).$$

Note that $\phi(\boldsymbol{e}^{(i)})$ is an element in $\mathbb{F}_{2^m}$ and is known to all parties. Therefore, all parties can locally compute $[y^{(i)}]_t := \phi(\boldsymbol{e}^{(i)}) \cdot [\phi(\boldsymbol{x})]_t$. In particular, we have $\mathsf{val}(y^{(i)}) = x_i$. In the honest majority setting, a degree-t Shamir sharing satisfies that the secret is determined by the shares of honest parties. In particular, corrupted parties cannot alter the secret of this sharing. Therefore, $[y^{(i)}]_t$ can be seen as a secure MAC for x_i. Thus for an element $x \in \mathbb{F}_2$, we set $\mathsf{MAC}(x) := [y]_t$, where $y \in \mathbb{F}_{2^m}$ satisfies that $\mathsf{val}(y) = x$. Note that $[y]_t$ can be used to check the correctness of x, and for all $x, x' \in \mathbb{F}_2$,

$$\mathsf{MAC}(x) + \mathsf{MAC}(x') = [y]_t + [y']_t = [y + y']_t = \mathsf{MAC}(x + x'),$$

where the last step follows from the fact that $\mathsf{val}(y + y') = \mathsf{val}(y) + \mathsf{val}(y')$.

Recall that $[\![x_i]\!] = (\langle x_i \rangle, \mathsf{MAC}(x_i))$. So far, we have obtained $\mathsf{MAC}(x_i)$ from $[\phi(\boldsymbol{x})]_t$. Therefore, the only task is to obtain $\langle x_i \rangle$. Let $\langle \boldsymbol{x} \rangle := (\langle x_1 \rangle, \langle x_2 \rangle, \ldots, \langle x_k \rangle)$ denote a vector of additive sharings of $\boldsymbol{x} \in \mathbb{F}_2^k$. For each party, its share of $\langle \boldsymbol{x} \rangle$ is a vector in $\mathbb{F}_2^k$. For the last t parties, they take the all-0 vector as their shares.

We note that for a degree-t Shamir sharing $[\phi(\boldsymbol{x})]_t$, the secret $\phi(\boldsymbol{x})$ can be written as a linear combination of the shares of the first $t+1$ parties. Therefore, the first $t+1$ parties can locally transform their shares of $[\phi(\boldsymbol{x})]_t$ to an additive sharing of $\phi(\boldsymbol{x})$, denoted by $\langle \phi(\boldsymbol{x}) \rangle$. Let u_i denote the i-th share of $\langle \phi(\boldsymbol{x}) \rangle$. Then we have $\phi(\boldsymbol{x}) = \sum_{i=1}^{t+1} u_i$. In Section 3.3, we give an explicit construction of an $\mathbb{F}_2$-linear map $\tilde{\phi}^{-1} : \mathbb{F}_{2^m} \to \mathbb{F}_2^k$ which satisfies that for all $\boldsymbol{x} \in \mathbb{F}_2^k$, $\tilde{\phi}^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}$. Utilizing $\tilde{\phi}^{-1}$, we have

$$\sum_{i=1}^{t+1} \tilde{\phi}^{-1}(u_i) = \tilde{\phi}^{-1}(\sum_{i=1}^{t+1} u_i) = \tilde{\phi}^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}.$$

Thus, the i-th party takes $\tilde{\phi}^{-1}(u_i)$ as its share of $\langle \boldsymbol{x} \rangle$.

In summary, we show that given $[\phi(\boldsymbol{x})]_t$, all parties can *locally* obtain $\{[\![x_i]\!]\}_{i=1}^k$. Together with RMFEs, the communication complexity per Beaver tuple is $O(n)$ bits. Relying on the prototype protocol of using Beaver tuples, we obtain a secure-with-abort MPC protocol for a *single* binary circuit which has communication complexity $O(Cn)$ bits. We note that these k sharings $\{[\![x_i]\!]\}_{i=1}^k$ are correlated since they are computed from a single degree-t Shamir sharing $[\phi(\boldsymbol{x})]_t$. Our protocol will make use of additional randomness as mask to protect the secrecy of these sharings when they are used. The preparation of this additional randomness is done in a batch way at the beginning of the protocol and does not affect the asymptotic communication complexity of the main protocol. We refer the readers to Section 6.3 and Section 6.4 for the additional randomness we need in the construction.

An Overview of Our Main Construction. Our main protocol follows the same structure as the prototype protocol of using Beaver tuples. Recall that for $x \in \mathbb{F}_2$, we use $\langle x \rangle$ to denote an additive sharing of x among the first $t+1$ parties, and the shares of the rest of parties are 0. Let (ϕ, ψ) be a RMFE, where $\phi : \mathbb{F}_2^k \to \mathbb{F}_{2^m}$ and $\psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^k$ are $\mathbb{F}_2$-linear maps. Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map, defined by $\mathsf{val}(y) = \sum_{i=1}^{k} y_i$, where $(y_1, y_2, \ldots, y_k) = \psi(y)$. For $x \in \mathbb{F}_2$, let $[\![x]\!] := (\langle x \rangle, [y]_t)$, where $\langle x \rangle$ is an additive sharing among the first $t+1$ parties in $\mathbb{F}_2$, and $[y]_t$ is a degree-t Shamir sharing of $y \in \mathbb{F}_{2^m}$ such that $\mathsf{val}(y) = x$.

In the preprocessing phase, all parties prepare a batch of Beaver tuples in the form of $([\![a]\!], [\![b]\!], [\![c]\!])$, where a, b are random bits and $c := a \cdot b$. The Beaver tuples are prepared by the following steps:

– All parties first prepare a batch of random tuples of sharings in the form of $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_2^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. In our protocol, preparing such a random tuple of sharings require the communication of $O(m \cdot n)$ bits.

– For each tuple of sharings $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, all parties locally transform it to k Beaver tuples in the form of $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$.

Note that the amortized cost per Beaver tuple is $O(n)$ bits.

In the online phase, all parties start with holding $\llbracket x \rrbracket$ for each input wire. Addition gates and multiplication gates are evaluated in a predetermined topological order.

– For an addition gate with input sharings $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, all parties locally compute $\llbracket z \rrbracket := \llbracket x \rrbracket + \llbracket y \rrbracket$.
– For a multiplication gate with input sharings $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$, let $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ be the first unused Beaver tuple. All parties use the additive sharings $\langle x + a \rangle, \langle y + b \rangle$ to reconstruct $x + a$ and $y + b$. Then all parties compute

$$\llbracket z \rrbracket := (x + a) \cdot (y + b) - (x + a) \cdot \llbracket b \rrbracket - (y + b) \cdot \llbracket a \rrbracket + \llbracket c \rrbracket.$$

All parties also locally compute $\llbracket x + a \rrbracket := \llbracket x \rrbracket + \llbracket a \rrbracket$ and $\llbracket y + b \rrbracket := \llbracket y \rrbracket + \llbracket b \rrbracket$. These sharings will be used to verify the reconstructions at the end of the protocol.

After evaluating the whole circuit, all parties together verify the value-sharing pairs in the form of $(x + a, \llbracket x + a \rrbracket)$, where $x + a$ is the reconstruction of $\llbracket x + a \rrbracket$. In Section 7.3, we show that all the value-sharing pairs can be verified together with sub-linear communication complexity in the number of pairs.

Note that addition gates can be computed locally, and the communication complexity per multiplication gate is $O(n)$ bits. Therefore, the communication complexity of our protocol is $O(Cn)$ bits.

Other Building Blocks and Security Issues. We note that the work [CCXY18] only focuses on the setting of 1/3 corruption. These protocols cannot be used directly in the honest majority setting. Some techniques even fail when the corruption threshold increases. In this work, we rebuild the protocols in [CCXY18] to fit the honest majority setting by combining known techniques in [BSFO12,GSZ20]. Concretely,

– We follow the definition of a general linear secret sharing scheme (GLSSS) in [CCXY18]. Following the idea in [BSFO12] of preparing random degree-t Shamir sharings, we introduce a protocol to allow all parties efficiently prepare random sharings of a given GLSSS. We use this protocol to prepare various kinds of random sharings in our main construction. Let $\mathcal{F}_{\text{rand}}$ denote the functionality of this protocol.
– To prepare Beaver tuples, we first prepare a random tuple of sharings

$$([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t),$$

where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_2^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. This random tuple of sharings is prepared as follows:
  - The first step is to prepare random sharings $[\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t$. We show that they can be prepared by using $\mathcal{F}_{\text{rand}}$.
  - Then all parties compute $[\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})]_t$. We rely on the multiplication protocol and the efficient multiplication verification in [GSZ20].
  - Finally, all parties need to transform a sharing $[w]_t$ to $[\phi(\psi(w))]_t$, where $w = \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})$. We model this process in the functionality $\mathcal{F}_{\text{re-encode}}$. We extend the idea in [CCXY18] from the 1/3 corruption setting to the honest majority setting, and construct an efficient protocol for the functionality $\mathcal{F}_{\text{re-encode}}$.

More details can be found in Section 4 and Section 6.

We note that the idea of using Beaver tuples to construct an MPC protocol in the honest majority setting has been used in several previous works [BTH08,BSFO12,CCXY18]. These protocols all have an additional term $O(D \cdot n^2)$ in the communication complexity, where D is the circuit depth. It is due to a verification of the computation in each layer. Recall that relying on Beaver tuples, an multiplication can be transformed to two reconstructions. In [GLS19], Goyal, et al. show that, without verification of the computation in each layer, corrupted parties can learn extra information when doing reconstructions for multiplications in the next layer. It turns out that our protocol has a similar security issue.

To avoid the verification of the computation per layer, Goyal, et al. [GLS19] rely on an n-out-of-n secret sharing to protect the shares of honest parties. In this way, even without verifications, the share of each

honest party is uniformly distributed. It allows Goyal, et al. to only check the correctness at the end of the protocol. We follows the idea in [GLS19]. Concretely, we want to protect the shares of honest parties when using $\langle x + a \rangle, \langle y + b \rangle$ to do reconstructions. To this end, we add a uniformly random additive sharing of 0 for each reconstruction. In this way, each honest party simply sends a uniformly random element to the first party. It allows us to delay the verification to the end of the protocol. More details can be found in Section 7.

3 Preliminaries

3.1 The Model

In this work, we focus on functions that can be represented as arithmetic circuits over a finite field $\mathbb{F}_q$ of size q with input, addition, multiplication, and output gates. We use κ to denote the security parameter and C to denote the size of the circuit. In the following, we will use an extension field of $\mathbb{F}_q$ denoted by $\mathbb{F}_{q^m}$ (of size q^m). We always assume that $|\mathbb{F}_{q^m}| = q^m \geq 2^\kappa$.

For the secure multi-party computation, we use 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 $n = 2t + 1$ 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 by the number of bits via private channels.

Security Definition. Let $\mathcal{F}$ be a secure function evaluation functionality. 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. Corrupted clients and servers can deviate from the protocol arbitrarily.

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

Ideal World Execution. In the ideal world, a simulator `Sim` simulates honest clients and servers and interacts with the adversary $\mathcal{A}$. Furthermore, `Sim` has a one-time access to $\mathcal{F}$, which includes providing inputs of corrupted clients and servers to $\mathcal{F}$, receiving the outputs of corrupted clients and servers, 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 clients and servers, and the view of the adversary.

We say that a protocol π securely computes $\mathcal{F}$ if there exists a simulator `Sim`, such that for all adversary $\mathcal{A}$, the distribution of the output of the real world execution is *statistically close* to the distribution of the output of the ideal world execution. If π allows a premature abort, then we say π securely computes $\mathcal{F}$ with abort. We refer the readers to [GIP+14] for a formal definition.

Benefits of the Client-Server Model. In our construction, the clients only participate in the input phase and the output phase. The main computation is conducted by the servers. For simplicity, we use $\{P_1, \ldots, P_n\}$ to denote the n servers, and refer to the servers as parties. Let $\mathcal{C}$ denote the set of all corrupted parties and $\mathcal{H}$ denote the set of all honest parties. One benefit of the client-server model is the following theorem shown in [GIP+14].

Theorem 2 (Lemma 5.2 [GIP+14]). *Let Π be a protocol computing a c-client circuit C using $n = 2t+1$ parties. Then, if Π is secure against any adversary controlling exactly t parties, then Π is secure against any adversary controlling at most t parties.*

This theorem allows us to only consider the case where the adversary controls exactly t parties. Therefore in the following, we assume that there are exactly t corrupted parties.

3.2 Secret Sharing Scheme

Shamir Secret Sharing Scheme. In this work, we will use the standard Shamir Secret Sharing Scheme [Sha79]. Let n be the number of parties and $\mathbb{G}$ be a finite field of size $|\mathbb{G}| \geq n+1$. Let $\alpha_1, \ldots, \alpha_n$ be n distinct non-zero elements in $\mathbb{G}$.

A *degree-d* Shamir sharing of $x \in \mathbb{G}$ is a vector $(x_1, \ldots, x_n)$ which satisfies that, there exists a polynomial $f(\cdot) \in \mathbb{G}[X]$ of degree at most d such that $f(0) = x$ and $f(\alpha_i) = x_i$ for $i \in \{1, \ldots, n\}$. Each party P_i holds a share x_i and the whole sharing is denoted by $[x]_d$.

We recall the properties of a degree-d Shamir sharing: (1) It requires $d+1$ shares to reconstruct the secret x, and (2) any d shares do not leak any information about x.

Abstract General Linear Secret Sharing Schemes. We adopt the notion of an abstract definition of a general linear secret sharing scheme (GLSSS) in [CCXY18]. The following notations are borrowed from [CCXY18].

For non-empty sets U and $\mathcal{I}$, $U^{\mathcal{I}}$ denotes the indexed Cartesian product $\prod_{i \in \mathcal{I}} U$. For a non-empty set $A \subset \mathcal{I}$, the natural projection π_A maps a tuple $u = (u_i)_{i \in \mathcal{I}} \in U^{\mathcal{I}}$ to the tuple $(u_i)_{i \in A} \in U^A$. Let K be a field.

Definition 1 (Abstract K-GLSSS [CCXY18]). *A general K-linear secret sharing scheme Σ consists of the following data:*

- *A set of parties $\mathcal{I} = \{1, \ldots, n\}$*
- *A finite-dimensional K-vector space Z, the secret space.*
- *A finite-dimensional K-vector space U, the share space.*
- *A K-linear subspace $C \subset U^{\mathcal{I}}$, where the latter is considered a K-vector space in the usual way (i.e., direct sum).*
- *A surjective K-linear map $\Phi : C \to Z$, its defining map.*

Definition 2 ([CCXY18]). *Suppose $A \subset \mathcal{I}$ is nonempty. Then A is a privacy set if the K-linear map*

$$(\Phi, \pi_A) : C \longrightarrow Z \times \pi_A(C), \quad x \mapsto (\Phi(x), \pi_A(x))$$

is surjective. Finally, A is a reconstruction set if, for all $x \in C$, it holds that

$$\pi_A(x) = 0 \Rightarrow \Phi(x) = 0.$$

A Tensoring-up Lemma. We follow the definition of interleaved GLSSS: the m-fold interleaved GLSSS $\Sigma^{\times m}$ is an n-party scheme which corresponds to m Σ-sharings. We have the following proposition from [CCXY18]:

Proposition 1 ([CCXY18]). *Let L be a degree-m extension field of K and let Σ be a K-GLSSS. Then the m-fold interleaved K-GLSSS $\Sigma^{\times m}$ is naturally viewed as an L-GLSSS, compatible with its K-linearity.*

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

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

An Example of a GLSSS and Using the Tensoring-up Lemma. We will use the standard Shamir secret sharing scheme as an example of a GLSSS and show how to use the tensoring-up lemma. For a field K (of size $|K| \geq n+1$), we may define a secret sharing Σ which takes an input $x \in K$ and outputs $[x]_t$, i.e., a degree-t Shamir sharing. The secret space and the share space of Σ are K. According to the Lagrange interpolation, the secret x can be written as a K-linear combination of all the shares. Therefore, the defining map of Σ is K-linear. Thus Σ is a K-GLSSS.

A sharing $[\boldsymbol{x}]_t = ([x_1]_t, [x_2]_t, \ldots, [x_m]_t) \in \Sigma^{\times m}$ is a vector of m sharings in Σ. Let L be a degree-m extension field of K. The tensoring-up lemma says that $\Sigma^{\times m}$ is a L-GLSSS. Therefore we can perform L-linear operations to the sharings in $\Sigma^{\times m}$.

3.3 Reverse Multiplication Friendly Embeddings

Definition 3 ([CCXY18]). *Let k, m be integers and $\mathbb{F}_q$ be a finite field. A pair (ϕ, ψ) is called an $(k, m)_q$-reverse multiplication friendly embedding (RMFE) if $\phi : \mathbb{F}_q^k \to \mathbb{F}_{q^m}$ and $\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^k$ are two $\mathbb{F}_q$-linear maps satisfying*

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

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

Note that when picking $\mathbf{1} = (1, 1, \ldots, 1)$, we have $\boldsymbol{x} * \mathbf{1} = \boldsymbol{x}$ and therefore, $\boldsymbol{x} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\mathbf{1}))$. It implies that ϕ is injective. Therefore, there exists $\phi^{-1} : \mathrm{Im}(\phi) \to \mathbb{F}_q^k$ such that for all $\boldsymbol{x} \in \mathbb{F}_q^k$, it satisfies that

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

It is easy to verify that ϕ^{-1} is also $\mathbb{F}_q$-linear.

Now we show that there exists an $\mathbb{F}_q$-linear map $\tilde{\phi}^{-1} : \mathbb{F}_{q^m} \to \mathbb{F}_q^k$ such that for all $\boldsymbol{x} \in \mathbb{F}_q^k$,

$$\tilde{\phi}^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}.$$

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

$$\tilde{\phi}^{-1}(\phi(\boldsymbol{x})) = \boldsymbol{x}.$$

Proof. Let $\mathbf{1} = (1, 1, \ldots, 1) \in \mathbb{F}_q^k$. We explicitly construct $\tilde{\phi}^{-1}$ as follows:

$$\tilde{\phi}^{-1} : \mathbb{F}_{q^m} \longrightarrow \mathbb{F}_q^k, \quad x \mapsto \psi(\phi(\mathbf{1}) \cdot x)$$

It is clear that $\tilde{\phi}^{-1}$ is $\mathbb{F}_q$-linear. For all $\boldsymbol{x} \in \mathbb{F}_q^k$, by the definition of RMFE, we have

$$\tilde{\phi}^{-1}(\phi(\boldsymbol{x})) = \psi(\phi(\mathbf{1}) \cdot \phi(\boldsymbol{x})) = \mathbf{1} * \boldsymbol{x} = \boldsymbol{x}.$$

□

In [CCXY18], Cascudo et al. show that there exist constant rate RMFEs, which is summarized in Theorem 3.

Theorem 3. *For every finite prime power q, there exists a family of constant rate $(k, m)_q$-RMFE where $m = \Theta(k)$.*

3.4 Useful Building Blocks

In this part, we will introduce three functionalities which will be used in our main construction.

- The first functionality $\mathcal{F}_{\text{coin}}$ allows all parties to generate a random element.
- The second functionality $\mathcal{F}_{\text{mult}}$ allows all parties to evaluate a multiplication with inputs being shared by degree-t Shamir sharings. While $\mathcal{F}_{\text{mult}}$ protects the secrets of the input sharings, it allows the adversary to add an arbitrary fixed value to the multiplication result.
- The third functionality $\mathcal{F}_{\text{multVerify}}$ allows all parties to verify the correctness of multiplications computed by $\mathcal{F}_{\text{mult}}$.

Generating Random Coin. The first functionality $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ allows all parties to generate a random field element in $\mathbb{F}_{q^m}$. The description of $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ appears in Functionality 1.

An instantiation of this functionality can be found in [GSZ20] (Protocol 6 in Section 3.5 of [GS20]), which has communication complexity $O(n^2)$ elements in $\mathbb{F}_{q^m}$ (i.e., $O(n^2 \cdot m)$ elements in $\mathbb{F}_q$).

Functionality 1: $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$

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

Functionality 2: $\mathcal{F}_{\text{mult}}$

1. Let $[x]_t, [y]_t$ denote the input sharings. $\mathcal{F}_{\text{mult}}$ receives from honest parties their shares of $[x]_t, [y]_t$. Then $\mathcal{F}_{\text{mult}}$ reconstructs the secrets x, y. $\mathcal{F}_{\text{mult}}$ further computes the shares of $[x]_t, [y]_t$ held by corrupted parties, and sends these shares to the adversary.
2. $\mathcal{F}_{\text{mult}}$ receives from the adversary a value d and a set of shares $\{z_i\}_{i\in\mathcal{C}}$.
3. $\mathcal{F}_{\text{mult}}$ computes $x \cdot y + d$. Based on the secret $z := x \cdot y + d$ and the t shares $\{z_i\}_{i\in\mathcal{C}}$, $\mathcal{F}_{\text{mult}}$ reconstructs the whole sharing $[z]_t$ and distributes the shares of $[z]_t$ to honest parties.

Computing Multiplication. In [GIP+14], Genkin et al. prove that several semi-honest MPC protocols in the honest-majority setting are secure up to an additive attack in the presence of a fully malicious adversary. An additive attack means that the adversary is able to change the multiplication result by adding an arbitrary fixed value. We model this kind of attack in $\mathcal{F}_{\text{mult}}$, which takes two degree-t Shamir sharings $[x]_t, [y]_t$ and outputs the multiplication result $[x \cdot y]_t$. The description of $\mathcal{F}_{\text{mult}}$ can be found in Functionality 2.

This functionality can be instantiated by the multiplication protocol in the semi-honest DN protocol [DN07]. In [GSZ20], Goyal et al. also provide a detailed proof of the security of the multiplication protocol in [DN07] (Lemma 4 in Section 4.1 of [GS20]). The amortized communication complexity per multiplication is $O(n)$ field elements per party.

Multiplication Verification. Note that $\mathcal{F}_{\text{mult}}$ does not guarantee the correctness of the multiplications. Therefore, all parties need to verify the multiplications computed by $\mathcal{F}_{\text{mult}}$. The functionality $\mathcal{F}_{\text{multVerify}}$ takes N multiplication tuples as input and outputs to all parties a single bit b indicating whether all multiplication tuples are correct. The description of $\mathcal{F}_{\text{multVerify}}$ can be found in Functionality 3.

Functionality 3: $\mathcal{F}_{\text{multVerify}}$

1. Let N denote the number of multiplication tuples. The multiplication tuples are denoted by
$$([x^{(1)}]_t, [y^{(1)}]_t, [z^{(1)}]_t), ([x^{(2)}]_t, [y^{(2)}]_t, [z^{(2)}]_t), \ldots, ([x^{(N)}]_t.[y^{(N)}]_t, [z^{(N)}]_t).$$
2. For all $i \in [N]$, $\mathcal{F}_{\text{multVerify}}$ receives from honest parties their shares of $[x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t$. Then $\mathcal{F}_{\text{multVerify}}$ reconstructs the secrets $x^{(i)}, y^{(i)}, z^{(i)}$. $\mathcal{F}_{\text{multVerify}}$ further computes the shares of $[x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t$ held by corrupted parties and sends these shares to the adversary.
3. For all $i \in [N]$, $\mathcal{F}_{\text{multVerify}}$ computes $d^{(i)} = z^{(i)} - x^{(i)} \cdot y^{(i)}$ and sends $d^{(i)}$ to the adversary.
4. Finally, let $b \in \{\texttt{abort}, \texttt{accept}\}$ denote whether there exists $i \in [N]$ such that $d^{(i)} \neq 0$. $\mathcal{F}_{\text{multVerify}}$ sends b to the adversary and waits for its response.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{multVerify}}$ sends b to honest parties.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{multVerify}}$ sends `abort` to honest parties.

An instantiation of $\mathcal{F}_{\text{multVerify}}$ can be found in [GSZ20] (Protocol 17 in Section 5.4 of [GS20]), which has communication complexity $O(n^2 \cdot \log N \cdot \kappa)$ bits, where n is the number of parties and κ is the security parameter. Note that the amortized communication per multiplication tuple is sub-linear.

4 Preparing Random Sharings for $\mathbb{F}_q$-GLSSS

In this section, we present the protocol for preparing random sharings for a given general $\mathbb{F}_q$-linear secret sharing scheme, denoted by Σ. Let $[x]$ denote a sharing in Σ of secret x. For a set $A \subset \mathcal{I}$, recall that $\pi_A([x])$ refers to the shares of $[x]$ held by parties in A. We assume that Σ satisfies the following property:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, let

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

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

The description of the functionality $\mathcal{F}_{\text{rand}}$ appears in Functionality 4. In short, $\mathcal{F}_{\text{rand}}$ allows the adversary to specify the shares held by corrupted parties. Based on these shares, $\mathcal{F}_{\text{rand}}$ generates a random sharing in Σ and distributes the shares to honest parties. Note that, when the set of corrupted parties is a privacy set, the secret is independent of the shares chosen by the adversary.

Functionality 4: $\mathcal{F}_{\text{rand}}$

1. $\mathcal{F}_{\text{rand}}$ receives from the adversary the set of corrupted parties, denoted by $\mathcal{C}$, and a set of shares $(s_i)_{i\in\mathcal{C}}$ such that $\Sigma(\mathcal{C}, (s_i)_{i\in\mathcal{C}}) \neq \emptyset$. Then $\mathcal{F}_{\text{rand}}$ randomly samples $[r] \in \Sigma(\mathcal{C}, (s_i)_{i\in\mathcal{C}})$.
2. $\mathcal{F}_{\text{rand}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{rand}}$ sends `abort` to honest parites.
   - If the adversary replies `continue`, for each honest party P_i, $\mathcal{F}_{\text{rand}}$ sends the i-th share of $[r]$ to P_i.

We will follow the idea in [BSFO12] of preparing random degree-t Shamir sharings to prepare random sharings in Σ. At a high-level, each party first deals a batch of random sharings in Σ. For each party, all parties together verify that the sharings dealt by this party have the correct form. Then all parties locally convert the sharings dealt by each party to random sharings such that the secrets are not known to any single party.

Recall that κ denotes the security parameter, and the extension field $\mathbb{F}_{q^m}$ satisfies that $|\mathbb{F}_{q^m}| = q^m \geq 2^\kappa$. In the following, instead of preparing random sharings in Σ, we choose to prepare random sharings in $\Sigma^{\times m}$, where each sharing $[\boldsymbol{x}]$ in $\Sigma^{\times m}$ is a vector of m sharings $([x_1], [x_2], \ldots, [x_m])$ in Σ. According to Proposition 1, $\Sigma^{\times m}$ is an $\mathbb{F}_{q^m}$-GLSSS.

Preparing Verified Sharings. The first step is to let each party deal a batch of random sharings in $\Sigma^{\times m}$. The protocol VerShare(P_d, N') (Protocol 5) allows the dealer P_d to deal N' random sharings in $\Sigma^{\times m}$. Suppose the share size of a sharing in Σ is sh field elements in $\mathbb{F}_q$. Then the share size of a sharing in $\Sigma^{\times m}$ is $m \cdot \mathsf{sh}$ field elements in $\mathbb{F}_q$. Recall that the communication complexity of the instantiation of $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ in [GSZ20] is $O(n^2)$ elements in $\mathbb{F}_{q^m}$, which is $O(n^2 \cdot m)$ elements in $\mathbb{F}_q$. The communication complexity of VerShare(P_d, N') is $O(N' \cdot n \cdot m \cdot \mathsf{sh} + n^2 \cdot m)$ elements in $\mathbb{F}_q$.

For a sharing $[s]$ and a nonempty set A, we say that $\pi_A([s])$ is valid if $\Sigma(A, \pi_A([s]))$ is nonempty. For a sharing $[\boldsymbol{s}]$ and a nonempty set A, we say $\pi_A([\boldsymbol{s}])$ is valid if $\Sigma^{\times m}(A, \pi_A([\boldsymbol{s}]))$ (which can be similarly defined) is nonempty.

Protocol 5: VERSHARE(P_d, N')

1. For $\ell = 1, 2, \ldots, N'$, P_d randomly samples a sharing $[\boldsymbol{s}^{(\ell)}]$ in $\Sigma^{\times m}$. P_d distributes the shares of $[\boldsymbol{s}^{(\ell)}]$ to all other parties.
2. P_d randomly samples a sharing $[\boldsymbol{s}^{(0)}]$ in $\Sigma^{\times m}$. P_d distributes the shares of $[\boldsymbol{s}^{(0)}]$ to all other parties. This sharing is used as a random mask when verifying the random sharings generated in Step 1.
3. All parties invoke $\mathcal{F}_{\mathrm{coin}}(\mathbb{F}_{q^m})$ and receive a random field element $\alpha \in \mathbb{F}_{q^m}$. Then, all parties locally compute
$$[\boldsymbol{s}] := [\boldsymbol{s}^{(0)}] + \alpha \cdot [\boldsymbol{s}^{(1)}] + \alpha^2 \cdot [\boldsymbol{s}^{(2)}] + \ldots + \alpha^{N'} \cdot [\boldsymbol{s}^{(N')}].$$
4. Each party P_j sends the j-th share of $[\boldsymbol{s}]$ to all other parties. Then, each party P_j verifies that $[\boldsymbol{s}]$ is a valid sharing in $\Sigma^{\times m}$. If not, P_j aborts. Otherwise, all parties take $\{[\boldsymbol{s}^{(\ell)}]\}_{\ell=1}^{N'}$ as output.

Lemma 2. *Let $\mathcal{H} \subset \mathcal{I}$ denote the set of all honest parties. If all parties accept the verification in the last step of* VERSHARE*(P_d, N'), then the probability that there exists $[\boldsymbol{s}^\star] \in \{[\boldsymbol{s}^{(\ell)}]\}_{\ell=1}^{N'}$ such that $\pi_{\mathcal{H}}([\boldsymbol{s}^\star])$ is invalid is bounded by $N'/2^\kappa$.*

Proof. Suppose that there exists $[\boldsymbol{s}^\star] \in \{[\boldsymbol{s}^{(\ell)}]\}_{\ell=1}^{N'}$ such that $\pi_{\mathcal{H}}([\boldsymbol{s}^\star])$ is invalid. Now we show that the number of $\alpha \in \mathbb{F}_{q^m}$ such that $[\boldsymbol{s}]$ passes the verification in the last step of VERSHARE(P_d, N') is bounded by N'. Then, the lemma follows from that α output by $\mathcal{F}_{\mathrm{coin}}$ is uniformly random in $\mathbb{F}_{q^m}$ and $|\mathbb{F}_{q^m}| = q^m \geq 2^\kappa$. Note that if $[\boldsymbol{s}]$ passes the verification, then $\pi_{\mathcal{H}}([\boldsymbol{s}])$ is valid since $[\boldsymbol{s}] \in \Sigma^{\times m}(\mathcal{H}, \pi_{\mathcal{H}}([\boldsymbol{s}]))$.

Now assume that there are $N'+1$ different values $\alpha_0, \alpha_1, \ldots, \alpha_{N'}$ such that for all $i \in \{0, 1, \ldots N'\}$,
$$[\boldsymbol{s}'_i] := [\boldsymbol{s}^{(0)}] + \alpha_i \cdot [\boldsymbol{s}^{(1)}] + \alpha_i^2 \cdot [\boldsymbol{s}^{(2)}] + \ldots + \alpha_i^{N'} \cdot [\boldsymbol{s}^{(N')}].$$
can pass the verification, which implies that for all $i \in \{0, 1, \ldots, N'\}$, $\pi_{\mathcal{H}}([\boldsymbol{s}'_i])$ is valid. Let $\boldsymbol{M}$ be a matrix of size $(N'+1) \times (N'+1)$ in $\mathbb{F}_{q^m}$ such that $\boldsymbol{M}_{ij} = \alpha_{i-1}^{j-1}$. Then we have
$$([\boldsymbol{s}'_0], [\boldsymbol{s}'_1], \ldots, [\boldsymbol{s}'_{N'}])^{\mathrm{T}} = \boldsymbol{M} \cdot ([\boldsymbol{s}^{(0)}], [\boldsymbol{s}^{(1)}], \ldots, [\boldsymbol{s}^{(N')}])^{\mathrm{T}}.$$
Note that $\boldsymbol{M}$ is a $(N'+1) \times (N'+1)$ Vandermonde matrix, which is invertible. Therefore,
$$([\boldsymbol{s}^{(0)}], [\boldsymbol{s}^{(1)}], \ldots, [\boldsymbol{s}^{(N')}])^{\mathrm{T}} = \boldsymbol{M}^{-1} \cdot ([\boldsymbol{s}'_0], [\boldsymbol{s}'_1], \ldots, [\boldsymbol{s}'_{N'}])^{\mathrm{T}}.$$
Since $\Sigma^{\times m}$ is $\mathbb{F}_{q^m}$-linear, and $\pi_{\mathcal{H}}([\boldsymbol{s}'_i])$ is valid for all $i \in \{0, 1, \ldots, N'\}$ by assumption, we have that $\pi_{\mathcal{H}}([\boldsymbol{s}^{(0)}]), \pi_{\mathcal{H}}([\boldsymbol{s}^{(1)}]), \ldots, \pi_{\mathcal{H}}([\boldsymbol{s}^{(N')}])$ are all valid, which leads to a contradiction. □

Converting to Random Sharings. Let $[\boldsymbol{s}_d]$ denote a sharing in $\Sigma^{\times m}$ dealt by P_d in VERSHARE. We will convert these n sharings, one sharing dealt by each party, to $(t+1)$ random sharings in $\Sigma^{\times m}$. As [BSFO12], this is achieved by making use of the fact that the transpose of a Vandermonde matrix acts as a randomness extractor. The description of CONVERT appears in Protocol 6.

Combining VERSHARE and CONVERT, we have RAND(N) (Protocol 7) which securely computes $\mathcal{F}_{\mathrm{rand}}$. The communication complexity of RAND(N) is $O(N \cdot n \cdot \mathsf{sh} + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

Lemma 3. *Protocol* RAND *securely computes $\mathcal{F}_{rand}$ with abort in the $\mathcal{F}_{coin}$-hybrid model in the presence of a malicious adversary who controls t parties.*

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

Protocol 6: Convert

1. For each party P_d, let $[\boldsymbol{s}_d]$ denote a sharing in $\Sigma^{\times m}$ dealt by P_d in VerShare. Let $\boldsymbol{M}^{\mathrm{T}}$ be an $n \times (t+1)$ Vandermonde matrix in $\mathbb{F}_{q^m}$. Then $\boldsymbol{M}$ is a matrix of size $(t+1) \times n$.
2. All parties compute
$$([\boldsymbol{r}_1], \ldots, [\boldsymbol{r}_{t+1}])^{\mathrm{T}} := \boldsymbol{M} \cdot ([\boldsymbol{s}_1], \ldots, [\boldsymbol{s}_n])^{\mathrm{T}}.$$
3. All parties take $[\boldsymbol{r}_1], \ldots, [\boldsymbol{r}_{t+1}]$ as output.

Protocol 7: Rand(N)

1. Let $N' = \frac{N}{m(t+1)}$. For each party P_d, all parties invoke VerShare(P_d, N'). Let $\{[\boldsymbol{s}_d^{(\ell)}]\}_{\ell=1}^{N'}$ denote the output.
2. For each $\ell \in \{1, \ldots, N'\}$, all parties invoke Convert on $\{[\boldsymbol{s}_d^{(\ell)}]\}_{d=1}^{n}$. Let $\{[\boldsymbol{r}_i^{(\ell)}]\}_{i=1}^{t+1}$ denote the output.
3. For each sharing $[\boldsymbol{r}_i^{(\ell)}]$, all parties separate it into m sharings in Σ. Note that there are in total $N' \cdot (t+1) \cdot m = N$ sharings in Σ.

Simulation for VerShare. We first consider the case where P_d is an honest party.

- In Step 1 and Step 2, P_d needs to distribute random sharings $\{[\boldsymbol{s}^{(\ell)}]\}_{\ell=0}^{N'}$ in $\Sigma^{\times m}$. For each sharing $[\boldsymbol{s}^{(\ell)}]$, $\mathcal{S}$ samples a random sharing $[\boldsymbol{s}^{(\ell)}] \in \Sigma^{\times m}$ and sends the shares of corrupted parties $\pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}])$ to $\mathcal{A}$.
- In Step 3, $\mathcal{S}$ emulates $\mathcal{F}_{\mathrm{coin}}$ and generates a random field element $\alpha \in \mathbb{F}_{q^m}$. Then, $\mathcal{S}$ computes the shares of $[\boldsymbol{s}]$ held by corrupted parties, i.e., $\pi_{\mathcal{C}}([\boldsymbol{s}])$. Based on $\pi_{\mathcal{C}}([\boldsymbol{s}])$, $\mathcal{S}$ randomly samples $[\boldsymbol{s}] \in \Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}]))$.
- In Step 4, $\mathcal{S}$ faithfully follows the protocol since the shares of $[\boldsymbol{s}]$ held by honest parties have been explicitly generated.

When P_d is corrupted, $\mathcal{S}$ simply follows the protocol. If there exists some $[\boldsymbol{s}^{(\ell)}]$ such that $\pi_{\mathcal{H}}([\boldsymbol{s}^{(\ell)}])$ is invalid, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\mathrm{rand}}$ and aborts in Step 4 (even if the verification passes). Otherwise, for each sharing $[\boldsymbol{s}^{(\ell)}]$ dealt by P_d, $\mathcal{S}$ randomly samples $[\tilde{\boldsymbol{s}}^{(\ell)}] \in \Sigma^{\times m}(\mathcal{H}, \pi_{\mathcal{H}}([\boldsymbol{s}^{(\ell)}]))$, and views it as the sharing dealt by P_d.

If some party aborts at the end of VerShare, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\mathrm{rand}}$ and aborts.

Simulation for Convert. Recall that Convert only involves local computation. For each sharing $[\boldsymbol{s}_d]$, $\mathcal{S}$ has computed $\pi_{\mathcal{C}}([\boldsymbol{s}_d])$ in the simulation of VerShare. In Convert, $\mathcal{S}$ computes $\pi_{\mathcal{C}}([\boldsymbol{r}_i])$ and sends them to $\mathcal{F}_{\mathrm{rand}}$. Then $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\mathrm{rand}}$.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Consider the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ simulates VerShare for honest parties when the dealer P_d is corrupted. Note that, $\mathcal{S}$ simply follows the protocol in this case and computes the shares held by corrupted parties. The only difference is that $\mathcal{S}$ will abort if there exists some $[\boldsymbol{s}^{(\ell)}]$ such that $\pi_{\mathcal{H}}([\boldsymbol{s}^{(\ell)}])$ is invalid even if the verification in Step 4 passes. According to Lemma 2, this happens with negligible probability. Therefore, the distribution of $\mathbf{Hybrid}_1$ is statistically close to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ first simulates VerShare for honest parties when the dealer P_d is honest. Then, for each $\ell \in \{1, \ldots, N'\}$, $\mathcal{S}$ re-samples a new random sharing $[\tilde{\boldsymbol{s}}^{(\ell)}] \in \Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}]))$. $\mathcal{S}$ takes $\{[\tilde{\boldsymbol{s}}^{(\ell)}]\}_{\ell=1}^{N'}$ as the sharings dealt by P_d.

Note that in Step 1 and Step 2, $\mathcal{A}$ only receives $\pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}])$. Therefore, $[\boldsymbol{s}^{(\ell)}]$ is a random sharing in $\Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}]))$. In Step 3, after receiving $\alpha \in \mathbb{F}_{q^m}$ from $\mathcal{F}_{\text{coin}}$, all parties compute

$$[\boldsymbol{s}] := [\boldsymbol{s}^{(0)}] + \alpha \cdot [\boldsymbol{s}^{(1)}] + \alpha^2 \cdot [\boldsymbol{s}^{(2)}] + \ldots + \alpha^{N'} \cdot [\boldsymbol{s}^{(N')}].$$

Therefore, $[\boldsymbol{s}]$ is a random sharing in $\Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}]))$. Note that $[\boldsymbol{s}]$ is masked by a random sharing $[\boldsymbol{s}^{(0)}]$. Thus, $[\boldsymbol{s}]$ is independent of $\{[\boldsymbol{s}^{(\ell)}]\}_{\ell=1}^{N'}$. Therefore, given the view of $\mathcal{A}$, each sharing $[\boldsymbol{s}^{(\ell)}]$ is a random sharing in $\Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}]))$. This means that $\mathcal{S}$ can re-sample and use a new random sharing $[\tilde{\boldsymbol{s}}^{(\ell)}] \in \Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{s}^{(\ell)}]))$ instead of using the sharing $[\boldsymbol{s}^{(\ell)}]$ generated in the beginning.

Thus, the distribution of $\mathbf{Hybrid}_2$ is the same as the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ does not re-sample the sharings $\{[\tilde{\boldsymbol{s}}^{(\ell)}]\}_{\ell=1}^{N'}$. Instead, $\mathcal{S}$ simulates Convert for honest parties, which does not need to generate the whole sharing $[\tilde{\boldsymbol{s}}^{(\ell)}]$.

Let $\boldsymbol{M}_{\mathcal{C}}$ denote the sub-matrix of $\boldsymbol{M}$ containing columns with indices in $\mathcal{C}$, and $\boldsymbol{M}_{\mathcal{H}}$ denote the sub-matrix containing columns with indices in $\mathcal{H}$. We have

$$([\boldsymbol{r}_1], \ldots, [\boldsymbol{r}_{t+1}])^{\mathrm{T}} = \boldsymbol{M} \cdot ([\boldsymbol{s}_1], \ldots, [\boldsymbol{s}_n])^{\mathrm{T}} = \boldsymbol{M}_{\mathcal{C}} \cdot ([\boldsymbol{s}_j])_{j\in\mathcal{C}}^{\mathrm{T}} + \boldsymbol{M}_{\mathcal{H}} \cdot ([\boldsymbol{s}_j])_{j\in\mathcal{H}}^{\mathrm{T}}.$$

Let

$$([\boldsymbol{r}_1^{(\mathcal{H})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{H})}])^{\mathrm{T}} := \boldsymbol{M}_{\mathcal{H}} \cdot ([\boldsymbol{s}_j])_{j\in\mathcal{H}}^{\mathrm{T}}$$
$$([\boldsymbol{r}_1^{(\mathcal{C})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{C})}])^{\mathrm{T}} := \boldsymbol{M}_{\mathcal{C}} \cdot ([\boldsymbol{s}_j])_{j\in\mathcal{C}}^{\mathrm{T}}.$$

Then $([\boldsymbol{r}_1], \ldots, [\boldsymbol{r}_{t+1}]) = ([\boldsymbol{r}_1^{(\mathcal{C})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{C})}]) + ([\boldsymbol{r}_1^{(\mathcal{H})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{H})}])$.

Recall that $\boldsymbol{M}^{\mathrm{T}}$ is a Vandermonde matrix of size $n \times (t+1)$. Therefore $\boldsymbol{M}_{\mathcal{H}}^{\mathrm{T}}$ is a Vandermonde matrix of size $(t+1) \times (t+1)$, which is invertible. There is a one-to-one map from $([\boldsymbol{r}_1^{(\mathcal{H})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{H})}])$ to $([\boldsymbol{s}_j])_{j\in\mathcal{H}}$. Given $([\boldsymbol{s}_j])_{j\in\mathcal{C}}$, there is a one-to-one map from $([\boldsymbol{r}_1], \ldots, [\boldsymbol{r}_{t+1}])$ to $([\boldsymbol{r}_1^{(\mathcal{H})}], \ldots, [\boldsymbol{r}_{t+1}^{(\mathcal{H})}])$. Recall that for each sharing $[\boldsymbol{s}_d]$ dealt by a corrupted party P_d, $\mathcal{S}$ received the shares of honest parties $\pi_{\mathcal{H}}([\boldsymbol{s}_d])$ and sampled a random sharing $[\tilde{\boldsymbol{s}}_d] \in \Sigma^{\times m}(\mathcal{H}, \pi_{\mathcal{H}}([\boldsymbol{s}_d]))$. Note that this may not be the sharing dealt by P_d since we do not know the shares held by corrupted parties. However, we show that this does not affect the distribution of the shares of honest parties generated by $\mathcal{F}_{\text{rand}}$.

To see this, note that for each valid $([\tilde{\boldsymbol{s}}_j])_{j\in\mathcal{C}}$, $\mathcal{S}$ computes $\{\pi_{\mathcal{C}}([\boldsymbol{r}_j])\}_{j=1}^{t+1}$ and sends them to $\mathcal{F}_{\text{rand}}$. Then for each $j \in \{1, \ldots, t+1\}$, $\mathcal{F}_{\text{rand}}$ samples a random sharing $[\tilde{\boldsymbol{r}}_j] \in \Sigma^{\times m}(\mathcal{C}, \pi_{\mathcal{C}}([\boldsymbol{r}_j]))$. These random sharings $\{[\tilde{\boldsymbol{r}}_j]\}_{j=1}^{t+1}$ correspond to random sharings $([\tilde{\boldsymbol{s}}_j])_{j\in\mathcal{H}}$, which are independent of $([\tilde{\boldsymbol{s}}_j])_{j\in\mathcal{C}}$. Thus, the distribution of $\mathbf{Hybrid}_3$ is the same as the distribution of $\mathbf{Hybrid}_2$.

Note that $\mathbf{Hybrid}_3$ is the execution in the ideal world and $\mathbf{Hybrid}_3$ is statistically close to $\mathbf{Hybrid}_0$, the execution in the real world. □

5 Hidden Additive Secret Sharing

Let (ϕ, ψ) be an $(k, m)_q$-RMFE. Recall that n denotes the number of parties and $\phi : \mathbb{F}_q^k \to \mathbb{F}_{q^m}$ is an $\mathbb{F}_q$-linear map. Recall that $|\mathbb{F}_{q^m}| = q^m \geq 2^\kappa \geq n+1$. Thus, the Shamir secret sharing scheme is well-defined in $\mathbb{F}_{q^m}$. In our construction, we will use ϕ to encode a vector $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(k)}) \in \mathbb{F}_q^k$. All parties will hold a degree-t Shamir sharing of $\phi(\boldsymbol{x})$, denoted by $[\phi(\boldsymbol{x})]_t$.

Defining Additive Sharings and Couple Sharings. For $x \in \mathbb{F}_q$, we use $\langle x \rangle$ to denote an additive sharing of x among the first $t+1$ parties in $\mathbb{F}_q$. Specifically, $\langle x \rangle = (x_1, \ldots, x_n)$ where the party P_i holds the share $x_i \in \mathbb{F}_q$ such that $x = \sum_{i=1}^{t+1} x_i$ and the last t shares $x_{t+2}, \ldots, x_n$ are all 0.

Recall that $\psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^k$ is an $\mathbb{F}_q$-linear map. For all $y \in \mathbb{F}_{q^m}$, if $\psi(y) = (y_1, y_2, \ldots, y_k)$, we define $\mathsf{val}(y) := \sum_{i=1}^{k} y_i$. Note that $\mathsf{val}(\cdot)$ is an $\mathbb{F}_q$-linear map from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$. We say a pair of sharings $(\langle x \rangle, [y]_t)$ is a pair of *couple sharings* if

– $\langle x \rangle$ is an additive sharing of $x \in \mathbb{F}_q$;
– $[y]_t$ is a degree-t Shamir sharing of $y \in \mathbb{F}_{q^m}$;
– $\mathsf{val}(y) = x$.

In the following, we will use $[\![x]\!] := (\langle x \rangle, [y]_t)$ to denote a pair of couple sharings of $x \in \mathbb{F}_q$. Note that for the additive sharing $\langle x \rangle$, a corrupted party in the first $t+1$ parties can easily change the secret by changing its own share. However, the secret of $[y]_t$ is determined by the shares of honest parties and cannot be altered by corrupted parties. Therefore, $[y]_t$ can be seen as a robust version of the sharing $\langle x \rangle$.

Properties of Couple Sharings. We note that couple sharings are $\mathbb{F}_q$-linear. Concretely, for all couple sharings $[\![x]\!] = (\langle x \rangle, [y]_t)$ and $[\![x']\!] = (\langle x' \rangle, [y']_t)$, and for all $\alpha, \beta \in \mathbb{F}_q$, the linear combination

$$\alpha \cdot [\![x]\!] + \beta \cdot [\![x']\!] := (\alpha \cdot \langle x \rangle + \beta \cdot \langle x' \rangle, \alpha \cdot [y]_t + \beta \cdot [y']_t)$$

is still a pair of couple sharings. This property follows from the fact that $\mathsf{val}(\cdot)$ is an $\mathbb{F}_q$-linear map.

We can also define the addition operation between a pair of couple sharings $[\![x]\!]$ and a field element x' in $\mathbb{F}_q$. This is done by transforming x' to a pair of couple sharings of x'. For $\langle x' \rangle$, we set the share of the first party to be x', and the shares of the rest of parties to be 0. For the degree-t Shamir sharing, we first need to find $y' \in \mathbb{F}_{q^m}$ such that $\mathsf{val}(y') = x'$. This is done by choosing two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^k$ such that:

– For $\boldsymbol{a}$, the first entry is 1 and the rest of entries are 0.
– For $\boldsymbol{b}$, the first entry is x' and the rest of entries are 0.

By the property of RMFE, $\psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})) = \boldsymbol{a} * \boldsymbol{b}$. In particular, the first entry of $\boldsymbol{a} * \boldsymbol{b}$ is x' and the rest of entries are 0. Therefore $y' := \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})$ satisfies that $\mathsf{val}(y') = x'$. For $[y']_t$, we set the share of each party to be y'. Finally, the addition operation between $[\![x]\!]$ and $x' \in \mathbb{F}_q$ is defined by

$$[\![x]\!] + x' := (\langle x \rangle, [y]_t) + (\langle x' \rangle, [y']_t).$$

Generating Couple Sharings from $[\phi(\boldsymbol{x})]_t$. In this part, we show how to *non-interactively* obtain k pairs of couple sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(k)}]\!]$ from a degree-t Shamir sharing $[\phi(\boldsymbol{x})]_t$, where $\boldsymbol{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(k)}) \in \mathbb{F}_q^k$. It allows us to prepare k pairs of random couple sharings with the cost of preparing one random sharing $[\phi(\boldsymbol{x})]_t$.

We first show how to obtain $[y^{(i)}]_t$ such that $\mathsf{val}(y^{(i)}) = x^{(i)}$ for all $i \in [k]$. Let $\boldsymbol{e}^{(i)}$ be a vector in $\mathbb{F}_q^k$ such that all entries are 0 except that the i-th entry is 1. By the property of RMFE, we have

$$\psi(\phi(\boldsymbol{e}^{(i)}) \cdot \phi(\boldsymbol{x})) = \boldsymbol{e}^{(i)} * \boldsymbol{x}.$$

For $\boldsymbol{e}^{(i)} * \boldsymbol{x}$, all entries are 0 except that the i-th entry is $x^{(i)}$. Therefore by the definition of $\mathsf{val}(\cdot)$, we have $\mathsf{val}(\phi(\boldsymbol{e}^{(i)}) \cdot \phi(\boldsymbol{x})) = x^{(i)}$. To obtain $[y^{(i)}]_t$, all parties compute

$$[y^{(i)}]_t := \phi(\boldsymbol{e}^{(i)}) \cdot [\phi(\boldsymbol{x})]_t.$$

Now we show how to obtain $\langle x^{(i)} \rangle$ from $[\phi(\boldsymbol{x})]$. Let $\langle \boldsymbol{x} \rangle := (\langle x^{(1)} \rangle, \ldots, \langle x^{(k)} \rangle)$ denote a vector of additive sharings of $\boldsymbol{x} \in \mathbb{F}_q^k$. For each party, its share of $\langle \boldsymbol{x} \rangle$ is a vector in $\mathbb{F}_q^k$. For the last t parties, they take the all-0 vector as their shares.

Recall that the degree-t Shamir sharing $[\phi(\boldsymbol{x})]_t$ corresponds to a degree-t polynomial $f(\cdot) \in \mathbb{F}_{q^m}[X]$ such that $f(\alpha_i)$ is the share of the i-th party P_i and $f(0) = \phi(\boldsymbol{x})$, where $\alpha_1, \ldots, \alpha_n$ are distinct non-zero elements in $\mathbb{F}_{q^m}$. In particular, relying on Lagrange interpolation, $f(0)$ can be written as a linear combination of the first $t+1$ shares. For $i \in \{1, \ldots, t+1\}$, let $c_i = \prod_{j \neq i, j \in [t+1]} \frac{\alpha_j}{\alpha_j - \alpha_i}$. We have

$$f(0) = \sum_{i=1}^{t+1} c_i f(\alpha_i).$$

Therefore, the Shamir sharing $[\phi(\boldsymbol{x})]_t$ can be locally converted to an additive sharing of $\phi(\boldsymbol{x})$ among the first $t+1$ parties by letting P_i take $c_i f(\alpha_i)$ as its share. For each $i \in \{1, \ldots, t+1\}$, P_i locally applies $\tilde{\phi}^{-1}(c_i f(\alpha_i))$, which outputs a vector in $\mathbb{F}_q^k$. It is sufficient to show that these $t+1$ shares correspond to an additive sharing of $\boldsymbol{x}$. Note that

$$\sum_{i=1}^{t+1} \tilde{\phi}^{-1}(c_i f(\alpha_i)) = \tilde{\phi}^{-1}(\sum_{i=1}^{t+1} c_i f(\alpha_i)) = \tilde{\phi}^{-1}(f(0)) = \boldsymbol{x}.$$

The description of SEPARATE appears in Protocol 8.

Protocol 8: SEPARATE$([\phi(\boldsymbol{x})]_t)$

1. For all $i \in [k]$, let $\boldsymbol{e}^{(i)}$ be a vector in $\mathbb{F}_q^k$ such that all entries are 0 except that the i-th entry is 1. All parties locally compute $[y^{(i)}]_t := \phi(\boldsymbol{e}^{(i)}) \cdot [\phi(\boldsymbol{x})]_t$.
2. Let $\alpha_1, \ldots, \alpha_n$ be n distinct elements in $\mathbb{F}_{q^m}$ defined in the Shamir secret sharing scheme.
   - For each $i \in \{1, \ldots, t+1\}$, P_i locally computes $c_i = \prod_{j \neq i, j \in [t+1]} \frac{\alpha_j}{\alpha_j - \alpha_i}$. Let $f(\alpha_i)$ denote the i-th share of $[\phi(\boldsymbol{x})]_t$. P_i locally computes $\tilde{\phi}^{-1}(c_i f(\alpha_i))$ and regards the result as the i-th share of $\langle \boldsymbol{x} \rangle = (\langle x^{(1)} \rangle, \ldots, \langle x^{(k)} \rangle)$.
   - For each $i \in \{t+2, \ldots, n\}$, P_i takes the all-0 vector as its share of $\langle \boldsymbol{x} \rangle$.
3. For all $i \in [k]$, all parties set $[\![x^{(i)}]\!] := (\langle x^{(i)} \rangle, [y^{(i)}]_t)$. All parties take the following k pairs of couple sharings as output:

$$[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(k)}]\!]$$

6 Building Blocks for Preprocessing Phase

In this section, we will introduce 4 functionalities which are used to prepare necessary correlated-randomness for the computation.

- The first functionality $\mathcal{F}_{\text{random}}$ allows all parties to prepare random sharings in the form of $[\phi(\boldsymbol{r})]_t$, where (ϕ, ψ) is a RMFE, and $\boldsymbol{r}$ is a random vector in $\mathbb{F}_q^k$.
- The second functionality $\mathcal{F}_{\text{tuple}}$ allows all parties to prepare random tuple of sharings in the form of $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_q^k$, and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. For each tuple, relying on SEPARATE, all parties can locally obtain k multiplication tuples in the form of $([\![a]\!], [\![b]\!], [\![c]\!])$, where a, b are random elements in $\mathbb{F}_q$, and $c = a \cdot b$. Such a multiplication tuple is referred to as a Beaver tuple. In the online phase, one Beaver tuple will be consumed to compute a multiplication gate.
- Recall that we use $\langle x \rangle$ to denote an additive sharing of $x \in \mathbb{F}_q$ among the first $t+1$ parties, and the shares of the rest of parties are 0. The third functionality $\mathcal{F}_{\text{zero}}$ allows all parties to prepare random additive sharings of 0. When evaluating a multiplication gate in the online phase, we will use random additive sharings of 0 to protect the shares of honest parties.
- Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map, defined by $\mathsf{val}(y) = \sum_{i=1}^k y_i$, where $(y_1, y_2, \ldots, y_k) = \psi(y)$. The last functionality $\mathcal{F}_{\text{parity}}$ allows all parties to prepare random sharings in the form of $[p]_t$, where $\mathsf{val}(p) = 0$. These random sharings are used at the end of the protocol to verify the computation.

6.1 Preparing Random Sharings

In this part, we introduce the functionality to let all parties prepare random sharings in the form of $[\phi(\boldsymbol{r})]_t$. Recall that (ϕ, ψ) is an $(k, m)_q$-RMFE. Here each $[\phi(\boldsymbol{r})]_t$ is a random degree-t Shamir sharing of the secret $\phi(\boldsymbol{r})$ where $\boldsymbol{r}$ is a random vector in $\mathbb{F}_q^k$. The description of $\mathcal{F}_{\text{random}}$ appears in Functionality 9.

Functionality 9: $\mathcal{F}_{\text{random}}$

1. $\mathcal{F}_{\text{random}}$ receives $\{s_i\}_{i\in\mathcal{C}}$ from the adversary, where $\mathcal{C}$ is the set of corrupted parties. Then $\mathcal{F}_{\text{random}}$ randomly samples $\boldsymbol{r} \in \mathbb{F}_q^k$ and generates a degree-t Shamir sharing $[\phi(\boldsymbol{r})]_t$ such that the share of $P_i \in \mathcal{C}$ is s_i.
2. $\mathcal{F}_{\text{random}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{random}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, for each honest party P_i, $\mathcal{F}_{\text{random}}$ sends the i-th share of $[\phi(\boldsymbol{r})]_t$ to P_i.

Consider a secret sharing scheme Σ which takes a vector $\boldsymbol{x} \in \mathbb{F}_q^k$ and outputs a Shamir sharing of $\phi(\boldsymbol{x})$ in $\mathbb{F}_{q^m}$, i.e., $[\phi(\boldsymbol{x})]_t$. Note that the Shamir secret sharing scheme is linear in $\mathbb{F}_{q^m}$ and ϕ is linear in $\mathbb{F}_q$. Therefore Σ is an $\mathbb{F}_q$-GLSSS. To use $\mathcal{F}_{\text{rand}}$ to prepare random sharings in Σ, we need to show that:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, there exists an efficient algorithm which outputs that either $\Sigma(A, (a_i)_{i\in A}) = \emptyset$, or a random sharing $[\phi(\boldsymbol{r})]_t \in \Sigma(A, (a_i)_{i\in A})$.

Depending on the size of A, there are two cases:

- If $|A| \geq t+1$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ can fully determine the whole sharing if exists. The algorithm checks whether the shares $\{a_i\}_{i\in A}$ lie on a polynomial $f(\cdot)$ of degree at most t in $\mathbb{F}_{q^m}$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm checks whether $f(0)$, i.e., the secret of this Shamir sharing, is in the image of ϕ. This can be done by checking whether $\phi(\tilde{\phi}^{-1}(f(0))) = f(0)$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs the Shamir sharing determined by $\{a_i\}_{i\in A}$.
- If $|A| \leq t$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ is independent of the secret. Therefore, $\Sigma(A, (a_i)_{i\in A}) \neq \emptyset$. The algorithm randomly samples $\boldsymbol{r} \in \mathbb{F}_q^k$ and randomly samples $t-|A|$ elements in $\mathbb{F}_{q^m}$ as the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$. Then, based on the secret $\phi(\boldsymbol{r})$, the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$, and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the whole Shamir sharing $[\phi(\boldsymbol{r})]_t$ and outputs $[\phi(\boldsymbol{r})]_t$.

Thus, $\mathcal{F}_{\text{random}}$ can be instantiated by $\mathcal{F}_{\text{rand}}$ with the secret sharing scheme Σ defined above. Note that the share size of a sharing in Σ is $\mathsf{sh} = m$ elements in $\mathbb{F}_q$. Therefore, when using $\textsc{Rand}(N)$ to instantiate $\mathcal{F}_{\text{rand}}$, the communication complexity of generating N random sharings in Σ is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

6.2 Preparing Beaver Tuples

In this part, we show how to prepare random tuples of sharings in $\mathbb{F}_{q^m}$ in the form of $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_q^k$, and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. The description of $\mathcal{F}_{\text{tuple}}$ appears in Functionality 10.

In the online phase, each tuple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ will be separated by $\textsc{Separate}$ (Protocol 8) to k Beaver tuples

$$([\![a^{(1)}]\!], [\![b^{(1)}]\!], [\![c^{(1)}]\!]), ([\![a^{(2)}]\!], [\![b^{(2)}]\!], [\![c^{(2)}]\!]), \ldots, ([\![a^{(k)}]\!], [\![b^{(k)}]\!], [\![c^{(k)}]\!]).$$

A Beaver tuple $([\![a^{(i)}]\!], [\![b^{(i)}]\!], [\![c^{(i)}]\!])$ satisfies that $a^{(i)}, b^{(i)}$ are random elements in $\mathbb{F}_q$ and $c^{(i)} = a^{(i)} \cdot b^{(i)}$. A multiplication gate is then evaluated by consuming one Beaver tuple. More details can be found in Section 7.2.

We will prepare these random tuples of sharings as follows:

1. All parties first prepare random tuples of sharings in the form of

   $$([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})]_t)$$

   where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_q^k$. This can be done by first invoking $\mathcal{F}_{\text{random}}$ to prepare random sharings $[\phi(\boldsymbol{a})]_t$ and $[\phi(\boldsymbol{b})]_t$. Then all parties inovke $\mathcal{F}_{\text{mult}}$ to compute $[\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})]_t$. Finally, all parties check the correctness of these tuples by invoking $\mathcal{F}_{\text{multVerify}}$.

Functionality 10: $\mathcal{F}_{\text{tuple}}$

1. $\mathcal{F}_{\text{tuple}}$ receives $\{(u_i, v_i, w_i)\}_{i\in\mathcal{C}}$ from the adversary, where $\mathcal{C}$ is the set of corrupted parties. Then $\mathcal{F}_{\text{tuple}}$ randomly samples $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^k$ and computes $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Finally, $\mathcal{F}_{\text{tuple}}$ generates 3 degree-t Shamir sharings $[\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t$ such that the shares of $P_i \in \mathcal{C}$ are u_i, v_i, w_i respectively.
2. $\mathcal{F}_{\text{tuple}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{tuple}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, for each honest party P_i, $\mathcal{F}_{\text{tuple}}$ sends the i-th shares of $[\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t$ to P_i.

2. For each tuple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})]_t)$ prepared in the first step, we need to transform $[\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})]_t$ to $[\phi(\boldsymbol{c})]_t$ where $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. By the property of RMFE, we have $\boldsymbol{c} = \psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}))$. Therefore, if we set $x := \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b})$, the task becomes to transform a sharing $[x]_t$ to $[\phi(\psi(x))]_t$. The functionality of this step is described in $\mathcal{F}_{\text{re-encode}}$ (Functionality 11).

Functionality 11: $\mathcal{F}_{\text{re-encode}}$

1. Let $[x]_t$ denote the input sharing. $\mathcal{F}_{\text{re-encode}}$ receives from honest parties their shares of $[x]_t$. Then $\mathcal{F}_{\text{re-encode}}$ reconstructs the secret x. $\mathcal{F}_{\text{re-encode}}$ further computes the shares of $[x]_t$ held by corrupted parties, and sends these shares to the adversary.
2. $\mathcal{F}_{\text{re-encode}}$ receives from the adversary a set of shares $\{y_i\}_{i\in\mathcal{C}}$, where $\mathcal{C}$ is the set of corrupted parties.
3. $\mathcal{F}_{\text{re-encode}}$ computes $y := \phi(\psi(x))$ and generates a degree-t Shamir sharing $[y]_t$ such that the share of $[y]_t$ held by $P_i \in \mathcal{C}$ is y_i.
4. $\mathcal{F}_{\text{re-encode}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{re-encode}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{re-encode}}$ distributes the shares of $[y]_t$ to honest parties.

Realization of $\mathcal{F}_{\text{re-encode}}$. We follow the idea in [CCXY18] to realize $\mathcal{F}_{\text{re-encode}}$. Concretely, for an input sharing $[x]_t$, all parties first prepare a pair of random sharings $([r]_t, [\phi(\psi(r))]_t)$. Then, all parties reconstruct the sharing $[x + r]_t := [x]_t + [r]_t$. Since both ϕ and ψ are $\mathbb{F}_q$-linear, all parties can compute $[\phi(\psi(x))]_t := \phi(\psi(x + r)) - [\phi(\psi(r))]_t$.

Preparing random correlated sharings. We will use $\mathcal{F}_{\text{rand}}$ to prepare the random sharings $([r]_t, [\phi(\psi(r))]_t)$. Consider a secret sharing scheme Σ which takes an element $r \in \mathbb{F}_{q^m}$ and outputs a pair of two sharings in $\mathbb{F}_{q^m}$, where the first one is a degree-t Shamir sharing of r, and the second one is a degree-t Shamir sharing of $\phi(\psi(r))$. Note that the Shamir secret sharing scheme is linear in $\mathbb{F}_{q^m}$, and ϕ, ψ are linear in $\mathbb{F}_q$. Therefore Σ is an $\mathbb{F}_q$-GLSSS. To use $\mathcal{F}_{\text{rand}}$ to prepare random sharings in Σ, we need to show that:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, there exists an efficient algorithm which outputs that either $\Sigma(A, (a_i)_{i\in A}) = \emptyset$, or a random sharing $([r]_t, [\phi(\psi(r))]_t) \in \Sigma(A, (a_i)_{i\in A})$. Note that each share a_i is a pair of elements $(a_i^{(0)}, a_i^{(1)})$ in $\mathbb{F}_{q^m}$.

  Depending on the size of A, there are two cases:

- If $|A| \geq t+1$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ can fully determine the whole sharings if exist. The algorithm parses each share a_i to $(a_i^{(0)}, a_i^{(1)})$. Then, the algorithm checks whether the shares $\{a_i^{(0)}\}_{i\in A}$ lie on a polynomial $f(\cdot)$ of degree at most t in $\mathbb{F}_{q^m}$, and, the shares $\{a_i^{(1)}\}_{i\in A}$ lie on a polynomial $g(\cdot)$ of degree at most t in $\mathbb{F}_{q^m}$. If not, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm checks whether $\phi(\psi(f(0))) = g(0)$. If not, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs the two Shamir sharings determined by $\{a_i\}_{i\in A}$.
- If $|A| \leq t$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ is independent of the secrets. Therefore, $\Sigma(A,(a_i)_{i\in A}) \neq \emptyset$. The algorithm randomly samples $r \in \mathbb{F}_{q^m}$ and randomly samples $t-|A|$ pairs of elements in $\mathbb{F}_{q^m}$ as the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$. Then, based on the secrets $r, \phi(\psi(r))$, the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$, and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the two Shamir sharings $[r]_t, [\phi(\psi(r))]_t$ and outputs $([r]_t, [\phi(\psi(r))]_t)$.

Thus, we can use $\mathcal{F}_{\text{rand}}$ with the secret sharing scheme Σ to prepare random sharings $([r]_t, [\phi(\psi(r))]_t)$. Note that the share size of a sharing in Σ is $\mathsf{sh} = 2m$ elements in $\mathbb{F}_q$. Therefore, when using $\textsc{Rand}(N)$ to instantiate $\mathcal{F}_{\text{rand}}$, the communication complexity of preparing N random sharings in Σ is $O(N\cdot n\cdot m + n^3\cdot m)$ elements in $\mathbb{F}_q$.

Reconstructing Shamir Sharings with Verification. The second step is to reconstruct a batch of N degree-t Shamir sharings. The description of $\mathcal{F}_{\text{open}}$ appears in Functionality 12.

Functionality 12: $\mathcal{F}_{\text{open}}$

1. Let $[x]_t$ denote the input sharing. $\mathcal{F}_{\text{open}}$ receives from honest parties their shares of $[x]_t$. Then $\mathcal{F}_{\text{open}}$ reconstructs the secret x. $\mathcal{F}_{\text{open}}$ further computes the shares of $[x]_t$ held by corrupted parties, and sends the shares to the adversary.
2. $\mathcal{F}_{\text{open}}$ sends the reconstruction result x to the adversary.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{open}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{open}}$ sends x to honest parties.

The reconstruction is done by asking the first party P_1 to collect all the shares of $[x]_t$, reconstruct the secret x, and send x to all other parties. To check the correctness of reconstructions, all parties will together examine a random linear combination of all reconstructed sharings. The description of $\textsc{Open}$ appears in Protocol 13. Regarding the communication complexity, reconstructing a single sharing in Step 1 requires the communication of $O(n\cdot m)$ elements in $\mathbb{F}_q$. In Step 2, recall that the instantiation of $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ in [GSZ20] requires the communication of $O(n^2\cdot m)$ elements in $\mathbb{F}_q$. In Step 3, all parties need to communicate $O(n^2\cdot m)$ elements in $\mathbb{F}_q$. Therefore, the communication complexity of $\textsc{Open}(N, \{[x^{(\ell)}]_t\}_{i\in[N]})$ is $O(N\cdot n\cdot m + n^2\cdot m)$ elements in $\mathbb{F}_q$.

Lemma 4. *Protocol* $\textsc{Open}$ *securely computes* $\mathcal{F}_{open}$ *with abort in the* $\mathcal{F}_{coin}$*-hybrid model in the presence of a malicious adversary who controls* t *parties.*

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

Simulation for $\textsc{Open}$. At the beginning, for each $\ell \in [N]$, $\mathcal{S}$ receives the shares of $[x^{(\ell)}]_t$ held by corrupted parties and the reconstruction result $x^{(\ell)}$ from $\mathcal{F}_{\text{open}}$. Using the shares of corrupted parties and the secret, $\mathcal{S}$ can reconstruct the whole sharing $[x^{(\ell)}]_t$.

Protocol 13: $\textsc{Open}(N, \{[x^{(\ell)}]_t\}_{i\in[N]})$

1. For each $\ell \in [N]$, P_1 receives from all other parties their shares of $[x^{(\ell)}]_t$. Then P_1 checks whether the shares of $[x^{(\ell)}]_t$ lie on a polynomial of degree at most t in $\mathbb{F}_{q^m}$. If not, P_1 aborts. Otherwise, P_1 reconstructs the secret $x^{(\ell)}$ and sends $x^{(\ell)}$ to all other parties.
2. All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ to generate a random element $r \in \mathbb{F}_{q^m}$. All parties locally compute
$$[x]_t := [x^{(1)}]_t + [x^{(2)}]_t \cdot r + \ldots + [x^{(N)}]_t \cdot r^{N-1},$$
$$x := x^{(1)} + x^{(2)} \cdot r + \ldots + x^{(N)} \cdot r^{N-1}.$$
3. For each party P_i, P_i receives from all other parties their shares of $[x]_t$. Then P_i checks whether the shares of $[x]_t$ lie on a polynomial of degree at most t in $\mathbb{F}_{q^m}$. If not, P_i aborts. Otherwise, P_i reconstructs the secret x and compares it with the value computed in Step 2. If they are the same, P_i accepts the reconstructions. Otherwise, P_i aborts.

In Step 1, $\mathcal{S}$ honestly sends the shares of honest parties to P_1. If P_1 is an honest party, $\mathcal{S}$ faithfully follows the protocol by checking the consistency of the sharings. If any sharing is inconsistent, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{open}}$ and aborts the protocol. Otherwise, $\mathcal{S}$ sends the reconstruction results received from $\mathcal{F}_{\text{open}}$ to corrupted parties. If P_1 is a corrupted party, $\mathcal{S}$ receives from P_1 the reconstruction results. If P_1 aborts, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{open}}$ and aborts.

In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling $r \in \mathbb{F}_{q^m}$. Note that $\mathcal{S}$ knows all the sharings. Therefore, $\mathcal{S}$ faithfully computes $[x]_t$ and x.

In Step 3, $\mathcal{S}$ honestly follows the protocol. If any party aborts, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{open}}$ and aborts. Finally, $\mathcal{S}$ checks whether the reconstruction results received from P_1 are the same as those received from $\mathcal{F}_{\text{open}}$. If not, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{open}}$ and aborts. Otherwise, $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\text{open}}$.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Conisder the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ simulates the first step as described above. Note that the main difference is that, in $\mathbf{Hybrid}_0$, $\mathcal{S}$ uses the real shares of honest parties, while in $\mathbf{Hybrid}_1$, $\mathcal{S}$ computes the shares of honest parties from the shares of corrupted parties and the secrets received from $\mathcal{F}_{\text{open}}$. However, this makes no difference since the shares of honest parties are determined by the secrets and the shares held by corrupted parties.

The other difference is that, when P_1 is an honest party and all the sharings are consistent, $\mathcal{S}$ sends to corrupted parties the reconstruction results received from $\mathcal{F}_{\text{open}}$ instead of the real reconstruction results. Note that, the secret of a degree-t Shamir secret sharing is determined by the shares of honest parties. Corrupted parties can only cause the sharing to be inconsistent by sending incorrect shares to P_1 but cannot change the secret. Thus, if all the sharings are consistent, it means that corrupted parties send the correct shares to P_1 and the reconstruction results should be the same as those received from $\mathcal{F}_{\text{open}}$.

Therefore, the distribution of $\mathbf{Hybrid}_1$ is identical to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ simulates the second step as described above. The only difference is that $\mathcal{S}$ emulates $\mathcal{F}_{\text{coin}}$ by randomly generating $r \in \mathbb{F}_{q^m}$. This does not change the distribution of r. Therefore, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ simulates the last step as described above. The only difference is that, $\mathcal{S}$ additionally checks whether the reconstruction results output by P_1 are the same as those received from $\mathcal{F}_{\text{open}}$, and aborts if the reconstruction results are incorrect. Note that in $\mathbf{Hybrid}_2$, all parties just accept the incorrect reconstruction results.

We use $x^{(1)}, \ldots, x^{(N)}$ to denote the secrets of $[x^{(1)}]_t, \ldots, [x^{(N)}]_t$, and $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}$ to denote the reconstruction results output by P_1. Assume that there exists $\ell \in [N]$ such that $x^{(\ell)} \neq \tilde{x}^{(\ell)}$. We show that the probability that all parties accept the reconstructions in Step 3 is negligible. Consider the following two polynomials:

$$f(r) = x^{(1)} + x^{(2)} \cdot r + \ldots + x^{(N)} \cdot r^{N-1}$$
$$\tilde{f}(r) = \tilde{x}^{(1)} + \tilde{x}^{(2)} \cdot r + \ldots + \tilde{x}^{(N)} \cdot r^{N-1}.$$

Then $f(\cdot)$ and $\tilde{f}(\cdot)$ are two different polynomials. The number of r such that $f(r) = \tilde{f}(r)$ is bounded by $N-1$. Since r is randomly sampled from $\mathbb{F}_{q^m}$, the probability that $f(r) = \tilde{f}(r)$ is at most $\frac{N-1}{q^m} \leq \frac{N-1}{2^\kappa}$. Thus, with overwhelming probability, $f(r) \neq \tilde{f}(r)$. Note that in Step 2, we have $[x]_t = [f(r)]_t$ and $x = \tilde{f}(r)$. If $f(r) \neq \tilde{f}(r)$, an honest party will either receive an inconsistent sharing $[x]_t$ or reject the reconstruction results. In either case, this honest party will abort.

In summary, the probability that all parties accept incorrect reconstruction results in $\mathbf{Hybrid}_2$ is negligible. Since this is the only difference between $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$, the distribution of $\mathbf{Hybrid}_3$ is statistically close to $\mathbf{Hybrid}_2$.

Note that $\mathbf{Hybrid}_3$ is the execution in the ideal world and $\mathbf{Hybrid}_3$ is statistically close to $\mathbf{Hybrid}_0$, the execution in the real world. □

Constructing the Protocol for $\mathcal{F}_{re\text{-}encode}$. We are ready to introduce the protocol for $\mathcal{F}_{\text{re-encode}}$. The description of RE-ENCODE appears in Protocol 14. When using RAND to instantiate $\mathcal{F}_{\text{rand}}$ and OPEN to instantiate $\mathcal{F}_{\text{open}}$, the communication complexity of RE-ENCODE$(N, \{[x^{(\ell)}]_t\}_{i\in[N]})$ is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

Protocol 14: RE-ENCODE$(N, \{[x^{(\ell)}]_t\}_{i\in[N]})$

1. Let Σ be a secret sharing scheme which takes an element $r \in \mathbb{F}_{q^m}$ and outputs a pair of two sharings $([r]_t, [\phi(\psi(r))]_t)$. All parties invoke $\mathcal{F}_{\text{rand}}$ to prepare N random sharings in Σ, which are denoted by
$$([r^{(1)}]_t, [\phi(\psi(r^{(1)}))]_t), ([r^{(2)}]_t, [\phi(\psi(r^{(2)}))]_t), \ldots, ([r^{(N)}]_t, [\phi(\psi(r^{(N)}))]_t).$$
2. For all $\ell \in [N]$, all parties locally compute $[x^{(\ell)} + r^{(\ell)}]_t := [x^{(\ell)}]_t + [r^{(\ell)}]_t$. Then invoke $\mathcal{F}_{\text{open}}$ to receive the reconstruction result $x^{(\ell)} + r^{(\ell)}$.
3. For all $\ell \in [N]$, all parties locally compute $[\phi(\psi(x))]_t := \phi(\psi(x^{(\ell)} + r^{(\ell)})) - [\phi(\psi(r))]_t$.

Lemma 5. *Protocol* RE-ENCODE *securely computes $\mathcal{F}_{re\text{-}encode}$ in the $(\mathcal{F}_{rand}, \mathcal{F}_{open})$-hybrid model in the presence of a malicious adversary who controls t parties.*

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

Simulation for RE-ENCODE. At the beginning, for all $\ell \in [N]$, $\mathcal{S}$ receives from $\mathcal{F}_{\text{re-encode}}$ the shares of $[x^{(\ell)}]_t$ held by corrupted parties.

In Step 1, for all $\ell \in [N]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{rand}}$ and receives from the adversary the set of shares of $([r^{(\ell)}]_t, [\phi(\psi(r^{(\ell)}))]_t)$ of corrupted parties. If $\mathcal{S}$ receives `abort` from the adversary, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{re-encode}}$ and aborts.

In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{\text{open}}$. Concretely, for all $\ell \in [N]$, $\mathcal{S}$ chooses a random field element as the reconstruction result and sends the reconstruction result to the adversary. Since $\mathcal{S}$ receives from $\mathcal{F}_{\text{re-encode}}$

the shares of $[x^{(\ell)}]_t$ held by corrupted parties and receives from the adversary the shares of $[r^{(\ell)}]_t$ held by corrupted parties, $\mathcal{S}$ computes the shares of $[x^{(\ell)} + r^{(\ell)}]_t$ held by corrupted parties and sends them to the adversary. If $\mathcal{S}$ receives `abort` from the adversary, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{re-encode}}$ and aborts.

In Step 3, for all $\ell \in [N]$, $\mathcal{S}$ uses the shares of $[\phi(\psi(r^{(\ell)}))]_t$ held by corrupted parties to compute the shares of $[\phi(\psi(x))]_t$ held by corrupted parties, and sends these shares to $\mathcal{F}_{\text{re-encode}}$. Then $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\text{re-encode}}$.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. Consider the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ simulates the last step as described above. Note that, for the degree-t Shamir sharing $[\phi(\psi(x))]_t$, the shares of honest parties are determined by the shares of corrupted parties and the secret $\phi(\psi(x))$. Therefore, the distribution of $\mathbf{Hybrid}_1$ is identical to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ simulates the second step as described above. The only difference is that, in $\mathbf{Hybrid}_1$, $\mathcal{S}$ uses the real reconstruction result $x^{(\ell)} + r^{(\ell)}$, while in $\mathbf{Hybrid}_2$, $\mathcal{S}$ samples a random field element as the reconstruction result. Note that in Step 1, $r^{(\ell)}$ is randomly sampled by $\mathcal{F}_{\text{rand}}$. Therefore, $x^{(\ell)} + r^{(\ell)}$ is uniformly random. Thus, this does not change the distribution of the reconstruction result. Therefore, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ simulates the first step as described above. Note that the only difference is that $\mathcal{S}$ does not generate the random sharings $([r^{(\ell)}]_t, [\phi(\psi(r^{(\ell)}))]_t)$ explicitly. Since they are not used anymore in the rest of the protocols, the distribution of $\mathbf{Hybrid}_3$ is identical to the distribution of $\mathbf{Hybrid}_2$.

Note that $\mathbf{Hybrid}_3$ is the execution in the ideal world and the distribution of $\mathbf{Hybrid}_3$ is identical to the distribution of $\mathbf{Hybrid}_0$, the execution in the real world. □

Realization of $\mathcal{F}_{\text{tuple}}$. In this part, we will introduce the protocol which realizes $\mathcal{F}_{\text{tuple}}$ (Functionality 10). The description of TUPLE appears in Protocol 15.

Protocol 15: TUPLE(N)

1. For all $\ell \in [N]$, all parties invoke $\mathcal{F}_{\text{random}}$ twice to prepare random sharings $[\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t$.
2. For all $\ell \in [N]$, all parties invoke $\mathcal{F}_{\text{mult}}$ on $([\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t)$ to compute $[z^{(\ell)}]_t := [\phi(\boldsymbol{a}^{(\ell)}) \cdot \phi(\boldsymbol{b}^{(\ell)})]_t$.
3. All parties invoke $\mathcal{F}_{\text{multVerify}}$ to check the correctness of the following N multiplication tuples:
$$\{([\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t, [z^{(\ell)}]_t)\}_{\ell=1}^{N}.$$
4. For all $\ell \in [N]$, all parties invoke $\mathcal{F}_{\text{re-encode}}$ on $[z^{(\ell)}]_t$ to compute $[\phi(\psi(z^{(\ell)}))]_t$. Let $\boldsymbol{c}^{(\ell)} := \psi(z^{(\ell)})$. All parties take
$$\{([\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t, [\phi(\boldsymbol{c}^{(\ell)})]_t)\}_{\ell=1}^{N}$$
as output.

Recall that

- the communication complexity of the instantiation of $\mathcal{F}_{\text{mult}}$ in [DN07,GSZ20] is $O(n)$ elements in $\mathbb{F}_{q^m}$, which is $O(n \cdot m)$ elements in $\mathbb{F}_q$;
- the communication complexity of th instantiation of $\mathcal{F}_{\text{multVerify}}$ in [GSZ20] is $O(n^2 \cdot \log N \cdot \kappa)$ bits. Since we have assumed that $q^m \geq 2^\kappa$, the communication complexity is bounded by $(n^2 \cdot \log N \cdot m)$ elements in $\mathbb{F}_q$.

Therefore, the communication complexity of $\textsc{Tuple}(N)$ is $O(N \cdot n \cdot m + n^3 \cdot m + n^2 \cdot \log N \cdot m)$ elements in $\mathbb{F}_q$.

Lemma 6. *Protocol* $\textsc{Tuple}$ *securely computes* $\mathcal{F}_{tuple}$ *in the* $(\mathcal{F}_{random}, \mathcal{F}_{mult}, \mathcal{F}_{multVerify}, \mathcal{F}_{re\text{-}encode})$*-hybrid model in the presence of a malicious adversary who controls* t *parties.*

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

Simulation for $\textsc{Tuple}$. In Step 1, $\mathcal{S}$ emulates $\mathcal{F}_{\text{random}}$ and receives the shares of corrupted parties. If $\mathcal{S}$ receives `abort` from the adversary, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{tuple}}$ and aborts.

In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{\text{mult}}$. Concretely, for all $\ell \in [N]$, $\mathcal{S}$ sends the shares of $[\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t$ held by corrupted parties to the adversary. Note that these shares are learnt when $\mathcal{S}$ emulates $\mathcal{F}_{\text{random}}$ in Step 1. Then $\mathcal{S}$ receives from the adversary a value $d^{(\ell)}$ and the shares of $[z^{(\ell)}]_t := [\phi(\boldsymbol{a}^{(\ell)}) \cdot \phi(\boldsymbol{b}^{(\ell)})]_t$ of corrupted parties.

In Step 3, $\mathcal{S}$ emulates $\mathcal{F}_{\text{multVerify}}$. Concretely, $\mathcal{S}$ sends to the adversary the shares of

$$\{([\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t, [z^{(\ell)}]_t)\}_{\ell=1}^{N}$$

held by corrupted parties. These shares are learnt either when $\mathcal{S}$ emulates $\mathcal{F}_{\text{random}}$ in Step 1 or when $\mathcal{S}$ emulates $\mathcal{F}_{\text{mult}}$ in Step 2. Then $\mathcal{S}$ sends $\{d^{(\ell)}\}_{\ell=1}^{N}$ to the adversary. These values are learnt when $\mathcal{S}$ emulates $\mathcal{F}_{\text{mult}}$ in Step 2. $\mathcal{S}$ faithfully follows the rest of steps in $\mathcal{F}_{\text{multVerify}}$. If either the final result $b = \texttt{abort}$ or the adversary replies `abort`, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{tuple}}$ and aborts.

In Step 4, $\mathcal{S}$ emulates $\mathcal{F}_{\text{re-encode}}$. Concretely, for all $\ell \in [N]$, $\mathcal{S}$ sends the shares of $[z^{(\ell)}]_t$ held by corrupted parties to the adversary. Then $\mathcal{S}$ receives from the adversary the set of shares of $[\phi(\psi(z^{(\ell)}))]_t$ of corrupted parties. $\mathcal{S}$ sets $\boldsymbol{c}^{(\ell)} := \psi(z^{(\ell)})$. If $\mathcal{S}$ receives `abort` from the adversary, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{tuple}}$ and aborts.

Finally, $\mathcal{S}$ sends the shares of

$$\{([\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t, [\phi(\boldsymbol{c}^{(\ell)})]_t)\}_{\ell=1}^{N}$$

held by corrupted parties to $\mathcal{F}_{\text{tuple}}$. Then $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\text{tuple}}$.

Hybrid Arguments. Now we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. Consider the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: $\mathcal{S}$ simulates the last step as described above. Note that, for a degree-t Shamir sharing, the shares of honest parties are determined by the shares of corrupted parties and the secret. Therefore, the only difference is that, in $\mathbf{Hybrid}_0$, $\mathcal{S}$ uses the real secrets $\phi(\boldsymbol{a}^{(\ell)}), \phi(\boldsymbol{b}^{(\ell)}), \phi(\boldsymbol{c}^{(\ell)})$, while in $\mathbf{Hybrid}_1$, $\phi(\boldsymbol{a}^{(\ell)}), \phi(\boldsymbol{b}^{(\ell)})$ are randomly sampled by $\mathcal{F}_{\text{tuple}}$ and $\phi(\boldsymbol{c}^{(\ell)})$ is computed by $\boldsymbol{c}^{(\ell)} := \boldsymbol{a}^{(\ell)} * \boldsymbol{b}^{(\ell)}$. Note that, in $\mathbf{Hybrid}_0$, (1) $\phi(\boldsymbol{a}^{(\ell)}), \phi(\boldsymbol{b}^{(\ell)})$ are randomly sampled by $\mathcal{F}_{\text{random}}$, (2) the functionalities $\mathcal{F}_{\text{mult}}, \mathcal{F}_{\text{multVerify}}$ guarantee the correctness of $z := \phi(\boldsymbol{a}^{(\ell)}) \cdot \phi(\boldsymbol{b}^{(\ell)})$, and (3) the property of RMFE guarantees that $\boldsymbol{c} := \psi(z) = \psi(\phi(\boldsymbol{a}^{(\ell)}) \cdot \phi(\boldsymbol{b}^{(\ell)})) = \boldsymbol{a}^{(\ell)} * \boldsymbol{b}^{(\ell)}$. Thus, the distribution of $\mathbf{Hybrid}_1$ is identical to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: $\mathcal{S}$ simulates the third step as described above. The only difference is that, in $\mathbf{Hybrid}_1$, $\mathcal{S}$ computes the shares of corrupted parties using the shares of honest parties, and computes the actual differences, while in $\mathbf{Hybrid}_2$, $\mathcal{S}$ uses the shares of corrupted parties and the differences received from the adversary when honestly emulating $\mathcal{F}_{\text{mult}}$. Note that the distribution of these shares of corrupted parties and the differences is the same in both hybrids. Therefore, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: $\mathcal{S}$ simulates the second step as described above. The only difference is that $\mathcal{S}$ does not explicitly generate the sharing $[z^{(\ell)}]_t$ for each $\ell \in [N]$. However, note that these sharings are not used in the rest of steps. Thus, the distribution of $\mathbf{Hybrid}_3$ is identical to the distribution of $\mathbf{Hybrid}_2$.

Hybrid$_4$: $\mathcal{S}$ simulates the first step as described above. The only difference is that $\mathcal{S}$ does not explicitly generate the sharings $[\phi(\boldsymbol{a}^{(\ell)})]_t, [\phi(\boldsymbol{b}^{(\ell)})]_t$ for each $\ell \in [N]$. However, note that these sharings are not used in the rest of steps. Thus, the distribution of **Hybrid**$_4$ is identical to the distribution of **Hybrid**$_3$.

Note that **Hybrid**$_4$ is the execution in the ideal world and the distribution of **Hybrid**$_4$ is identical to the distribution of **Hybrid**$_0$, the execution in the real world. □

6.3 Preparing Zero Additive Sharings

With Beaver tuples prepared in the preprocessing phase, all parties only need to do reconstructions in the online phase. To protect the shares held by honest parties, for each reconstruction, we will prepare a random additive sharing of 0 among the first $t+1$ parties. We summarize the functionality for zero additive sharings in Functionality 16.

Functionality 16: $\mathcal{F}_{\text{zero}}$

1. $\mathcal{F}_{\text{zero}}$ receives $\{s_i\}_{i \in \mathcal{C} \bigcap \{1,\ldots,t+1\}}$ from the adversary, where $\mathcal{C}$ is the set of corrupted parties. Then $\mathcal{F}_{\text{zero}}$ randomly samples an additive sharing $\langle o \rangle$ such that $o = 0$, and for each $i \in \mathcal{C} \bigcap \{1, \ldots, t+1\}$, the i-th share of $\langle o \rangle$ is s_i.
2. $\mathcal{F}_{\text{zero}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{zero}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{zero}}$ distributes the shares of $\langle o \rangle$ to parties in $\mathcal{H} \bigcap \{1, \ldots, t+1\}$, where $\mathcal{H}$ is the set of honest parties.

As $\mathcal{F}_{\text{random}}$, we will use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{zero}}$. Consider a secret sharing scheme Σ which outputs an additive sharing of 0 in $\mathbb{F}_q$ among the first $t+1$ parties (and the shares of the rest of t parties are fixed to be 0). The secret space of Σ is $\{0\}$. It is clear that Σ is an $\mathbb{F}_q$-GLSSS. To use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{zero}}$, we need to show that:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i \in A}$ for parties in A, there exists an efficient algorithm which outputs that either $\Sigma(A, (a_i)_{i \in A}) = \emptyset$, or a random sharing $\langle o \rangle \in \Sigma(A, (a_i)_{i \in A})$.

The algorithm first checks that, for all $i \in A \backslash \{1, \ldots, t+1\}$, $a_i = 0$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Depending on the size of $A \bigcap \{1, \ldots, t+1\}$, there are two cases:

- If $|A \bigcap \{1, \ldots, t+1\}| = t+1$, then the whole sharing is determined by $\{a_i\}_{i \in A}$. The algorithm checks whether the summation of a_i for all $i \in A \backslash \{1, \ldots, t+1\}$ is 0. If not, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Otherwise, the algorithm sets the shares of $\langle o \rangle$ of parties in A to be $\{a_i\}_{i \in A}$, and the shares of parties in $\mathcal{I} \backslash A$ to be 0. Then, the algorithm outputs $\langle o \rangle$.
- If $|A \bigcap \{1, \ldots, t+1\}| < t+1$, then the set $\Sigma(A, (a_i)_{i \in A})$ is non-empty. Without loss of generality, assume $P_1 \notin A$. To generate a random sharing in $\Sigma(A, (a_i)_{i \in A})$, the algorithm samples random elements in $\mathbb{F}_q$ as the shares of $\langle o \rangle$ of parties in $\{2, \ldots, t+1\} \backslash A$. Then the algorithm reconstructs the whole sharing $\langle o \rangle$ based on the secret $o = 0$ and the shares of parties in $\{2, \ldots, t+1\}$, and outputs $\langle o \rangle$.

Thus, $\mathcal{F}_{\text{zero}}$ can be instantiated by $\mathcal{F}_{\text{rand}}$ with the secret sharing scheme Σ defined above. Note that the share size of a sharing in Σ is $\mathsf{sh} = 1$ element in $\mathbb{F}_q$. Therefore, when using $\text{RAND}(N)$ to instantiate $\mathcal{F}_{\text{rand}}$, the communication complexity of generating N random sharings in Σ is $O(N \cdot n + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

6.4 Preparing Parity Sharings

Recall that all parties only need to do reconstructions in the online phase. At the end of the online phase, it is sufficient to only verify the reconstructions. To this end, we first define what we call parity elements and parity sharings.

Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is defined by $\mathsf{val}(y) = \sum_{i=1}^{k} y_i$, where $(y_1, y_2, \ldots, y_k) = \psi(y)$. For an element $p \in \mathbb{F}_{q^m}$, we say p is a parity element if $\mathsf{val}(p) = 0$. A parity sharing is a degree-t Shamir sharing of a parity element. At the end of the protocol, we will use uniformly random parity sharings as masks when checking the correctness of the reconstructions. We summarize the functionality for preparing random parity sharings in Functionality 17.

Functionality 17: $\mathcal{F}_{\text{parity}}$

1. $\mathcal{F}_{\text{parity}}$ receives $\{u_i\}_{i\in\mathcal{C}}$ from the adversary, where $\mathcal{C}$ is the set of corrupted parties. Then $\mathcal{F}_{\text{parity}}$ randomly samples $p \in \mathbb{F}_{q^m}$ such that $\mathsf{val}(p) = 0$. Finally, $\mathcal{F}_{\text{parity}}$ generates a degree-t sharing $[p]_t$ such that the share of $P_i \in \mathcal{C}$ is u_i.
2. $\mathcal{F}_{\text{parity}}$ asks the adversary whether it should continue or not.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{parity}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, for each honest party P_i, $\mathcal{F}_{\text{parity}}$ sends the i-th share of $[p]_t$ to P_i.

Properties of Parity Elements. Let V_{parity} denote the set of all parity elements in $\mathbb{F}_{q^m}$. We show that V_{parity} is an $\mathbb{F}_q$-subspace. Recall that $\mathsf{val}(\cdot)$ is $\mathbb{F}_q$-linear. Therefore, for all $p, p' \in V_{\text{parity}}$ and $\alpha, \beta \in \mathbb{F}_q$, we have

$$\mathsf{val}(\alpha \cdot p + \beta \cdot p') = \alpha \cdot \mathsf{val}(p) + \beta \cdot \mathsf{val}(p') = 0,$$

which implies that $\alpha \cdot p + \beta \cdot p' \in V_{\text{parity}}$. We show that there exists an efficient algorithm to sample a uniformly random element in V_{parity}.

Let $\boldsymbol{e}$ be a vector in $\mathbb{F}_q^k$ such that the first entry is 1 and the rest of entries are 0. The algorithm computes $\delta = \phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})$. By the property of RMFE, $\psi(\phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})) = \boldsymbol{e} * \boldsymbol{e} = \boldsymbol{e}$. Therefore, $\mathsf{val}(\delta) = 1$. The algorithm randomly samples $x \in \mathbb{F}_{q^m}$ and computes $v := \mathsf{val}(x)$. Then, the algorithm outputs $x - v \cdot \delta$.

Note that $\mathsf{val}(x - v \cdot \delta) = \mathsf{val}(x) - v \cdot \mathsf{val}(\delta) = 0$. Therefore, $x - v \cdot \delta \in V_{\text{parity}}$. Also note that, for all $p \in V_{\text{parity}}$, p will be output by the algorithm only when x satisfies that $x = p + v \cdot \delta$ for some $v \in \mathbb{F}_q$. Since x is randomly sampled in $\mathbb{F}_{q^m}$, the probability that the algorithm outputs p is $|\mathbb{F}_q|/|\mathbb{F}_{q^m}|$. Note that this is independent of p. Therefore, the algorithm outputs a uniformly random element in V_{parity}.

Realizing $\mathcal{F}_{parity}$. Consider a secret sharing scheme Σ which takes an element $p \in V_{\text{parity}}$ and outputs a Shamir sharing of p in $\mathbb{F}_{q^m}$. Since V_{parity} is an $\mathbb{F}_q$-subspace and the Shamir secret sharing scheme is linear in $\mathbb{F}_{q^m}$, Σ is an $\mathbb{F}_q$-GLSSS. To use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{parity}}$, we need to show that:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, there exists an efficient algorithm which outputs that either $\Sigma(A, (a_i)_{i\in A}) = \emptyset$, or a random sharing $[p]_t \in \Sigma(A, (a_i)_{i\in A})$.

  Depending on the size of A, there are two cases:

- If $|A| \geq t + 1$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ can fully determine the whole sharing if exists. The algorithm checks whether the shares $\{a_i\}_{i\in A}$ lie on a polynomial $f(\cdot)$ of degree at most t in $\mathbb{F}_{q^m}$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm checks whether $f(0)$, i.e., the secret of this Shamir sharing, is a parity element. If not, the algorithm outputs $\Sigma(A, (a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm outputs the Shamir sharing determined by $\{a_i\}_{i\in A}$.

– If $|A| \leq t$, by the property of the Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ is independent of the secret. Therefore, $\Sigma(A, (a_i)_{i\in A}) \neq \emptyset$. The algorithm randomly samples $p \in V_{\text{parity}}$ and randomly samples $t-|A|$ elements in $\mathbb{F}_{q^m}$ as the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$. Then, based on the secret p, the shares of the first $t-|A|$ parties in $\mathcal{I}\backslash A$, and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the whole Shamir sharing $[p]_t$ and outputs $[p]_t$.

Thus, $\mathcal{F}_{\text{parity}}$ can be instantiated by $\mathcal{F}_{\text{rand}}$ with the secret sharing scheme Σ defined above. Note that the share size of a sharing in Σ is $\mathsf{sh} = m$ elements in $\mathbb{F}_q$. Therefore, when using $\textsc{Rand}(N)$ to instantiate $\mathcal{F}_{\text{rand}}$, the communication complexity of generating N sharings in Σ is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

7 Online Phase

Let (ϕ, ψ) be an $(k, m)_q$-RMFE. Recall that

– $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is defined by $\mathsf{val}(y) = \sum_{i=1}^{k} y_i$, where $(y_1, \ldots, y_k) = \psi(y)$.
– We use $\langle x \rangle$ to denote an additive sharing of $x \in \mathbb{F}_q$ among the first $t+1$ parties, and the shares of the rest of parties are 0.
– A pair of couple sharings $[\![x]\!] := (\langle x \rangle, [y]_t)$ contains an additive sharing of $x \in \mathbb{F}_q$ and a degree-t Shamir sharing of $y \in \mathbb{F}_{q^m}$ such that $\mathsf{val}(y) = x$.

In the online phase, our idea is to compute a pair of couple sharings for each wire. For an addition gate, given two pairs of couple sharings as input, all parties can locally compute the addition of these two sharings. For a multiplication gate, relying on Beaver tuples prepared in the preprocessing phase, all parties only need to reconstruct two pairs of couple sharings. We note that for the two sharings in a pair of couple sharings:

– The first sharing is an additive sharing in $\mathbb{F}_q$. The share of each party is just a field element in $\mathbb{F}_q$. We will use this sharing to do reconstruction. However, the correctness cannot be guaranteed since a single corrupted party can change the secret by changing its own share.
– The second sharing is a degree-t Shamir sharing in $\mathbb{F}_{q^m}$. The share of each party is a field element in $\mathbb{F}_{q^m}$. Note that the secret is determined by the shares of honest parties, and cannot be altered by corrupted parties. However, using this sharing to do reconstruction is expensive. Therefore, we will use this sharing to verify the correctness of reconstruction at the end of the protocol.

7.1 Input Gates

Recall that we are in the client-server model. In particular, all the inputs belong to the clients. In this part, we introduce a protocol $\textsc{Input}$, which allows a client to share k inputs to all parties. In the main protocol, we will invoke $\textsc{Input}$ for every client with k inputs.

The description of $\textsc{Input}$ appears in Protocol 18. The communication complexity of $\textsc{Input}(\mathsf{Client}, \{x^{(1)}, \ldots, x^{(k)}\})$ is $O(m+k)$ elements in $\mathbb{F}_q$ plus one call of $\mathcal{F}_{\text{random}}$.

Note that this protocol guarantees the security of the inputs of honest clients. This is because the input of honest clients are masked by random vectors $\boldsymbol{r}$'s which are chosen by $\mathcal{F}_{\text{random}}$. However, a corrupted client can send different values to different parties, which leads to incorrect or inconsistent couple sharings in the final step. We will address this issue by checking consistency of the values distributed by all clients at the end of the protocol.

7.2 Addition Gates and Multiplication Gates

For each fan-in two addition gate with input sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!]$, all parties locally compute

$$[\![x^{(0)}]\!] := [\![x^{(1)} + x^{(2)}]\!] = [\![x^{(1)}]\!] + [\![x^{(2)}]\!].$$

Protocol 18: INPUT(Client, $\{x^{(1)}, \ldots, x^{(k)}\}$)

1. All parties invoke $\mathcal{F}_{\text{random}}$ to prepare a random sharing $[\phi(\boldsymbol{r})]_t$, where $\boldsymbol{r}$ is a random vector in $\mathbb{F}_q^k$. Then, all parties send their shares of $[\phi(\boldsymbol{r})]_t$ to the Client.
2. After receiving the shares of $[\phi(\boldsymbol{r})]_t$, the Client checks whether all the shares lie on a polynomial of degree at most t in $\mathbb{F}_{q^m}$. If not, the Client aborts. Otherwise, the Client reconstructs the secret $\phi(\boldsymbol{r})$.
3. The Client computes $\boldsymbol{r}$ from $\phi(\boldsymbol{r})$. Then, the Client sets $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(k)})$, where $x^{(1)}, \ldots, x^{(k)}$ are its input. The Client sends $\boldsymbol{x} + \boldsymbol{r}$ to all parties.
4. After receiving $\boldsymbol{x} + \boldsymbol{r}$ from the Client, all parties locally compute $[\phi(\boldsymbol{x})]_t := \phi(\boldsymbol{x} + \boldsymbol{r}) - [\phi(\boldsymbol{r})]_t$.
5. All parties invoke SEPARATE on $[\phi(\boldsymbol{x})]_t$ to obtain couple sharings for the input of the Client:

$$(\langle x^{(1)} \rangle, [y^{(1)}]_t), \ldots, (\langle x^{(k)} \rangle, [y^{(k)}]_t)$$

For each multiplication gate with input sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!]$, we want to obtain a pair of couple sharings $[\![x^{(0)}]\!]$ such that $x^{(0)} = x^{(1)} \cdot x^{(2)}$. To this end, we will use one Beaver tuple $([\![a]\!], [\![b]\!], [\![c]\!])$ prepared in Section 6.2. It satisfies that a, b are random field elements in $\mathbb{F}_q$ and $c = a \cdot b$. Note that

$$\begin{aligned} x^{(0)} &= x^{(1)} \cdot x^{(2)} \\ &= (a + x^{(1)} - a) \cdot (b + x^{(2)} - b) \\ &= (a + x^{(1)}) \cdot (b + x^{(2)}) - (b + x^{(2)}) \cdot a - (a + x^{(1)}) \cdot b + a \cdot b \\ &= (a + x^{(1)}) \cdot (b + x^{(2)}) - (b + x^{(2)}) \cdot a - (a + x^{(1)}) \cdot b + c. \end{aligned}$$

Therefore, all parties only need to reconstruct the sharings $[\![a]\!] + [\![x^{(1)}]\!]$ and $[\![b]\!] + [\![x^{(2)}]\!]$, and the resulting sharing can be computed by

$$[\![x^{(0)}]\!] = (a + x^{(1)}) \cdot (b + x^{(2)}) - (b + x^{(2)}) \cdot [\![a]\!] - (a + x^{(1)}) \cdot [\![b]\!] + [\![c]\!].$$

To reconstruct $[\![a]\!] + [\![x^{(1)}]\!]$, we will use the additive sharing $\langle a + x^{(1)} \rangle := \langle a \rangle + \langle x^{(1)} \rangle$. We first add a random additive sharing $\langle o \rangle$ of 0 (prepared in Section 6.3) to protect the shares of honest parties. The first $t+1$ parties locally compute $\langle a \rangle + \langle x^{(1)} \rangle + \langle o \rangle$ and send their shares to P_1. P_1 reconstructs the secret $a + x^{(1)}$ and sends the result to all other parties. Similar process is done when reconstructing $\langle b + x^{(2)} \rangle := \langle b \rangle + \langle x^{(2)} \rangle$.

Note that $\langle a \rangle + \langle o \rangle$ is a random additive sharing. The share of each honest party in $\{P_1, \ldots, P_{t+1}\}$ is uniformly distributed. Essentially, each honest party in $\{P_1, \ldots, P_{t+1}\}$ uses a random element as mask to protect its own share. The protocol MULT appears in Protocol 19. The communication complexity of MULT is $O(n)$ elements in $\mathbb{F}_q$ plus two calls of $\mathcal{F}_{\text{zero}}$.

The protocol MULT can go wrong at three places:

- A corrupted party may send an incorrect share to P_1.
- P_1 is corrupted and distributes an incorrect reconstruction result to all other parties.
- P_1 is corrupted and distributes different values to different parties.

Note that, relying on the random additive sharing of 0, honest parties in the first $t + 1$ parties only send random elements to P_1. Therefore, MULT does not leak any information about the shares of honest parties *even if the input sharings of the multiplication gate are not in the correct form.* It allows us to delay the verification of the values distributed by P_1 to the end of the protocol. It also allows us to delay the verification of the values distributed by clients in the input phase to the end of the protocol since a corrupted client distributing different values to different parties has the same effect as P_1 distributing different values to different parties. During the verification of the computation, we will first check whether all parties receive the same values to resolve the third issue. Then, for the first two issues, it is sufficient to check the correctness of the reconstructions.

Protocol 19: $\textsc{Mult}(\llbracket x^{(1)} \rrbracket, \llbracket x^{(2)} \rrbracket, (\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket))$

1. All parties invoke $\mathcal{F}_{\text{zero}}$ to prepare two random additive sharings $\langle o^{(1)} \rangle, \langle o^{(2)} \rangle$ where $o^{(1)} = o^{(2)} = 0$.
2. Let $\langle x^{(1)} + a \rangle, \langle x^{(2)} + b \rangle$ denote the additive sharings in $\llbracket x^{(1)} + a \rrbracket, \llbracket x^{(2)} + b \rrbracket$ respectively. The first $t + 1$ parties locally compute $\langle u^{(1)} \rangle := \langle x^{(1)} + a \rangle + \langle o^{(1)} \rangle$ and $\langle u^{(2)} \rangle := \langle x^{(2)} + b \rangle + \langle o^{(2)} \rangle$. Then, they send their shares of $\langle u^{(1)} \rangle, \langle u^{(2)} \rangle$ to the first party P_1.
3. P_1 reconstructs the secrets $u^{(1)}, u^{(2)}$ by computing the summation of the shares of $\langle u^{(1)} \rangle$ and $\langle u^{(2)} \rangle$ respectively. Then, P_1 sends $u^{(1)}, u^{(2)}$ to all other parties (including the last t parties).
4. After receiving $u^{(1)}, u^{(2)}$, all parties locally compute the resulting couple sharings

$$\llbracket x^{(0)} \rrbracket = u^{(1)} \cdot u^{(2)} - u^{(2)} \cdot \llbracket a \rrbracket - u^{(1)} \cdot \llbracket b \rrbracket + \llbracket c \rrbracket,$$

and take $\llbracket x^{(0)} \rrbracket$ as output.

7.3 Verification of the Computation

Before all parties revealing the outputs, we need to verify the computation. Concretely, we need to verify that (1) the clients distributed the same values in the input phase, and P_1 distributed the same values when evaluating multiplication gates, and (2) the reconstructions are correct.

Checking the Correctness of Distribution. All parties first check whether they receive the same values when handling input gates and multiplication gates. Note that these values are all in $\mathbb{F}_q$. Assume that these values are denoted by $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$. The protocol $\textsc{CheckConsistency}$ appears in Protocol 20. The communication complexity of $\textsc{CheckConsistency}(N, \{x^{(1)}, \ldots, x^{(N)}\})$ is $O(n^2 \cdot m)$ elements in $\mathbb{F}_q$.

Protocol 20: $\textsc{CheckConsistency}(N, \{x^{(1)}, \ldots, x^{(N)}\})$

1. All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ to generate a random element $r \in \mathbb{F}_{q^m}$. All parties locally compute

$$x := x^{(1)} + x^{(2)} \cdot r + \ldots + x^{(N)} \cdot r^{N-1}.$$

2. All parties exchange their results x's and check whether they are the same. If a party P_i receives different x's, P_i aborts.

Lemma 7. *If there exists two honest parties who receive different set of values $\{x^{(1)}, \ldots, x^{(N)}\}$, then with overwhelming probability, at least one honest party will abort in the protocol* $\textsc{CheckConsistency}$.

Proof. Suppose P_i, P_j are two honest parties and they receive $\{x^{(1)}, \ldots, x^{(N)}\}$ and $\{\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}\}$ respectively. Suppose that there exists $i \in [N]$ such that $x^{(i)} \neq \tilde{x}^{(i)}$. Consider the following two polynomials in $\mathbb{F}_{q^m}$:

$$f(r) = x^{(1)} + x^{(2)} \cdot r + \ldots + x^{(N)} \cdot r^{N-1}$$
$$\tilde{f}(r) = \tilde{x}^{(1)} + \tilde{x}^{(2)} \cdot r + \ldots + \tilde{x}^{(N)} \cdot r^{N-1}$$

Since there exists $i \in [N]$ such that $x^{(i)} \neq \tilde{x}^{(i)}$, $f(\cdot)$ and $\tilde{f}(\cdot)$ are two different polynomials. The number of r such that $f(r) = \tilde{f}(r)$ is bounded by the degree of $f(\cdot) - \tilde{f}(\cdot)$, which is $N - 1$. Since r is uniformly

chosen from $\mathbb{F}_{q^m}$, the probability that $f(r) = \tilde{f}(r)$ is at most $\frac{N-1}{q^m} \leq \frac{N-1}{2^\kappa}$. Therefore, with overwhelming probability, $f(r) \neq \tilde{f}(r)$, which means that P_i, P_j will receive different values from each other and abort in the protocol CheckConsistency. □

This step makes sure that all (honest) parties receive the same values from clients and P_1. Therefore, the remaining task is to verify the correctness of the reconstructions.

Verification of Reconstructions. Recall that a pair of couple sharings $[\![x]\!] := (\langle x \rangle, [y]_t)$ satisfies that $\langle x \rangle$ is an additive sharing of x and $[y]_t$ is a degree-t Shamir sharing of y such that $\mathsf{val}(y) = x$. For a multiplication gate with input sharings $(\langle x^{(1)} \rangle, [y^{(1)}]_t), (\langle x^{(2)} \rangle, [y^{(2)}]_t)$, one Beaver tuple $((\langle a \rangle, [\alpha]_t), (\langle b \rangle, [\beta]_t), (\langle c \rangle, [\gamma]_t))$ is consumed to compute the resulting sharing. All parties reconstruct

$$(\langle x^{(1)} \rangle, [y^{(1)}]_t) + (\langle a \rangle, [\alpha]_t) \text{ and } (\langle x^{(2)} \rangle, [y^{(2)}]_t) + (\langle b \rangle, [\beta]_t),$$

and learn $x^{(1)} + a$ and $x^{(2)} + b$. Note that, the secret of a degree-t Shamir sharing is determined by the shares held by honest parties. Therefore, the correctness can be verified by checking $\mathsf{val}(y^{(1)} + \alpha) = x^{(1)} + a$ and $\mathsf{val}(y^{(2)} + \beta) = x^{(2)} + b$.

This task can be summarized as follows: Given N value-sharing pairs

$$(u^{(1)}, [w^{(1)}]_t), \ldots, (u^{(N)}, [w^{(N)}]_t),$$

where $u^{(i)} \in \mathbb{F}_q$ and $w^{(i)} \in \mathbb{F}_{q^m}$ for all $i \in [N]$, we want to verify that for all $i \in [N]$, $\mathsf{val}(w^{(i)}) = u^{(i)}$. Here $u^{(i)}$ corresponds to $x^{(1)} + a$ and $[w^{(i)}]_t$ corresponds to $[y^{(1)} + \alpha]_t$. The functionality $\mathcal{F}_{\text{checkRecon}}$ appears in Functionality 21.

Functionality 21: $\mathcal{F}_{\text{checkRecon}}$

1. Let N denote the number of value-sharing pairs. These value-sharing pairs are denoted by
$$(u^{(1)}, [w^{(1)}]_t), (u^{(2)}, [w^{(2)}]_t), \ldots, (u^{(N)}, [w^{(N)}]_t).$$
$\mathcal{F}_{\text{checkRecon}}$ will check whether $\mathsf{val}(w^{(i)}) = u^{(i)}$ for all $i \in [N]$.
2. For all $i \in [N]$, $\mathcal{F}_{\text{checkRecon}}$ receives from honest parties their shares of $[w^{(i)}]_t$. Then $\mathcal{F}_{\text{checkRecon}}$ reconstructs the secret $w^{(i)}$. $\mathcal{F}_{\text{checkRecon}}$ further computes the shares of $[w^{(i)}]_t$ held by corrupted parties and sends these shares to the adversary.
3. For all $i \in [N]$, $\mathcal{F}_{\text{checkRecon}}$ computes $\mathsf{val}(w^{(i)})$ and sends $u^{(i)}, \mathsf{val}(w^{(i)})$ to the adversary.
4. Finally, let $b \in \{\texttt{abort}, \texttt{accept}\}$ denote whether there exists $i \in [N]$ such that $\mathsf{val}(w^{(i)}) \neq u^{(i)}$. $\mathcal{F}_{\text{checkRecon}}$ sends b to the adversary and waits for its response.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{checkRecon}}$ sends `abort` to honest parties.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{checkRecon}}$ sends b to honest parties.

Consider a secret sharing scheme Σ which takes a field element $u \in \mathbb{F}_q$ as input and outputs $(u, [w]_t)$, where u is the same as the input and $[w]_t$ is a degree-t Shamir sharing of $w \in \mathbb{F}_{q^m}$ such that $\mathsf{val}(w) = u$. Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map, and the Shamir secret sharing scheme is $\mathbb{F}_{q^m}$-linear. Therefore, Σ is an $\mathbb{F}_q$-GLSSS. Therefore, $\mathcal{F}_{\text{checkRecon}}$ essentially checks the membership of $(u^{(1)}, [w^{(1)}]_t), \ldots, (u^{(N)}, [w^{(N)}]_t)$ in Σ.

The protocol CheckRecon appears in Protocol 22. At a high-level, instead of checking the membership in Σ, we choose to use the m-fold interleaved sharing scheme $\Sigma^{\times m}$. All parties first transform the N sharings in Σ (i.e., the value-sharing pairs) into $N' = N/m$ sharings in $\Sigma^{\times m}$. Then all parties compute a random linear combination of these N' sharings, and only test the membership of the resulting sharing in $\Sigma^{\times m}$. To

further protect the shares of honest parties, we need to prepare a random sharing of 0 in $\Sigma^{\times m}$. Recall that a parity sharing $[p]_t$ satisfies that $\mathsf{val}(p) = 0$. Therefore, $(0, [p]_t)$ is a random sharing of 0 in Σ. All parties will invoke $\mathcal{F}_{\text{parity}}$ to prepare m random parity sharings and transform them to m random sharings of 0 in Σ, and then, to a random sharing of 0 in $\Sigma^{\times m}$.

The communication complexity of $\textsc{CheckRecon}(N, \{(u^{(1)}, [w^{(1)}]_t), \ldots, (u^{(N)}, [w^{(N)}]_t)\})$ is $O(n^2 \cdot m^2)$ elements in $\mathbb{F}_q$ plus m calls of $\mathcal{F}_{\text{parity}}$.

Protocol 22: $\textsc{CheckRecon}(N, \{(u^{(1)}, [w^{(1)}]_t), \ldots, (u^{(N)}, [w^{(N)}]_t)\})$

1. Let $N' = N/m$. All parties first transform the N sharings in Σ to N' sharings in $\Sigma^{\times m}$. Concretely, for all $i \in [N']$, let
$$(\boldsymbol{v}^{(i)}, [\boldsymbol{z}^{(i)}]_t) := ((u^{((i-1)m+1)}, [w^{((i-1)m+1)}]_t), \ldots, (u^{(i \cdot m)}, [w^{(i \cdot m)}]_t)).$$
2. All parties invoke $\mathcal{F}_{\text{parity}}$ to generate m random parity sharings $[p^{(1)}]_t, [p^{(2)}]_t, \ldots, [p^{(m)}]_t$. All parties transform these m parity sharings to a sharing of 0 in $\Sigma^{\times m}$:
$$(\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t) := ((0, [p^{(1)}]_t), \ldots, (0, [p^{(m)}]_t)).$$
3. All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ to generate a random element $r \in \mathbb{F}_{q^m}$. All parties locally compute
$$(\boldsymbol{v}, [\boldsymbol{z}]_t) := (\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t) + (\boldsymbol{v}^{(1)}, [\boldsymbol{z}^{(1)}]_t) \cdot r + \ldots + (\boldsymbol{v}^{(N')}, [\boldsymbol{z}^{(N')}]_t) \cdot r^{N'}$$
4. For each party P_i, P_i receives from all other parties their shares of $[\boldsymbol{z}]_t$. Let $(\boldsymbol{v}, [\boldsymbol{z}]_t) = ((v_1, [z_1]_t), \ldots, (v_m, [z_m]_t))$. For all $j \in [m]$, P_i checks whether $[z_j]_t$ is a valid degree-t Shamir sharing and $\mathsf{val}(z_j) = v_j$. If not, P_i aborts. Otherwise, P_i accepts the reconstructions.

Lemma 8. *Protocol* $\textsc{CheckRecon}$ *securely computes* $\mathcal{F}_{checkRecon}$ *with abort in the* $\{\mathcal{F}_{parity}, \mathcal{F}_{coin}\}$*-hybrid model in the presence of a malicious adversary who controls* t *parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map. For a vector $\boldsymbol{z} := (z_1, \ldots, z_m) \in (\mathbb{F}_{q^m})^m$, we use $\mathsf{val}(\boldsymbol{z})$ to denote $(\mathsf{val}(z_1), \ldots, \mathsf{val}(z_m))$.

Simulation for $\textsc{CheckRecon}$. At the beginning, for each $i \in [N]$, $\mathcal{S}$ receives the shares of $[w^{(i)}]_t$ held by corrupted parties and $u^{(i)}, \mathsf{val}(w^{(i)})$ from $\mathcal{F}_{\text{checkRecon}}$.

In Step 1, $\mathcal{S}$ computes $\{\boldsymbol{v}^{(i)}\}_{i=1}^{N'}$ and the shares of $\{[\boldsymbol{z}^{(i)}]_t\}_{i=1}^{N'}$ held by corrupted parties from $\{u^{(i)}\}_{i=1}^{N}$ and the shares of $\{[w^{(i)}]_t\}_{i=1}^{N}$ held by corrupted parties.

In Step 2, $\mathcal{S}$ emulates $\mathcal{F}_{\text{parity}}$ and receives the shares of $[p^{(1)}]_t, \ldots, [p^{(m)}]_t$ held by corrupted parties. If $\mathcal{S}$ receives `abort` from the adversary, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{checkParity}}$ and aborts. $\mathcal{S}$ computes $\boldsymbol{v}^{(0)}$ and the shares of $[\boldsymbol{z}^{(0)}]_t$ held by corrupted parties.

In Step 3, $\mathcal{S}$ emulates $\mathcal{F}_{\text{coin}}$ by randomly sampling $r \in \mathbb{F}_{q^m}$. $\mathcal{S}$ computes $\boldsymbol{v}$ and the shares of $[\boldsymbol{z}]_t$ held by corrupted parties. Let $(\boldsymbol{v}, [\boldsymbol{z}]_t) = ((v_1, [z_1]_t), \ldots, (v_m, [z_m]_t))$. According to Protposition 1, for all $i \in [m]$, $(v_i, [z_i]_t)$ is an $\mathbb{F}_q$-linear combination of $\{(0, [p^{(i)}]_t)\}_{i=1}^{m}$ and $\{(u^{(i)}, [w^{(i)}]_t)\}_{i=1}^{N}$. Therefore, $\mathcal{S}$ can compute $\mathsf{val}(z_i)$ from $\{\mathsf{val}(p^{(i)})\}_{i=1}^{m}$ and $\{\mathsf{val}(w^{(i)})\}_{i=1}^{N}$. Note that each of $\mathsf{val}(p^{(i)})$ is 0 and $\{\mathsf{val}(w^{(i)})\}_{i=1}^{N}$ are received from $\mathcal{F}_{\text{checkRecon}}$.

Also note that for all $i \in [m]$, $p^{(i)}$ is a uniformly random parity element. Since z_i is masked by $p^{(i)}$, z_i is uniformly distributed given $\mathsf{val}(z_i)$. Let $\boldsymbol{e}$ be a vector in $\mathbb{F}_q^k$ such that the first entry is 1 and the rest of entries are 0. Let $\delta = \phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})$. By the property of RMFE, $\psi(\phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})) = \boldsymbol{e} * \boldsymbol{e} = \boldsymbol{e}$. Therefore, $\mathsf{val}(\delta) = 1$. For all $i \in [m]$, to sample a random element z_i given $\mathsf{val}(z_i)$, $\mathcal{S}$ randomly samples a parity element r and

sets $z_i := r + \mathsf{val}(z_i) \cdot \delta$. Then, based on the secret z_i and the shares of $[z_i]_t$ held by corrupted parties, $\mathcal{S}$ computes the shares of $[z_i]_t$ held by honest parties.

In Step 4, since $\mathcal{S}$ has computed $\boldsymbol{v}$ and the shares of $[\boldsymbol{z}]_t$ held by honest parties, $\mathcal{S}$ faithfully follows the protocol. If any party aborts, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{checkRecon}}$ and aborts. Otherwise, $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\text{checkRecon}}$.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Conisder the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ simulates the first step as described above. It does not change the messages sent from honest parties to corrupted parties. Therefore, the distribution of $\mathbf{Hybrid}_1$ is identical to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ simulates the second step as described above. Note that the only difference is that, in $\mathbf{Hybrid}_1$, the secrets $p^{(1)}, \ldots, p^{(m)}$ are chosen by $\mathcal{F}_{\text{parity}}$, while in $\mathbf{Hybrid}_2$, these secrets are chosen by $\mathcal{S}$. However, it makes no difference since in both hybrids, $p^{(1)}, \ldots, p^{(m)}$ are random parity elements. Therefore, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ simulates the third step as described above. There are two differences between $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$:

- Instead of using the real $\boldsymbol{z}$, $\mathcal{S}$ randomly chooses $\boldsymbol{z}$ based on $\mathsf{val}(\boldsymbol{z}) := (\mathsf{val}(z_1), \ldots, \mathsf{val}(z_m))$.
- Instead of using the real shares of honest parties, $\mathcal{S}$ computes the shares of $[\boldsymbol{z}]_t$ held by honest parties based on the secret $\boldsymbol{z}$ and the shares of $[\boldsymbol{z}]_t$ held by corrupted parties.

According to Proposition 1, $\mathsf{val}(\boldsymbol{z})$ can be computed (and therefore determined) by $\mathsf{val}(\boldsymbol{z}^{(0)}), \ldots, \mathsf{val}(\boldsymbol{z}^{(N)})$ (i.e., $\{\mathsf{val}(p^{(i)})\}_{i=1}^{m}$ and $\{\mathsf{val}(w^{(i)})\}_{i=1}^{N}$). Since $\boldsymbol{z}^{(0)} = (p^{(1)}, \ldots, p^{(m)})$ is a vector of random parity elements chosen by $\mathcal{S}$, $\boldsymbol{z}$ is a random vector of elements given $\mathsf{val}(\boldsymbol{z})$. Therefore, the distributions of $\boldsymbol{z}$ in both hybrids are identical. Since the shares of honest parties are determined by the secret $\boldsymbol{z}$ and the shares held by corrupted parties, the distributions of the shares of $[\boldsymbol{z}]_t$ held by honest parties in both hybrids are identical.

Therefore, the distribution of $\mathbf{Hybrid}_3$ is identical to the distribution of $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_4$: In this hybrid, $\mathcal{S}$ simulates the last step as described above. The only difference is that, in $\mathbf{Hybrid}_3$, all parties accept the reconstructions if $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$, while in $\mathbf{Hybrid}_4$, honest parties accept the reconstructions (by receiving `accept` from $\mathcal{F}_{\text{checkRecon}}$) only if for all $i \in N$, $\mathsf{val}(w^{(i)}) = u^{(i)}$.

Assume that there exists $i \in [N]$ such that $\mathsf{val}(w^{(i)}) \neq u^{(i)}$. This implies that there exists $i \in [N']$ such that $\mathsf{val}(\boldsymbol{z}^{(i)}) \neq \boldsymbol{v}^{(i)}$. We show that the number of r such that $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$ is bounded by N'. Suppose there are $N'+1$ different values $r_0, r_1, \ldots, r_{N'}$ such that for all $j \in \{0, 1, \ldots N'\}$,

$$(\tilde{\boldsymbol{v}}_j, [\tilde{\boldsymbol{z}}_j]_t) := (\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t) + (\boldsymbol{v}^{(1)}, [\boldsymbol{z}^{(1)}]_t) \cdot r_j + \ldots + (\boldsymbol{v}^{(N')}, [\boldsymbol{z}^{(N')}]_t) \cdot r_j^{N'}$$

satisfies that $\mathsf{val}(\tilde{\boldsymbol{z}}_j) = \tilde{\boldsymbol{v}}_j$. Let $\boldsymbol{M}$ be a matrix of size $(N'+1) \times (N'+1)$ in $\mathbb{F}_{q^m}$ such that $\boldsymbol{M}_{ji} = r_{j-1}^{i-1}$. Then we have

$$((\tilde{\boldsymbol{v}}_0, [\tilde{\boldsymbol{z}}_0]_t), \ldots, (\tilde{\boldsymbol{v}}_{N'}, [\tilde{\boldsymbol{z}}_{N'}]_t))^{\mathrm{T}} = \boldsymbol{M} \cdot ((\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t), \ldots, (\boldsymbol{v}^{(N')}, [\boldsymbol{z}^{(N')}]_t))^{\mathrm{T}}.$$

Note that $\boldsymbol{M}$ is an $(N'+1) \times (N'+1)$ Vandermonde matrix, which is invertible. Therefore,

$$((\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t), \ldots, (\boldsymbol{v}^{(N')}, [\boldsymbol{z}^{(N')}]_t))^{\mathrm{T}} = \boldsymbol{M}^{-1} \cdot ((\tilde{\boldsymbol{v}}_0, [\tilde{\boldsymbol{z}}_0]_t), \ldots, (\tilde{\boldsymbol{v}}_{N'}, [\tilde{\boldsymbol{z}}_{N'}]_t))^{\mathrm{T}}.$$

According to the assumption, for all $j \in \{0, 1, \ldots, N'\}$, $(\tilde{\boldsymbol{v}}_j, [\tilde{\boldsymbol{z}}_j]_t)$ is a valid sharing in $\Sigma^{\times m}$. Since $\Sigma^{\times m}$ is $\mathbb{F}_{q^m}$-linear, the above equation implies that for all $i \in \{0, 1, \ldots, N'\}$, $(\boldsymbol{v}^{(i)}, [\boldsymbol{z}^{(i)}]_t)$ is a valid sharing in $\Sigma^{\times m}$, which means that $\mathsf{val}(\boldsymbol{z}^{(i)}) = \boldsymbol{v}^{(i)}$. This leads to a contradiction.

Therefore, the number of r such that $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$ is bounded by N'. Since r is randomly sampled from $\mathbb{F}_{q^m}$, the probability that $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$ is bounded by $\frac{N'}{q^m} \leq \frac{N'}{2^\kappa}$, which is negligible.

Therefore, the distribution of $\mathbf{Hybrid}_4$ is statistically close to $\mathbf{Hybrid}_3$.

Note that $\mathbf{Hybrid}_4$ is the execution in the ideal world and $\mathbf{Hybrid}_4$ is statistically close to $\mathbf{Hybrid}_0$, the execution in the real world. □

7.4 Output Gates

Recall that we are in the client-server model. In particular, only the clients receive the outputs. In this part, we will introduce a functionality $\mathcal{F}_{\text{output}}$ which reconstructs the output couple sharings to the client who should receive them. In the main protocol, we will invoke $\mathcal{F}_{\text{output}}$ for every client.

Suppose we need to reconstruct the following N pairs of couple sharings to the Client:

$$[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!].$$

Recall that a pair of couple sharings $[\![x]\!] := (\langle x \rangle, [y]_t)$ satisfies that $\langle x \rangle$ is an additive sharing of x, and $[y]_t$ is a degree-t Shamir sharing of y such that $\mathsf{val}(y) = x$. The functionality $\mathcal{F}_{\text{output}}$ appears in Functionality 23.

Functionality 23: $\mathcal{F}_{\text{output}}$

1. Let N denote the number of output gates belonging to the Client. The couple sharings are denoted by
$$[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!].$$
$\mathcal{F}_{\text{output}}$ will reconstruct $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ to the Client.
2. For all $i \in [N]$, suppose $[\![x^{(i)}]\!] = (\langle x^{(i)} \rangle, [y^{(i)}]_t)$. $\mathcal{F}_{\text{output}}$ receives from honest parties their shares of $(\langle x^{(i)} \rangle, [y^{(i)}]_t)$. Then $\mathcal{F}_{\text{output}}$ reconstructs the secret $y^{(i)}$ and computes $\mathsf{val}(y^{(i)})$.
   - For $[y^{(i)}]_t$, $\mathcal{F}_{\text{output}}$ computes the shares of $[y^{(i)}]_t$ held by corrupted parties and sends these shares to the adversary.
   - For $\langle x^{(i)} \rangle$, note that the summation of all the shares should be $\mathsf{val}(y^{(i)})$. $\mathcal{F}_{\text{output}}$ computes the summation of the shares of corrupted parties, denoted by $x_{\mathcal{C}}^{(i)}$, which can be computed from $\mathsf{val}(y^{(i)})$ and the shares of $\langle x^{(i)} \rangle$ held by honest parties. $\mathcal{F}_{\text{output}}$ sends $x_{\mathcal{C}}^{(i)}$ to the adversary.
3. Depending on whether the Client is honest, there are two cases:
   - If the Client is corrupted, $\mathcal{F}_{\text{output}}$ sends $\{\mathsf{val}(y^{(i)})\}_{i=1}^{N}$ to the adversary. If the adversary replies $\texttt{abort}$, $\mathcal{F}_{\text{output}}$ sends $\texttt{abort}$ to all honest parties.
   - If the Client is honest, $\mathcal{F}_{\text{output}}$ asks the adversary whether it should continue. If the adversary replies $\texttt{abort}$, $\mathcal{F}_{\text{output}}$ sends $\texttt{abort}$ to the Client and all honest parties. If the adversary replies $\texttt{continue}$, $\mathcal{F}_{\text{output}}$ sends $\{\mathsf{val}(y^{(i)})\}_{i=1}^{N}$ to the Client.

The protocol Output appears in Protocol 24. At a high-level, the first $t+1$ parties send their shares of the additive sharings to the Client to allow the Client to reconstruct the output. To verify the correctness of the reconstructions, we follow a similar approach as that in Section 7.3. Concretely, suppose the reconstruction results are $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(N)}$. We need to verify that for all $i \in [N]$, the value-sharing pair $(\tilde{x}^{(i)}, [y^{(i)}]_t)$ satisfies that $\mathsf{val}(y^{(i)}) = \tilde{x}^{(i)}$. The only difference here is that only the Client knows the reconstruction results. Note that all the operations on value-sharing pairs in CheckRecon are coordinate-wise. Therefore, we simply let the Client compute the part related to $\{\tilde{x}^{(i)}\}_{i=1}^{N}$ and let all parties compute the part related to $\{[y^{(i)}]_t\}_{i=1}^{N}$.

The communication complexity of Output($\mathsf{Client}, \{[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!]\}$) is $O(N \cdot n + n^2 \cdot m + n \cdot m^2)$ elements in $\mathbb{F}_q$ plus N calls of $\mathcal{F}_{\text{zero}}$ and m calls of $\mathcal{F}_{\text{parity}}$.

Lemma 9. *Protocol* Output *securely computes $\mathcal{F}_{output}$ with abort in the $\{\mathcal{F}_{zero}, \mathcal{F}_{parity}, \mathcal{F}_{coin}\}$-hybrid model in the presence of a malicious adversary who controls t parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map. For a vector $\boldsymbol{z} := (z_1, \ldots, z_m) \in (\mathbb{F}_{q^m})^m$, we use $\mathsf{val}(\boldsymbol{z})$ to denote $(\mathsf{val}(z_1), \ldots, \mathsf{val}(z_m))$.

Protocol 24: $\textsc{Output}(\mathsf{Client}, \{[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!]\})$

1. For all $i \in [N]$, suppose $[\![x^{(i)}]\!] = (\langle x^{(i)} \rangle, [y^{(i)}]_t)$. All parties invoke $\mathcal{F}_{\text{zero}}$ to prepare a random additive sharing $\langle o^{(i)} \rangle$ such that $o^{(i)} = 0$. The first $t+1$ parties send their shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ to the Client.
2. For all $i \in [N]$, the Client reconstructs $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$. Let $\tilde{x}^{(i)}$ denote the reconstruction result.
3. Let $N' = N/m$. For all $i \in [N']$, let
$$(\boldsymbol{v}^{(i)}, [\boldsymbol{z}^{(i)}]_t) := ((\tilde{x}^{((i-1)m+1)}, [y^{((i-1)m+1)}]_t), \ldots, (\tilde{x}^{(i \cdot m)}, [y^{(i \cdot m)}]_t)).$$
The Client takes $\boldsymbol{v}^{(i)}$ and all parties take $[\boldsymbol{z}^{(i)}]_t$.
4. All parties invoke $\mathcal{F}_{\text{parity}}$ to generate m random parity sharings $[p^{(1)}]_t, [p^{(2)}]_t, \ldots, [p^{(m)}]_t$. Let
$$(\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t) := ((0, [p^{(1)}]_t), \ldots, (0, [p^{(m)}]_t)).$$
The Client takes $\boldsymbol{v}^{(0)}$ (which is just an all-0 vector) and all parties take $[\boldsymbol{z}^{(0)}]_t$.
5. All parties invoke $\mathcal{F}_{\text{coin}}(\mathbb{F}_{q^m})$ to generate a random element $r \in \mathbb{F}_{q^m}$. All parties send r to the Client. The Client takes the majority value as r.
6. Let
$$(\boldsymbol{v}, [\boldsymbol{z}]_t) := (\boldsymbol{v}^{(0)}, [\boldsymbol{z}^{(0)}]_t) + (\boldsymbol{v}^{(1)}, [\boldsymbol{z}^{(1)}]_t) \cdot r + \ldots + (\boldsymbol{v}^{(N')}, [\boldsymbol{z}^{(N')}]_t) \cdot r^{N'}.$$
The Client computes the part $\boldsymbol{v}$ and all parties compute the part $[\boldsymbol{z}]_t$.
7. All parties send their shares of $[\boldsymbol{z}]_t$ to the Client. Let $(\boldsymbol{v}, [\boldsymbol{z}]_t) = ((v_1, [z_1]_t), \ldots, (v_m, [z_m]_t))$. For all $i \in [m]$, the Client checks whether $[z_i]_t$ is a valid degree-t Shamir sharing and $\mathsf{val}(z_i) = v_i$. If not, the Client aborts. Otherwise, the Client accepts the reconstructions.

Simulation for $\textsc{Open}$. At the beginning, for each $i \in [N]$, $\mathcal{S}$ receives the shares of $[y^{(i)}]_t$ held by corrupted parties and $x_{\mathcal{C}}^{(i)}$ from $\mathcal{F}_{\text{output}}$. Recall that $x_{\mathcal{C}}^{(i)}$ is the summation of the shares of $\langle x^{(i)} \rangle$ held by corrupted parties. Depending on whether the Client is honest or not, there are two cases.

Case 1: The Client is honest.

In this case, since the Client is honest, the simulator only checks whether the Client can reconstruct the correct outputs using the shares sent by corrupted parties in the first step.

In Step 1, for all $i \in [N]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{zero}}$ and receives the shares of $\langle o^{(i)} \rangle$ held by corrupted parties. Then $\mathcal{S}$ computes the summation of the shares of $\langle o^{(i)} \rangle$ held by corrupted parties, denoted by $o_{\mathcal{C}}^{(i)}$. $\mathcal{S}$ further computes $x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)}$.

Next, $\mathcal{S}$ receives the shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ held by corrupted parties. $\mathcal{S}$ checks that whether the summation of the shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ held by corrupted parties is $x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)}$. If not, $\mathcal{S}$ marks this execution as `fail`.

From Step 2 to Step 6, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\text{parity}}$ and $\mathcal{F}_{\text{coin}}$ when they are invoked. $\mathcal{S}$ further computes the shares of $[\boldsymbol{z}]_t$ held by corrupted parties. These shares can be computed from the shares of $\{[p^{(i)}]_t\}_{i=1}^m$ held by corrupted parties which are received when emulating $\mathcal{F}_{\text{parity}}$, and the shares of $\{[y^{(i)}]_t\}_{i=1}^N$ held by corrupted parties which are received from $\mathcal{F}_{\text{output}}$.

In Step 7, $\mathcal{S}$ receives from the adversary the shares of $[\boldsymbol{z}]_t$ held by corrupted parties. If they are different from the shares computed by $\mathcal{S}$, $\mathcal{S}$ marks this execution as `fail`.

Finally, if $\mathcal{S}$ has marked this execution as `fail`, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{output}}$ and aborts. Otherwise, $\mathcal{S}$ sends `continue` to $\mathcal{F}_{\text{output}}$.

Case 2: The Client is corrupted.

In this case, $\mathcal{S}$ receives from $\mathcal{F}_{\text{output}}$ $\{\mathsf{val}(y^{(i)})\}_{i=1}^N$.

In Step 1, for all $i \in [N]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{zero}}$ and receives the shares of $\langle o^{(i)} \rangle$ held by corrupted parties. Then $\mathcal{S}$ computes the summation of the shares of $\langle o^{(i)} \rangle$ held by corrupted parties, denoted by $o_{\mathcal{C}}^{(i)}$. $\mathcal{S}$ further computes $x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)}$.

Note that $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ should be a random additive sharing of $x^{(i)} = \mathsf{val}(y^{(i)})$. $\mathcal{S}$ randomly samples the shares of honest parties in $\{P_1, \ldots, P_{t+1}\}$ such that the summation of these shares is $\mathsf{val}(y^{(i)}) - (x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)})$. Finally, $\mathcal{S}$ sends these shares to the Client.

From Step 2 to Step 6, $\mathcal{S}$ faithfully emulates $\mathcal{F}_{\text{parity}}$ and $\mathcal{F}_{\text{coin}}$ when they are invoked. $\mathcal{S}$ further computes $\boldsymbol{v}$ and the shares of $[\boldsymbol{z}]_t$ held by corrupted parties. These values can be computed from $\{\mathsf{val}(y^{(i)})\}_{i=1}^N$, the shares of $\{[p^{(i)}]_t\}_{i=1}^m$ held by corrupted parties which are received when emulating $\mathcal{F}_{\text{parity}}$, and the shares of $\{[y^{(i)}]_t\}_{i=1}^N$ held by corrupted parties which are received from $\mathcal{F}_{\text{output}}$.

Let $(\boldsymbol{v}, [\boldsymbol{z}]_t) = ((v_1, [z_1]_t), \ldots, (v_m, [z_m]_t))$. Note that for all $i \in [N]$, we should have $\mathsf{val}(z_i) = v_i$. Also note that for all $i \in [m]$, $p^{(i)}$ is a uniformly random parity element. Since z_i is masked by $p^{(i)}$, z_i is uniformly distributed given $\mathsf{val}(z_i) = v_i$. Let $\boldsymbol{e}$ be a vector in $\mathbb{F}_q^k$ such that the first entry is 1 and the rest of entries are 0. Let $\delta = \phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})$. By the property of RMFE, $\psi(\phi(\boldsymbol{e}) \cdot \phi(\boldsymbol{e})) = \boldsymbol{e} * \boldsymbol{e} = \boldsymbol{e}$. Therefore, $\mathsf{val}(\delta) = 1$. For all $i \in [m]$, to sample a random element z_i given $\mathsf{val}(z_i) = v_i$, $\mathcal{S}$ randomly samples a parity element r and sets $z_i := r + v_i \cdot \delta$. Then, based on the secret z_i and the shares of $[z_i]_t$ held by corrupted parties, $\mathcal{S}$ computes the shares of $[z_i]_t$ held by honest parties.

In Step 7, since $\mathcal{S}$ has computed the shares of $[\boldsymbol{z}]_t$ held by honest parties, $\mathcal{S}$ faithfully follows the protocol. If the Client aborts, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{output}}$ and aborts.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Conisder the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ simulates the behaviors of honest parties and the Client when the Client is honest. Note that when the Client is honest, $\mathcal{S}$ does not need to send any message to the adversary. Therefore, we only need to focus on the outputs of honest parties and the Client. There are two possible outcomes: (1) the Client receives the correct outputs and honest parties receive nothing; (2) the Client aborts and honest parties receive `abort`. We first show the following three points:

- In Step 1, for all $i \in [N]$, if and only if $\mathcal{S}$ marks the execution as `fail` when checking the shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ held by corrupted parties, $\tilde{x}^{(i)}$ reconstructed by the Client in $\mathbf{Hybrid}_0$ is incorrect. Note that the sharing $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ is reconstructed by computing the summation of all the shares. In this hybrid, $\mathcal{S}$ checks the correctness of the summation of the shares held by corrupted parties. If the check does not pass, $\mathcal{S}$ will mark the execution as `fail`, and the Client will not reconstruct the correct output.
- In Step 7, if $\mathcal{S}$ marks the execution as `fail` when checking the shares of $[\boldsymbol{z}]_t$ held by corrupted parties, the Client in $\mathbf{Hybrid}_0$ will abort. Note that a degree-t Shamir sharing is determined by the shares held by honest parties. Therefore, if the shares of $[\boldsymbol{z}]_t$ held by corrupted parties do not match the shares computed by $\mathcal{S}$, the Client in $\mathbf{Hybrid}_0$ will receive inconsistent sharings $[\boldsymbol{z}]_t$ and abort.
- If there exists $i \in [N]$ such that $\tilde{x}^{(i)}$ reconstructed by the Client in $\mathbf{Hybrid}_0$ is incorrect, then with overwhelming probability, the Client will abort in Step 7. This follows from the same argument as that in the proof of Lemma 8.

Consider the following three cases:

- If $\mathcal{S}$ does not mark the execution as `fail`, then in $\mathbf{Hybrid}_1$, the Client receives the correct outputs and honest parties receive nothing. According to the first point, the Client reconstructs the correct outputs in $\mathbf{Hybrid}_0$. Also, corrupted parties provide the correct shares of $[\boldsymbol{z}]_t$ in Step 7. Therefore, in $\mathbf{Hybrid}_0$, the Client receives the correct outputs and honest parties receive nothing.
- If $\mathcal{S}$ marks the execution as `fail` in Step 1, then in $\mathbf{Hybrid}_1$, the Client aborts and honest parties receive `abort`. According to the first point, at least one $\tilde{x}^{(i)}$ reconstructed by the Client in $\mathbf{Hybrid}_0$ is incorrect. According to the third point, the Client in $\mathbf{Hybrid}_0$ will abort with overwhelming probability.
- If $\mathcal{S}$ marks the execution as `fail` in Step 7, then in $\mathbf{Hybrid}_1$, the Client aborts and honest parties receive `abort`. According to the second point, the Client in $\mathbf{Hybrid}_0$ will also abort.

Therefore, the distribution of $\mathbf{Hybrid}_1$ is statistically close to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ simulates the behaviors of honest parties when the Client is corrupted. Note that in this case, honest parties do not receive any message from corrupted parties or the Client. Therefore, we only need to focus on the shares which are sent from honest parties to the Client. There are two places where honest parties need to send shares.

- In Step 1, for all $i \in [N]$, honest parties need to send their shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ to the Client. Note that the secret of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ should be $x^{(i)} = \mathsf{val}(y^{(i)})$, which is received from $\mathcal{F}_{\text{output}}$. Also the summation of the shares held by corrupted parties should be $x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)}$. Therefore, the summation of the shares held by honest parties should be $\mathsf{val}(y^{(i)}) - (x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)})$. Since $\langle o^{(i)} \rangle$ is a uniformly random additive sharing of 0, the shares of $\langle x^{(i)} \rangle + \langle o^{(i)} \rangle$ held by honest parties are uniformly distributed with the constraint that the summation of these shares is $\mathsf{val}(y^{(i)}) - (x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)})$. In this hybrid, $\mathcal{S}$ uniformly samples the shares of honest parties in the first $t+1$ parties such that the summation of these shares is $\mathsf{val}(y^{(i)}) - (x_{\mathcal{C}}^{(i)} + o_{\mathcal{C}}^{(i)})$. Thus, the shares of honest parties in both hybrids have the same distribution.
- In Step 7, honest parties need to send their shares of $[\boldsymbol{z}]_t$ to the Client. Note that the secret $\boldsymbol{z}$ should satisfies that $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$. Since $\boldsymbol{z}^{(0)} = (p^{(1)}, p^{(2)}, \ldots, p^{(m)})$ is a vector of m uniformly random parity elements, $\boldsymbol{z}$ should be a vector of random elements in $\mathbb{F}_{q^m}$ given $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$. In this hybrid, $\mathcal{S}$ uniformly samples $\boldsymbol{z}$ such that $\mathsf{val}(\boldsymbol{z}) = \boldsymbol{v}$. Therefore, $\boldsymbol{z}$ in both hybrids have the same distribution. Since the shares of $[\boldsymbol{z}]_t$ held by honest parties are determined by the shares of corrupted parties and the secret $\boldsymbol{z}$, we conclude that $[\boldsymbol{z}]_t$ in both hybrids have the same distribution.

Therefore, the distribution of $\mathbf{Hybrid}_2$ is identical to the distribution of $\mathbf{Hybrid}_1$.

Note that $\mathbf{Hybrid}_2$ is the execution in the ideal world and $\mathbf{Hybrid}_2$ is statistically close to $\mathbf{Hybrid}_0$, the execution in the real world. □

7.5 Main Protocol

Now we are ready to introduce our main construction. Recall that we are in the client-server model. In particular, all the inputs belong to the clients, and only the clients receive the outputs. The functionality $\mathcal{F}_{\text{main}}$ is described in Functionality 25. The protocol Main appears in Protocol 26.

Functionality 25: $\mathcal{F}_{\text{main}}$

1. $\mathcal{F}_{\text{main}}$ receives from all clients their inputs.
2. $\mathcal{F}_{\text{main}}$ evaluates the circuit and computes the outputs. $\mathcal{F}_{\text{main}}$ first sends the outputs of corrupted clients to the adversary.
   - If the adversary replies `continue`, $\mathcal{F}_{\text{main}}$ distributes the outputs to honest clients.
   - If the adversary replies `abort`, $\mathcal{F}_{\text{main}}$ sends `abort` to honest clients.

Theorem 4. *Let c be the number of clients and $n = 2t + 1$ be the number of parties. The protocol* Main *securely computes $\mathcal{F}_{main}$ with abort in $\{\mathcal{F}_{tuple}, \mathcal{F}_{random}, \mathcal{F}_{zero}, \mathcal{F}_{coin}, \mathcal{F}_{checkRecon}, \mathcal{F}_{output}\}$-hybrid model in the presence of a fully malicious adversary controlling up to c clients and t parties.*

Proof. According to Theorem 2, we assume that the adversary controls exactly t parties. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties, and $\mathcal{H}$ denote the set of honest parties.

Protocol 26: MAIN

Let (ϕ, ψ) be an $(k, m)_q$-RMFE. Recall that $\mathsf{val}(\cdot) : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is an $\mathbb{F}_q$-linear map, which is defined by $\mathsf{val}(y) = \sum_{i=1}^{k} y_i$ where $(y_1, \ldots, y_k) = \psi(y)$. A pair of couple sharings $[\![x]\!] := (\langle x \rangle, [y]_t)$ satisfies that $\mathsf{val}(y) = x$.

1. **Preparing Beaver Tuples**: Let c_M denote the number of multiplication gates in the circuit. All parties invoke $\mathcal{F}_{\text{tuple}}$ to prepare c_M/k random tuples in the form of

$$([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t),$$

   where $\boldsymbol{a}, \boldsymbol{b}$ are random vectors in $\mathbb{F}_q^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Then all parties invoke SEPARATE to locally transform these c_M/k tuples into c_M random Beaver tuples in the form of

$$([\![a]\!], [\![b]\!], [\![c]\!]),$$

   where a, b are random elements in $\mathbb{F}_q$ and $c = a \cdot b$.
2. **Input Phase**: For every Client with k inputs $x^{(1)}, \ldots, x^{(k)} \in \mathbb{F}_q$, all parties and the Client invoke $\text{INPUT}(\mathsf{Client}, \{x^{(1)}, \ldots, x^{(k)}\})$. At the end of the protocol, all parties take the couple sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(k)}]\!]$ as output.
3. **Computation Phase**: All parties start with holding a pair of couple sharings for each input gate. The circuit is evaluated in a predetermined topological order.
   - For each addition gate with input sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!]$, all parties locally compute $[\![x^{(0)}]\!] := [\![x^{(1)} + x^{(2)}]\!] = [\![x^{(1)}]\!] + [\![x^{(2)}]\!]$.
   - For each multiplication gate with input sharings $[\![x^{(1)}]\!], [\![x^{(2)}]\!]$, all parties invoke MULT with the first *unused* Beaver tuple $([\![a]\!], [\![b]\!], [\![c]\!])$ to compute $[\![x^{(0)}]\!]$. Let $u^{(1)}, u^{(2)}$ denote the reconstruction results of $[\![x^{(1)} + a]\!], [\![x^{(2)} + b]\!]$ sent by P_1 in Step 3 of MULT.
     Suppose $[w^{(1)}]_t$ is the degree-t Shamir sharing in $[\![x^{(1)} + a]\!]$, and $[w^{(2)}]_t$ is the degree-t Shamir sharing in $[\![x^{(2)} + b]\!]$. All parties will use $(u^{(1)}, [w^{(1)}]_t)$ and $(u^{(2)}, [w^{(2)}]_t)$ to verify the reconstructions.
4. **Verification phase**:
   - Suppose that $u^{(1)}, u^{(2)}, \ldots, u^{(c_I)}$ are the values all parties receive from the clients in INPUT, and $u^{(c_I+1)}, \ldots, u^{(c_I+2 \cdot c_M)}$ are the values all parties receive from P_1 in MULT, where c_I denotes the number of inputs and c_M denotes the number of multiplications. All parties invoke $\text{CHECKCONSISTENCY}(c_I + 2 \cdot c_M, \{u^{(1)}, \ldots, u^{(c_I+2 \cdot c_M)}\})$ to verify that they receive the same values.
   - Suppose $(u^{(1)}, [w^{(1)}]_t), \ldots, (u^{(2 \cdot c_M)}, [w^{(2 \cdot c_M)}]_t)$ are the value-sharing pairs generated when evaluating multiplication gates. All parties invoke $\mathcal{F}_{\text{checkRecon}}$ to verify that for all $i \in [2 \cdot c_M]$, $\mathsf{val}(w^{(i)}) = u^{(i)}$.
5. **Output Phase**: For every Client, let $[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!]$ denote the sharings associated with the output gates, which should be reconstructed to the Client. All parties and the Client invoke $\mathcal{F}_{\text{output}}$ on these N pairs of couple sharings.

Simulation for MAIN. We describe the strategy of $\mathcal{S}$ phase by phase.

- **Preparing Beaver Tuples**: In this step, $\mathcal{S}$ emulates $\mathcal{F}_{\text{tuple}}$ and receives the shares of

$$\{[\phi(\boldsymbol{a}^{(i)})]_t, [\phi(\boldsymbol{b}^{(i)})]_t, [\phi(\boldsymbol{c}^{(i)})]_t\}_{i=1}^{c_M/k}$$

  held by corrupted parties. Then $\mathcal{S}$ follows SEPARATE to compute the shares of

$$\{([\![a^{(i)}]\!], ([\![b^{(i)}]\!], [\![c^{(i)}]\!])\}_{i=1}^{c_M}$$

  held by corrupted parties.
- **Input Phase**: In this step, $\mathcal{S}$ emulates $\mathcal{F}_{\text{random}}$ and receives the shares of $[\phi(\boldsymbol{r})]_t$ held by corrupted parties. Depending on whether the Client is corrupted, there are two cases:
  - If the Client is corrupted, $\mathcal{S}$ faithfully generates the whole sharing $[\phi(\boldsymbol{r})]_t$ when emulating $\mathcal{F}_{\text{random}}$. Then $\mathcal{S}$ sends the shares of $[\phi(\boldsymbol{r})]_t$ held by honest parties to the Client. After receiving $\boldsymbol{x} + \boldsymbol{r}$ from

the Client, $\mathcal{S}$ extracts the input of the Client. If the Client sends different values to different honest parties, $\mathcal{S}$ uses the values of the first honest party and marks this execution as `fail`.

- If the Client is honest, $\mathcal{S}$ receives the shares of $[\phi(\boldsymbol{r})]_t$ from corrupted parties. $\mathcal{S}$ checks whether these shares are the same as the shares received when emulating $\mathcal{F}_{\text{random}}$. If not, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{main}}$ and aborts. Otherwise, $\mathcal{S}$ samples a random vector in $\mathbb{F}_q^k$ and regards it as $\boldsymbol{x}+\boldsymbol{r}$. $\mathcal{S}$ sends $\boldsymbol{x}+\boldsymbol{r}$ to corrupted parties.

If $\mathcal{S}$ successfully extracts all the inputs of corrupted clients, $\mathcal{S}$ sends these inputs to $\mathcal{F}_{\text{main}}$. Finally, $\mathcal{S}$ follows SEPARATE to compute the shares of $[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(k)}]\!]$ held by corrupted parties.

– **Computation Phase**: In this step, $\mathcal{S}$ will compute the shares of each pair of couple sharings held by corrupted parties. Note that these shares have computed in the input phase for couple sharings associated with the input gates.

For each addition gate, $\mathcal{S}$ follows the protocol to locally compute the shares of the resulting couple sharings held by corrupted parties.

For each multiplication gate, We describe the strategy of $\mathcal{S}$ for MULT.

1. $\mathcal{S}$ emulates $\mathcal{F}_{\text{zero}}$ and receives the shares of $\langle o^{(1)}\rangle, \langle o^{(2)}\rangle$ held by corrupted parties.
2. For the additive sharing $\langle u^{(1)}\rangle = \langle x^{(1)}+a\rangle + \langle o^{(1)}\rangle$, for each honest party in $\{P_1, \ldots, P_{t+1}\}$, $\mathcal{S}$ samples a random element in $\mathbb{F}_q$ as its share. Similar process is done for $\langle u^{(2)}\rangle = \langle x^{(2)}+b\rangle + \langle o^{(2)}\rangle$. Then $\mathcal{S}$ sends the shares of $\langle u^{(1)}\rangle, \langle u^{(2)}\rangle$ held by honest parties to P_1. $\mathcal{S}$ then uses the shares held by corrupted parties to reconstruct $u^{(1)}, u^{(2)}$. Let $[w^{(1)}]_t, [w^{(2)}]_t$ denote the degree-t Shamir sharings in $[\![x^{(1)}+a]\!], [\![x^{(2)}+b]\!]$ respectively. $\mathcal{S}$ computes the shares of $[w^{(1)}]_t, [w^{(2)}]_t$ held by corrupted parties.
3. If P_1 is honest, $\mathcal{S}$ faithfully follows the protocol by reconstructing $\langle u^{(1)}\rangle, \langle u^{(2)}\rangle$ using the shares *received from other parties*. Let $\tilde{u}^{(1)}, \tilde{u}^{(2)}$ denote the reconstruction results. $\mathcal{S}$ sends $\tilde{u}^{(1)}, \tilde{u}^{(2)}$ to all other parties. If P_1 is corrupted, $\mathcal{S}$ receives $\tilde{u}^{(1)}, \tilde{u}^{(2)}$ from P_1. If P_1 sends different values to different honest parties, $\mathcal{S}$ uses the values of the first honest party and marks this execution as `fail`.
4. $\mathcal{S}$ follows the protocol to compute the shares of $[\![x^{(0)}]\!]$ held by corrupted parties.

– **Verification Phase**: For CHECKCONSISTENCY, note that $u^{(1)}, \ldots, u^{(c_I+2\cdot c_M)}$ are either received from corrupted clients and the corrupted P_1 or explicitly generated by $\mathcal{S}$. $\mathcal{S}$ follows the protocol honestly. If any party aborts *or $\mathcal{S}$ has marked this execution as* `fail`, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{main}}$ and aborts.

For $\mathcal{F}_{\text{checkRecon}}$, $\mathcal{S}$ emulates the functionality as follows:

1. The value-sharing pairs are denoted by $(\tilde{u}^{(1)}, [w^{(1)}]_t), \ldots, (\tilde{u}^{(2\cdot c_M)}, [w^{(2\cdot c_M)}]_t)$. Here $\tilde{u}^{(1)}, \ldots, \tilde{u}^{(2\cdot c_M)}$ are the results reconstructed by P_1. Recall that $\mathcal{S}$ also computes the correct reconstruction results when simulating MULT, which are denoted by $u^{(1)}, \ldots, u^{(2\cdot c_M)}$.
2. For all $i \in [2\cdot c_M]$, $\mathcal{S}$ sends the shares of $[w^{(i)}]_t$ held by corrupted parties to the adversary.
3. For all $i \in [2\cdot c_M]$, $\mathcal{S}$ sends $\tilde{u}^{(i)}, u^{(i)}$ to the adversary.
4. If there exists $i \in [2\cdot c_M]$ such that $\tilde{u}^{(i)} \neq u^{(i)}$, $\mathcal{S}$ sets $b = $ `abort`. Otherwise, $\mathcal{S}$ sets $b = $ `accept`. Then $\mathcal{S}$ follows this step in the functionality.

– **Output Phase**: $\mathcal{S}$ receives from $\mathcal{F}_{\text{main}}$ the outputs of corrupted clients. For every Client who should receive the reconstruction results of

$$[\![x^{(1)}]\!], [\![x^{(2)}]\!], \ldots, [\![x^{(N)}]\!],$$

we describe the strategy of $\mathcal{S}$ for $\mathcal{F}_{\text{output}}$. For all $i \in [N]$, recall that $\mathcal{S}$ has computed the shares of $[\![x^{(i)}]\!] = (\langle x^{(i)}\rangle, [y^{(i)}]_t)$ held by corrupted parties. For $[y^{(i)}]_t$, $\mathcal{S}$ sends the shares held by corrupted parties to the adversary. For $\langle x^{(i)}\rangle$, $\mathcal{S}$ computes the summation of the shares of corrupted parties, denoted by $x_{\mathcal{C}}^{(i)}$, and sends $x_{\mathcal{C}}^{(i)}$ to the adversary. Depending on whether the Client is honest, there are two cases.

- If the Client is corrupted, $\mathcal{S}$ sends $\{\mathsf{val}(y^{(i)})\}_{i=1}^N$ to the adversary. Note that these are the outputs of the Client, which are received from $\mathcal{F}_{\text{main}}$. Then $\mathcal{S}$ receives the response from the adversary.
- If the Client is honest, $\mathcal{S}$ receives the response from the adversary.

If the adversary replies `abort` in any call of $\mathcal{F}_{\text{output}}$, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{main}}$ and aborts. Otherwise, $\mathcal{S}$ sends `continue`.

Hybrid Arguments. Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties with overwhelming probability. Consider the following hybrids.

$\mathbf{Hybrid}_0$: The execution in the real world.

$\mathbf{Hybrid}_1$: In this hybrid, $\mathcal{S}$ computes the inputs of corrupted clients and sends them to $\mathcal{F}_{\text{main}}$. The distribution of $\mathbf{Hybrid}_1$ is identical to the distribution of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ simulates CheckConsistency. Concretely, $\mathcal{S}$ checks whether all honest parties receive the same values from clients in Input and P_1 in Mult. If not, $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{main}}$ and aborts. According to Lemma 7, the probability that an honest party aborts in this case is overwhelming. Therefore, the distribution of $\mathbf{Hybrid}_2$ is statistically close to the distribution of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ uses the outputs of corrupted clients received from $\mathcal{F}_{\text{main}}$ in the output phase. Note that in $\mathbf{Hybrid}_2$, $\mathcal{S}$ has checked that all parties receive the same values from clients in Input and P_1 in Mult. The functionality $\mathcal{F}_{\text{checkRecon}}$ ensures that the reconstructions are correct if no party aborts. Therefore, the computation is correct and corrupted clients should receive the same outputs as those computed by $\mathcal{F}_{\text{main}}$. Thus, the distribution of $\mathbf{Hybrid}_3$ is statistically close to the distribution of $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_4$: In this hybrid, $\mathcal{S}$ computes the shares of couple sharings held by corrupted parties as described above. Then $\mathcal{S}$ simulates the output phase as described above. Note that when using the shares computed by $\mathcal{S}$ for corrupted parties, all the couple sharings are valid. For $[y^{(i)}]_t$, the shares of corrupted parties are determined by the shares of honest parties. Therefore, the shares of $[y^{(i)}]_t$ held by corrupted parties provided by $\mathcal{S}$ in both hybrids are identical. For $\langle x^{(i)} \rangle$, note that the secret $x^{(i)} = \mathsf{val}(y^{(i)})$ is the summation of the shares of all parties. Therefore, the summation of the shares held by corrupted parties is determined by $x^{(i)}$ and the shares held by honest parties. Therefore, $x_{\mathcal{C}}^{(i)}$ provided by $\mathcal{S}$ in both hybrids are identical. Thus, the distribution of $\mathbf{Hybrid}_4$ is identical to the distribution of $\mathbf{Hybrid}_3$.

$\mathbf{Hybrid}_5$: In this hybrid, $\mathcal{S}$ simulates the computation phase. Note that for addition gates, $\mathcal{S}$ simply computes the shares of corrupted parties, which is identical to $\mathbf{Hybrid}_4$. For multiplication gates, the only difference is that $\mathcal{S}$ uses uniformly random elements as the shares of $\langle u^{(1)} \rangle, \langle u^{(2)} \rangle$ for honest parties in the first $t+1$ parties. Recall that $\langle u^{(1)} \rangle = \langle x^{(1)} + a \rangle + \langle o^{(1)} \rangle$. Since a is uniformly sampled by $\mathcal{F}_{\text{tuple}}$ and $\langle o^{(1)} \rangle$ is a uniformly random additive sharing of 0 given the shares of corrupted parties, $\langle x^{(1)} + a \rangle + \langle o^{(1)} \rangle$ is a uniformly random additive sharing given the shares of corrupted parties. Therefore, the share of $\langle u^{(1)} \rangle = \langle x^{(1)} + a \rangle + \langle o^{(1)} \rangle$ held by an honest party in $\{P_1, \ldots, P_{t+1}\}$ is uniformly distributed. A similar argument works for $\langle u^{(2)} \rangle$. Therefore, the distribution of $\mathbf{Hybrid}_4$ is identical to the distribution of $\mathbf{Hybrid}_5$.

$\mathbf{Hybrid}_6$: In this hybrid, $\mathcal{S}$ simulates $\mathcal{F}_{\text{checkRecon}}$. Recall that in $\mathbf{Hybrid}_4$, $\mathcal{S}$ has computed the shares of couple sharings held by corrupted parties. When using the shares computed by $\mathcal{S}$ for corrupted parties, all the couple sharings are valid. For $[w^{(i)}]_t$, the shares of corrupted parties are determined by the shares of honest parties. Therefore, the shares of $[w^{(i)}]_t$ held by corrupted parties provided by $\mathcal{S}$ in both hybrids are identical. Since $\mathcal{S}$ has computed the correct reconstructions when simulating Mult in $\mathbf{Hybrid}_5$, the values $\tilde{u}^{(i)}, u^{(i)}$ provided by $\mathcal{S}$ in $\mathbf{Hybrid}_6$ are identical to $\tilde{u}^{(i)}, \mathsf{val}(w^{(i)})$ provided by $\mathcal{F}_{\text{checkRecon}}$ in $\mathbf{Hybrid}_5$. Therefore, the distribution of $\mathbf{Hybrid}_6$ is identical to the distribution of $\mathbf{Hybrid}_5$.

$\mathbf{Hybrid}_7$: In this hybrid, $\mathcal{S}$ simulates the input phase. Note that the main difference is that $\mathcal{S}$ does not generate the whole sharing $[\phi(\boldsymbol{r})]_t$ when the Client is honest, and $\boldsymbol{x} + \boldsymbol{r}$ is randomly sampled in $\mathbb{F}_q^k$. Since $\boldsymbol{r}$ is uniformly distributed in $\mathbb{F}_q^k$, $\boldsymbol{x} + \boldsymbol{r}$ is also a uniformly random vector in $\mathbb{F}_q^k$. Therefore, the distribution of $\boldsymbol{x} + \boldsymbol{r}$ in this hybrid is the same as that in $\mathbf{Hybrid}_6$. The other difference is that, when the Client is honest, $\mathcal{S}$ only checks whether the shares of $[\phi(\boldsymbol{r})]_t$ received from corrupted parties are identical to the shares received when emulating $\mathcal{F}_{\text{random}}$. Note that the shares of $[\phi(\boldsymbol{r})]_t$ held by corrupted parties are determined by the shares of honest parties. If corrupted parties send a different set of values, then the sharing $[\phi(\boldsymbol{r})]_t$ must be inconsistent in $\mathbf{Hybrid}_6$. In summary, the distribution of $\mathbf{Hybrid}_7$ is identical to the distribution of $\mathbf{Hybrid}_6$.

$\mathbf{Hybrid}_8$: In this hybrid, $\mathcal{S}$ simulates $\mathcal{F}_{\text{tuple}}$. Note that only the shares of corrupted parties are used in the rest of the simulation. The distribution of $\mathbf{Hybrid}_8$ is identical to the distribution of $\mathbf{Hybrid}_7$.

Note that $\mathbf{Hybrid}_8$ is the execution in the ideal world, and the distribution of $\mathbf{Hybrid}_8$ is statistically close to the distribution of $\mathbf{Hybrid}_0$, the execution in the real world. □

Analysis of the Communication Complexity of Main. Let c_I, c_M, c_O denote the numbers of input gates, multiplication gates, and output gates. Recall that c is the number of clients. In Main, we need to invoke

- c_M/k times of $\mathcal{F}_{\text{tuple}}$ in Step 1, which has communication complexity $O(c_M \cdot n \cdot m/k + n^3 \cdot m + n^2 \cdot \log(c_M/k) \cdot m)$ elements in $\mathbb{F}_q$,
- c_I/k times of Input in Step 2, which has communication complexity $O(c_I \cdot (m+k)/k)$ elements in $\mathbb{F}_q$ and c_I/k calls of $\mathcal{F}_{\text{random}}$,
- c_M times of Mult in Step 3, which has communication complexity $O(c_M \cdot n)$ elements in $\mathbb{F}_q$ and $2 \cdot c_M$ calls of $\mathcal{F}_{\text{zero}}$,
- one time of CheckConsistency in Step 4, which has communication complexity $O(n^2 \cdot m)$ elements in $\mathbb{F}_q$,
- one time of $\mathcal{F}_{\text{checkRecon}}$ in Step 4, which has communication complexity $O(n^2 \cdot m^2)$ elements in $\mathbb{F}_q$ plus m calls of $\mathcal{F}_{\text{parity}}$,
- c times of $\mathcal{F}_{\text{output}}$ in Step 5, which has communication complexity $O(c_O \cdot n + c \cdot n^2 \cdot m + c \cdot n \cdot m^2)$ elements in $\mathbb{F}_q$ plus c_O calls of $\mathcal{F}_{\text{zero}}$ and $c \cdot m$ calls of $\mathcal{F}_{\text{parity}}$.

For $\mathcal{F}_{\text{random}}, \mathcal{F}_{\text{zero}}, \mathcal{F}_{\text{parity}}$, we will instantiate them using Rand with suitable secret sharing schemes. As analyzed in Section 6,

- the communication complexity for c_I/k calls of $\mathcal{F}_{\text{random}}$ is $O(c_I \cdot n \cdot m/k + n^3 \cdot m)$ elements in $\mathbb{F}_q$,
- the communication complexity for $2 \cdot c_M + c_O$ calls of $\mathcal{F}_{\text{zero}}$ is $O((2 \cdot c_M + c_O) \cdot n + n^3 \cdot m)$ elements in $\mathbb{F}_q$,
- the communication complexity for $(c+1) \cdot m$ calls of $\mathcal{F}_{\text{parity}}$ is $O((c+1) \cdot n \cdot m^2 + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

Let $C = c_I + c_M + c_O$ be the size of the circuit. In summary, the communication complexity of Main is

$$O(C \cdot n \cdot m/k + n^2 \cdot \log(C/k) \cdot m + n^3 \cdot m + n^2 \cdot m^2 + c \cdot (n^2 \cdot m + n \cdot m^2))$$

elements in $\mathbb{F}_q$. Recall that we require the extension field $\mathbb{F}_{q^m}$ to satisfy that $q^m \geq 2^\kappa$. Therefore, we use κ as an upper bound of m. According to theorem 3, there exists a family of constant rate $(k,m)_q$-RMFEs with $m = \Theta(k)$. Thus, m/k is a constant. The communication complexity becomes

$$O(C \cdot n + n^2 \cdot \log C \cdot \kappa + n^3 \cdot \kappa + n^2 \cdot \kappa^2 + c \cdot (n^2 \cdot \kappa + n \cdot \kappa^2)) = O(C \cdot n + \mathsf{poly}(c, n, \kappa, \log C))$$

elements in $\mathbb{F}_q$.

Theorem 5. *In the client-server model, let c denote the number of clients, and $n = 2t+1$ denote the number of parties (servers). Let κ denote the security parameter, and $\mathbb{F}_q$ denote a finite field of size q. For an arithmetic circuit over $\mathbb{F}_q$ of size C, there exists an information-theoretic MPC protocol which securely computes the arithmetic circuit with abort in the presence of a fully malicious adversary controlling up to c clients and t parties. The communication complexity of this protocol is $O(C \cdot n + \mathsf{poly}(c, n, \kappa, \log C))$ elements in $\mathbb{F}_q$.*

References

BBCG+19. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 67–97, Cham, 2019. Springer International Publishing.

Bea89. Donald Beaver. Multiparty protocols tolerating half faulty processors. In *Conference on the Theory and Application of Cryptology*, pages 560–572. Springer, 1989.

BGIN20. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 244–276, Cham, 2020. Springer International Publishing.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

BSFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 663–680, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography*, pages 213–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure mpc revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 395–426, Cham, 2018. Springer International Publishing.

CG20. Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based mpc protocol for boolean circuits with good amortized complexity. Cryptology ePrint Archive, Report 2020/162, 2020. `https://eprint.iacr.org/2020/162`.

CGH+18. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority mpc for malicious adversaries. In *Annual International Cryptology Conference*, pages 34–64. Springer, 2018.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

DZ13. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In Amit Sahai, editor, *Theory of Cryptography*, pages 621–641, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

GIP+14. Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 85–114, Cham, 2019. Springer International Publishing.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

GS20. Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority mpc. Cryptology ePrint Archive, Report 2020/134, 2020. https://eprint.iacr.org/2020/134.

GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 618–646, Cham, 2020. Springer International Publishing.

HVW20. Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 184–215, Cham, 2020. Springer International Publishing.

LN17. Yehuda Lindell and Ariel Nof. A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–276. ACM, 2017.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 681–700, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

NV18. Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority mpc by batchwise multiplication verification. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security*, pages 321–339, Cham, 2018. Springer International Publishing.

RBO89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

Yao82. Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.