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.

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.

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]. 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

[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].

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:

- 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.

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.

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

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

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 locally compute $[\![z]\!] := [\![x]\!] + [\![y]\!]$.
- For a multiplication gate with input sharings $[\![x]\!]$ and $[\![y]\!]$, let $([\![a]\!], [\![b]\!], [\![c]\!])$ 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

$$[\![z]\!] := (x + a) \cdot (y + b) - (x + a) \cdot [\![b]\!] - (y + b) \cdot [\![a]\!] + [\![c]\!].$$

  All parties also locally compute $[\![x + a]\!] := [\![x]\!] + [\![a]\!]$ and $[\![y + b]\!] := [\![y]\!] + [\![b]\!]$. 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 $(x + a, [\![x + a]\!])$, where $x + a$ is the reconstruction of $[\![x + a]\!]$. 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.

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. We refer the readers to Section 3.1 in the full version of this paper [PS20] for the security 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. 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 $\mathbb{F}_q$).
- 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. 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.
- The third functionality $\mathcal{F}_{\text{multVerify}}$ allows all parties to verify the correctness of multiplications computed by $\mathcal{F}_{\text{mult}}$. 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.

We refer the readers to Section 3.4 in the full version of this paper [PS20] for the descriptions of these functionalities.

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 1. 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 1: $\mathcal{F}_{\text{rand}}$

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.

We refer the readers to Section 4 in the full version of this paper [PS20] for the construction for $\mathcal{F}_{\text{rand}}$. Suppose the share size of a sharing in Σ is sh field elements in $\mathbb{F}_q$. The communication complexity of preparing N random sharings in Σ is $O(N \cdot n \cdot \mathsf{sh} + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

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 2.

Protocol 2: 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 $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$, 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 3. In Section 6.1 of the full version of this paper [PS20], we show how to use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{random}}$. Relying on the protocol (Section 4 in [PS20]) for $\mathcal{F}_{\text{rand}}$, we can generate N random sharings in the form of $[\phi(\boldsymbol{r})]_t$ with communication of $O(N \cdot n \cdot m + n^3 \cdot m)$ 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 4. In Section 6.2 of the full version of this paper [PS20], we introduce an instantiation of $\mathcal{F}_{\text{tuple}}$. The communication complexity of preparing N tuples of sharings in the form of $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ is $O(N \cdot n \cdot m + n^3 \cdot m + n^2 \cdot \log N \cdot m)$ elements in $\mathbb{F}_q$.

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

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.

In the online phase, each tuple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ will be separated by Separate (Protocol 2) to k Beaver tuples

$$(\llbracket a^{(1)} \rrbracket, \llbracket b^{(1)} \rrbracket, \llbracket c^{(1)} \rrbracket), (\llbracket a^{(2)} \rrbracket, \llbracket b^{(2)} \rrbracket, \llbracket c^{(2)} \rrbracket), \ldots, (\llbracket a^{(k)} \rrbracket, \llbracket b^{(k)} \rrbracket, \llbracket c^{(k)} \rrbracket).$$

A Beaver tuple $(\llbracket a^{(i)} \rrbracket, \llbracket b^{(i)} \rrbracket, \llbracket c^{(i)} \rrbracket)$ 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.

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

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.

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 5. In Section 6.3 of the full version of this paper [PS20], we show how to use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{zero}}$. Relying on the protocol (Section 4 in [PS20]) for $\mathcal{F}_{\text{rand}}$, we can generate N random sharings in the form of $\langle o \rangle$ with communication of $O(N \cdot n + n^3 \cdot m)$ elements in $\mathbb{F}_q$.

Functionality 5: $\mathcal{F}_{\text{zero}}$

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.

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 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 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 6. In Section 6.4 of the full version of this paper [PS20], we show how to use $\mathcal{F}_{\text{rand}}$ to instantiate $\mathcal{F}_{\text{parity}}$. Relying on the protocol (Section 4 in [PS20]) for $\mathcal{F}_{\text{rand}}$, we can generate N random parity sharings with communication of $O(N\cdot n\cdot m+n^3\cdot m)$ elements in $\mathbb{F}_q$.

Functionality 6: $\mathcal{F}_{\text{parity}}$

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.

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 Input, which allows a client to share k inputs to all parties. In the main protocol, we will invoke Input for every client with k inputs.

The description of Input appears in Protocol 7. The communication complexity of 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 7: 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 8. 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.

Protocol 8: $\textsc{Mult}(\llbracket x^{(1)} \rrbracket, \llbracket x^{(2)} \rrbracket, (\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket))$

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.

- 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, $\textsc{Mult}$ 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.

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 9.

The communication complexity of $\textsc{CheckConsistency}(N, \{x^{(1)}, \ldots, x^{(N)}\})$ is $O(n^2 \cdot m)$ elements in $\mathbb{F}_q$.

Protocol 9: $\textsc{CheckConsistency}(N, \{x^{(1)}, \ldots, x^{(N)}\})$

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 2. *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* $\textsc{CheckConsistency}$.

We refer the readers to Section 7.3 in the full version of this paper [PS20] for the proof of Lemma 2.

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 10. In Section 7.3 of the full version of this paper [PS20], we introduce an instantiation of $\mathcal{F}_{\text{checkRecon}}$. The communication complexity of this instantiation is $O(n^2 \cdot m^2)$ elements in $\mathbb{F}_q$ plus m calls of $\mathcal{F}_{\text{parity}}$.

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

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.

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 11. In Section 7.4 of the full version of this paper [PS20], we introduce an instantiation of $\mathcal{F}_{\text{output}}$. The communication complexity of this instantiation 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 $\mathcal{F}_{\text{parity}}$.

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 12. The protocol Main appears in Protocol 13.

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 $\{\mathcal{F}_{tuple}, \mathcal{F}_{random},$*

Functionality 11: $\mathcal{F}_{\text{output}}$

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 `abort`, $\mathcal{F}_{\text{output}}$ sends `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 `abort`, $\mathcal{F}_{\text{output}}$ sends `abort` to the Client and all honest parties. If the adversary replies `continue`, $\mathcal{F}_{\text{output}}$ sends $\{\mathsf{val}(y^{(i)})\}_{i=1}^{N}$ to the Client.

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

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.

$\mathcal{F}_{zero}, \mathcal{F}_{coin}, \mathcal{F}_{checkRecon}, \mathcal{F}_{output}\}$*-hybrid model in the presence of a fully malicious adversary controlling up to* c *clients and* t *parties.*

We refer the readers to Section 7.5 in the full version of this paper [PS20] for the proof of Theorem 4.

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

Protocol 13: 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 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 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.

- 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 *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.

PS20. Antigoni Polychroniadou and Yifan Song. Constant-overhead unconditionally secure multiparty computation over binary fields. Cryptology ePrint Archive, Report 2020/1412, 2020. `https://eprint.iacr.org/2020/1412`.

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.