Improved Multi-Party Fixed-Point Multiplication

Saikrishna Badrinarayanan, Eysa Lee, Peihan Miao and Peter Rindal

Abstract. Machine learning is widely used for a range of applications and is increasingly offered as a service by major technology companies. However, the required massive data collection raises privacy concerns during both training and inference. Privacy-preserving machine learning aims to solve this problem. In this setting, a collection of servers secret share their data and use secure multi-party computation to train and evaluate models on the joint data. All prior work focused on the scenario where the number of servers is two or three. In this work, we study the problem where there are $N \geq 3$ servers amongst whom the data is secret shared.
A key component of machine learning algorithms is to perform fixed-point multiplication with truncation of secret shared decimal values. In this work, we design new protocols for multi-party secure fixed-point multiplication where each of the N parties have one share each of the two values to be multiplied and receive one share of the product at the end of the protocol. We consider three forms of secret sharing - replicated, Shamir, and additive, and design an efficient protocol secure in the presence of a semi-honest adversary for each of the forms. Our protocols are more communication efficient than all prior work on performing multi-party fixed-point multiplication. Additionally, for replicated secret sharing, we design another efficient protocol that is secure in the presence of a malicious adversary. Finally, we leverage our fixed-point multiplication protocols to design secure multi-party computation (MPC) protocols for arbitrary arithmetic circuits that have addition and fixed-point multiplication with truncation gates. All our protocols are proven secure using a standard simulation based security definition. Our protocols for replicated and Shamir sharing work in the presence of an honest majority of parties while the one for additive sharing can tolerate a dishonest majority as well.[1]

Keywords: Fixed-Point Multiplication, Efficient MPC

1 Introduction

Machine learning is widely used to produce models that classify images, perform medical diagnosis, provide recommendations and identify fraudulent transactions among several other applications. Major technology companies provide

[1] This manuscript was written in 2021 and does not reflect any more recent related work.

cloud-based machine learning services [1–4] to their customers that perform both training models using customer data as well as performing inference on pre-trained models. The data being classified or used for training is often sensitive and may come from multiple sources with different privacy requirements. Additionally, for regulatory reasons, or even to retain competitive advantage, entities might be unable to share their private data. Thus, privacy preserving machine learning, which aims to perform both training and inference while maintaining the privacy of user's data, has become increasingly important and is an actively studied research direction.

Secure multi-party computation (MPC) provides a promising solution to this problem. It ensures that, during training, the only information revealed about the data is the final model (or an encrypted version), and during evaluation, the only information revealed is the output label. Though MPC does not provide a full solution to the problem of privacy-preserving machine learning (the models themselves or interactions with them can leak information about data), it provides a first line of defense with strong security guarantees that can be strengthened when combined with orthogonal mechanisms such as differential privacy [5]. The most common setting considered in this line of work is where data owners (clients) secret share their data among multiple servers[2] who perform training on the combined data or apply a (secret shared) pre-trained model to evaluate new data samples. There have been several efficient solutions proposed in literature [6, 17, 20, 30, 32, 34–39, 41, 44]. However, most of them focus on the setting of two or three servers[3] where an adversary can corrupt at most one of them. For larger number of parties, there is still a significant gap between plaintext training/evaluation and privacy-preserving solutions.

In this work, we focus on the scenario where there are $N \geq 3$ servers and a majority of them are honest. While there has been a lot of recent research on improving the efficiency of honest majority N party MPC protocols [10, 19, 25, 29, 31], they are typically suited only for computation over integers. In the case of machine learning algorithms, both training data and intermediate parameters involve decimal values that cannot be natively handled using modular arithmetic. An immediate idea to handle decimal values would be to represent them as integers where the least significant bits represent the fractional part, and choose a large enough modulus to avoid a wrap around. This approach fails when performing many floating point multiplications, which is the case in standard training algorithms where millions of sequential multiplications are performed. Moreover, a large modulus implies a more expensive multiplication that further reduces performance. Alternatively, one could perform fixed-point multiplication using a boolean multiplication circuit inside the MPC protocol. Such a boolean circuit can be evaluated using either secret sharing or garbled circuit based techniques, leading to a significant increase in either round or communication cost of the solution, respectively. Indeed, much of the effort in the space of two or three

[2] Each server can be an independent party or the representative of a data owner.

[3] A few works [11, 18, 21, 32] also consider the setting of four parties with one corruption.

party privacy preserving machine learning via MPC has focused on designing new protocols to perform efficient (rounds and communication cost) fixed point multiplication where the two inputs to be multiplied are secret shared amongst the parties (and so is the output). On the other hand, very few works [16, 36] study *multi-party* fixed point multiplication. Motivated by the above bottleneck in performing efficient privacy preserving machine learning for a large number of parties, in this work, we ask the following question:

Can we design efficient multi-party fixed point multiplication protocols where the inputs and outputs are secret shared amongst the parties?

1.1 Our Contributions

Our main contribution is a conceptually new method for truncating secret shared values, i.e. computing $[\![y]\!] := [\![x]\!]/2^d$ where d is a public value (looking ahead, this denotes the number of decimal bits in a fixed-point value). This core idea can then be combined with a standard multiplication protocol to obtain a fixed-point multiplication protocol. We consider three forms of secret sharing to represent the inputs to be multiplied and the output: replicated, Shamir and additive secret sharing.

Replicated sharing. We design two efficient protocols for multi-party fixed point multiplication where the inputs and output are represented using replicated secret sharing over modulus 2^k. The first protocol is secure against a semi-honest adversary in the presence of an honest majority and requires no offline communication prior to the protocol. The protocol has two forms: a single round protocol requiring $(n^2 - nt)k$ bits of online communication or two rounds requiring only $2nk$ bits of online communication, where n is the number of parties and $t < n/2$ is the number of corrupt parties. Our communication overhead is significantly better than the previous work of Mohassel and Rindal [36] for multi-party fixed point multiplication in the same setting. We compare the cost in Fig. 1. We elaborate on the protocol in Sect. 5.
The second protocol, which builds on the first one, is secure against a malicious adversary in the presence of an honest majority and requires two rounds in the online phase. We refer to Sect. 6 for more details.

Shamir sharing. We design an efficient multi-party fixed point multiplication protocol where the inputs and output are represented using Shamir secret sharing. The protocol requires two rounds in the online phase and is secure against a semi-honest adversary in the presence of an honest majority. Compared to the prior work of Catrina and Saxena [16] for the same setting, our online communication is the same while offline communication is significantly lower since $n, t \ll k$ typically in practice (Fig. 1). Our protocol is described in Sect. 7.

General truncation and additive secret sharing. The aforementioned approaches for replicated and Shamir secret sharing require an honest majority among the parties. This is inherently required by these secret sharing schemes.

Protocol	Online Rounds	Online Comm	Offline Comm
Replicated (Ours)	1 or 2	$(n^2 - nt)k$ or $2nk$	0
Replicated ([36])	1	$(n^2 - nt)k$	$(n^2 - nt)tk$
Shamir (Ours)	2	$2nk$	$(n - t - 1)(t + 1)k$
Shamir ([16])	2	$2nk$	$2k^2n.$

Fig. 1: Communication (in bits) and round complexity of our protocols compared to [16, 36]. n is the number of parties, $t < n/2$ is the number of corrupt parties. For replicated, 2^k is the modulus. For Shamir, $k = \lfloor \mathsf{log}_2(q) \rfloor$ where q is the prime modulus.

Nevertheless, it is not a requirement of our new truncation technique. In fact, our new truncation protocol can be viewed as a general technique that works for even for dishonest majority. We demonstrate this by applying our approach to design an efficient multi-party fixed point multiplication protocol where the inputs and output are represented using an n-out-of-n additive secret sharing scheme. Our four-round protocol is secure against a semi-honest adversary that can corrupt any $t < n$ parties. We refer to Sect. 8 for more details.

It is our expectation that future work will apply our core technique to various other secret sharing schemes.

Performance. The communication and computation cost of all three protocols is close to a standard secret shared modular multiplication. For replicated secret sharing the protocol is exactly as the modular counter-part. The same holds for Shamir for a dishonest majority. Curiously, our Shamir protocol with an honest majority is mildly less efficient, requiring $t + 1$ parties to secret share a random value. This is contrasted by prior works which required preprocessing k secret shared bits [16] or performing a binary MPC protocol consisting of kt AND gates [36].

General MPC. For each form of secret sharing, we leverage our fixed-point multiplication protocol to design an MPC protocol for computing arbitrary arithmetic circuits that contain addition and fixed-point multiplication with truncation gates. We then prove the resulting MPC protocol secure via a standard real world-ideal world simulation based security definition.

1.2 Related Work

Work in this area of secret sharing decimal values has largely taken two approaches. The first has been to implement some type of floating point number system. One could compile a circuit which implements the IEEE 754 floating point standard using either a binary or arithmetic circuit as done by [40]. However, this leads to relatively poor performance compared to a standard secret sharing due to difficulties in expressing it as an MPC circuit. Others have abandoned the IEEE standard and implemented more MPC friendly floating point number systems [8, 13–15, 24, 33]. While significant progress has been made, all

of these protocols require many rounds of interaction and are inefficient when compared to their standard secret sharing counter-parts.

A second approach has been to forgo floating-point and instead perform all operations on fixed-point values. This has the advantage of enabling significantly more efficient protocols at the expense of a loss in accuracy. For fixed-point, the core idea is that standard secret sharing can be used with the added required that after each multiplication, the result is scaled down by the size of the decimal. This scaling down is referred to as *truncation*. One of the first methods for achieving this involves decomposing a secret share of x into secret shares of the bits of x. This is known as bit decomposition [7, 42]. Given the bits it is a relatively easy task to truncate the secret share by any power of 2. In 2010 Catrina and Saxena [16] presented an efficient method for truncating an secret sharing with a prime modulus. Their protocol requires preprocessing k random secret shared bits, where k is the bit length of the shares. The online phase of their protocol is quite efficient, consisting of local operations and revealing a value. More recently, Mohassel and Zhang [37] presented an even more efficient protocol for a two-out-of-two linear secret scheme with modulus 2^k. This protocol requires no preprocessing. There have also been several follow up works to these, e.g. [12, 22, 26, 36]. Some of these protocols are described in detailed later.

Organization. In Sect. 2, we discuss some preliminaries. In Sect. 3, we discuss the techniques used in our protocols in detail, and this is followed by a performance comparison of our protocols with prior work in Sect. 4. In Sect. 5 and Sect. 6, we describe our protocols for replicated secret sharing. Our protocol for Shamir secret sharing is in Sect. 7 and finally, Sect. 8 discusses our protocol for additive secret sharing.

2 Preliminaries

2.1 Notation

We use κ to refer to the computational security parameter and λ for the statistical security parameter. The parties are enumerated as $P_1, ..., P_n$. We use the notation $[\![x]\!]$ to denote a secret sharing of x. This sharing can be one of several different types - $[\![x]\!]^{\mathsf{R}}$, $[\![x]\!]^{\mathsf{S}}$ or $[\![x]\!]^{\mathsf{A}}$ as described later. Regardless, $[\![x]\!]_i$ will refer to the share held by P_i.

We assign a variable a the value of b by $a := b$. When uniformly sampling from a set S or sampling a randomized function f, we use the notation $a \leftarrow S$ and $a \leftarrow f(...)$ respectively. Let $[a, b]$ denote the set $\{a, a+1, ..., b\}$ and the shorthand $[b] := [1, b]$. A parentheses on the left or right side denotes that corresponding side of the range is exclusive, e.g. $(a, b] := \{a+1, a+2, ..., b\}$. We refer the reader to [27] for the definition of pseudorandom functions (PRFs), which is one of the cryptographic primitives we use in our constructions.

2.2 Representations

In secure computation, functions are generally represented as a circuit where each wire can hold an integer value modulo some public modulus $N \in \mathbb{N}$ such as $N = 2$ for Boolean and $N = 2^k$ or a prime q for arithmetic operations. In this work, we consider arithmetic circuits with $N = 2^k$ or a prime q. In the case of a prime modulus q, we define $k := \lfloor \log_2(q) \rfloor$ such that 2^k is just smaller than q. In addition to being able to perform these types of modular arithmetics, this framework supports both signed and fixed-point arithmetics such that fractional numbers can efficiently be represented.

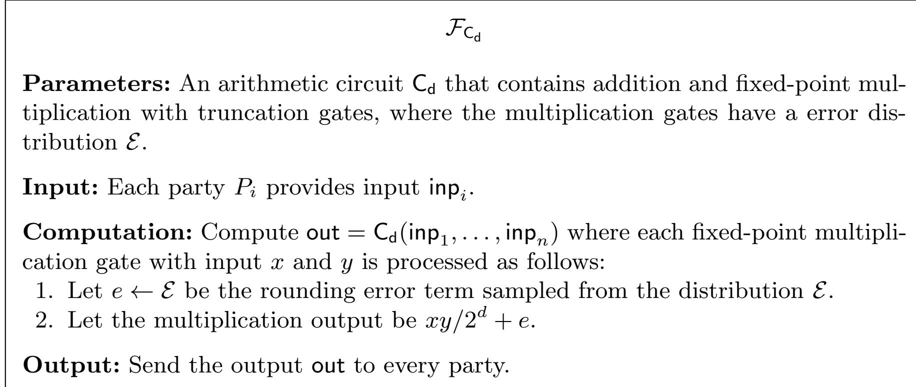
Signed Values. A value $x \in [-N/2, N/2)$ can be represented as $\hat{x} := (x \mod N)$. For $N = 2^k$, this is also known as two's complement, and it is straightforward to see the highest order bit indicates sign. The same approach also works when N is a prime. Sometimes it will be more convenient to consider a signed value as simply an element of $\mathbb{Z}_N$ while at other times $x \in [-N/2, N/2)$ is more natural. In particular, we will use the notation $\mathbb{S}_N := [-N/2, N/2)$ to denote this range. Regardless, performing addition, subtraction and multiplication in $\mathbb{S}_N$ or $\mathbb{Z}_N$ has a one-to-one correspondence. We define the symmetric mod operator $x' := (x \mathsf{\ symod\ } N)$ as the unique value $x' \in \mathbb{S}_N$ such that $(x \mod N) = (x' \mod N)$.

Fixed-Point Values. Fixed-point values have support for representing rational numbers $\mathbb{Q}$ such as $x = 1.25$. This can be achieved by having the bottom d bits of an integer represent the fractional part. That is, let $x_i \in \mathbb{Z}_2$ be the i'th bit of an unsigned x, then $x = \sum 2^{i-d} x_i$. We parameterize a set of fixed point values by two integers N, d. The set of (signed) values can then be described as $\mathsf{Fx}_{N,d} := \{x/2^d \mid x \in \mathbb{S}_N\}$ where the division is performed over $\mathbb{Q}$ and it is assumed that $N > 2^d$. We define addition and subtraction in the natural way. For multiplication of $x, y \in \mathsf{Fx}_{N,d}$, we define the result as $xy \in \mathbb{Q}$ rounded down to the next multiple of 2^{-d}.

2.3 Security Model

Consider $\mathbb{S}_N$ and a value k' where $2^{k'} \ll N$. Consider any arithmetic circuit $\mathsf{C_d}$ with addition and fixed-point multiplication with truncation gates. The multiplication gates are associated with a truncation parameter d and an error distribution $\mathcal{E}$ - given two inputs x and y, the output of the gate is $(xy/2^d + e)$ where e is a rounding error term sampled from the distribution $\mathcal{E}$. Each party P_i has an input $\mathsf{inp}_i \in \mathbb{S}_{2^{k'}}$ and they wish to evaluate the circuit $\mathsf{C_d}$ on these n joint inputs without revealing any information about their respective inputs. The input values satisfy the property that when evaluating circuit $\mathsf{C_d}$, the inputs and output of any multiplication gate lies in $\mathbb{S}_{2^{k'}}$.

We follow the standard real-ideal world simulation based security definition for secure multiparty computation (MPC) [27]. The ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ is defined in Fig. 2. We consider an adversary that corrupts at most t of the parties and study both semi-honest and malicious adversaries.



$\mathcal{F}_{\mathsf{C_d}}$

Parameters: An arithmetic circuit $\mathsf{C_d}$ that contains addition and fixed-point multiplication with truncation gates, where the multiplication gates have a error distribution $\mathcal{E}$.

Input: Each party P_i provides input inp_i.

Computation: Compute $\mathsf{out} = \mathsf{C_d}(\mathsf{inp}_1, \ldots, \mathsf{inp}_n)$ where each fixed-point multiplication gate with input x and y is processed as follows:

1. Let $e \leftarrow \mathcal{E}$ be the rounding error term sampled from the distribution $\mathcal{E}$.
2. Let the multiplication output be $xy/2^d + e$.

Output: Send the output out to every party.

Fig. 2: Ideal functionality for computing an arithmetic circuit $\mathsf{C_d}$ over ring $\mathbb{S}_N$

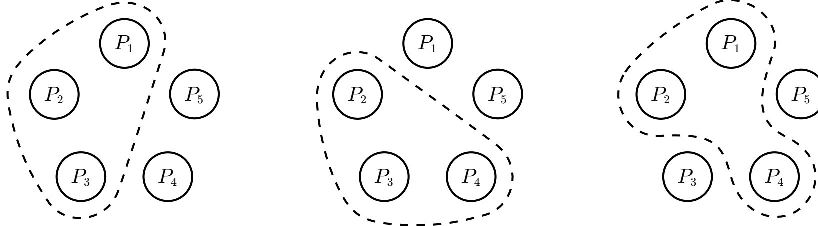
2.4 Secret Sharing Types

We consider different types of secret sharing schemes depending on the number of parties: replicated secret sharing $[\![x]\!]^{\mathsf{R}}$ for a small number of parties, Shamir and additive secret sharing $[\![x]\!]^{\mathsf{S}}$ for a larger number of parties.

Replicated Secret Sharing A value $x \in \mathbb{G}$ is replicated secret shared by uniformly sampling $x_1, ..., x_m \in \mathbb{G}$ for $m := \binom{n}{n-t}$ such that $x = \sum_i x_i$. $\mathbb{G}$ can be any group such as $\mathbb{S}_{2^k}, \mathbb{Z}_{2^k}$ or $\mathbb{F}_q$ for a prime q. To obtain a t-out-of-n replicated secret sharing of x, it is possible to distribute these shares such that every subset of $(t+1)$ parties holds all m shares of x, yet any smaller coalition is missing at least one share. One method for doing this is as follows: Define the sets $D_1, ..., D_m \subset [n]$ as the m distinct subsets of size $(n-t)$. Without loss of generality, for $i \in [n]$, let $D_i = \{i, i+1, i+2, ...\}$ be the contiguous set of party indices starting at index i, with wrap around (see Fig. 3). The replicated secret share of the ith party P_i is then $[\![x]\!]^{\mathsf{R}}_i := \{x_j \mid i \in D_j\}$. We will use the group $\mathbb{S}_{2^k}$.

Due to the linearity of the scheme, addition, subtraction or multiplication with a public constant can be performed locally by applying the corresponding operation to the shares. For example, computing $[\![z]\!]^{\mathsf{R}} := [\![x]\!]^{\mathsf{R}} + [\![y]\!]^{\mathsf{R}}$ can be done by having each party P_i define the output shares $[\![z]\!]^{\mathsf{R}}_j := [\![x]\!]^{\mathsf{R}}_j + [\![y]\!]^{\mathsf{R}}_j$. Subtraction works in the same way while public multiplication $[\![z]\!]^{\mathsf{R}} := c[\![x]\!]^{\mathsf{R}}$ can be perform by defining $[\![z]\!]^{\mathsf{R}}_j := c[\![x]\!]^{\mathsf{R}}_j$. To multiply two secret shared values $[\![z]\!]^{\mathsf{R}} := [\![x]\!]^{\mathsf{R}}[\![y]\!]^{\mathsf{R}}$, parties must perform an interactive protocol. We elaborate more on this later in the paper.

Shamir Secret Sharing [43]. To Shamir secret share a value $x \in \mathbb{F}$, the party who holds x samples a uniformly random polynomial $p_x(\cdot) \in \mathbb{F}[\cdot]$ of degree t such that $p_x(0) = x$. Each party P_i is given the point $[\![x]\!]^{\mathsf{S}}_i := p_x(i)$, which is the evaluation of the polynomial p_x at i. A degree t polynomial is uniquely defined by $(t+1)$ points, so it follows that any subset of $(t+1)$ parties can use their shares to reconstruct p_x and retrieve the secret value $x = p_x(0)$, while any subset



(a) Parties indexed by D_1. (b) Parties indexed by D_2. (c) Parties indexed by D_6.

Fig. 3: When $n = 5,\ t = 2$, sets $D_1, \ldots, D_5$ refer to the first 5 contiguous subsets (e.g Fig. 3a and Fig. 3b), and $D_6, \ldots, D_{10}$ are the remaining non-contiguous, distinct subsets (e.g Fig. 3c). In replicated secret sharing, a share x_i is held by all of the parties indexed by D_i.

of fewer than $(t + 1)$ parties learn no information about the secret. We use the notation $[\![x]\!]^{\mathsf{S},t'}$ to denote the Shamir secret sharing of x using a polynomial of degree t'. Typically, we will work with polynomials of degree t and omit the degree in this case for brevity (e.g $[\![x]\!]^{\mathsf{S}}$) and consider $\mathbb{F} = \mathbb{F}_q$ for a prime q.

As in the case of replicated secret sharing, observe that addition and multiplication by a constant can be performed locally on each share. That is, to compute $[\![z]\!]^{\mathsf{S}} := [\![x]\!]^{\mathsf{S}} + c$ or $[\![z]\!]^{\mathsf{S}} := c[\![x]\!]^{\mathsf{S}}$, each party P_i locally compute $[\![z]\!]^{\mathsf{S}}_i := [\![x]\!]^{\mathsf{S}}_i + c$ or $[\![z]\!]^{\mathsf{S}}_i := c[\![x]\!]^{\mathsf{S}}_i$, respectively. Similarly, to calculate the sum of two shared values $[\![z]\!]^{\mathsf{S}} := [\![x]\!]^{\mathsf{S}} + [\![y]\!]^{\mathsf{S}}$, each party P_i can locally compute their shares as $[\![z]\!]^{\mathsf{S}}_i := [\![x]\!]^{\mathsf{S}}_i + [\![y]\!]^{\mathsf{S}}_i$. More care is required when multiplying two shared values and we elaborate on this later.

Additive Secret Sharing A value $x \in \mathbb{G}$ is additive secret shared by uniformly sampling $x_1, ..., x_n \in \mathbb{G}$ such that $x = \sum_i x_i$. The additive secret share of the ith party P_i is then $[\![x]\!]^{\mathsf{A}}_i := x_i$. Observe that unless all the n parties come together, no information about the secret x is learnt. We will use the group $\mathbb{S}_{2^k}$. Similar to the other two sharing techniques, addition, subtraction or multiplication with a public constant can be performed locally by applying the corresponding operation to the shares.

3 Techniques

Generally speaking, this work builds on two main techniques for performing fixed-point computation. The first is an improvement for replicated secret sharing over the ring $\mathbb{Z}_{2^k}$ as performed by [36, 37]. The second approach is tailored for secret sharing over a prime field $\mathbb{F}_q$ such as Shamir secret sharing.

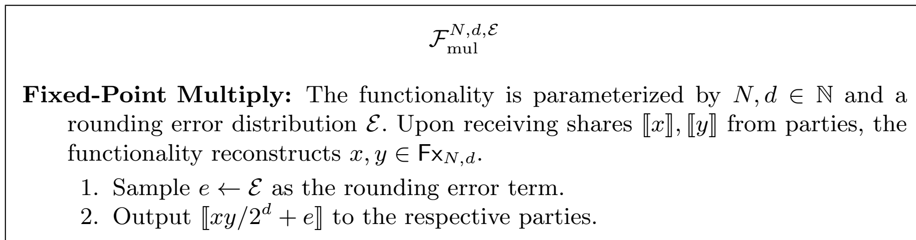
Regardless of whether the scheme naturally supports $\mathbb{Z}_{2^k}$ or $\mathbb{F}_q$, the basic operations follow a standard approach. For party P_i to input a value $x \in \mathsf{Fx}_{N,d}$,

P_i maps x into the target group and secret shares it. Specifically, for $\mathbb{Z}_{2^k}$ and $N = 2^k$, party P_i compute $x' := x2^d \mod 2^k$ and then runs the standard input sharing protocol. In the case of secret sharing over $\mathbb{F}_q$, P_i computes $x' := x2^d \mod q$ and inputs x'. Addition, subtraction of shared values and multiplication by a public integer follow the same approach as the standard secret sharing. To reveal a shared fixed-point value $[\![x']\!]$ to some of the parties, first the standard reveal procedure is performed to reveal x'. The final result is then computed as $x := x'/2^d \in \mathsf{Fx}_{N,d}$.

The core difficulty in multiplying this style of fixed-point secret shares is that the standard protocol would result in a semantic value in $\mathsf{Fx}_{N,2d}$ given input values in $\mathsf{Fx}_{N,d}$ and ignores the possibility of the product overflowing N. That is, let $[\![x']\!], [\![y']\!]$ be sharing of $x, y \in \mathsf{Fx}_{N,d}$ as described above. Then computing $[\![z']\!] := [\![x']\!][\![y']\!]$ using the standard $\mathbb{Z}_{2^k}$ or $\mathbb{F}_q$ protocols would result in z' such that $z'/2^{2d} \approx xy$ where the approximation comes from the rounding of the lower order terms. An approach known as *truncation* can then be used to take a shared value in $\mathsf{Fx}_{N,2d}$ and produce shares of that value rounded into $\mathsf{Fx}_{N,d}$.

Ideally, this rounding would exactly correspond to multiplication in $\mathsf{Fx}_{N,d}$ (rounding down to the nearest 2^{-d}) but several works [16, 36, 37] have shown that a significant performance improvement can be achieved by allowing probabilistic rounding. We capture this in our ideal functionality $\mathcal{F}_{\text{mul}}^{N,d,\mathcal{E}}$ which is parameterized by a modulus N, the number of decimal bits d and a rounding error distribution $\mathcal{E}$. This distribution will specify how the rounding should be perform.

We adapt and improve these prior approaches beyond the two party setting with a focus on reducing the communication overhead.


$\mathcal{F}_{\text{mul}}^{N,d,\mathcal{E}}$

Fixed-Point Multiply: The functionality is parameterized by $N, d \in \mathbb{N}$ and a rounding error distribution $\mathcal{E}$. Upon receiving shares $[\![x]\!], [\![y]\!]$ from parties, the functionality reconstructs $x, y \in \mathsf{Fx}_{N,d}$.

1. Sample $e \leftarrow \mathcal{E}$ as the rounding error term.
2. Output $[\![xy/2^d + e]\!]$ to the respective parties.

Fig. 4: Probabilistic functionality for fixed-point multiplication

3.1 Replicated Secret Sharing

The approach of [36, 37]. At its core, this approach is tailored for a 2-out-of-2 secret sharing scheme over $\mathbb{S}_{2^k}$. A value $z \in \mathbb{S}_{2^{k'}}$ is secret shared into two random values $z_1, z_2 \in \mathbb{S}_{2^k}$ such that $z_1 + z_2 = z$ and for some $k \geq k'$. Mohassel and Zhang [37] showed that so long as $|z| \ll 2^k$, then with high probability

$$\frac{z_1}{2^d} + \frac{z_2}{2^d} \in \left\{\frac{z}{2^d}, \frac{z}{2^d} - 1\right\}$$

where the division is the quotient over the integers. Here, the possibility of having a minus one comes from the fact that if $(z_1 \mod 2^d) + (z_2 \mod 2^d) \geq 2^d$, then this would have generated a "carry bit" which was eliminated by performing separate division operations on the shares.

There are two subtleties in understanding this approach. First it is crucial to interpret the individual shares z_1, z_2 as being signed integers in the range $\mathbb{S}_{2^k} = [-2^{k-1}, 2^{k-1})$. Second is the necessity for $|z| \ll 2^{k-1}$. In the event that $z_1 + z_2$ *wraps around*[4] in $\mathbb{S}_{2^k}$, then it must hold that z_1, z_2 are both positive or negative and that z has the opposite sign. However, after dividing z_1, z_2 by 2^d, the addition of these values can no longer wrap around $\mathbb{S}_{2^k}$ and as such does not sum to $z/2^d$. This event where $z_1 + z_2$ wraps around is referred to as a *catastrophic error* [37].

To fix these catastrophic errors, [37] observed the following. Let us redefine the shares by $r \xleftarrow{\$} \mathbb{S}_{2^k}$ and $z_1 := -r, z_2 := r + z$ and require $z \in \mathbb{S}_{2^{k'}}$ where $2^{k'} \ll 2^k$. Then, with probability at least $1 - 2^{k'-k}$ it holds that z_1 and z_2 do not have the same sign since this would imply that $r + z$ wraps around $\mathbb{S}_{2^k}$ and the top $k - k'$ bits of r are set to 1. Given that z_1, z_2 do not have the same sign it is impossible for them to wrap around $\mathbb{S}_{2^k}$ and therefore $\frac{z_1}{2^d} + \frac{z_2}{2^d}$ will be equal to $\frac{z}{2^d}$ or $\frac{z}{2^d} - 1$ as described above.

Unfortunately, this approach does not generalized to more than two parties. The core reason is that it is likely for the shares to wrap around the modulus, regardless of the distribution of x. Mohassel and Rindal [36] were able to circumvent this issue by emulating the two party protocol in the multi-party setting. This required the addition of a preprocessing phase where a *truncation pair* $[\![r]\!]^{\mathsf{R}}, [\![r/2^d]\!]^{\mathsf{R}}$ is computed. Given this, the protocol above can be emulated by revealing $[\![z_1]\!]^{\mathsf{R}} := [\![z]\!]^{\mathsf{R}} - [\![r]\!]^{\mathsf{R}}$ and then letting $[\![z_2/2^d]\!]^{\mathsf{R}} := [\![r/2^d]\!]^{\mathsf{R}}$. Since z_1 is made public, all parties can compute $[\![z/2^d]\!] := [\![z_2/2^d]\!]^{\mathsf{R}} + z_1/2^d$.

The main limitation of [36] is the need to generate the truncation pair $[\![r]\!]^{\mathsf{R}}, [\![r/2^d]\!]^{\mathsf{R}}$. [36] present a three-party preprocessing protocol which requires $2k$ binary AND gates in the semi-honest setting and $4k$ in the malicious setting. Generalizing this approach results in $2tk$ and $2mk$ AND gates respectively in the multi-party scenario. Since $m = \binom{n}{t} = O(2^n)$, the overhead of this approach does not scale well as n grows.

Our approach. The core contribution of our new approach is a new technique for generating a multi-party sharing of x which is guaranteed not to wrap around the modulus. As a starting point, consider secret sharing $x \in \mathbb{S}_{2^{k'}}$ over $\mathbb{S}_{2^k}$ (where $k \geq k' + \lambda + \log_2 m$) by sampling each share $x_2, \ldots, x_m \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ and then defining the last share as $x_1 = x - x_2 - \cdots - x_m$. Observe that random shares are sampled over a 2^λ times larger range as compared to the underlying value, where λ is the statistical security parameter. This is required in order to sufficiently hide the value of x.

[4] i.e. if when interpreting z_1, z_2 as integers and adding them results in $(z_1 + z_2) > 2^{k-1}$ or $(z_1 + z_2) < -2^{k-1} + 1$.

By definition, this type of sharing cannot wrap around the modulus and as such, locally dividing each share approximately truncates the underlying secret shared value. In particular, we have

$$\begin{aligned} x =& x_1 + \cdots + x_m \\ =& 2^d \left(\frac{x_1}{2^d} + \cdots + \frac{x_m}{2^d} \right) + (x_1 \textsf{ symod } 2^d + \cdots + x_m \textsf{ symod } 2^d) \\ x/2^d - e =& \frac{x_1}{2^d} + \cdots + \frac{x_m}{2^d} \end{aligned}$$

where $e := (x_1 \textsf{ symod } 2^d + \cdots + x_m \textsf{ symod } 2^d)/2^d \in [-m, m]$. That is, locally dividing the shares results in the secret shared value being similarly truncated with an absolute error $|e|$ of at most $\log m$ bits.

This approach does not immediately give a practical secret sharing scheme due to the size of the shares doubling after each multiplication. We overcome this by effectively converting a traditional secret sharing (e.g. replicated) into the sharing described above, performing the division and then converting back.

New replicated secret sharing. First let us consider the task of dividing a replicated secret sharing of $[\![x]\!]^{\mathsf{R}}$ by 2^d to obtain a sharing $[\![z]\!]^{\mathsf{R}}$ where $z \approx x/2^d$ over the integers. Later we will discuss how to combine this with the multiplication protocol with little added overhead.

Recall that each replicated secret sharing consists of m shares, distributed among n parties. In particular, we have $x = (x_1 + \cdots + x_m) \textsf{ symod } 2^k$. As with prior art, our technique will require that $x \in \mathbb{S}_{2^{k'}}$ for some $2^{k'} \ll 2^k$, e.g. $k = k' + \lambda + \log_2 m$.

The parties jointly sample a random sharing $[\![r]\!]^{\mathsf{R}}$ where share r_i is defined as $r_i \leftarrow \mathbb{S}_{2^{k'+\lambda}}$. This type of secret sharing has three critical properties: the first is that $[\![r]\!]^{\mathsf{R}}$ does not wrap around $\mathbb{S}_{2^k}$ when reconstructed, i.e. $r = (r_1 + \cdots + r_m) = (r_1 + \cdots + r_m \textsf{ symod } 2^k)$. The second is that the distributions of r and $w := r + x$ are statistically close. This latter property follows from r being distributed over a range that is approximately $2^{\lambda + \log_2 m}$ times bigger than x. Thirdly, $[\![r]\!]^{\mathsf{R}}$ can be sampled non-interactively by sampling the shares as the output of a pseudorandom function (PRF).

The protocol proceeds by revealing $[\![w]\!]^{\mathsf{R}} := [\![r]\!]^{\mathsf{R}} + [\![x]\!]^{\mathsf{R}}$ to all parties. Let us assume that $r + x$ does not wrap around and observe that

$$\begin{aligned} w &= x + r \\ w/2^d &= \frac{x + r}{2^d} \\ w/2^d &= \left(\frac{x}{2^d} + \frac{r_1}{2^d} + \cdots + \frac{r_m}{2^d} \right) + e, \\ &\quad \text{where } e := (x \textsf{ symod } 2^d + r_1 \textsf{ symod } 2^d + \cdots + r_m \textsf{ symod } 2^d)/2^d \\ x/2^d + e &= \frac{w}{2^d} - \frac{r_1}{2^d} - \cdots - \frac{r_m}{2^d} \end{aligned}$$

As before we have that $e \in [-m, m]$. More importantly, the parties can locally compute shares of $[\![z]\!]^{\mathsf{R}} = [\![x/2^d + e]\!]^{\mathsf{R}}$ given w and $[\![r]\!]^{\mathsf{R}}$ by defining $z_1 := w/2^d - r_1/2^d$ and $z_i := -r_i/2^d$ for $i \in [2, m]$.

Fused multiply and truncate. Now we consider the challenge of performing a fused multiply and truncate protocol. The trivial way of doing this is to first perform the standard multiplication protocol to compute $[\![z']\!]^{\mathsf{R}} := [\![x]\!]^{\mathsf{R}}[\![y]\!]^{\mathsf{R}}$ and then the truncation protocol to compute $[\![z]\!]^{\mathsf{R}} :\approx [\![z']\!]^{\mathsf{R}}/2^d$. The main downside of this approach is the need to perform two rounds of interaction.

First let us recall the normal multiplication protocol. Observe that $xy = (x_1 + \cdots + x_m)(y_1 + \cdots + y_m) = \sum_{j,j'} x_j y_{j'}$ and that the shares are distributed such that at least one party can compute each cross term $x_j y_{j'}$. The shares x_j, y_j will be held by party P_i for all $i \in D_j$ (recall that D_j consists of a set of $(n-t)$ parties and by definition, $P_i \in D_i$). Let us assign each cross term $x_j y_{j'}$ to a single P_i and let u_i be the sum of them. For $j \in (n, m]$, the parties in D_j non-interactively sample $z'_j \leftarrow \mathbb{S}_{2^k}$. In addition, the parties non-interactively sample an n-out-of-n zero-sharing $[\![s]\!]$ where P_i holds s_i and $\sum_i s_i = 0$. Given these, P_i defines $z'_i := u_i + s_i - \sum_{j \in N_i} z'_j$ and distributes z'_i to all parties in D_i. Here $N_i \subset (n, m]$ index the set of z'_j terms that are assigned to P_i. A detailed description of this protocol is given in Fig. 10.

Observe there are two sets of randomized shares, z'_j for $j \in (n, m]$ and r_i for $i \in [n]$, which play slightly different roles. Given that the adversary corrupts t parties, there will be exactly one index $j^* \in [m]$ for which they do not hold the share, e.g. y_{j^*}. As such, its critical that z'_{j^*} is sampled uniformly as is done above. The zero sharing $[\![s]\!]$ serves a slightly different purpose. For exposition, let's assume x is somehow known to the adversary while y, and therefore y_{j^*}, is not. The honest parties P_i will each use y_{j^*} to compute their share z'_i. As such, if s_i was not included, for some of these z'_i values, the adversary will know all the other terms and can solve for y_{j^*}. However, by including the zero sharing, we effectively "distribute" the uncertainty the adversary has about z'_{j^*} into all of the z'_i messages that are sent.

Now, we are ready to combine the multiplication and division protocols into a single fused operation. The core change is that instead of distributing shares of z', the parties will directly reveal $w = z' + r$ and then perform the division step as described before. Beyond this conceptual change, we observe that sampling the z'_j shares for $j \in (n, m]$ are no longer necessary due to z' being sufficiently masked by r.

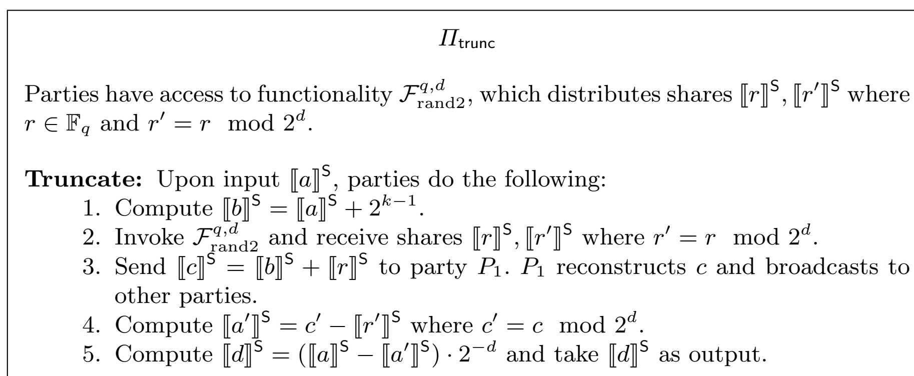
In more detail, each party P_i computes the sum u_i of their cross terms $x_j, y_{j'}$ along with their zero sharing s_i. The sharing $[\![r]\!]^{\mathsf{R}}$ is sampled as $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ which ensures that r will not wrap around. Party P_i computes $w_i := u_i + s_i + \sum_{j \in N_i} r_j$ and reveal it to all parties. The final output shares are defined as $z_1 := w_1/2^d - r_1/2^d$ and $z_i := -r_i/2^d$ for $i \in [2, m]$. A detailed description of our semi-honest secure protocol is given in Fig. 8.

Malicious security. To achieve malicious security, we add an information theoretic MAC $\alpha z'$ along with the term z'. The MAC key α is unknown to the adversary and is secret shared amongst all the parties. Using $[\![\alpha]\!]^{\mathsf{R}}$, parties can computes secret shares of $\alpha z'$. Then, we run a MAC check protocol (from in Figure 10 of [23]) that checks that shares of the MAC term $\alpha z'$ are consistent with $[\![\alpha]\!]^{\mathsf{R}}$ and z'. From this, we have the guarantee that the term z' was correctly

computed by the adversary. Finally, after computing $[\![z]\!]^{\mathsf{R}}$ as in the semi-honest protocol, we now also compute the MAC $[\![\alpha z]\!]^{\mathsf{R}}$. We refer to Fig. 12 for more details.

3.2 Shamir Secret Sharing

The [12, 16] approach. For the sake of comparison we first give a detailed description of the protocol of Catrina and Saxena [16] for computing $[\![z]\!]^{\mathsf{S}} :\approx [\![x]\!]^{\mathsf{S}}/2^d$. The full protocol is shown in Fig. 5. The core of their approach is to observe that integer division by 2^d and modular division in $\mathbb{F}_q$ are the same when the numerator is divisible by 2^d. The intuition is then, given an arbitrary numerator $a \in \mathbb{F}_q$, we compute $a/2^d$ in the integers by computing $(a-(a \mod 2^d))\cdot 2^{-d}$ in $\mathbb{F}_q$. The challenge is to support signed values and to give an efficient way to compute a sharing of $[\![a \mod 2^d]\!]^{\mathsf{S}}$ given $[\![a]\!]^{\mathsf{S}}$.



Π_{trunc}

Parties have access to functionality $\mathcal{F}^{q,d}_{\text{rand2}}$, which distributes shares $[\![r]\!]^{\mathsf{S}}, [\![r']\!]^{\mathsf{S}}$ where $r \in \mathbb{F}_q$ and $r' = r \mod 2^d$.

Truncate: Upon input $[\![a]\!]^{\mathsf{S}}$, parties do the following:

1. Compute $[\![b]\!]^{\mathsf{S}} = [\![a]\!]^{\mathsf{S}} + 2^{k-1}$.
2. Invoke $\mathcal{F}^{q,d}_{\text{rand2}}$ and receive shares $[\![r]\!]^{\mathsf{S}}, [\![r']\!]^{\mathsf{S}}$ where $r' = r \mod 2^d$.
3. Send $[\![c]\!]^{\mathsf{S}} = [\![b]\!]^{\mathsf{S}} + [\![r]\!]^{\mathsf{S}}$ to party P_1. P_1 reconstructs c and broadcasts to other parties.
4. Compute $[\![a']\!]^{\mathsf{S}} = c' - [\![r']\!]^{\mathsf{S}}$ where $c' = c \mod 2^d$.
5. Compute $[\![d]\!]^{\mathsf{S}} = ([\![a]\!]^{\mathsf{S}} - [\![a']\!]^{\mathsf{S}}) \cdot 2^{-d}$ and take $[\![d]\!]^{\mathsf{S}}$ as output.

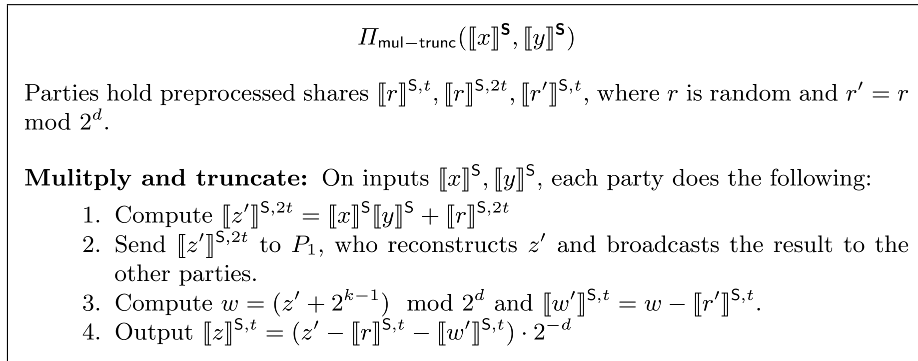
Fig. 5: Protocol for truncation [16, Protocol 3.1].

To support signed values, they require that a is in $\mathbb{S}_{2^k} = [-2^{k-1}, 2^{k-1})$ where $2^k < q$. They then shift a into the range $[0, 2^k)$ by computing $b := a + 2^{k-1} \in \mathbb{F}_q$. Now observe that $a' := b \mod 2^d$ is precisely the "semantic" low order bits of a. Effectively, shifting a allows us to handle the positive and negative cases simultaneously.

Given $[\![b]\!]^{\mathsf{S}} = [\![a + 2^{k-1}]\!]^{\mathsf{S}}$ the parties approximate $[\![b \mod 2^d]\!]^{\mathsf{S}}$ using a mask and open technique. First, the pair $[\![r]\!]^{\mathsf{S}}, [\![r']\!]^{\mathsf{S}}$ are preprocessed where $r \xleftarrow{\$} \mathbb{F}_q$ and $r' := (r \mod 2^d)$. The masked value $c := b + r$ is then revealed to all parties. Importantly, we require that $b + r$ does not wrap around $\mathbb{F}_q$ which we will discuss later. The parties define $c' := c \mod 2^d = b + r \mod 2^d$. Observe that $[\![a']\!]^{\mathsf{S}} := c' - [\![r']\!]^{\mathsf{S}} = (b + r \mod 2^d) - [\![r']\!]^{\mathsf{S}}$ is equal to $(b \mod 2^d)$ or $(b - 1 \mod 2^d)$. This -1 term is due to the possibility of $b + r$ generating a carry bit at bit position d which is eliminated by the $\mod 2^d$ operation. As such, this truncation will result in the same rounding error as in [36, 37]. The final result

can then be computed as $\llbracket d \rrbracket^{\mathsf{S}} := (\llbracket a \rrbracket^{\mathsf{S}} - \llbracket a' \rrbracket^{\mathsf{S}}) \cdot 2^{-d}$ which equals $a/2^d$ or $a/2^d - 1$ over the integers.

In [16], the multiplication and division are given separately. A followup work [12] gave an optimization which combined the multiplication and division into a single round as shown in Fig. 6. At a high level, the protocol begins with the standard strategy of computing the higher degree product and opening a masked version of it. Then, following the truncation protocol of [16], the masked value z' is shifted to the range $[0, 2^k)$ and shares of the unmasked bottom d-bits are computed. These shares are then used to compute a sharing of the product that can have the bottom d-bits truncated.



$\Pi_{\mathsf{mul-trunc}}(\llbracket x \rrbracket^{\mathsf{S}}, \llbracket y \rrbracket^{\mathsf{S}})$

Parties hold preprocessed shares $\llbracket r \rrbracket^{\mathsf{S},t}, \llbracket r \rrbracket^{\mathsf{S},2t}, \llbracket r' \rrbracket^{\mathsf{S},t}$, where r is random and $r' = r \mod 2^d$.

Mulitply and truncate: On inputs $\llbracket x \rrbracket^{\mathsf{S}}, \llbracket y \rrbracket^{\mathsf{S}}$, each party does the following:

1. Compute $\llbracket z' \rrbracket^{\mathsf{S},2t} = \llbracket x \rrbracket^{\mathsf{S}} \llbracket y \rrbracket^{\mathsf{S}} + \llbracket r \rrbracket^{\mathsf{S},2t}$
2. Send $\llbracket z' \rrbracket^{\mathsf{S},2t}$ to P_1, who reconstructs z' and broadcasts the result to the other parties.
3. Compute $w = (z' + 2^{k-1}) \mod 2^d$ and $\llbracket w' \rrbracket^{\mathsf{S},t} = w - \llbracket r' \rrbracket^{\mathsf{S},t}$.
4. Output $\llbracket z \rrbracket^{\mathsf{S},t} = (z' - \llbracket r \rrbracket^{\mathsf{S},t} - \llbracket w' \rrbracket^{\mathsf{S},t}) \cdot 2^{-d}$

Fig. 6: Protocol for multiplication for Shamir secret sharing.

Our approach. Our protocol for Shamir secret sharing is conceptually similar to our replicated secret sharing technique. This generality speaks to the versatility of our approach in that it can be applied to many settings.

The first step in our protocol is for the parties to sample a linear secret sharing of r which does not wrap around the modulus. One method for achieving this is to have P_i for $i \in [t+1]$ sample $r_i \xleftarrow{\$} \mathbb{S}_{q/(t+1)}$ and generate a sharing of $\llbracket r_i \rrbracket^{\mathsf{S}}$ and $\llbracket r_i/2^d \rrbracket^{\mathsf{S}}$. Given these, the parties can define $\llbracket r \rrbracket^{\mathsf{S}} := \sum_i \llbracket r_i \rrbracket^{\mathsf{S}}$ and $\llbracket r' \rrbracket^{\mathsf{S}} := \sum_i \llbracket r_i/2^d \rrbracket^{\mathsf{S}}$. By the same analysis as above, we have that $r/2^d = r' + e$ for some e with at most $\log_2 t$ bits.

To then truncate an existing sharing $\llbracket x \rrbracket^{\mathsf{S}}$ by 2^d the parties reveal $\llbracket w \rrbracket^{\mathsf{S}} := \llbracket x \rrbracket^{\mathsf{S}} + \llbracket r \rrbracket^{\mathsf{S}}$ to all parties. The output shares are defined as $\llbracket z \rrbracket^{\mathsf{S}} := w/2^d - \llbracket r' \rrbracket^{\mathsf{S}}$. Since r is not uniformly distributed over $\mathbb{F}_q$, this protocol requires that $|x| \ll q$ in order to argue that x remains hidden. We note that this is a common property to all efficient division protocols. The detailed protocol is presented in Fig. 16.

3.3 General Truncation and Additive Secret Sharing

The aforementioned approaches for replicated and Shamir secret sharing require honest majority among the parties. This is inherently required by these secret

sharing schemes. Nevertheless, it is not a requirement of our new truncation technique. In fact, our new truncation protocol can be viewed as a general technique that works for any $t < n$.

To truncate a secret shared value $[\![x]\!] \in \mathbb{S}_N$ where $x \in \mathbb{S}_{2^{k'}}$ and $\log_2 N \geq k' + \lambda + \log_2 n$, we first let the parties sample a linear secret sharing of r which does not wrap round $\mathbb{S}_N$. In particular, each party P_i for $i \in [t+1]$ samples $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ and generates sharings of $[\![r_i]\!]$ and $[\![r_i/2^d]\!]$ in $\mathbb{S}_N$. Given these, the parties can define $[\![r]\!] := \sum_i [\![r_i]\!]$ and $[\![r']\!] := \sum_i [\![r_i/2^d]\!]$. By the same analysis as above, we have that $r/2^d = r' + e$ for some e with at most $\log_2 t$ bits.

To illustrate this idea, we show how to incorporate the truncation protocol into additive secret sharing with dishonest majority and present the fixed point multiplication protocol in Fig. 18.

4 Performance Comparison

We now compare the concrete performance of our fixed point multiplication schemes with that of existing work. In particular, we compare to [36] for multi-party (semi-honest) replicated secret sharing over $\mathbb{S}_{2^k}$ and with [16] for Shamir secret sharing over $\mathbb{F}_q$ which is widely implemented [9, 31].

Replicated. Mohassel and Zhang [37] study two party secret sharing over $\mathbb{S}_{2^k}$ and perform stand-alone multiplication and truncation operations in a single round of communication. Our approach (if seen in a two-party setting) is effectively identical to theirs.

Mohassel and Rindal [36] generalized to more than two parties by effectively emulating the two party protocol of [37] within another MPC protocol. In essence, the protocol of [36] inputs the shares of the parties into a binary MPC protocol where the underlying value is reconstructed, truncated and then a new arithmetic sharing is generated and output to the parties. Their approach can be optimized to have practical concrete performance. In particular, in the three party case, it involves a pre-processing phase with $2k$ binary gates and almost no overhead in the online phase. However, in the multi-party case, when more than 3 parties are involved, it requires $((n^2 - nt)tk)$ bits of offline communication and $(n^2 - nt)k)$ bits of online communication where $t < n/2$ is a bound on the number of corrupt parties.

Our approach eliminates the need to emulate the two party protocol within a binary protocol. As suggested in Fig. 7, our protocol sends approximately t or $(n-t)t/2$ times less data, depending on how z' is revealed. In particular, all the shares of z' can either be sent to all $(n-t)$ parties in D_1 resulting in a single round protocol or a single party can receive them and then send them to the remaining parties in D_1 at the cost of an extra round. Regardless, our protocol requires significantly less communication than [36], especially in the offline phase where our protocol is completely non-interactive.

Shamir. Our protocol designed for Shamir secret sharing is also a significant improvement over the work of Catrina and Saxena [16]. The primary differ-

Protocol	Operation	Online Rounds	Online Comm	Offline Comm
This, Fig. 8	$[\![x]\!]^{\mathsf{R}} \cdot [\![y]\!]^{\mathsf{R}}$	1 or 2	$(n^2 - nt)k$ or $2nk$	0
[36]	$[\![x]\!]^{\mathsf{R}} \cdot [\![y]\!]^{\mathsf{R}}$	1	$(n^2 - nt)k$	$(n^2 - nt)tk$
This, Fig. 16	$[\![x]\!]^{\mathsf{S}} \cdot [\![y]\!]^{\mathsf{S}}$	2	$2nk$	$(n - t - 1)(t + 1)k$
[16]	$[\![x]\!]^{\mathsf{S}} \cdot [\![y]\!]^{\mathsf{S}}$	2	$2nk$	$2k^2n.$

Fig. 7: Communication (bits) and round complexity of our protocol compared to [16, 36]. For Shamir, we do not include the cost of generating beaver triples, if applicable.

ence is that the protocol of [16] requires preprocessing secret shares of k bits, $[\![r_1]\!]^{\mathsf{S}}, ..., [\![r_k]\!]^{\mathsf{S}}$ which are then used to perform the truncation operation. The typical method [23] for generating a random bit requires sharing a random value and a single multiplication. Therefore, the overall cost of the fixed-point multiplication protocol of [16] is effectively k multiplications.

Our protocol on the other hand requires $(t + 1)$ parties to each generate a random sharing. For an dishonest majority, this can be generated with *no interaction* using pre-shared keys, i.e. our protocol would be as efficient as a standard multiplication. More generally, this offline phase requires a total of $(n - t - 1)(t + 1)$ elements to be sent, along with the overhead of the standard multiplication protocol. This is contrasted with [16] which requires generating k random shares and k multiplications. We compare the numbers in Fig. 7. In practice, we would typically expect $n, t \ll k$ and so, this represents a significant reduction in communication.

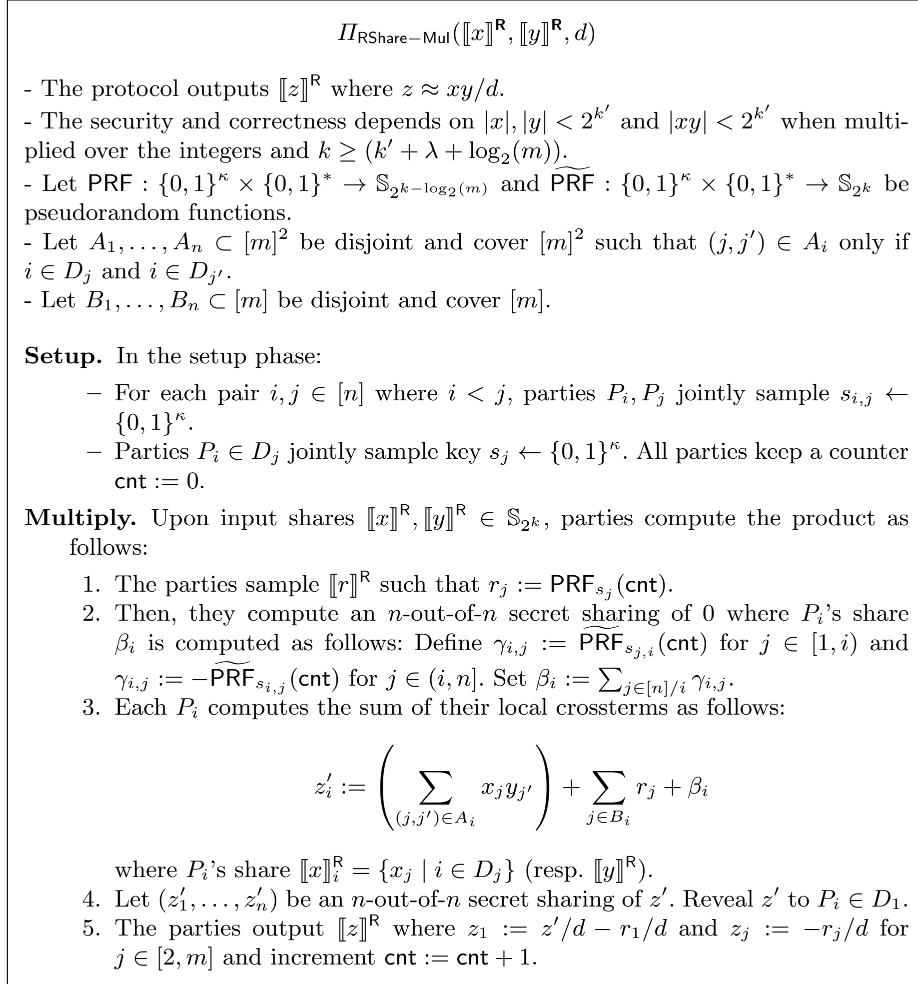
5 Replicated Secret Sharing: Semi-Honest

We first develop a new technique for fixed point multiplication with replicated secret sharing in the presence of a semi-honest adversary. We then show how to incorporate this into a general MPC protocol to compute any arithmetic circuit.

5.1 Fixed Point Multiplication

In this section, we present our new semi-honest protocol for fixed point multiplication with replicated secret sharing. Consider two fixed point values x, y represented in twos-complement form in $\mathbb{S}_{2^{k'}}$ where the bottom d bits denote the decimal. The parties hold replicated secret shares $[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$ in an extended ring $\mathbb{S}_{2^k}$ where $k \geq (k' + \lambda + \log_2(m))$. Their goal is to compute a replicated secret share $[\![z]\!]^{\mathsf{R}}$ where $z \approx \frac{xy}{2^d}$. The protocol works among n parties $P_1, \ldots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \ldots, D_m \subset [n]$ be the m distinct subsets of size $(n - t)$. We describe our protocol formally in Fig. 8.

Overview and proof sketch. In the setup phase, every pair of parties P_i and P_j jointly sample one PRF key $s_{i,j}$ and the set of parties in each set D_j sample a key s_j. Then, in the online phase, for each fixed point multiplication, all the



$\Pi_{\mathsf{RShare-Mul}}([\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}, d)$

- The protocol outputs $[\![z]\!]^{\mathsf{R}}$ where $z \approx xy/d$.
- The security and correctness depends on $|x|, |y| < 2^{k'}$ and $|xy| < 2^{k'}$ when multiplied over the integers and $k \geq (k' + \lambda + \log_2(m))$.
- Let $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{S}_{2^{k-\log_2(m)}}$ and $\widetilde{\mathsf{PRF}} : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{S}_{2^k}$ be pseudorandom functions.
- Let $A_1, \ldots, A_n \subset [m]^2$ be disjoint and cover $[m]^2$ such that $(j, j') \in A_i$ only if $i \in D_j$ and $i \in D_{j'}$.
- Let $B_1, \ldots, B_n \subset [m]$ be disjoint and cover $[m]$.

Setup. In the setup phase:

– For each pair $i, j \in [n]$ where $i < j$, parties P_i, P_j jointly sample $s_{i,j} \leftarrow \{0,1\}^\kappa$.
– Parties $P_i \in D_j$ jointly sample key $s_j \leftarrow \{0,1\}^\kappa$. All parties keep a counter $\mathsf{cnt} := 0$.

Multiply. Upon input shares $[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}} \in \mathbb{S}_{2^k}$, parties compute the product as follows:

1. The parties sample $[\![r]\!]^{\mathsf{R}}$ such that $r_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$.
2. Then, they compute an n-out-of-n secret sharing of 0 where P_i's share β_i is computed as follows: Define $\gamma_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$. Set $\beta_i := \sum_{j \in [n]/i} \gamma_{i,j}$.
3. Each P_i computes the sum of their local crossterms as follows:

$$z'_i := \left(\sum_{(j,j') \in A_i} x_j y_{j'} \right) + \sum_{j \in B_i} r_j + \beta_i$$

where P_i's share $[\![x]\!]^{\mathsf{R}}_i = \{x_j \mid i \in D_j\}$ (resp. $[\![y]\!]^{\mathsf{R}}$).
4. Let $(z'_1, \ldots, z'_n)$ be an n-out-of-n secret sharing of z'. Reveal z' to $P_i \in D_1$.
5. The parties output $[\![z]\!]^{\mathsf{R}}$ where $z_1 := z'/d - r_1/d$ and $z_j := -r_j/d$ for $j \in [2, m]$ and increment $\mathsf{cnt} := \mathsf{cnt} + 1$.

Fig. 8: Protocol for fixed point multiplication with replicated secret shares against semi-honest adversaries.

parties jointly generate replicated shares of a value r by generating each share r_j at random, locally using the PRF key s_j. Observe that while the inputs x and y lie in $\mathbb{S}_{2^{k'}}$, each share r_i lies in $\mathbb{S}_{2^{k'+\lambda}}$ and the value r lies in $\mathbb{S}_{2^{k'+\lambda+\log_2(m)}}$. The parties then use the PRF keys $s_{i,j}$ to generate a replicated sharing of 0 - represented by the β's where each β_i lies in $\mathbb{S}_{2^k}$.

The rest of the protocol follows similar to the standard multiplication technique for replicated secret sharing. Specifically, each party first generates a share of $z' = x \cdot y + r$, masked with a share of 0. The value z' is revealed to all parties in D_1. Note that the sharing of 0 ensures that no individual share of z' is revealed in the clear to any party - this is because each individual share is not

over the whole ring $\mathbb{S}_{2^k}$ and hence might leak information about the terms in $x \cdot y$ if revealed in the clear.

Now, when the parties in D_1 learn $z' = x \cdot y + r$, observe that although r is not entirely uniform in the ring $\mathbb{S}_{2^k}$, its sampling space is sufficiently larger than the value of $x \cdot y$ and it is in fact statistically close to a uniform distribution, hence the value z' looks sufficiently random to all the parties. Finally, all the parties can obtain the sharing of $[\![z]\!]^{\mathsf{R}}$ by setting the first share $z_1 = z'/d - r_1/d$ and the rest of the shares as $z_j = -r_j/d$. The formal security proof is presented for the entire MPC protocol in Sect. 5.2.

Remarks.

1. **Rounds vs Communication tradeoff.** In the multiply phase, the protocol as written requires only one round of communication. However, every party sends its share of z' to every party in D_1 and so the total number of messages exchanged is $n \cdot (n-t)$ which is quadratic in the number of parties. Alternatively, we could have a two-round protocol where in the first round all parties send their shares of z' only to one party in D_1 (say P_1) who then forwards it to the others in D_1. This reduces the number of messages to $(n-1) + (n-t-1)$ which is linear in the number of parties.
2. **Information-theoretic.** Computational assumptions are used only for the PRF. However, the use of a PRF is needed only to reduce communication. In fact, to make the protocol completely information theoretic, we could replace the PRF calls by having one party pick the corresponding random string and share it with the others (at the cost of more communication).

5.2 MPC Protocol

In this section, we show the MPC protocol with replicated secret sharing leveraging our new fixed point multiplication protocol. Protocol $\Pi_{\mathsf{RShare-MPC-SH}}$, presented in Fig. 9, can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation. The protocol is parameterized by $t < n/2$ - which denotes the maximum number of parties the adversary can corrupt.

5.3 Security Proof

We now prove that the protocol $\Pi_{\mathsf{RShare-MPC-SH}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ against semi-honest adversaries with an honest majority. The rounding error distribution $\mathcal{E}$ is defined by the random variable $(\sum_{i\in[m]} s_i)/d$ where s_i is uniform over $\mathbb{S}_d$. Formally, we prove the following theorem:

Theorem 1. *Assuming one way functions exist, protocol $\Pi_{\mathsf{RShare-MPC-SH}}$ presented in Fig. 9 securely computes $\mathcal{F}_{\mathsf{C_d}}$ (Fig. 2) with rounding error distribution $\mathcal{E}$ defined above, in the presence of a semi-honest adversary that corrupts a set of $t < n/2$ parties.*

$\Pi_{\mathsf{RShare-MPC-SH}}$

- The protocol works among n parties $P_1, \ldots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \ldots, D_m \subset [n]$ be the m distinct subsets of size $(n-t)$. For each $j \in [m]$, let S_j be the set of parties indexed by D_j.
- During the circuit computation, all the inputs and outputs of every multiplication gate have absolute value $< 2^{k'}$ where $k \geq (k' + \lambda + \log_2(m))$.

Setup: Run the setup of protocol $\Pi_{\mathsf{RShare-Mul}}$ in Fig. 8.
Share: Every party $P \in \{P_1, \ldots, P_n\}$ shares an input value $x \in \mathbb{S}_{2^{k'}}$ as follows:

1. Sample $x_1, \ldots, x_m \leftarrow \mathbb{S}_{2^k}$ such that $x = \sum_{i \in [m]} x_i$.
2. Send x_j to all parties P_i such that $i \in D_j$.
3. Each party P_i sets $[\![x]\!]^{\mathsf{R}}_i := \{x_i \mid i \in D_j\}$

Linear operations: For any constant c and share $[\![y]\!]^{\mathsf{R}}$, parties perform linear operations on share $[\![x]\!]^{\mathsf{R}}$ to compute the output $[\![z]\!]^{\mathsf{R}}$ as follows:

1. **Addition by a constant** c**:** $z_1 = x_1 + c$ and $z_j = x_j$ for $j \in [m]/1$
2. **Addition of shares:** $z_j = x_j + y_j$ for $j \in [m]$
3. **Multiplication by a constant** c**:** $z_j = cx_j$ for $j \in [m]$

Fixed point multiplication: For any two shares $[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$, parties perform fixed point multiplication by running the multiply phase of $\Pi_{\mathsf{RShare-Mul}}$ in Fig. 8.
Output reconstruction: Each party P_i holds $[\![z]\!]^{\mathsf{R}}_i$ and publicly reconstructs z as follows:

1. For $j \in [m]$ where $1 \notin D_j$, $P_{\min(D_j)}$ sends z_j to P_1.
2. P_1 computes $z = \sum_{i \in [m]} z_i$ and sends z to all other parties.

Fig. 9: Protocol for MPC with replicated secret sharing against semi-honest adversaries.

Proof. Consider an adversary $\mathcal{A}$ that corrupts a set S^* of t parties where $t < n/2$. Let the honest parties be denoted by set H comprising $P_{H_1}, \ldots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{\mathsf{C_d}}$ and the set $(\mathsf{inp}_{i^*}, r_{i^*})_{P_{i^*} \in S^*}$ indicating the corrupt parties' inputs and randomness. The strategy of $\mathcal{S}$ is described below.

Share: On behalf of each honest party P_i, do:

1. Sample $x_1, \ldots, x_m \leftarrow \mathbb{S}_{2^k}$ such that $0 = \sum_{i \in [m]} x_i$.
2. Send x_j to $\mathcal{A}$ for each corrupt party P_{i^*} where $i^* \in D_j$.

Linear Operations: These are local operations that don't need to be simulated.

Fixed point multiplication: $\mathcal{S}$ participates in the setup phase as in Fig. 8. For each multiplication, for any two shares $[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$, $\mathcal{S}$ does:

1. Recall that S_j is the set of parties indexed by D_j. Sample $[\![r]\!]^{\mathsf{R}}$ as follows: For each $j \in [m]$, if $S_j \cap S^* = \emptyset$, sample $r_j \leftarrow \mathbb{S}_{2^{k-\log_2(m)}}$. Else, set $r_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$ as in Fig. 8.
2. For each honest party P_i, compute $\beta_i := \sum_{j \in [n]/i} \gamma_{i,j}$ where $\gamma_{i,j}$ is computed as follows:

- If $P_j \in S^*$, define $\gamma_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$ as in Fig. 8.
- If $P_j \notin S^*$, and $j > i$, sample $\gamma_{i,j} \leftarrow \mathbb{S}_{2^k}$ and store the value in $\Gamma[i][j]$. If $P_j \notin S^*$, and $j < i$, look up $\Gamma[i][j]$ and set $\gamma_{i,j} = -\Gamma[i][j]$.

3. Run Step 3 to Step 5 as in Fig. 8.

Output reconstruction: $\mathcal{S}$ reconstructs output out as follows.

- **If $P_1 \notin S^*$:**
  1. Receive a set of messages from $\mathcal{A}$.
  2. On behalf of P_1, send output out to all other parties.
- **If $P_1 \in S^*$:**
  1. Set $z^* = 0$.
  2. From $\mathcal{A}$'s inputs and randomness $(\mathsf{inp}_{i^*}, r_{i^*})_{P_{i^*} \in S^*}$, and the protocol transcript so far, for each $j \in [m]$, if $S_j \cap S^* \neq \emptyset$, compute z_j and set $z^* = z^* + z_j$. If $P_{\min(D_j)} \in H$, send z_j to $\mathcal{A}$.
  3. Let $j_1, \ldots, j_\ell \in [m]$ denote the indices for which $S_j \cap S^* = \emptyset$.
  4. Pick $z_{j_1}, \ldots, z_{j_{\ell-1}} \leftarrow \mathbb{S}_{2^k}$. Compute $z_{j_\ell} \in \mathbb{S}_{2^k}$ as $z_{j_\ell} = \mathsf{out} - \sum_{i \in [\ell-1]} z_{j_i} - z^*$.
  5. On behalf of the honest parties, send $\{z_{j_1}, \ldots, z_{j_\ell}\}$ to $\mathcal{A}$.

In Appendix A, we prove that the above simulation strategy is successful.

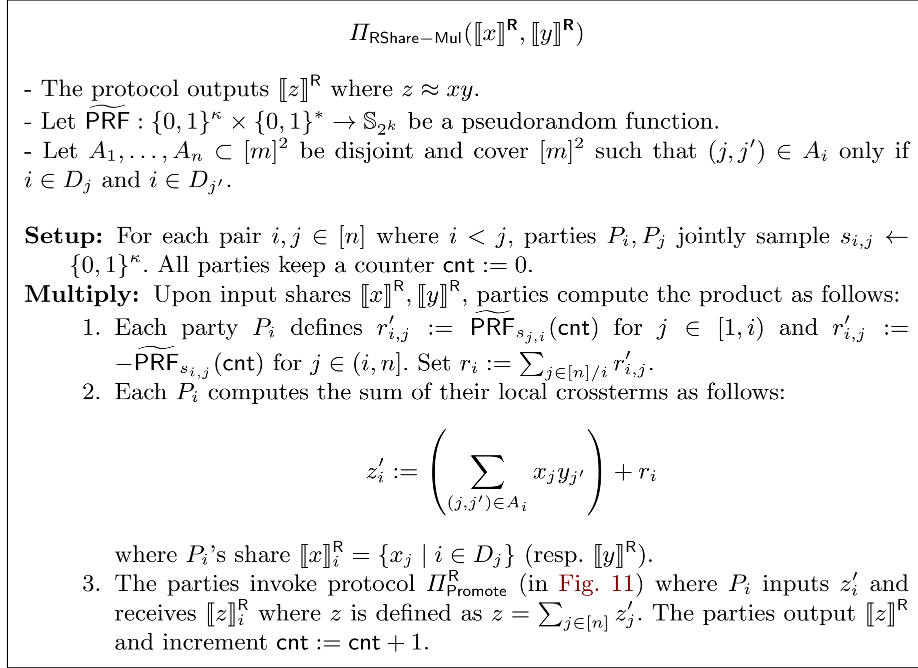
6 Replicated Secret Sharing: Malicious

In this section, we build on the ideas from the previous section to develop a new protocol for fixed point multiplication with replicated secret sharing in the presence of a malicious adversary. We then show how to incorporate this into a general MPC protocol to compute any arithmetic circuit. Along the way, we design a new protocol for multiplication with replicated secret sharing.

6.1 Multiplication

In this section, we describe a protocol to multiply two values represented using replicated secret sharing for any n-parties where $n \geq 3$. We use this protocol as a sub-routine in the next section when we design our fixed point multiplication with truncation protocol.

Let the ring be $\mathbb{Z}_{2^k}$. Consider two values x, y. The parties have replicated secret shares $[\![x]\!]^{\mathsf{R}}$ and $[\![y]\!]^{\mathsf{R}}$. Their goal is to compute a replicated secret share $[\![z]\!]^{\mathsf{R}}$ where $z \approx xy$. Replicated multiplication can logically be split into two parts: (1) locally computing a n-of-n additive sharing of the product and (2) "promoting" the n-of-n share into a replicated sharing. We give the basic replicated multiplication protocol in Figure 10 and separately give the "promote" protocol in Figure 11.



$\Pi_{\mathsf{RShare-Mul}}(\llbracket x \rrbracket^{\mathsf{R}}, \llbracket y \rrbracket^{\mathsf{R}})$

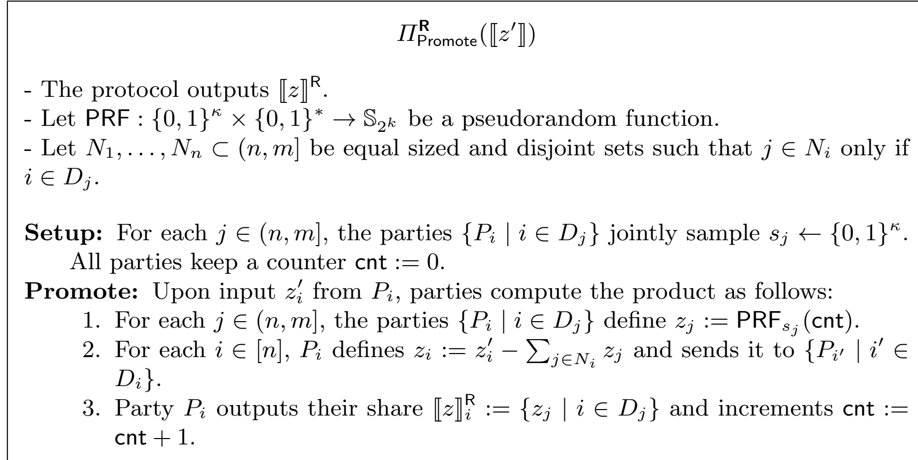
- The protocol outputs $\llbracket z \rrbracket^{\mathsf{R}}$ where $z \approx xy$.
- Let $\widetilde{\mathsf{PRF}} : \{0,1\}^{\kappa} \times \{0,1\}^{*} \rightarrow \mathbb{S}_{2^k}$ be a pseudorandom function.
- Let $A_1, \ldots, A_n \subset [m]^2$ be disjoint and cover $[m]^2$ such that $(j, j') \in A_i$ only if $i \in D_j$ and $i \in D_{j'}$.

Setup: For each pair $i, j \in [n]$ where $i < j$, parties P_i, P_j jointly sample $s_{i,j} \leftarrow \{0,1\}^{\kappa}$. All parties keep a counter $\mathsf{cnt} := 0$.

Multiply: Upon input shares $\llbracket x \rrbracket^{\mathsf{R}}, \llbracket y \rrbracket^{\mathsf{R}}$, parties compute the product as follows:

1. Each party P_i defines $r'_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $r'_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$. Set $r_i := \sum_{j \in [n]/i} r'_{i,j}$.
2. Each P_i computes the sum of their local crossterms as follows:
$$z'_i := \left(\sum_{(j,j') \in A_i} x_j y_{j'} \right) + r_i$$
where P_i's share $\llbracket x \rrbracket^{\mathsf{R}}_i = \{x_j \mid i \in D_j\}$ (resp. $\llbracket y \rrbracket^{\mathsf{R}}$).
3. The parties invoke protocol $\Pi^{\mathsf{R}}_{\mathsf{Promote}}$ (in Fig. 11) where P_i inputs z'_i and receives $\llbracket z \rrbracket^{\mathsf{R}}_i$ where z is defined as $z = \sum_{j \in [n]} z'_j$. The parties output $\llbracket z \rrbracket^{\mathsf{R}}$ and increment $\mathsf{cnt} := \mathsf{cnt} + 1$.

Fig. 10: Protocol for multiplication with replicated secret shares.



$\Pi^{\mathsf{R}}_{\mathsf{Promote}}(\llbracket z' \rrbracket)$

- The protocol outputs $\llbracket z \rrbracket^{\mathsf{R}}$.
- Let $\mathsf{PRF} : \{0,1\}^{\kappa} \times \{0,1\}^{*} \rightarrow \mathbb{S}_{2^k}$ be a pseudorandom function.
- Let $N_1, \ldots, N_n \subset (n, m]$ be equal sized and disjoint sets such that $j \in N_i$ only if $i \in D_j$.

Setup: For each $j \in (n, m]$, the parties $\{P_i \mid i \in D_j\}$ jointly sample $s_j \leftarrow \{0,1\}^{\kappa}$. All parties keep a counter $\mathsf{cnt} := 0$.

Promote: Upon input z'_i from P_i, parties compute the product as follows:

1. For each $j \in (n, m]$, the parties $\{P_i \mid i \in D_j\}$ define $z_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$.
2. For each $i \in [n]$, P_i defines $z_i := z'_i - \sum_{j \in N_i} z_j$ and sends it to $\{P_{i'} \mid i' \in D_i\}$.
3. Party P_i outputs their share $\llbracket z \rrbracket^{\mathsf{R}}_i := \{z_j \mid i \in D_j\}$ and increments $\mathsf{cnt} := \mathsf{cnt} + 1$.

Fig. 11: Protocol for promoting an n-out-of-n sharing into a replicated secret sharing.

6.2 Fixed Point Multiplication

In this section, we present our new protocol for fixed point multiplication with replicated secret sharing secure in the presence of a malicious adversary. Consider two fixed point values x, y represented in twos-complement form in $\mathbb{S}_{2^{k'}}$ where the bottom d bits denote the decimal. The parties hold replicated secret shares

$[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$ in an extended ring $\mathbb{S}_{2^k}$ where $k \geq (k' + 2\lambda + \log_2(m))$. Their goal is to compute a replicated secret share $[\![z]\!]^{\mathsf{R}}$ where $z \approx \frac{xy}{2^d} \in \mathbb{S}_{2^{k'}}$. The protocol works among n parties $P_1, \ldots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \ldots, D_m \subset [n]$ be the m distinct subsets of size $(n - t)$. Prior to the execution of the fixed point multiplication protocol (and as part of the overall MPC protocol), parties jointly sample replicated secret shares of a MAC key $[\![\alpha]\!]^{\mathsf{R}}$, where α is randomly sampled from $\mathbb{S}_{2^\lambda}$. Then, they generate shares of the MAC of one of the inputs $[\![\alpha x]\!]^{\mathsf{R}}$ using the protocol in Fig. 10. We describe our protocol that is secure against a malicious adversary in Fig. 12.

Remark: We can instantiate functionality $\mathcal{F}_{\mathsf{MAC.Check}}$ using the "MAC Check" protocol in Figure 10 of [23].

Overview and proof sketch. In addition to the protocol for semi-honest security, we add an information theoretic MAC $\alpha z'$ along with the term z'. Then, since α is unknown to the adversary, if the MAC Check protocol does not abort, we have the guarantee that the term z' was correctly computed by the adversary. Finally, after computing $[\![z]\!]^{\mathsf{R}}$ as in the semi-honest protocol, we now also compute the MAC $[\![\alpha z]\!]^{\mathsf{R}}$.

6.3 MPC Protocol

In this section, we show the MPC protocol with replicated secret sharing leveraging our new fixed point multiplication protocol. Protocol $\Pi_{\mathsf{RShare-MPC-Mal}}$, presented in Fig. 14, can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation. The protocol is parameterized by $t < n/2$ - which denotes the maximum number of parties the adversary can corrupt. We postpone the security proof to Appendix B.

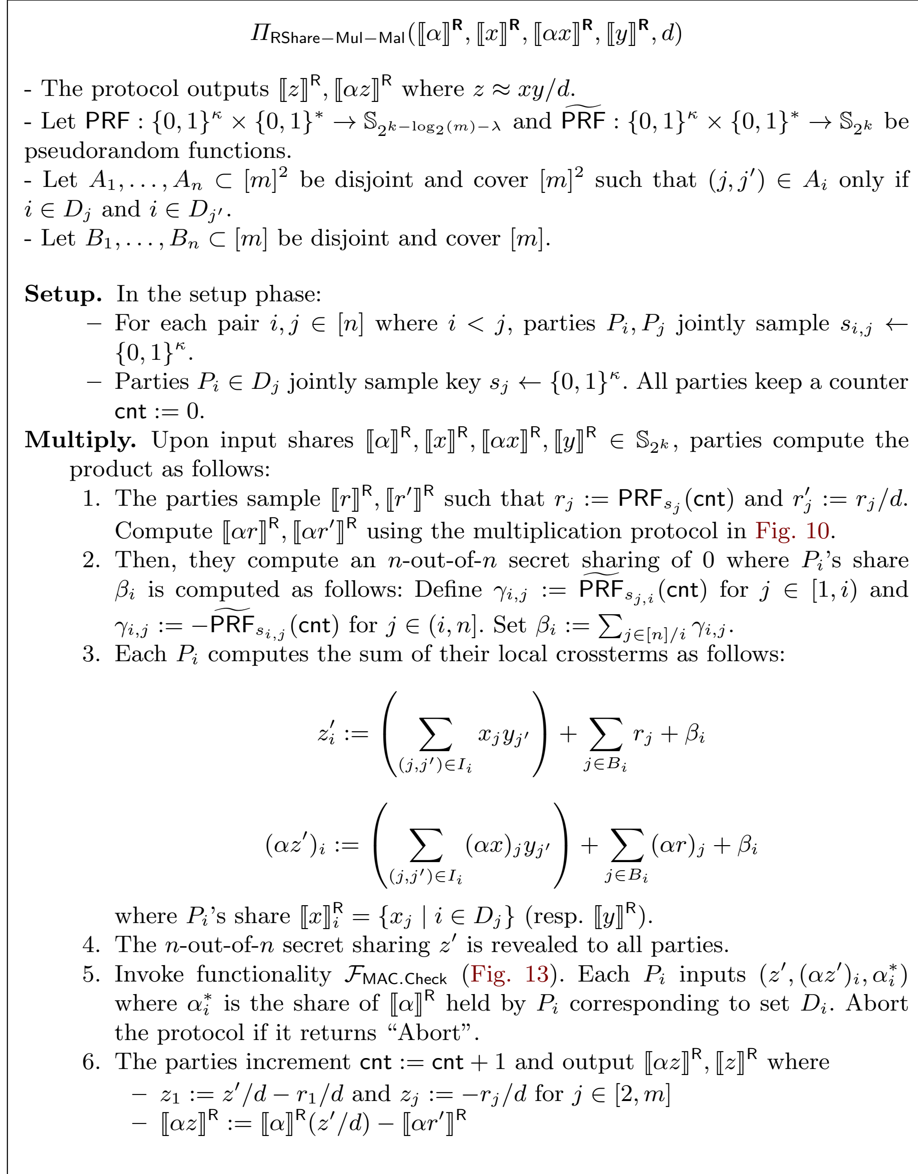
7 Shamir Secret Sharing

In this section, we develop a new approach to fixed point multiplication with Shamir secret sharing, and show how to incorporate this new approach in the general MPC protocol.

7.1 Fixed Point Multiplication

In this section, we present our semi-honest protocol for fixed point multiplication with Shamir secret sharing. Consider two fixed point values $x, y \in \mathbb{S}_{2^{k'}}$ held by the parties in Shamir secret sharing $[\![x]\!]^{\mathsf{S}}, [\![y]\!]^{\mathsf{S}}$ on a field $\mathbb{F}_q$ where $q \geq 2^{k'+\lambda} \cdot (t+1)$. For each multiplication, the parties want to learn a Shamir secret share $[\![z]\!]^{\mathsf{S}}$ where $z \approx xy/d$. We describe our protocol in Fig. 16.

Sketched Security Analysis. At a high level, for each fixed point multiplication, all the parties will first perform a preprocessing where they obtain a $2t$-out-of-n Shamir secret sharing of a random value $[\![r]\!]^{\mathsf{S},2t}$ in $\mathbb{F}_q$ and a t-out-of-n Shamir secret sharing of $[\![r']\!]^{\mathsf{S}}$ in $\mathbb{F}_q$ where $r' = r/d$. r and r' are generated



$\Pi_{\mathsf{RShare-Mul-Mal}}(\llbracket\alpha\rrbracket^{\mathsf{R}}, \llbracket x\rrbracket^{\mathsf{R}}, \llbracket\alpha x\rrbracket^{\mathsf{R}}, \llbracket y\rrbracket^{\mathsf{R}}, d)$

- The protocol outputs $\llbracket z\rrbracket^{\mathsf{R}}, \llbracket\alpha z\rrbracket^{\mathsf{R}}$ where $z \approx xy/d$.
- Let $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \mathbb{S}_{2^{k-\log_2(m)-\lambda}}$ and $\widetilde{\mathsf{PRF}} : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \mathbb{S}_{2^k}$ be pseudorandom functions.
- Let $A_1, \ldots, A_n \subset [m]^2$ be disjoint and cover $[m]^2$ such that $(j, j') \in A_i$ only if $i \in D_j$ and $i \in D_{j'}$.
- Let $B_1, \ldots, B_n \subset [m]$ be disjoint and cover $[m]$.

Setup. In the setup phase:
- For each pair $i, j \in [n]$ where $i < j$, parties P_i, P_j jointly sample $s_{i,j} \leftarrow \{0,1\}^\kappa$.
- Parties $P_i \in D_j$ jointly sample key $s_j \leftarrow \{0,1\}^\kappa$. All parties keep a counter $\mathsf{cnt} := 0$.

Multiply. Upon input shares $\llbracket\alpha\rrbracket^{\mathsf{R}}, \llbracket x\rrbracket^{\mathsf{R}}, \llbracket\alpha x\rrbracket^{\mathsf{R}}, \llbracket y\rrbracket^{\mathsf{R}} \in \mathbb{S}_{2^k}$, parties compute the product as follows:
1. The parties sample $\llbracket r\rrbracket^{\mathsf{R}}, \llbracket r'\rrbracket^{\mathsf{R}}$ such that $r_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$ and $r'_j := r_j/d$. Compute $\llbracket\alpha r\rrbracket^{\mathsf{R}}, \llbracket\alpha r'\rrbracket^{\mathsf{R}}$ using the multiplication protocol in Fig. 10.
2. Then, they compute an n-out-of-n secret sharing of 0 where P_i's share β_i is computed as follows: Define $\gamma_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$. Set $\beta_i := \sum_{j\in[n]/i} \gamma_{i,j}$.
3. Each P_i computes the sum of their local crossterms as follows:

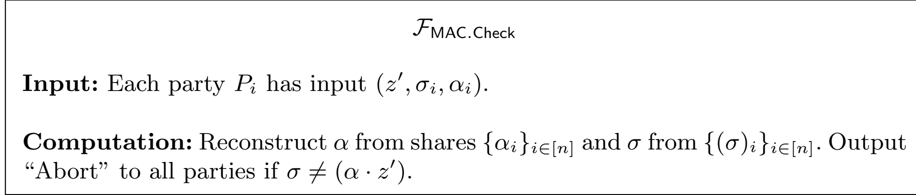
$$z'_i := \left(\sum_{(j,j')\in I_i} x_j y_{j'}\right) + \sum_{j\in B_i} r_j + \beta_i$$

$$(\alpha z')_i := \left(\sum_{(j,j')\in I_i} (\alpha x)_j y_{j'}\right) + \sum_{j\in B_i} (\alpha r)_j + \beta_i$$

where P_i's share $\llbracket x\rrbracket^{\mathsf{R}}_i = \{x_j \mid i \in D_j\}$ (resp. $\llbracket y\rrbracket^{\mathsf{R}}$).
4. The n-out-of-n secret sharing z' is revealed to all parties.
5. Invoke functionality $\mathcal{F}_{\mathsf{MAC.Check}}$ (Fig. 13). Each P_i inputs $(z', (\alpha z')_i, \alpha^*_i)$ where α^*_i is the share of $\llbracket\alpha\rrbracket^{\mathsf{R}}$ held by P_i corresponding to set D_i. Abort the protocol if it returns "Abort".
6. The parties increment $\mathsf{cnt} := \mathsf{cnt} + 1$ and output $\llbracket\alpha z\rrbracket^{\mathsf{R}}, \llbracket z\rrbracket^{\mathsf{R}}$ where
   - $z_1 := z'/d - r_1/d$ and $z_j := -r_j/d$ for $j \in [2, m]$
   - $\llbracket\alpha z\rrbracket^{\mathsf{R}} := \llbracket\alpha\rrbracket^{\mathsf{R}}(z'/d) - \llbracket\alpha r'\rrbracket^{\mathsf{R}}$

Fig. 12: Protocol for fixed point multiplication with replicated secret shares against malicious adversaries.

as a summation of n random values, namely $r = \sum_{i\in[n]} r_i$ and $r' = \sum_{i\in[n]} r'_i$, where $r'_i = r_i/d$ and they are contributed by all the parties. We make sure there is no overflow in the summation by sampling r_i from $\mathbb{S}_{2^{k'+\lambda}}$. Once we generate the double sharing with truncation, the rest of the protocol follows from the standard multiplication techniques for Shamir secret sharing.

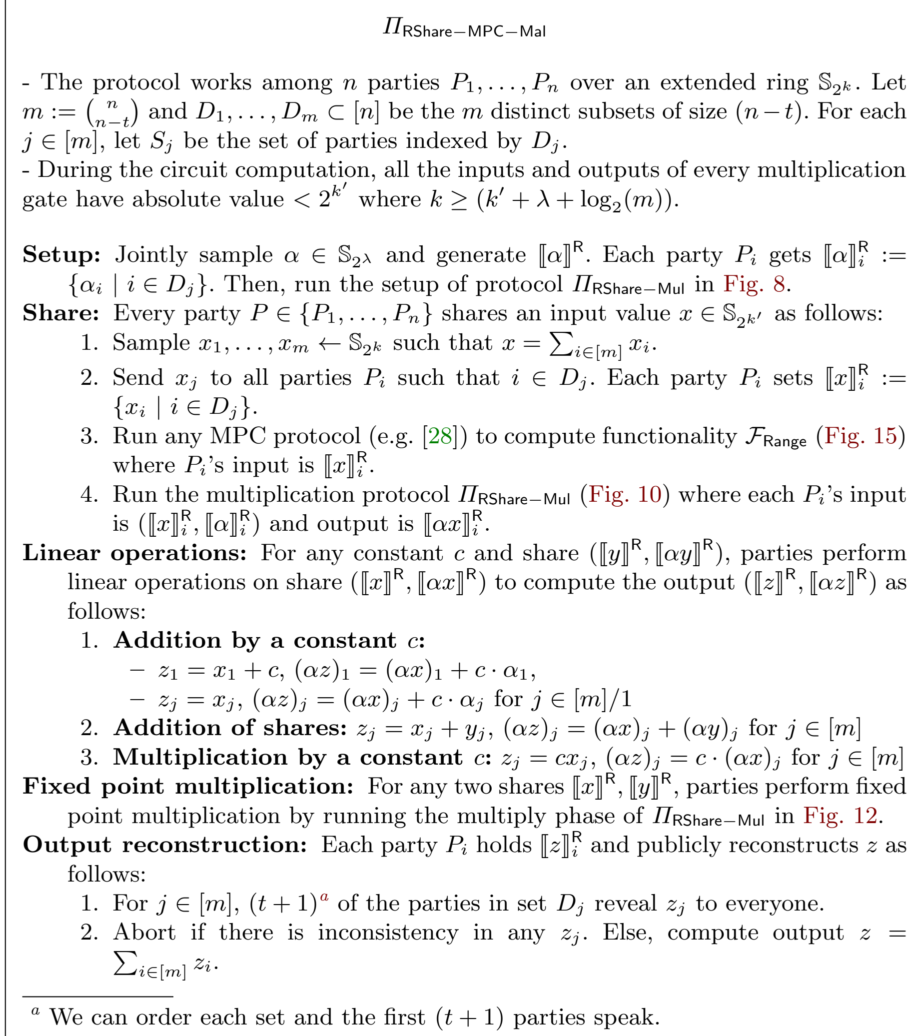


$\mathcal{F}_{\mathsf{MAC.Check}}$

Input: Each party P_i has input (z', σ_i, α_i).

Computation: Reconstruct α from shares $\{\alpha_i\}_{i\in[n]}$ and σ from $\{(\sigma)_i\}_{i\in[n]}$. Output "Abort" to all parties if $\sigma \neq (\alpha \cdot z')$.

Fig. 13: Ideal functionality for checking the MAC on a secret shared value.



$\Pi_{\mathsf{RShare-MPC-Mal}}$

- The protocol works among n parties $P_1, \ldots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \ldots, D_m \subset [n]$ be the m distinct subsets of size $(n-t)$. For each $j \in [m]$, let S_j be the set of parties indexed by D_j.
- During the circuit computation, all the inputs and outputs of every multiplication gate have absolute value $< 2^{k'}$ where $k \geq (k' + \lambda + \log_2(m))$.

Setup: Jointly sample $\alpha \in \mathbb{S}_{2^\lambda}$ and generate $[\![\alpha]\!]^{\mathsf{R}}$. Each party P_i gets $[\![\alpha]\!]_i^{\mathsf{R}} := \{\alpha_i \mid i \in D_j\}$. Then, run the setup of protocol $\Pi_{\mathsf{RShare-Mul}}$ in Fig. 8.

Share: Every party $P \in \{P_1, \ldots, P_n\}$ shares an input value $x \in \mathbb{S}_{2^{k'}}$ as follows:
1. Sample $x_1, \ldots, x_m \leftarrow \mathbb{S}_{2^k}$ such that $x = \sum_{i\in[m]} x_i$.
2. Send x_j to all parties P_i such that $i \in D_j$. Each party P_i sets $[\![x]\!]_i^{\mathsf{R}} := \{x_i \mid i \in D_j\}$.
3. Run any MPC protocol (e.g. [28]) to compute functionality $\mathcal{F}_{\mathsf{Range}}$ (Fig. 15) where P_i's input is $[\![x]\!]_i^{\mathsf{R}}$.
4. Run the multiplication protocol $\Pi_{\mathsf{RShare-Mul}}$ (Fig. 10) where each P_i's input is $([\![x]\!]_i^{\mathsf{R}}, [\![\alpha]\!]_i^{\mathsf{R}})$ and output is $[\![\alpha x]\!]_i^{\mathsf{R}}$.

Linear operations: For any constant c and share $([\![y]\!]^{\mathsf{R}}, [\![\alpha y]\!]^{\mathsf{R}})$, parties perform linear operations on share $([\![x]\!]^{\mathsf{R}}, [\![\alpha x]\!]^{\mathsf{R}})$ to compute the output $([\![z]\!]^{\mathsf{R}}, [\![\alpha z]\!]^{\mathsf{R}})$ as follows:
1. **Addition by a constant c:**
   - $z_1 = x_1 + c$, $(\alpha z)_1 = (\alpha x)_1 + c \cdot \alpha_1$,
   - $z_j = x_j$, $(\alpha z)_j = (\alpha x)_j + c \cdot \alpha_j$ for $j \in [m]/1$
2. **Addition of shares:** $z_j = x_j + y_j$, $(\alpha z)_j = (\alpha x)_j + (\alpha y)_j$ for $j \in [m]$
3. **Multiplication by a constant c:** $z_j = c x_j$, $(\alpha z)_j = c \cdot (\alpha x)_j$ for $j \in [m]$

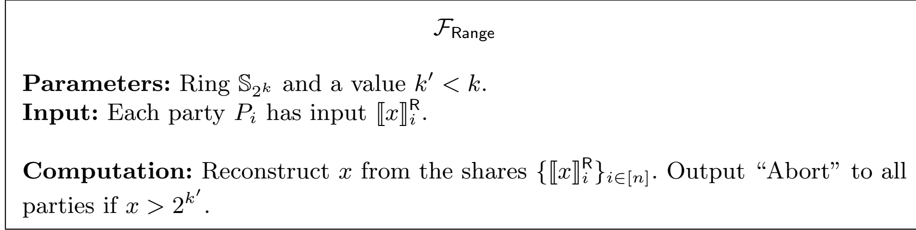
Fixed point multiplication: For any two shares $[\![x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$, parties perform fixed point multiplication by running the multiply phase of $\Pi_{\mathsf{RShare-Mul}}$ in Fig. 12.

Output reconstruction: Each party P_i holds $[\![z]\!]_i^{\mathsf{R}}$ and publicly reconstructs z as follows:
1. For $j \in [m]$, $(t+1)$[a] of the parties in set D_j reveal z_j to everyone.
2. Abort if there is inconsistency in any z_j. Else, compute output $z = \sum_{i\in[m]} z_i$.

[a] We can order each set and the first $(t+1)$ parties speak.

Fig. 14: Protocol for MPC with replicated secret sharing against malicious adversaries.

Specifically, P_1 first learns a value $z' = x \cdot y + r$ and sends it to all the other parties. Although r is not entirely uniform in the field $\mathbb{F}_q$, its sampling space is

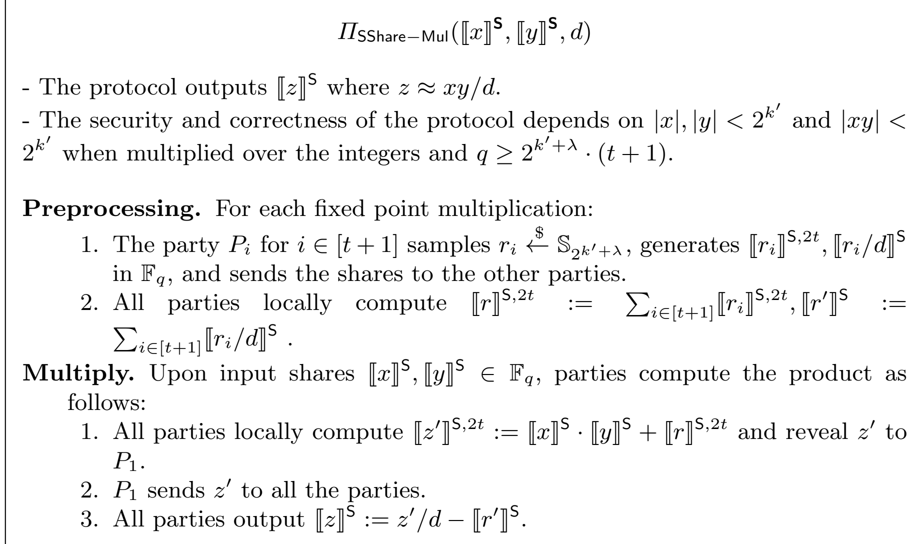


$\mathcal{F}_{\mathsf{Range}}$

Parameters: Ring $\mathbb{S}_{2^k}$ and a value $k' < k$.
Input: Each party P_i has input $[\![x]\!]^{\mathsf{R}}_i$.

Computation: Reconstruct x from the shares $\{[\![x]\!]^{\mathsf{R}}_i\}_{i \in [n]}$. Output "Abort" to all parties if $x > 2^{k'}$.

Fig. 15: Ideal functionality for checking the range of a secret shared value



$\Pi_{\mathsf{SShare-Mul}}([\![x]\!]^{\mathsf{S}}, [\![y]\!]^{\mathsf{S}}, d)$

- The protocol outputs $[\![z]\!]^{\mathsf{S}}$ where $z \approx xy/d$.
- The security and correctness of the protocol depends on $|x|, |y| < 2^{k'}$ and $|xy| < 2^{k'}$ when multiplied over the integers and $q \geq 2^{k'+\lambda} \cdot (t+1)$.

Preprocessing. For each fixed point multiplication:

1. The party P_i for $i \in [t+1]$ samples $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$, generates $[\![r_i]\!]^{\mathsf{S},2t}, [\![r_i/d]\!]^{\mathsf{S}}$ in $\mathbb{F}_q$, and sends the shares to the other parties.
2. All parties locally compute $[\![r]\!]^{\mathsf{S},2t} := \sum_{i \in [t+1]} [\![r_i]\!]^{\mathsf{S},2t}, [\![r']\!]^{\mathsf{S}} := \sum_{i \in [t+1]} [\![r_i/d]\!]^{\mathsf{S}}$.

Multiply. Upon input shares $[\![x]\!]^{\mathsf{S}}, [\![y]\!]^{\mathsf{S}} \in \mathbb{F}_q$, parties compute the product as follows:

1. All parties locally compute $[\![z']\!]^{\mathsf{S},2t} := [\![x]\!]^{\mathsf{S}} \cdot [\![y]\!]^{\mathsf{S}} + [\![r]\!]^{\mathsf{S},2t}$ and reveal z' to P_1.
2. P_1 sends z' to all the parties.
3. All parties output $[\![z]\!]^{\mathsf{S}} := z'/d - [\![r']\!]^{\mathsf{S}}$.

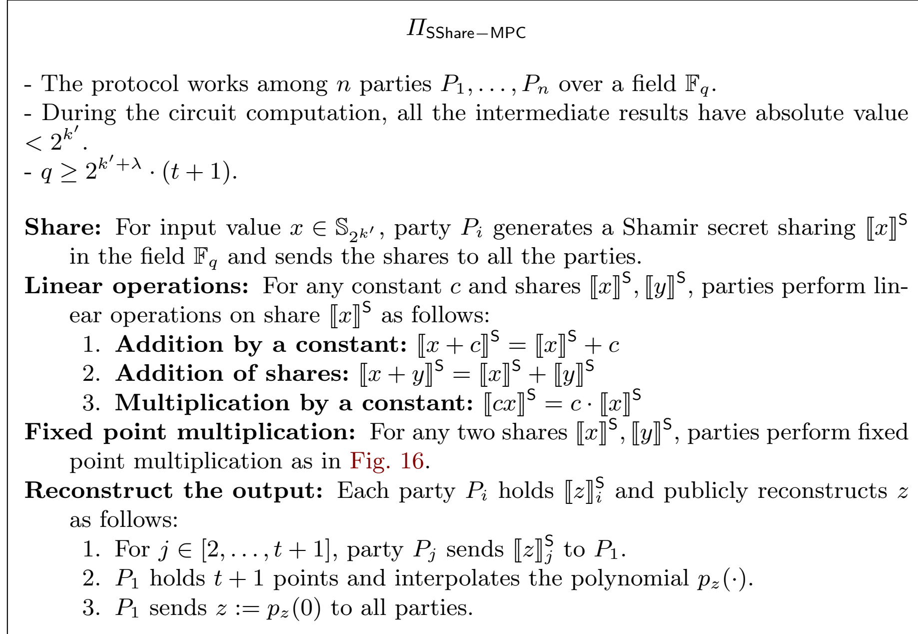
Fig. 16: Semi-honest protocol for fixed point multiplication with Shamir secret shares.

sufficiently larger than the value of $x \cdot y$ and it is in fact statistically close to a uniform distribution, hence the value z' looks sufficiently random to all the parties. Finally, by the correctness of double sharing with truncation for r and r', all the parties can obtain the sharing of $[\![z]\!]^{\mathsf{S}}$ by taking $z'/d - [\![r']\!]^{\mathsf{S}}$. The formal security proof will be presented in Sect. 7.3.

Remark. Our truncation technique does not have to follow a multiplication step and it does not necessarily require honest majority. It can be used as a general approach for truncating Shamir-shared value for any $t < n$.

7.2 MPC with Shamir Secret Sharing

In this section, we show the MPC protocol with Shamir secret sharing leveraging our new fixed point multiplication protocol. The protocol, presented in Fig. 17, can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation.



$\Pi_{\mathsf{SShare-MPC}}$

- The protocol works among n parties $P_1, \ldots, P_n$ over a field $\mathbb{F}_q$.
- During the circuit computation, all the intermediate results have absolute value $< 2^{k'}$.
- $q \geq 2^{k'+\lambda} \cdot (t+1)$.

Share: For input value $x \in \mathbb{S}_{2^{k'}}$, party P_i generates a Shamir secret sharing $[\![x]\!]^{\mathsf{S}}$ in the field $\mathbb{F}_q$ and sends the shares to all the parties.

Linear operations: For any constant c and shares $[\![x]\!]^{\mathsf{S}}, [\![y]\!]^{\mathsf{S}}$, parties perform linear operations on share $[\![x]\!]^{\mathsf{S}}$ as follows:
1. **Addition by a constant:** $[\![x+c]\!]^{\mathsf{S}} = [\![x]\!]^{\mathsf{S}} + c$
2. **Addition of shares:** $[\![x+y]\!]^{\mathsf{S}} = [\![x]\!]^{\mathsf{S}} + [\![y]\!]^{\mathsf{S}}$
3. **Multiplication by a constant:** $[\![cx]\!]^{\mathsf{S}} = c \cdot [\![x]\!]^{\mathsf{S}}$

Fixed point multiplication: For any two shares $[\![x]\!]^{\mathsf{S}}, [\![y]\!]^{\mathsf{S}}$, parties perform fixed point multiplication as in Fig. 16.

Reconstruct the output: Each party P_i holds $[\![z]\!]_i^{\mathsf{S}}$ and publicly reconstructs z as follows:
1. For $j \in [2, \ldots, t+1]$, party P_j sends $[\![z]\!]_j^{\mathsf{S}}$ to P_1.
2. P_1 holds $t+1$ points and interpolates the polynomial $p_z(\cdot)$.
3. P_1 sends $z := p_z(0)$ to all parties.

Fig. 17: Protocol for MPC with Shamir secret sharing.

7.3 Security Proof

In this section, we prove that the protocol $\Pi_{\mathsf{SShare-MPC}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ against semi-honest adversaries corrupting $t < n/2$ parties. The rounding error distribution $\mathcal{E}$ is defined by the random variable $(\sum_{i\in[t+1]} s_i)/d$ where s_i is uniform over $\mathbb{S}_d$. Formally, we prove the following theorem:

Theorem 2. *The protocol $\Pi_{\mathsf{SShare-MPC}}$ presented in Fig. 17 statistically securely realizes the functionality $\mathcal{F}_{\mathsf{C_d}}$ (defined in Fig. 2) with rounding error distribution $\mathcal{E}$ in the presence of a semi-honest adversary that corrupts $t < n/2$ parties.*

Proof. Correctness of the protocol follows from the invariant that after every linear operation or fixed point multiplication, parties hold t-out-of-n Shamir secret sharing of the output. This is straightforward for linear operations. In terms of fixed point multiplication by truncation, the invariant relies on the fact that $(\sum_{i\in[t+1]} r_i)/d \approx \sum_{i\in[t+1]} r_i/d$. Note that each $|r_i| < 2^{k'+\lambda}$, hence $\sum_{i\in[t+1]} |r_i| < 2^{k'+\lambda} \cdot (t+1)$, which suggests that $\sum_{i\in[t+1]} r_i$ does not overflow. Therefore $(\sum_{i\in[t+1]} r_i)/d \approx \sum_{i\in[t+1]} r_i/d$, where the rounding error follows the distribution $\mathcal{E}$ defined above.

For security, consider an adversary $\mathcal{A}$ that corrupts a set C of t parties, denoted as $P_{C_1}, \ldots, P_{C_t}$. Let the honest parties be denoted by set H comprising $P_{H_1}, \ldots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{\mathsf{C_d}}$ and the set $\{(\mathsf{inp}_{C_i}, r_{C_i})\}_{i\in[t]}$ indicating the corrupt parties' inputs and randomness. We construct a PPT simulator $\mathcal{S}$ that follows

the protocol description of the corrupted parties to generate $\mathcal{A}$'s view and does the following on behalf of honest parties:

Share: On behalf of each honest party P_{H_i}:

1. Sample $s_1, \ldots, s_t \xleftarrow{\$} \mathbb{F}_d$.
2. Send s_j to $\mathcal{A}$ for each corrupt party P_{C_j}.

Linear operations: These are local operations that don't need to be simulated.

Fixed point multiplication: For each fixed point multiplication:

- **Preprocessing:**
  1. For each corrupt party P_{C_i} where $C_i \in [t+1]$, sample $r_{C_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$.
  2. For each honest party P_{H_i} where $H_i \in [t+1]$, sample $r_{H_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$, generate $[\![r_{H_i}]\!]^{\mathsf{S},2t}$, $[\![r_{H_i}/d]\!]^{\mathsf{S}}$, and send the shares to the corrupted parties.
  3. Let $r := \sum_{i \in [t+1]} r_i$.
- **Online multiplication:**
  - If P_1 is an honest party, let $z' := r$ and send it to $\mathcal{A}$ on behalf of P_1.
  - If P_1 is a corrupt party, compute the shares of $[\![z']\!]^{\mathsf{S},2t}$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z' = r$, and send them to P_1 on behalf of the honest parties.

Reconstruct the output:

- If P_1 is an honest party, then let $z := \mathsf{out}$ and send it to $\mathcal{A}$ on behalf of P_1.
- If P_1 is a corrupt party, compute the shares of $[\![z]\!]^{\mathsf{S}}$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z = \mathsf{out}$, and send these shares to P_1 on behalf of the honest parties.

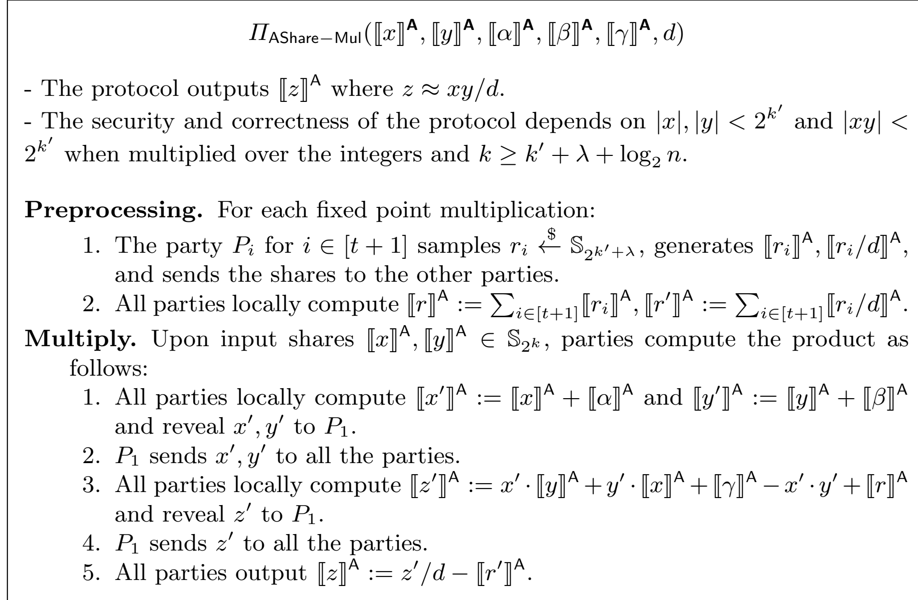
In Appendix C, we prove that the above simulation strategy is successful.

8 Additive Secret Sharing

In this section, we develop a new approach to fixed point multiplication with additive secret sharing, and show how to incorporate this new approach in the general MPC protocol.

8.1 Fixed Point Multiplication

In this section, we present our new semi-honest protocol for fixed point multiplication with additive secret sharing. Consider two fixed point values x, y represented in twos-complement form in $\mathbb{S}_{2^{k'}}$. The parties hold additive secret shares $[\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}}$ in an extended ring $\mathbb{S}_{2^k}$ where $k \geq k' + \lambda + \log_2 n$. Further, we assume the parties hold preprocessed Beaver triples of the form $[\![\alpha]\!]^{\mathsf{A}}, [\![\beta]\!]^{\mathsf{A}}, [\![\gamma]\!]^{\mathsf{A}}$ in $\mathbb{S}_{2^k}$ where $\alpha, \beta \xleftarrow{\$} \mathbb{S}_{2^k}$ and $\gamma = \alpha \cdot \beta$. For each fixed point multiplication, the



$\Pi_{\mathsf{AShare-Mul}}([\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}}, [\![\alpha]\!]^{\mathsf{A}}, [\![\beta]\!]^{\mathsf{A}}, [\![\gamma]\!]^{\mathsf{A}}, d)$

- The protocol outputs $[\![z]\!]^{\mathsf{A}}$ where $z \approx xy/d$.
- The security and correctness of the protocol depends on $|x|, |y| < 2^{k'}$ and $|xy| < 2^{k'}$ when multiplied over the integers and $k \geq k' + \lambda + \log_2 n$.

Preprocessing. For each fixed point multiplication:

1. The party P_i for $i \in [t+1]$ samples $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$, generates $[\![r_i]\!]^{\mathsf{A}}, [\![r_i/d]\!]^{\mathsf{A}}$, and sends the shares to the other parties.
2. All parties locally compute $[\![r]\!]^{\mathsf{A}} := \sum_{i\in[t+1]} [\![r_i]\!]^{\mathsf{A}}$, $[\![r']\!]^{\mathsf{A}} := \sum_{i\in[t+1]} [\![r_i/d]\!]^{\mathsf{A}}$.

Multiply. Upon input shares $[\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}} \in \mathbb{S}_{2^k}$, parties compute the product as follows:

1. All parties locally compute $[\![x']\!]^{\mathsf{A}} := [\![x]\!]^{\mathsf{A}} + [\![\alpha]\!]^{\mathsf{A}}$ and $[\![y']\!]^{\mathsf{A}} := [\![y]\!]^{\mathsf{A}} + [\![\beta]\!]^{\mathsf{A}}$ and reveal x', y' to P_1.
2. P_1 sends x', y' to all the parties.
3. All parties locally compute $[\![z']\!]^{\mathsf{A}} := x' \cdot [\![y]\!]^{\mathsf{A}} + y' \cdot [\![x]\!]^{\mathsf{A}} + [\![\gamma]\!]^{\mathsf{A}} - x' \cdot y' + [\![r]\!]^{\mathsf{A}}$ and reveal z' to P_1.
4. P_1 sends z' to all the parties.
5. All parties output $[\![z]\!]^{\mathsf{A}} := z'/d - [\![r']\!]^{\mathsf{A}}$.

Fig. 18: Semi-honest protocol for fixed point multiplication with additive secret shares.

parties want to learn an additive secret share $[\![z]\!]^{\mathsf{A}}$ where $z \approx xy/d$. We describe our protocol in Fig. 18.

Remark. Our multiplication protocol is a standard multiplication with Beaver triples followed by a our new truncation technique. The truncation step does not have to follow a multiplication step and can be used as a general approach whenever truncation is required.

8.2 MPC with Additive Secret Sharing

In this section, we show an MPC protocol with additive secret sharing leveraging our new fixed point multiplication protocol. The protocol, presented in Fig. 19, can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation. We postpone the security proof to Appendix D.

References

1. Google cloud ai. cloud.google.com/products/machine-learning/.
2. Machine learning on aws. aws.amazon.com/machine-learning/.
3. Microsoft azure. https://azure.microsoft.com/en-us/services/ machine-learning-studio/.
4. Watson machine learning. www.ibm.com/cloud/machine-learning.

5. Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *CCS*, 2016.
6. Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. *PETS*, 2021.
7. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO*, 2002.
8. Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. In *NDSS*, 2013.
9. Abdelrahaman Aly, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Nigel P Smart, and Tim Wood. Scale-mamba v1. 2: Documentation, 2018.
10. Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale P2P mpc-as-a-service and low-bandwidth MPC for weak participants. In *CCS*, 2018.
11. Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: fast and robust framework for privacy-preserving machine learning. *PETs*, 2020.
12. Octavian Catrina. Round-efficient protocols for secure multiparty fixed-point arithmetic. In *COMM*, 2018.
13. Octavian Catrina. Towards practical secure computation with floating-point numbers. *Balkan CryptSec*, 2018.
14. Octavian Catrina. Efficient secure floating-point arithmetic using shamir secret sharing. In *ICETE*, 2019.
15. Octavian Catrina. Optimizing secure floating-point arithmetic: Sums, dot products, and polynomials. 2020.
16. Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography*, 2010.
17. Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: high throughput 3pc over rings with application to secure prediction. In *CCSW*, 2019.
18. Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *NDSS*, 2020.
19. 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 *CRYPTO*, 2018.
20. Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *PETS*, 2020.
21. Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. *IACR ePrint*, 2020.
22. Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE S&P*, 2019.
23. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, 2013.
24. Vassil Dimitrov, Liisi Kerik, Toomas Krips, Jaak Randmets, and Jan Willemson. Alternative implementations of secure real numbers. In *CCS*, 2016.
25. Daniel Escudero and Anders P. K. Dalskov. Honest majority MPC with abort with minimal online communication. *IACR ePrint*, 2020.

26. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *CRYPTO*, 2020.
27. Oded Goldreich. *The Foundations of Cryptography.* 2004.
28. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
29. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *CRYPTO*, 2020.
30. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security*, 2018.
31. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS*, 2020.
32. Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: super-fast and robust privacy-preserving machine learning. *IACR ePrint*, 2020.
33. Toomas Krips and Jan Willemson. Hybrid model of fixed and floating point numbers in secure multiparty computations. In *International Conference on Information Security*, 2014.
34. Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE S&P*, 2020.
35. Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security*, 2020.
36. Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In *CCS*, 2018.
37. Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.
38. Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. In *USENIX Security*, 2020.
39. Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. In *NDSS*, 2020.
40. Pille Pullonen and Sander Siim. Combining secret sharing and garbled circuits for efficient private ieee 754 floating-point computations. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography*, 2015.
41. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, 2020.
42. Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In *ICISC*, 2009.
43. Adi Shamir. How to share a secret. *Commun. ACM*, 1979.
44. Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *PETs*, 2019.

Supplementary Material

A Replicated Semi-Honest: Proof

We now show that the simulation strategy in Sect. 5.3 is successful via a hybrid argument.

Hyb_0**:** Consider a simulator SimHyb that plays the role of the honest parties as in Fig. 9. This is the real world.

Hyb_1**: Switching output.** This is similar to Hyb_0 except that SimHyb now runs the output reconstruction step as done by $\mathcal{S}$ in the ideal world by using the output out from the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$. Honest parties also get their output from $\mathcal{F}_{\mathsf{C_d}}$ in this hybrid.

The only difference between Hyb_0 and Hyb_1 is the manner in which parties learn their output. In the real world, correctness of the protocol follows from the invariant that after every linear operation or fixed point multiplication, parties hold a replicated secret sharing of the output. In the case of fixed point multiplication, it is easy to observe that the rounding error associated with the output follows the distribution $\mathcal{E}$ defined above. From the correctness of the protocol, it is easy to observe that the real world output is same as that from the ideal functionality except with negligible error. For the corrupt parties, in Hyb_1, observe that the values $\{z_{j_1}, \ldots, z_{j_\ell}\}$ are picked by SimHyb to ensure that the output reconstructed by the corrupt parties is same as the value out from the ideal functionality. Hence, the two hybrids are statistically indistinguishable.

Hyb_2**: PRF to random.** In this hybrid, for each fixed-point multiplication, on behalf of the honest parties, for each $j \in [m]$, if $S_j \cap S^* = \emptyset$, SimHyb samples $r_j \leftarrow \mathbb{S}_{2^{k-\log_2(m)}}$ instead of as the output of PRF.

In the multiply phase, for each $j \in [m]$, if $S_j \cap S^* = \emptyset$, in Hyb_1, the honest parties sample the string r_j using the pseudorandom function PRF while in Hyb_2, they are sampled uniformly at random. Since the corresponding PRF key s_j is not used anywhere else in the protocol, it is easy to observe that if there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids with non-negligible probability, we can build a reduction $\mathcal{A}_{\mathsf{PRF}}$ that breaks the security of the pseudorandom function PRF which is a contradiction. Thus, the two hybrids are computationally indistinguishable.

Hyb_3**: PRF to random.** In this hybrid, for each fixed-point multiplication, on behalf of each honest party P_i, the 0-share β_i is computed as in the ideal world. In particular, the terms $\gamma_{i,j}$ are sampled randomly from $\mathbb{S}_{2^k}$ if $P_j \notin S^*$ and not as the output of $\widetilde{\mathsf{PRF}}$.

As in the previous case, the computational indistinguishability between Hyb_2 and Hyb_3 follows from the security of the pseudorandom function $\widetilde{\mathsf{PRF}}$.

Hyb_4**: Switching input.** On behalf of each honest party, in the input sharing step, SimHyb now generates shares of 0 instead of the actual input. This hybrid is identical to the ideal world.

First, it is easy to observe that the joint distribution of the honest parties' outputs and $\mathcal{A}$'s view, at the end of the input sharing phase alone, is identical. This follows from the security of the replicated secret sharing scheme since the adversary corrupts at most $t < n/2$ parties and so learns at most t shares for each honest party input. The same argument can be extended if we additionally include the linear operations.

We now argue that in each fixed point multiplication, the adversary's view remains statistically close. In the multiplication protocol, the value z' is comprised of n values $(z'_1, \ldots, z'_n)$. For each honest party P_i, the z'_i it generates consists of a term of the product $(\sum\limits_{(j,j') \in A_i} x_j y_{j'})$ masked by a value $\beta_i \in \mathbb{S}_{2^k}$. Observe that, aside from the fact that $\mathcal{A}$ knows $\sum_{P_i \in H} \beta_i$ since they form a sharing of 0, each individual share β_i appears uniformly random in $\mathbb{S}_{2^k}$. Therefore, it is easy to argue that each value z'_i reveals no information to $\mathcal{A}$ about the product term masked by β_i. Then, all parties in D_1 learn $z' = x \cdot y + r$. Now, $r = \sum_{j \in [m]} r_j$ where at least one of the r_j values are known only to the honest parties. In particular, for each j such that $S^* \cap S_j = \emptyset$, r_j appears uniformly random in the space $\mathbb{S}_{2^{k'+\lambda}}$ to $\mathcal{A}$. Thus, the product $x \cdot y$ which lies in $\mathbb{S}_{2^{k'}}$ is masked by at least one random string that belongs to a larger space $\mathbb{S}_{2^{k'+\lambda}}$. Thus, we can argue that $\mathcal{A}$ can distinguish $z' = x \cdot y + r$ between the two hybrids only with probability at most $2^{-\lambda}$ and this completes the proof of statistically indistinguishability between the two hybrids.

B Replicated Malicious: Proof

In this section, we prove that the protocol $\Pi_{\mathsf{RShare-MPC-Mal}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ against semi-honest adversaries with an honest majority. The rounding error distribution $\mathcal{E}$ is defined by the random variable $(\sum_{i \in [m]} s_i)/d$ where s_i is uniform over $\mathbb{S}_d$. Formally, we prove the following theorem:

Theorem 3. *Assuming one way functions exist, protocol* $\Pi_{\mathsf{RShare-MPC-Mal}}$ *presented in Fig. 14 securely computes* $\mathcal{F}_{\mathsf{C_d}}$ *(Fig. 2) with rounding error distribution* $\mathcal{E}$ *defined above, in the presence of a malicious adversary that corrupts a set of* $t < n/2$ *parties.*

Proof. Consider an adversary $\mathcal{A}$ that corrupts a set S^* of t parties where $t < n/2$. Let the honest parties be denoted by set H comprising $P_{H_1}, \ldots, P_{H_{n-t}}$. The sim-

ulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{\mathsf{C_d}}$ and the set $(\mathsf{inp}_{i^*}, r_{i^*})_{P_{i^*} \in S^*}$ indicating the corrupt parties' inputs and randomness. The strategy of $\mathcal{S}$ is described below.

Setup: Sample $\alpha^* \in \mathbb{S}_{2^\lambda}$. Run the simulator of the MPC protocol where the parties generate α jointly to force $\alpha = \alpha^*$ and receive $\{[\![\alpha]\!]_i^{\mathsf{R}}\}_{P_i \in H}$ on behalf of the honest parties. Then, on behalf of each honest party, run the setup of protocol $\Pi_{\mathsf{RShare-Mul}}$ in Fig. 8 as in the real world.

Share: On behalf of each honest party P:

1. Sample $x_1, \ldots, x_m \leftarrow \mathbb{S}_{2^k}$ such that $0 = \sum_{i \in [m]} x_i$. Send x_j to $\mathcal{A}$ for each corrupt party P_{i^*} where $i^* \in D_j$.
2. Simulate the MPC protocol used to compute functionality $\mathcal{F}_{\mathsf{Range}}$ (Fig. 15).
3. **Switch PRF to random:** Run the multiplication protocol $\Pi_{\mathsf{RShare-Mul}}$ (Fig. 10, Fig. 11) with the only difference that the PRF values not locally computed by $\mathcal{A}$ are now sampled uniformly at random by $\mathcal{S}$. In more detail,
   - In Step 1 of Fig. 10, on behalf of each honest party P_i, compute terms r'_j as follows: If $P_j \in S^*$, define $r'_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $r'_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$ as in Fig. 10. If $P_j \notin S^*$, and $j > i$, sample $r'_{i,j} \leftarrow \mathbb{S}_{2^k}$ and store the value in $R'[i][j]$. If $P_j \notin S^*$, and $j < i$, look up $R'[i][j]$ and set $r'_{i,j} = -R'[i][j]$.
   - In Step 1 of Fig. 11, for each $j \in (n, m]$, if $S_j \cap S^* = \emptyset$, sample $z_j \leftarrow \mathbb{S}_{2^k}$. Else, set $z_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$ as in Fig. 11.

For each corrupt party P_{i^*}'s input sharing:

1. Using the honest parties' shares $\{[\![x_{i^*}]\!]_i^{\mathsf{R}}\}_{P_i \in H}$ and the simulator of the MPC protocol for $\mathcal{F}_{\mathsf{Range}}$, extract x_{i^*}. If the extraction is unsuccessful, output "Extraction Abort".
2. Query the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ with inputs $\{x_{i^*}\}_{P_{i^*} \in S^*}$ and receive output out.

Linear Operations: These are local operations that don't need to be simulated.

Fixed point multiplication: For each multiplication, for any two input shares $[\![x]\!]^{\mathsf{R}}, [\![\alpha x]\!]^{\mathsf{R}}, [\![y]\!]^{\mathsf{R}}$, as in the semi-honest case, the only change is the PRF values not locally computed by $\mathcal{A}$ are now sampled uniformly at random by $\mathcal{S}$. In more detail, $\mathcal{S}$ does the following:

1. Recall that S_j is the set of parties indexed by D_j. For each $j \in [m]$, if $S_j \cap S^* = \emptyset$, sample $r_j \leftarrow \mathbb{S}_{2^{k-\log_2(m)-\lambda}}$. Else, set $r_j := \mathsf{PRF}_{s_j}(\mathsf{cnt})$ as in Fig. 12. Compute $r'_j = r_j/d$ and run the multiplication protocol (Fig. 10, Fig. 11) to compute $[\![\alpha r]\!]^{\mathsf{R}}, [\![\alpha r']\!]^{\mathsf{R}}$ with the same changes as listed above in Step 3.
2. For each honest party P_i, compute $\beta_i := \sum_{j \in [n]/i} \gamma_{i,j}$ where $\gamma_{i,j}$ is computed as follows:
   - If $P_j \in S^*$, define $\gamma_{i,j} := \widetilde{\mathsf{PRF}}_{s_{j,i}}(\mathsf{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\mathsf{PRF}}_{s_{i,j}}(\mathsf{cnt})$ for $j \in (i, n]$ as in Fig. 12.

– If $P_j \notin S^*$, and $j > i$, sample $\gamma_{i,j} \leftarrow \mathbb{S}_{2^k}$ and store the value in $\Gamma[i][j]$. If $P_j \notin S^*$, and $j < i$, look up $\Gamma[i][j]$ and set $\gamma_{i,j} = -\Gamma[i][j]$.

3. Run Step 3 to Step 6 as in Fig. 12.
4. **MAC Check Failure:** From the knowledge of all shares of $[\![\alpha]\!]^{\mathsf{R}}$, $[\![x]\!]^{\mathsf{R}}$, $[\![y]\!]^{\mathsf{R}}$, $[\![r]\!]^{\mathsf{R}}$ and the PRF keys, compute the values $\{z'_{i^*}\}_{P_{i^*} \in S^*}$ to be sent by the adversary. Output "MAC Abort" if, in Step 5, $\mathcal{F}_{\mathsf{MAC.Check}}$ does not abort but the values sent by $\mathcal{A}$ are inconsistent with those calculated above $\{z'_{i^*}\}_{P_{i^*} \in S^*}$.

Output reconstruction:

1. Set $z^* = 0$. For each $j \in [m]$: note that since $S_j \setminus S^* \neq \emptyset$, i.e, each S_j has at least one honest party, $\mathcal{S}$ knows z_j. Update $z^* = z^* + z_j$ if $S_j \cap S^* \neq \emptyset$.
2. Let $j_1, \ldots, j_\ell \in [m]$ denote the indices for which $S_j \cap S^* = \emptyset$.
3. Pick $z_{j_1}, \ldots, z_{j_{\ell-1}} \leftarrow \mathbb{S}_{2^k}$. Compute $z_{j_\ell} \in \mathbb{S}_{2^k}$ as $z_{j_\ell} = \mathsf{out} - \sum_{i \in [\ell-1]} z_{j_i} - z^*$.
4. On behalf of the honest parties, send their shares of z as in the real protocol.
5. Receive z_j values from $\mathcal{A}$. As in the real protocol, if $\mathcal{A}$ sent inconsistent values to any honest party P_i, instruct the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ to deliver output "Abort" to that honest party. Else, instruct $\mathcal{F}_{\mathsf{C_d}}$ to deliver the correct output out.

We now show that the above simulation strategy is successful via a hybrid argument.

Hyb_0**:** Consider a simulator SimHyb that plays the role of the honest parties as in Fig. 14. This is the real world.

Hyb_1**: Random α.** In the setup phase, sample $\alpha^* \in \mathbb{S}_{2^\lambda}$. Run the simulator of the MPC protocol where the parties generate α jointly to force $\alpha = \alpha^*$.

Indistinguishability of Hyb_0 and Hyb_1 is implied by the security of the MPC protocol used to jointly generate α.

Hyb_2**: Input extraction.** In the input sharing phase, SimHyb runs step Step 1 as done in the ideal world. That is, using the honest parties' shares $\{[\![x_{i^*}]\!]_i^{\mathsf{R}}\}_{P_i \in H}$ and the simulator of the MPC protocol for $\mathcal{F}_{\mathsf{Range}}$, extract $\{x_{i^*}\}_{* \in S^*}$ and query $\mathcal{F}_{\mathsf{C_d}}$ to learn output out. If the extraction is unsuccessful, output "Extraction Abort".

From the security of the MPC protocol for $\mathcal{F}_{\mathsf{Range}}$, the probability that the Simulator of MPC protocol for $\mathcal{F}_{\mathsf{Range}}$ aborts without successfully extracting the inputs but in the real execution, the honest parties don't abort is negligible. As a result, the probability that SimHyb outputs "Extraction Abort" in Hyb_2 is negligible. Apart from that, notice that the distributions produced by both Hyb_1 and Hyb_2 are the same. Hence, the two hybrids are indistinguishable.

Hyb_3**: Switching output.** SimHyb now runs the output reconstruction step as done by $\mathcal{S}$ in the ideal world by using the output out received from $\mathcal{F}_{\mathsf{C_d}}$.

Honest parties also get their output from $\mathcal{F}_{\mathsf{C_d}}$ in this hybrid.

The only difference between Hyb_2 and Hyb_3 is the manner in which parties learn their output. For the corrupt parties, in Hyb_3, observe that the values $\{z_{j_1}, \ldots, z_{j_\ell}\}$ are picked by SimHyb to ensure that the output reconstructed by the corrupt parties is same as the value out from the ideal functionality. In the real world, correctness of the protocol is easy to observe (as in the semi-honest case). Finally, observe that as in the real world, if SimHyb detects any inconsistent shares sent by $\mathcal{A}$ to any honest party, it instructs $\mathcal{F}_{\mathsf{C_d}}$ to deliver output "Abort" to that honest party. Thus, as in the semi-honest case, it is easy to observe that the real world output is same as that from the ideal functionality except with negligible error. Hence, the two hybrids are statistically indistinguishable.

Hyb_4**: PRF to random.** In the multiplication protocols, all the pseudorandom function outputs not locally computed by $\mathcal{A}$ are now sampled uniformly at random as done by $\mathcal{S}$ in the ideal world.

As in the semi-honest protocol, it is easy to see that the computational indistinguishability between Hyb_3 and Hyb_4 follows from the security of the pseudorandom functions PRF and $\widetilde{\mathsf{PRF}}$.

Hyb_5**: MAC Check.** SimHyb runs Step 4 as in the ideal world and outputs "MAC Abort" if $\mathcal{A}$ successfully sends incorrect shares of z' while making sure $\mathcal{F}_{\mathsf{MAC.Check}}$ succeeds.

Since the MAC key α appears uniformly random to $\mathcal{A}$, the security of $\mathcal{F}_{\mathsf{MAC.Check}}$ guarantees that the probability that $\mathcal{F}_{\mathsf{MAC.Check}}$ succeeds while $\mathcal{A}$ successfully sends incorrect shares of z' is negligible. As a result, the probability that SimHyb outputs "MAC Abort" in Hyb_4 is negligible and the two hybrids are indistinguishable.

Hyb_6**: Simulate $\mathcal{F}_{\mathsf{Range}}$.** In the input sharing step, simulate the MPC protocol used to compute functionality $\mathcal{F}_{\mathsf{Range}}$.

Indistinguishability of Hyb_6 and Hyb_7 follows from the security of the MPC protocol used to compute $\mathcal{F}_{\mathsf{Range}}$.

Hyb_7**: Switching input.** On behalf of each honest party, in the input sharing step, SimHyb now generates shares of 0 instead of the actual input. This hybrid is identical to the ideal world.

The argument is similar to that in the semi-honest protocol. The only difference here is that, unlike the semi-honest protocol, $\mathcal{A}$ does not necessarily follow the protocol honestly. However, the below two things guarantee that any deviation from honest behavior by $\mathcal{A}$ will be detected by the honest parties

with overwhelming probability without revealing anything about the input: (i) In the fixed point multiplication protocol, the MAC check (functionality $\mathcal{F}_{\mathsf{MAC.Check}}$) guarantees that every share of z' is correctly generated by $\mathcal{A}$. (ii) Apart from the shares of z' computed in the fixed point multiplication, for any other value $[\![x]\!]^{\mathsf{R}}$ computed as part of the circuit evaluation, every share of $[\![x]\!]^{\mathsf{R}}$ is held by at least one honest party - this is because $t < n/2$ and so $(n-t) > t$. Thus, the two hybrids are statistically indistinguishable and this completes the proof.

C Shamir: Proof

We now show that the simulation strategy in Sect. 7.3 is successful via a hybrid argument.

Hyb_0**:** $\mathcal{A}$'s view and the honest parties' output in the real world.

Hyb_1**:** Same as Hyb_0 except that in the final step to reconstruct the output, the simulator does the following on behalf of the honest parties: if P_1 is an honest party, then let $z := \mathsf{out}$ from the ideal functionality and send it to $\mathcal{A}$ on behalf of P_1; otherwise, let the shares of the honest parties be random such that the reconstructed $z = \mathsf{out}$, and send these shares to P_1 on behalf of the honest parties. In the meanwhile, replace the honest parties' output in the real world by their output in the ideal world.

Hyb_0 and Hyb_1 are statistically identical, which follows from the correctness of the protocol. Since there is at least one share held by honest parties, it is statistically identical to $\mathcal{A}$ if the honest parties sample their shares to be consistent with the output.

Hyb_2**:** Same as Hyb_1 but for each fixed point multiplication by truncation, the simulator manipulates the honest parties' shares to be random such that $z' = x \cdot y + r$.

This hybrid is statistically identical to Hyb_1 because there is at least one r_i contributed by honest parties in preprocessing. It is thus statistically identical to $\mathcal{A}$ if the honest parties randomly sample their shares to be consistent with z'.

Hyb_3**:** Same as Hyb_2 but for each fixed point multiplication by truncation, the simulator manipulates the honest parties' shares to be random such that $z' = r$ instead of $x \cdot y + r$.

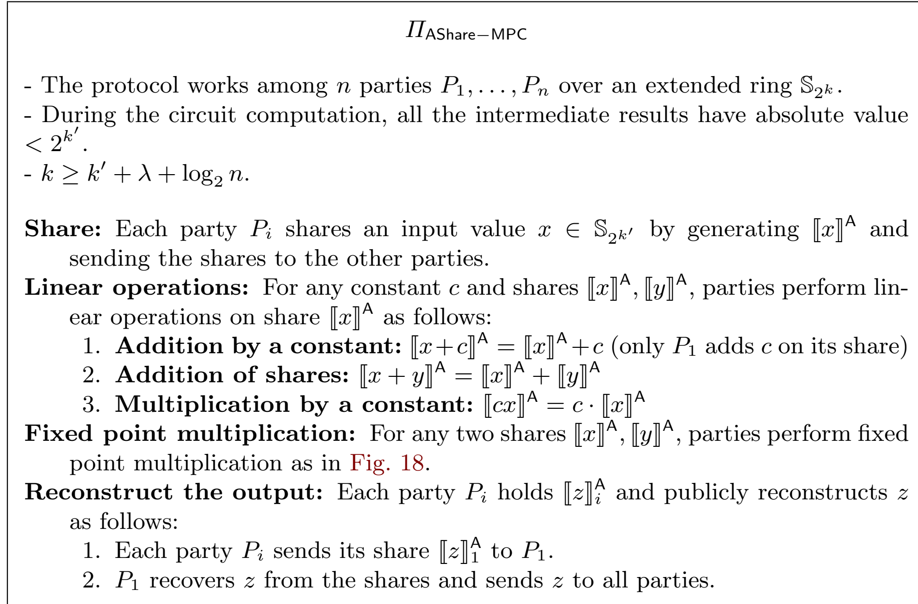
Since $x \cdot y \in \mathbb{S}_{2^{k'}}$, and there is at least one r_i (sampled from $\mathbb{S}_{2^{k'+\lambda}}$) contributed by honest parties in preprocessing, the distribution of r and $x \cdot y + r$ are statistically close. Hence this hybrid is statistically indistinguishable from

Hyb_2.

Hyb_4**:** Same as Hyb_3 except that on behalf of each honest party P_{H_i}, the simulator sends random shares to the corrupt parties for its input.

This hybrid is statistically identical to Hyb_3, which follows from the security of Shamir secret sharing. This hybrid outputs the simulated view along with the honest parties' output in the ideal world, which concludes the proof.

D Additive: Proof



$\Pi_{\mathsf{AShare-MPC}}$

- The protocol works among n parties $P_1, \ldots, P_n$ over an extended ring $\mathbb{S}_{2^k}$.
- During the circuit computation, all the intermediate results have absolute value $< 2^{k'}$.
- $k \geq k' + \lambda + \log_2 n$.

Share: Each party P_i shares an input value $x \in \mathbb{S}_{2^{k'}}$ by generating $[\![x]\!]^{\mathsf{A}}$ and sending the shares to the other parties.

Linear operations: For any constant c and shares $[\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}}$, parties perform linear operations on share $[\![x]\!]^{\mathsf{A}}$ as follows:

1. **Addition by a constant:** $[\![x+c]\!]^{\mathsf{A}} = [\![x]\!]^{\mathsf{A}} + c$ (only P_1 adds c on its share)
2. **Addition of shares:** $[\![x+y]\!]^{\mathsf{A}} = [\![x]\!]^{\mathsf{A}} + [\![y]\!]^{\mathsf{A}}$
3. **Multiplication by a constant:** $[\![cx]\!]^{\mathsf{A}} = c \cdot [\![x]\!]^{\mathsf{A}}$

Fixed point multiplication: For any two shares $[\![x]\!]^{\mathsf{A}}, [\![y]\!]^{\mathsf{A}}$, parties perform fixed point multiplication as in Fig. 18.

Reconstruct the output: Each party P_i holds $[\![z]\!]_i^{\mathsf{A}}$ and publicly reconstructs z as follows:

1. Each party P_i sends its share $[\![z]\!]_1^{\mathsf{A}}$ to P_1.
2. P_1 recovers z from the shares and sends z to all parties.

Fig. 19: Protocol for MPC with additive secret sharing.

In this section, we prove that the protocol $\Pi_{\mathsf{AShare-MPC}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{C_d}}$ against semi-honest adversaries corrupting $t < n$ parties. The rounding error distribution $\mathcal{E}$ is defined by the random variable $(\sum_{i \in [t+1]} s_i)/d$ where s_i is uniform over $\mathbb{S}_d$. Formally, we prove the following theorem:

Theorem 4. *Given preprocessed Beaver triples, the protocol $\Pi_{\mathsf{AShare-MPC}}$ presented in Fig. 19 statistically securely realizes the functionality $\mathcal{F}_{\mathsf{C_d}}$ (defined in Fig. 2) with rounding error distribution $\mathcal{E}$ in the presence of a semi-honest adversary that corrupts $t < n$ parties.*

Proof. Correctness of the protocol follows from the invariant that after every linear operation or fixed point multiplication, parties hold an additive sharing of the output. This is straightforward for linear operations. For fixed point multiplication by truncation, our protocol is standard multiplication with Beaver triples followed by a truncation using $[\![r]\!]^{\mathsf{A}}$ and $[\![r']\!]^{\mathsf{A}}$. The invariant relies on the fact that $(\sum_{i\in[t+1]} r_i)/d \approx \sum_{i\in[t+1]} r_i/d$. Note that each $|r_i| < 2^{k'+\lambda}$, hence $\sum_{i\in[t+1]} |r_i| < 2^{k'+\lambda} \cdot (t+1)$, which suggests that $\sum_{i\in[t+1]} r_i$ does not overflow. Therefore $(\sum_{i\in[t+1]} r_i)/d \approx \sum_{i\in[t+1]} r_i/d$, where the rounding error follows the distribution $\mathcal{E}$ defined above.

For security, consider an adversary $\mathcal{A}$ that corrupts a set C of t parties, denoted as $P_{C_1}, \ldots, P_{C_t}$. Let the honest parties be denoted by set H comprising $P_{H_1}, \ldots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{\mathsf{C_d}}$ and the set $\{(\mathsf{inp}_{C_i}, r_{C_i})\}_{i\in[t]}$ indicating the corrupt parties' inputs and randomness. We construct a PPT simulator $\mathcal{S}$ that follows the protocol description of the corrupted parties to generate $\mathcal{A}$'s view and does the following on behalf of honest parties:

Share: On behalf of each honest party P_{H_i}:

1. Sample $s_1, \ldots, s_t \xleftarrow{\$} \mathbb{S}_{2^k}$.
2. Send s_j to $\mathcal{A}$ for each corrupt party P_{C_j}.

Linear operations: These are local operations that don't need to be simulated.

Fixed point multiplication: For each fixed point multiplication with truncation:

- **Preprocessing:**
  1. For each corrupt party P_{C_i} where $C_i \in [t+1]$, sample $r_{C_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$.
  2. On behalf of each honest party P_{H_i} where $H_i \in [t+1]$, sample $r_{H_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$, generate $[\![r_{H_i}]\!]^{\mathsf{A}}, [\![r_{H_i}/d]\!]^{\mathsf{A}}$, and send the shares to the corrupted parties.
  3. Let $r := \sum_{i\in[t+1]} r_i$.
- **Online multiplication:**
  - If P_1 is an honest party, then sample $x', y' \xleftarrow{\$} \mathbb{S}_{2^k}$ and let $z' := r$; send x', y', z' to $\mathcal{A}$ on behalf of P_1.
  - If P_1 is a corrupt party, then let the shares of $[\![x']\!]^{\mathsf{A}}$ and $[\![y']\!]^{\mathsf{A}}$ of the honest parties be random and send them to P_1 on behalf of them. Next, compute the shares of $[\![z']\!]^{\mathsf{A}}$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z' = r$, and send these shares to P_1 on behalf of the honest parties.

Reconstruct the output:

- If P_1 is an honest party, then let $z := \mathsf{out}$ and send it to $\mathcal{A}$ on behalf of P_1.
- If P_1 is a corrupt party, then compute the shares of $[\![z]\!]^{\mathsf{A}}$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z = \mathsf{out}$, and send these shares to P_1 on behalf of the honest parties.

We now show that the above simulated view together with the honest parties' output in the ideal world is statistically indistinguishable from $\mathcal{A}$'s view and the honest parties' output in the real world via a hybrid argument.

Hyb_0**:** $\mathcal{A}$'s view and the honest parties' output in the real world.

Hyb_1**:** Same as Hyb_0 except that in the final step to reconstruct the output, the simulator does the following on behalf of the honest parties: if P_1 is an honest party, then let $z := \mathsf{out}$ from the ideal functionality and send it to $\mathcal{A}$ on behalf of P_1; otherwise, let the shares of the honest parties be random such that the reconstructed $z = \mathsf{out}$, and send these shares to P_1 on behalf of the honest parties. In the meanwhile, replace the honest parties' output in the real world by their output in the ideal world.

Hyb_0 and Hyb_1 are statistically identical, which follows from the correctness of the protocol. Since there is at least one share held by honest parties, it is statistically identical to $\mathcal{A}$ if the honest parties sample their shares to be consistent with the output.

Hyb_2**:** Same as Hyb_1 but for each fixed point multiplication by truncation, the simulator manipulates the honest parties' shares of z' to be random such that $z' = x \cdot y + r$.

This hybrid is statistically identical to Hyb_1 because there is at least one r_i contributed by honest parties in preprocessing. It is thus statistically identical to $\mathcal{A}$ if the honest parties randomly sample their shares to be consistent with z'.

Hyb_3**:** Same as Hyb_2 but for each fixed point multiplication by truncation, the simulator manipulates the honest parties' shares of z' to be random such that $z' = r$ instead of $x \cdot y + r$.

Since $x \cdot y \in \mathbb{S}_{2^{k'}}$, and there is at least one r_i (sampled from $\mathbb{S}_{2^{k'+\lambda}}$) contributed by honest parties in preprocessing, the distribution of r and $x \cdot y + r$ are statistically close. Hence this hybrid is statistically indistinguishable from Hyb_2.

Hyb_4**:** Same as Hyb_2 but for each fixed point multiplication by truncation, the simulator manipulates the honest parties' shares of x' and y' to be random.

First, $x' = x + \alpha$ is uniformly random in $\mathbb{S}_{2^k}$ as $\alpha \xleftarrow{\$} \mathbb{S}_{2^k}$. Since there is at least one share of α held by honest parties, the distribution of x' is statistically identical if the honest parties sample their shares randomly. For the same reason, the distribution of y' is statistically identical if the honest parties sample their shares randomly. Hence this hybrid is statistically identical from Hyb_3.

Hyb_5: Same as Hyb_4 except that on behalf of each honest party P_{H_i}, the simulator sends random shares to the corrupt parties for its input.

This hybrid is statistically identical to Hyb_4, which follows from the security of additive secret sharing. This hybrid outputs the simulated view along with the honest parties' output in the ideal world, which concludes the proof.