

MULTI-PARTY COMPUTATION FOR MANY COMPUTERS

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 63/192,933, filed May 25, 2021, which is herein incorporated by reference in its entirety for all purposes.

BACKGROUND

[0002] Machine learning models can be used for a wide range of applications. In many instances, to train and evaluate machine learning models, large volumes of data may be obtained from various data sources. However, in many instances, portions of received data can include sensitive or private information.

[0003] Computers (e.g., servers) performing secret share of data can use a multi-party computation technique to train and evaluate machine learning models using the jointly-obtained data. The multi-party computation technique can allow for retrieval of the data for evaluating and training of the machine learning models without requiring the dissemination of any sensitive/private information across the nodes.

SUMMARY

[0004] One embodiment of the present disclosure is directed to a method performed by a first computer of a plurality of computer for implementing a replication protocol for multi-party fixed point multiplication. The first computer can obtain a first secret share of a first fixed value (x) and a second secret share of a second fixed value (y). The first computer can then jointly sample a first pseudo-random key ($s_{i,j}$) with another computer paired with the first computer and a second pseudo-random key (s_j) with a first set (D_j) of multiple sets in the plurality of computers. The first computer can generate a replicated random share of a random value ($[[r]]^R$) using the second pseudo-random key (s_j) and then generate a replicated sharing value (β) using the first pseudo-random key ($s_{i,j}$). The first computer can compute a masked share (z'_i) by providing a product of the first secret share and the second secret share masked with the replicated sharing value (β) and the replicated random shares of the random value ($[[r]]^R$). The first computer can receive, from computers in the first set (D_j), a plurality of masked shares (z'_i), thereby revealing

a set shared value z' to the first set (D_j). The first computer can then generate a replicated secret share ($[[z]]^R$) using the received plurality of masked shares (z'_i) and the replicated random shares of the random value ($[[r]]^R$).

[0005] Another embodiment of the present disclosure is directed to a method performed by a first computer of a plurality of computers for implementing a sharing protocol for multi-party fixed point multiplication. The first computer can obtain a secret share of a first value (x), a secret share of a second value (y). The first computer can then sample a first random value (r_i) to obtain a first random share ($[[r_i]]^{S, 2t}$) of the first random value (r_i) and a second random share ($[[r_i/d]]^S$). The first computer can send the first random share ($[[r_i]]^{S, 2t}$) and the second random share $[[r_i/d]]^S$ to the plurality of computers. The first computer can also obtain each share of the first random share ($[[r]]^{S, 2t}$) and each share of the second random share $[[r/d]]^S$ from the plurality of computers. The first computer can then compute a third random share ($[[r]]^{S, 2t}$) and a fourth random share ($[[r']]^S$) based on each share of the first random share ($[[r]]^{S, 2t}$) and each share of the second random share $[[r/d]]^S$, respectively. The first computer can compute a masked share ($[[z']]^{S, 2t}$) that provides a product of the secret share of the first value (x) and the secret share of the second value (y) masked with the third random share ($[[r]]^{S, 2t}$). The first computer can then obtain a masked value z' based on a plurality of masked shares ($[[z']]^{S, 2t}$) determined by the plurality of computers and then generate an output secret share ($[[z]]^S$) using the masked value z' and the fourth random share ($[[r']]^S$).

[0006] Another embodiment of the present disclosure is directed to a method performed by a first computer of a plurality of computers for implementing an additive sharing protocol for multi-party fixed point multiplication. The first computer can obtain a secret share ($[[x]]^A$) of a first value (x), a secret share ($[[y]]^A$) of a second value (y), a decimal bit value (d), a secret share ($[[\alpha]]^A$) of a third value (α), a secret share ($[[\beta]]^A$) of a fourth value (β), and a secret share ($[[\gamma]]^A$) of a fifth value (γ). The first computer can determine determining a share of a first shared random value ($[[r']]^A$) and a share of a second shared random value ($[[r]]^A$). The first computer can determine a first intermediate share ($[[x']]^A$) based on the secret share ($[[x]]^A$) of the first value (x) and the secret share ($[[\alpha]]^A$) of the third value (α) and then determine a second intermediate share ($[[y']]^A$) based on the secret share ($[[y]]^A$) of the second value (y) and the secret share ($[[\beta]]^A$)

of the fourth value (β). The first computer can receive a plurality of first intermediate shares and a plurality of second intermediate shares from the plurality of computers to reveal a first intermediate value (x') and a second intermediate value (y'). The first computer can send the first intermediate value (x') and the second intermediate value (y') to the plurality of computers. The first computer can then determine a third intermediate share ($[[z']]^A$) based on the first intermediate value (x'), the second intermediate value (y'), the secret share ($[[x]]^A$) of the first value (x), the secret share ($[[y]]^A$) of the second value (y), the secret share ($[[\gamma]]^A$) of the fifth value (γ) and the share of the second shared random value ($[[r]]^A$). The first computer can receive a plurality of third intermediate shares from the plurality of computers to reveal a third intermediate value (z'). The first computer can send the third intermediate value (z') to the plurality of computers. The first computer can then determine determining an output secret share ($[[z]]^A$) based on the third intermediate value (z'), the decimal bit value (d), and the share of the first shared random value ($[[r']]^A$).

[0007] These and other embodiments of the disclosure are described in detail below. For example, other embodiments are directed to systems, devices, and computer readable media associated with methods described herein.

[0008] A better understanding of the nature and advantages of embodiments of the present disclosure may be gained with reference to the following detailed description and the accompanying drawings.

[0009] Reference to the remaining portions of the specification, including the drawings and claims, will realize other features and advantages of the present disclosure. Further features and advantages of the present disclosure, as well as the structure and operation of various embodiments of the present disclosure, are described in detail below with respect to the accompanying drawings. In the drawings, like reference numbers can indicate identical or functionally similar elements.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] FIG. 1 shows a multiparty fixed point multiplication system according to an embodiment of the present disclosure.

[0011] FIG. 2 shows a computer system comprising sets of computers according to an embodiment of the present disclosure.

[0012] FIG. 3 is an example signaling process for a replication protocol for multi-party fixed point multiplication in a semi-honest setting according to an embodiment of the present disclosure.

[0013] FIG. 4 is a block diagram illustrating an example replication protocol for multi-party fixed point multiplication in a malicious setting according to an embodiment of the present disclosure.

[0014] FIG. 5 is a block diagram of an example Shamir sharing protocol for multi-party fixed point multiplication according to an embodiment of the present disclosure.

[0015] FIG. 6 is a block diagram of an example additive sharing protocol for multi-party fixed point multiplication according to an embodiment of the present disclosure.

[0016] FIG. 7 shows a block diagram of an example computer system usable with systems and methods according to embodiments of the present disclosure.

TERMS

[0017] Prior to discussing embodiments of the invention, description of some terms may be helpful in understanding embodiments of the invention.

[0018] The term “server computer” may include a powerful computer or cluster of computers. For example, the server computer can be a large mainframe, a minicomputer cluster, or a group of computers functioning as a unit. In one example, the server computer may be a database server coupled to a web server. The server computer may be coupled to a database and may include any hardware, software, other logic, or combination of the preceding for servicing the requests from one or more other computers. The term “computer system” may generally refer to a system including one or more server computers, which may be coupled to one or more databases.

[0019] A “machine learning model” can refer to a set of software routines and parameters that can predict an output(s) of a real-world process (e.g., a diagnosis or treatment of a patient, identification of an attacker of a computer network, authentication of a computer, a suitable recommendation based on a user search query, etc.) based on a set of input features. A structure of the software routines (e.g., number of subroutines and relation between them) and/or the values of the parameters can be determined in a training process, which can use actual results of the real-world process that is being modeled.

[0020] The term “training computer” can refer to any computer that is used in training the machine learning model. As examples, a training computer can be one of a set of client computers from which the input data is obtained, or a server computer that is separate from the client computers.

[0021] “Multi-party computation” (MPC) may refer to a computation that is performed by multiple parties. Each party, such as a computer, server, or cryptographic device, may have some inputs to the computation. Each party can collectively calculate the output of the computation using the inputs.

[0022] “Secure multi-party computation” (secure MPC) may refer to a multi-party computation that is secure. In some cases, “secure multi-party computation” refers to a multi-party computation in which the parties do not share information or other inputs with each other. Determining a PSI can be accomplished using a secure MPC. The inputs from each computer in an MPC may be secret shares of a value that is known to one computer. For example, a first computer can split a database value into multiple secret shares and distribute the shares among the computers. Only the first computer may know how to assemble the shares to reconstruct the database value. One way to generate the shares is by selecting a random number and then taking an XOR of the random number and a value of an individual database table, which can be done independently for each value in a database.

[0023] The term “secret-sharing” can refer to any one of various techniques that can be used to store a data item on a set of training computers such that each training computer cannot determine the value of the data item on its own. As examples, the secret-sharing can involve

splitting a data item up into shares that require a sufficient number (e.g., all) of training computers to reconstruct and/or encryption mechanisms where decryption requires collusion among the training computers.

[0024] The term “secret” can include a value not known or seen or not meant to be known or seen by others. A secret can be a value that is intended to remain only known by a computer, entity, group, etc. A secret can be shared securely in an obfuscated type manner using a secret-sharing technique to create a plurality of secret shares.

[0025] The term “secret share” can include portion of a value not known or seen or not meant to be known or seen by others. A secret share, in isolation, may not reveal information regarding the secret. A secret share can be created using a secret sharing technique. The combination of all, or a portion of all, of the secret shares can reveal the secret.

[0026] The term “set” can refer to a group of things. A set of computers can be a group of computers within an overall network of computers. A network of computers can include any number of sets of computers. In some cases, a computer can belong (e.g., be a part of) to one or more sets of computers.

[0027] The term “set shared value” can include a value held by entities in a set. A set shared value can be a value that is known by each computer in a set of computers, but not by computers in a computer network that are not in the set of computers.

DETAILED DESCRIPTION

[0028] Machine learning is used to produce models that classify images, perform medical diagnosis, provide recommendations, and identify fraudulent transactions among several other applications. Cloud-based machine learning services [1-4] can perform both training models using user 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. 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.

[0029] Secure multi-party communication (MPC) can be used for transmitting data for training and/or evaluating machine learning models while retaining privacy of the data. Secure MPC can ensure that, during training, the only information leaked about the data may be the final model (or an encrypted version of the model). Further, during evaluation, the only information leaked can include the output label. One setting considered is where data owners (e.g., clients) secret share their data among multiple independent servers that perform training on the combined data or apply a (secret shared) pre-trained model to evaluate new data samples. There have been several 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 where an adversary can corrupt at most one of them. In a context where multiple parties are interacting (e.g., greater than three parties), there is a gap between plaintext training/evaluation and privacy-preserving solutions for secure MPC.

[0030] The present embodiments can provide multiple forms of secret sharing to represent input to be multiplied and the output. For example, embodiments provide for a replicated secret sharing protocol, a Shamir secret sharing protocol, and an additive secret sharing protocol. The protocols as described herein can relate to a scenario where there are more than three parties and a number of the parties are honest (e.g., can be trusted to implement a protocol for secure MPC).

[0031] In the case of many machine learning algorithms, both training data and the intermediate parameters may include decimal values that cannot be natively handled using modular arithmetic. Many approaches may fail when performing floating point multiplications, and a large modulus can imply a more complex multiplication that can reduce performance. Embodiments described herein can provide efficient multi-party fixed point multiplication where the inputs and outputs are secret amongst interacting parties. Embodiments provide a technical solution to, at least, the technical problem of how to efficiently perform multi-party fixed point multiplication methods where the inputs and outputs are secret shared amongst a plurality of computers.

[0032] Further, the protocols as described herein can include efficient multi-party fixed point multiplication protocols where one or more parties can include covert adversaries that may not follow the protocol. In such instances, embodiments can detect malicious behavior by the covert adversarial parties by invoking the multiplication protocols as described herein.

I. SECURE MULTI-PARTY COMMUNICATION AND SECRET SHARING OVERVIEW

[0033] Embodiments provide for a method for truncating secret shared values, for example, by computing $\llbracket y \rrbracket := \llbracket x \rrbracket / 2^d$ where d is a public value that denotes a number of decimal bits in a fixed-point value. The aforementioned truncation method can then be combined with multiplication protocol(s) to obtain a fixed-point multiplication protocol(s). Embodiments provide for three forms of secret sharing to represent the inputs to be multiplied and the output: replicated, Shamir, and additive secret sharing. However, it is understood that embodiments can apply to various other secret sharing schemes.

A. *Replicated Secret Sharing*

[0034] The present embodiments provide multiple replicated protocols for multi-party fixed point multiplication where the inputs and output are represented using replicated secret sharing over modulus 2^k , where k is the bit length of the shares.

[0035] The first protocol is secure against a semi-honest adversary in the presence of an honest majority and does not utilize offline communication prior to the protocol. The protocol has two forms according to embodiments: 1) a single round protocol utilizing $(n^2 - nt)k$ bits of online communication and 2) two rounds utilizing only $2nk$ bits of online communication, where n is the number of computers (also referred to as parties) and $t < n/2$ is the number of corrupt computers. The communication overhead of such methods is significantly better than the previous work of Mohassel and Rindal [36] for multi-party fixed point multiplication in the same setting. The communication cost is compared in Table 1, below.

[0036] The second protocol can require two rounds in the online phase and can be secure in the presence of a covert adversary (e.g., a party that can behave arbitrarily maliciously).

B. *Shamir Secret Sharing*

[0037] Embodiments provide for an efficient multi-party fixed point multiplication protocol where the inputs and output are represented using Shamir secret sharing. The protocol utilizes 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,

according to embodiments the online communication overhead is the same while offline communication is significantly lower since $n, t \ll k$ typically in practice (see Table 1).

C. Additive Secret Sharing

[0038] The aforementioned approaches for replicated and Shamir secret sharing can occur in a setting where an honest majority is present among the computers. This is inherently required by these secret sharing schemes.

[0039] However, it is understood that it is not a requirement of the truncation method according to embodiments. In fact, the truncation protocol can be viewed as a technique that works for even for dishonest majority. This can be demonstrated by utilizing the truncation method to create 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. Such a method can be a four-round protocol, which is secure against a semi-honest adversary that can corrupt any $t < n$ parties.

D. Performance and General MPC

[0040] The communication and computation cost of the three aforementioned protocols is close to a standard secret shared modular multiplication. For replicated secret sharing the protocol is as computationally complex as the modular counter-part. The same holds for Shamir for a dishonest majority.

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

Table 1: Communication (in bits) and round complexity of embodiments compared to [16, 36]. n is the number of parties, $t < n/2$ is the number of corrupt parties. 2^k is the modulus for

replicated secret sharing. $k = \lfloor \log_2(q) \rfloor$ where q is the prime modulus is for Shamir secret sharing.

[0041] For each form of secret sharing, embodiments can leverage the fixed-point multiplication protocol to design an MPC protocol for computing arbitrary arithmetic circuits that contain addition and fixed-point multiplication with truncation gates. It can then be proved that the resulting MPC protocol is secure via a standard real world-ideal world simulation based security definition.

II. PRELIMINARIES

[0042] Prior to describing further details of secret sharing methods, a notational base will be described that can aid in the understanding and description of embodiments.

A. Notation

[0043] A computational security parameter can be referred to using κ and a statistical security parameter can be referred to using λ . The parties (also referred to as nodes or computers) are enumerated as $P_1, \dots, P_n$, where there are a total of n parties. The notation $\llbracket x \rrbracket$ denotes a secret sharing of a value x . This sharing can be one of several different types - $\llbracket x \rrbracket^R$, $\llbracket x \rrbracket^S$ or $\llbracket x \rrbracket^A$ as described in further detail below. Regardless of the sharing type, the notation of $\llbracket x \rrbracket_i$ will refer to the share of the value x held by the party i , denoted as P_i .

[0044] The assignation of a variable can be denoted by $:=$, for example a variable a can be assigned to the value of b by $a := b$. When uniformly sampling from a set S or sampling a randomized function f , embodiments use the notation $a \leftarrow S$ and $a \leftarrow f(\dots)$ respectively. Let $[a, b]$ denote the set $\{a, a + 1, \dots, b\}$ and the shorthand $[b] := [1, b]$. A parentheses on the left or right side denotes that corresponding side of the range is exclusive, such as $(a, b] := \{a + 1, a + 2, \dots, b\}$. Discussion of the definitions of pseudorandom functions (PRFs), which are used as one of the cryptographic primitives utilized in the constructions herein, can be found in [27].

B. Computation Representations

[0045] In secure computation, functions can be 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 Booleans and $N =$

2^k or a prime q for arithmetic operations. According to embodiments, arithmetic circuits can be considered with $N = 2^k$ or a prime q . In the case of a prime modulus q , $k := \lfloor \log_2(q) \rfloor$ can be defined 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.

[0046] For signed values, a value $x \in [-N/2, N/2)$ can be represented as $\hat{x} := (x \bmod N)$. For $N = 2^k$, this is also known as two's complement, where the highest order bit indicates sign (e.g., positive or negative). The same approach also works when N is a prime. Sometimes it can be more convenient to consider a signed value as simply an element of $\mathbb{Z}_N$ (integer numbers set), while at other times $x \in [-N/2, N/2)$ is more natural. In particular, embodiments can 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. The symmetric mod operator $x' := (x \text{ symod } N)$ can be defined as the unique value $x' \in \mathbb{S}_N$ such that $(x \bmod N) = (x' \bmod N)$.

[0047] 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 $x = \sum 2^{i-d} x_i$. A set of fixed point values can be parameterized by two integers N, d . The set of (signed) values can then be described as $Fx_{N,d} := \{x/2^d \mid x \in \mathbb{S}_N\}$ where the division is performed over $\mathbb{Q}$ and it is assumed that $N > 2^d$. Addition and subtraction can be defined in the natural/standard way. For multiplication of $x, y \in Fx_{N,d}$, the result can be defined as $xy \in \mathbb{Q}$ rounded down to the next multiple of 2^{-d} .

C. Security Model

[0048] Consider a ring of values $\mathbb{S}_N$ and a value k' where $2^{k'} \ll N$. Consider any arithmetic circuit 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 input values 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 $\text{inp}_i \in \mathbb{S}_{2^{k'}}$ and their goal may be to evaluate the circuit 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 C_d , the inputs and output of any multiplication gate lies in $\mathbb{S}_{2^{k'}}$.

[0049] Embodiments can follow the standard real-ideal world simulation based security definition for secure multiparty computation (MPC) [27]. The ideal functionality $\mathcal{F}_{C_d}$ is defined in Table 2, below. Various embodiments consider an adversary that corrupts at most t of the parties and study both semi-honest and malicious adversaries.

$\mathcal{F}_{C_d}$
Parameters: An arithmetic circuit 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 $\text{out} = C_d(\text{inp}_1, \dots, \text{inp}_n)$ 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.

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

D. Computer System

[0050] FIG. 1 shows a system 100 according to embodiments of the disclosure. The system 100 comprises a plurality of computers including a computer A 110, a computer B 120, a computer C 130, and a computer D 140. Each computer of the plurality of computers can be in operative communication with any other computer in the system 100.

[0051] The computer A 110 includes a memory 112, a processor 114, a network interface 116, and a computer readable medium 118. The processor 114 can be coupled to the memory 112, the network interface 116, and the computer readable medium 118.

[0052] For simplicity of illustration, a certain number of components are shown in FIG. 1. It is understood, however, that embodiments of the invention may include more than one of each component. In addition, some embodiments of the invention may include fewer than or greater than all of the components shown in FIG. 1. For example, even though the components of computer A 110 are illustrated in FIG. 1, it is understood that the computer B 120, the computer C 130, and/or the computer D 140 can have similar components.

[0053] Messages between the computers in FIG. 1 can be transmitted using a secure communications protocols such as, but not limited to, File Transfer Protocol (FTP); HyperText Transfer Protocol (HTTP); Secure Hypertext Transfer Protocol (HTTPS), SSL, ISO (e.g., ISO 8583) and/or the like. The communications network may include any one and/or the combination of the following: a direct interconnection; the Internet; a Local Area Network (LAN); a Metropolitan Area Network (MAN); an Operating Missions as Nodes on the Internet (OMNI); a secured custom connection; a Wide Area Network (WAN); a wireless network (e.g., employing protocols such as, but not limited to a Wireless Application Protocol (WAP), I-mode, and/or the like); and/or the like. The communications network can use any suitable communications protocol to generate one or more secure communication channels. A communications channel may, in some instances, comprise a secure communication channel, which may be established in any known manner, such as through the use of mutual authentication and a session key, and establishment of a Secure Socket Layer (SSL) session.

[0054] The memory 112 of the computer A 110 can be used to store data and code. For example, the memory 112 can store secret shared values. The memory 112 may be coupled to the processor 114 internally or externally (e.g., cloud based data storage), and may comprise any combination of volatile and/or non-volatile memory, such as RAM, DRAM, ROM, flash, or any other suitable memory device.

[0055] The network interface 116 may include an interface that can allow the computer A 110 to communicate with external computers. The network interface 116 may enable the computer A 110 to communicate data to and from another device (e.g., to other computers in the system 100, etc.). Some examples of the network interface 116 may include a modem, a physical network interface (such as an Ethernet card or other Network Interface Card (NIC)), a virtual network

interface, a communications port, a Personal Computer Memory Card International Association (PCMCIA) slot and card, or the like. The wireless protocols enabled by the network interface 116 may include Wi-Fi™. Data transferred via the network interface 116 may be in the form of signals which may be electrical, electromagnetic, optical, or any other signal capable of being received by the external communications interface (collectively referred to as “electronic signals” or “electronic messages”). These electronic messages that may comprise data or instructions may be provided between the network interface 116 and other devices via a communications path or channel. As noted above, any suitable communication path or channel may be used such as, for instance, a wire or cable, fiber optics, a telephone line, a cellular link, a radio frequency (RF) link, a WAN or LAN network, the Internet, or any other suitable medium.

[0056] The computer readable medium 118 may comprise code, executable by the processor 114, for performing methods according to embodiments described herein. As an illustrative example, the computer readable medium 118 can comprise code, executable by the processor 114 for performing a method performed by a first computer of a plurality of computer for implementing a replication protocol for multi-party fixed point multiplication, the method comprising: obtaining a first secret share of a first fixed value (x) and a second secret share of a second fixed value (y); jointly sampling a first pseudo-random key ($s_{i,j}$) with another computer paired with the first computer and a second pseudo-random key (s_j) with a first set (D_j) of multiple sets in the plurality of computers; generating a replicated random share of a random value ($[[r]]^R$) using the second pseudo-random key (s_j); generating a replicated sharing value (β) using the first pseudo-random key ($s_{i,j}$); computing a masked share (z'_i) by providing a product of the first secret share and the second secret share masked with the replicated sharing value (β) and the replicated random shares of the random value ($[[r]]^R$); receiving, by the first computer from computers in the first set (D_j), a plurality of masked shares (z'_i), thereby revealing a set shared value z' to the first set (D_j); and generating a replicated secret share ($[[z]]^R$) using the received plurality of masked shares (z'_i) and the replicated random shares of the random value ($[[r]]^R$).

III. SECRET SHARING TECHNIQUES

[0057] Embodiments provide for several techniques for performing fixed-point computation. A first method includes efficient replicated secret sharing over the ring $\mathbb{Z}_{2^k}$, which is an improvement over the secret sharing methods as performed by [36, 37]. A second approach is for secret sharing over a prime field $\mathbb{F}_q$ such as Shamir secret sharing. A third approach is for additive secret sharing. Embodiments consider different types of secret sharing schemes depending on the number of computers and the setting: replicated secret sharing $[[x]]^R$ for a small number of computers, and Shamir and additive secret sharing $[[x]]^S$ for a larger number of computers.

[0058] Regardless of whether the sharing scheme naturally supports $\mathbb{Z}_{2^k}$ (e.g., as in replicated secret sharing) or $\mathbb{F}_q$ (e.g., as in Shamir secret sharing), the operations according to embodiments follow a similar approach. For example, for a computer P_i to input a value $x \in Fx_{N,d}$, the computer P_i maps a value x into a target group and secret shares the value x . Specifically, for a replicated secret sharing system including $\mathbb{Z}_{2^k}$ and $N = 2^k$, the computer P_i can compute $x' := x2^d \bmod 2^k$ and then run an input sharing protocol. In the case of Shamir secret sharing over $\mathbb{F}_q$, the computer P_i can compute $x' := x2^d \bmod q$ and can input x' . Addition and 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 $x := x'/2^d \in Fx_{N,d}$.

[0059] A technical problem that is present in multiplying this style of fixed-point secret shares is that the standard protocol would result in a semantic value in $Fx_{N,2d}$ given input values in $Fx_{N,d}$ and also ignores the possibility of the product overflowing N . That is, let $[[x']]$, $[[y']]$ be sharing of values $x, y \in Fx_{N,d}$ as described above. Then, when computing $[[z']] := [[x']][[y']]$ (e.g., multiplying the values) using the standard $\mathbb{Z}_{2^k}$ (e.g., replicated secret sharing) or $\mathbb{F}_q$ (e.g., Shamir secret sharing) protocols would result in a value z' such that $z'/2^d \approx xy$ where the approximation comes from the rounding of the lower order terms. Truncation can then be used to take a shared value in $Fx_{N,2d}$ and produce shares of that value rounded into $Fx_{N,d}$.

[0060] Ideally, this rounding would exactly correspond to multiplication in $Fx_{N,d}$ (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. Embodiments capture this probabilistic rounding in an ideal functionality $\mathcal{F}_{mul}^{N,d,\mathcal{E}}$ which is parameterized by a modulus N , the number of decimal bits d , and a rounding error distribution $\mathcal{E}$. This distribution can specify how the rounding can be performed.

[0061] Embodiments adapt and improve these prior approaches beyond the two party setting while reducing the communication overhead.

$\mathcal{F}_{mul}^{N,d,\mathcal{E}}$
Fixed-point Multiply: The functionality is parameterized by a modulus $N \in \mathbb{N}$, a number of decimal bits $d \in \mathbb{N}$, and a rounding error distribution $\mathcal{E}$. Upon receiving shares $\llbracket x \rrbracket, \llbracket y \rrbracket$ from parties, the functionality reconstructs $x, y \in Fx_{N,d}$. 1. Sample $e \leftarrow \mathcal{E}$ as the rounding error term. 2. Output $\llbracket xy/2^d + e \rrbracket$ to the respective parties.

Table 3: Probabilistic functionality for fixed-point multiplication.

A. Replicated Secret Sharing

[0062] A value $x \in \mathbb{G}$ is replicated secret shared by uniformly sampling $x_1, \dots, x_m \in \mathbb{G}$ 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 the value x , it is possible to distribute these shares such that every subset of $(t+1)$ parties holds all m shares of the value x , yet any smaller grouping of parties is missing at least one share. One method for doing this is as follows. Define the sets $D_1, \dots, D_m \subset [n]$ as the m distinct subsets of size $(n-t)$. Without loss of generality, for $i \in [n]$, let $D_i = \{i, i+1, i+2, \dots\}$ be the contiguous set of computer indices starting at index i , with wrap around, as illustrated in respect to FIG. 2. The replicated secret share of the i^{th} computer P_i is then $\llbracket x \rrbracket_i^R = \{x_j | i \in D_j\}$. Embodiments can use the group $\mathbb{S}_{2^k}$.

[0063] FIG. 2 illustrates a computer system comprising sets of computers according to an embodiment of the present disclosure. FIG. 2 illustrates three different sets of computers grouped from the same network of computers. FIG. 2 includes a first set of computers 210 (D_1), a second set of computers 220 (D_2), and a third set of computers 230 (D_3). The network of computers can comprise any number of computers grouped into any number of overlapping sets of computers.

[0064] The first set of computers 210 includes a first computer P_1 , a second computer P_2 , and a third computer P_3 . The second set of computers 220 includes the second computer P_2 , the third computer P_3 , and a fourth computer P_4 . The third set of computers 230 includes the first computer P_1 , and the fourth computer P_4 . In replicated secret sharing, a share x_i can be held (e.g., stored) by all of the computers indexed by D_i .

[0065] Due to the linearity of the method, addition, subtraction, or multiplication with a public constant can be performed locally by applying the corresponding operation to the shares. For example, computing $\llbracket z \rrbracket^R := \llbracket x \rrbracket^R + \llbracket y \rrbracket^R$ can be done by having each computer P_i define the output shares $\llbracket z \rrbracket_j^R := \llbracket x \rrbracket_j^R + \llbracket y \rrbracket_j^R$. Subtraction works in the same way while public multiplication $\llbracket z \rrbracket^R := c \llbracket x \rrbracket^R$ can be performed by defining $\llbracket z \rrbracket_j^R := c \llbracket x \rrbracket_j^R$. To multiply two secret shared values $\llbracket z \rrbracket^R := \llbracket x \rrbracket^R \llbracket y \rrbracket^R$, parties can perform an interactive protocol due to cross terms between the values of x and y . Multiplication of two secret shared values is described in more detail herein.

[0066] The replicated secret sharing methods according to embodiments can be utilized for a 2-out-of-2 secret sharing scheme over a ring $\mathbb{S}_{2^k}$. 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 where the division is the quotient over the integers. Here, the possibility of having a minus one comes from the fact that if $(z_1 \bmod 2^d) + (z_2 \bmod 2^d) \geq 2^d$, then this would have generated a “carry bit” which was eliminated by performing separate division operations on the shares, mathematically shown below.

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

[0067] There are two subtleties in understanding this approach: 1) the individual shares z_1, z_2 should be interpreted as being signed integers in the range $\mathbb{S}_{2^k} = [-2^{k-1}, 2^{k-1})$ and 2) the necessity for $|z| \ll 2^{k-1}$. In the event that $z_1 + z_2$ wraps around in the ring $\mathbb{S}_{2^k}$ (e.g., if when interpreting z_1, z_2 as integers and adding them results in $(z_1 + z_2) > 2^{k-1}$ or $(z_1 + z_2) > 2^{k-1} + 1$), then it must hold that the values z_1, z_2 are both positive or negative and that the value 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].

[0068] To fix these catastrophic errors, [37] observed the following. First, redefine the shares by a random value $r \xleftarrow{\$} \mathbb{S}_{2^k}$ 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 the values 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.

[0069] Unfortunately, this approach does not generalize to more than two parties. A reason is that it is likely for the shares to wrap around the modulus, regardless of the distribution of the value 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 $\llbracket r \rrbracket^R, \llbracket r/2^d \rrbracket^R$ is computed. Given this, the protocol above can be emulated by revealing $\llbracket z \rrbracket^R := \llbracket z \rrbracket^R - \llbracket r \rrbracket^R$ and then letting $\llbracket z_2/2^d \rrbracket^R := \llbracket r/2^d \rrbracket^R$. Since z_1 is made public, all parties can compute $\llbracket z/2^d \rrbracket := \llbracket z/2^d \rrbracket^R + z_1/2^d$.

[0070] The main limitation of [36] is the need to generate the truncation pair $\llbracket r \rrbracket^R, \llbracket r/2^d \rrbracket^R$. [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 as n grows.

[0071] Embodiments solve the aforementioned technical problems, which stem from the original “carry bit” technical problem discussed above. Embodiments provide for an efficient technique for generating a multi-party sharing of a value 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 $x_2, \dots, x_m \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ and then defining the last share as $x_1 = x - x_2 - \dots - x_m$. 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 utilized in order to sufficiently hide the value of x .

[0072] 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:

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

[0073] where $e := (x_1 \text{ symod } 2^d + \dots + x_m \text{ symod } 2^d)/2^d \in [-m, m]$. 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.

[0074] This approach does not, in isolation, provide a practical secret sharing scheme due to the size of the shares doubling after each multiplication. Embodiments overcome this value size problem by effectively converting a traditional secret sharing (e.g., replicated) into the sharing described above, performing the division and then converting back.

[0075] The efficient replicated secret sharing technique will now be discussed. First let us consider the task of dividing a replicated secret sharing $\llbracket x \rrbracket^R$ of a value x by 2^d to obtain a sharing $\llbracket z \rrbracket^R$ where $z \approx x/2^d$ over the integers. Combining this with the multiplication protocol with little added overhead, will be described in further detail herein.

[0076] Recall that each replicated secret sharing consists of m shares, distributed among n parties, in particular, $x = (x_1 + \dots + x_m) \text{ symod } 2^k$. Embodiments can utilize a secret value in a ring $x \in \mathbb{S}_{2^{k'}}$ for some $2^{k'} \ll 2^k$ (e.g., $k = k' + \lambda + \log_2 m$).

[0077] The parties jointly sample a random sharing $\llbracket r \rrbracket^R$ of a random value r , where share r_i is defined as $r_i \leftarrow \mathbb{S}_{2^{k'+\lambda}}$. This type of secret sharing has three noteworthy properties. A first property is that $\llbracket r \rrbracket^R$ does not wrap around $\mathbb{S}_{2^k}$ when reconstructed (e.g., $r = (r_1 + \dots + r_m) = (r_1 + \dots + r_m \text{ symod } 2^k)$). A second property is that the distributions of the random value r and the sum of the random value r and the secret value x (e.g., $w := r + x$) are statistically close. This second property follows from the random value r being distributed over a range that is approximately $2^{\lambda + \log_2 m}$ times bigger than the secret value x . A third property is that the random sharing $\llbracket r \rrbracket^R$ can be sampled non-interactively by sampling the shares as the output of a pseudorandom function (PRF).

[0078] The protocol can proceed by revealing a revealed value w , which is equal to $\llbracket w \rrbracket^R := \llbracket r \rrbracket^R + \llbracket x \rrbracket^R$, to all parties. Assume that $r + x$ does not wrap around and observe that:

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

[0079] As before, the error value $e \in [-m, m]$. The parties can locally compute shares of $\llbracket z \rrbracket^R = \llbracket x/d^2 + e \rrbracket^R$ given the revealed value w and a shared random value $\llbracket r \rrbracket^R$ by defining $z_1 := w/2^d - r_1/2^d$ and $z_i := -r_i/2^d$ for $i \in [2, m]$.

[0080] Next, fused multiplication and truncation will be described. The trivial way of performing a fused multiply and truncate method is to first perform the standard multiplication

protocol to compute $\llbracket z' \rrbracket^R := \llbracket x \rrbracket^R \llbracket y \rrbracket^R$ and then the truncation protocol to compute $\llbracket z \rrbracket^R: \approx \llbracket z' \rrbracket^R / 2^d$. The main downside of this approach is the need to perform two rounds of interaction.

[0081] First, recall the normal multiplication protocol. Observe that the multiplied secret values $xy = (x_1 + \dots + x_m)(y_1 + \dots + y_m) = \sum_{j,j'} x_j y_{j'}$, and that the shares are distributed such that at least one computer can compute each cross term $x_j y_{j'}$. The shares x_j, y_j will be held by computer P_i for all $i \in D_j$ (the set of parties D_j consists of a set of $(n - t)$ parties and by definition, $P_i \in D_i$). Each cross term $x_j y_{j'}$ can be assigned to a single P_i and let u_i be the sum of the crossterms. For $j \in (n, m]$, the parties in the set of parties 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 $\llbracket s \rrbracket$ where a computer P_i holds a zero-sharing share s_i and the sum of all zero-sharing shares equals zero $\sum_i s_i = 0$. Given these, the computer P_i defines $z'_i := u_i + s_i - \sum_{j \in N_i} z'_j$ and distributes z'_i to all parties in the set of parties D_i . Here $N_i \subset (n, m]$ index the set of z'_j terms that are assigned to the computer P_i . A detailed description of this protocol is given in Table 7.

[0082] There are two sets of randomized shares, 1) z'_j for $j \in (n, m]$ and 2) 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, z'_{j^*} should be sampled uniformly, as is done above.

[0083] The zero sharing $\llbracket s \rrbracket$ serves a slightly different purpose. For exposition, let's assume a first secret value x is somehow known to the adversary while a second secret value y , and therefore y_{j^*} , is not. The honest parties P_i will each use y_{j^*} to compute their share z'_i , when determining the product of the first secret value x and the second secret value y . 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, embodiments effectively “distribute” the uncertainty the adversary has about z'_{j^*} into all of the z'_i messages that are sent.

[0084] Now, the multiplication and division protocols can be combined into a single fused operation. A 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, it can be observed that sampling the z'_j shares for $j \in (n, m]$ are no longer necessary due to z' being sufficiently masked by the random value r .

[0085] In more detail, each computer P_i computes the sum u_i of their cross terms x_j, y_j , along with their zero sharing s_i . The sharing of the random value $\llbracket r \rrbracket^R$ is sampled as $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ which ensures that the random value r will not wrap around. The computer 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 the semi-honest secure protocol is given in Table 6.

[0086] To achieve malicious security, embodiments can include an information theoretic MAC az' along with the term z' . The MAC key α is unknown to the adversary and is secret shared amongst all the parties. Using $\llbracket \alpha \rrbracket^R$, parties can compute secret shares of az' . Then, a MAC check protocol (e.g., as an example, from in Figure 10 of [23]) can be run that checks that shares of the MAC term az' are consistent with $\llbracket \alpha \rrbracket^R$ and z' . From this, it can be guaranteed that the term z' was correctly computed by the adversary. Finally, after computing $\llbracket z \rrbracket^R$ as in the semi-honest protocol, embodiments can also compute the MAC $\llbracket az \rrbracket^R$. Table 9, below, describes the method in further detail.

B. Shamir Secret Sharing

[0087] To Shamir secret share [43] a value $x \in \mathbb{F}$, a computer that holds a value x can sample a uniformly random polynomial $p_x(\cdot) \in \mathbb{F}[\cdot]$ of degree t such that a first point of the polynomial lies at the value x (e.g., $p_x(0) = x$). Each computer P_i is given a point $\llbracket x \rrbracket_i^S := p_x(i)$, which is the evaluation of the polynomial p_x at a point 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 the polynomial p_x and retrieve the secret value $x = p_x(0)$, while any subset of fewer than $(t + 1)$ parties learn no information about the secret. The notation $\llbracket x \rrbracket^{S,t'}$ can denote the Shamir secret sharing of the value x using a polynomial of degree t' . Typically, embodiments will be described in reference to polynomials of degree t and omit the degree in this case for brevity.

(e.g., $\llbracket x \rrbracket^S$) and consider $\mathbb{F} = \mathbb{F}_q$ for a prime q , however it is understood that embodiments are not limited thereto.

[0088] 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 $\llbracket z \rrbracket^S := \llbracket x \rrbracket^S + c$ or $\llbracket z \rrbracket^S := c \llbracket x \rrbracket^S$, each computer P_i locally compute $\llbracket z \rrbracket_i^S := \llbracket x \rrbracket_i^S + c$ or $\llbracket z \rrbracket_i^S := c \llbracket x \rrbracket_i^S$, respectively. Similarly, to calculate the sum of two shared values $\llbracket z \rrbracket^S := \llbracket x \rrbracket^S + \llbracket y \rrbracket^S$, each computer P_i can locally compute their shares as $\llbracket z \rrbracket_i^S := \llbracket x \rrbracket_i^S + \llbracket y \rrbracket_i^S$. However, additional processing is needed when multiplying two shared values and is described in further detail herein.

[0089] For the sake of comparison a detailed description of the protocol of Catrina and Saxena [16] for computing $\llbracket z \rrbracket^S := \llbracket x \rrbracket^S / 2^d$ will be first discussed in reference to Table 4. The core of the 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 $a \in \mathbb{F}_q$, embodiments compute $a/2^d$ in the integers by computing $(a - (a \bmod 2^d)) \cdot 2^{-d}$ in $\mathbb{F}_q$. The challenge is to support signed values and to give an efficient way to compute a sharing of $\llbracket a \bmod 2^d \rrbracket^S$ given $\llbracket a \rrbracket^S$.

Π_{trunc}

Parties have access to functionality $\mathcal{F}_{\text{rand}2}^{q,d}$, which distributes shares $\llbracket r \rrbracket^S, \llbracket r' \rrbracket^S$ where $r \in \mathbb{F}_q$ and $r' = r \bmod 2^d$.

Truncate: Upon input $\llbracket a \rrbracket^S$, parties do the following:

- (a) Compute $\llbracket b \rrbracket^S = \llbracket a \rrbracket^S + 2^{k-1}$.
- (b) Invoke $\mathcal{F}_{\text{rand}2}^{q,d}$, and receive shares $\llbracket r \rrbracket^S, \llbracket r' \rrbracket^S$ where $r' = r \bmod 2^d$.
- (c) Send $\llbracket c \rrbracket^S = \llbracket b \rrbracket^S + \llbracket r \rrbracket^S$ to party P_1 . P_1 reconstructs c and broadcasts to other parties.
- (d) Compute $\llbracket a' \rrbracket^S = c' - \llbracket r' \rrbracket^S$ where $c' = c \bmod 2^d$.
- (e) Compute $\llbracket d \rrbracket^S = (\llbracket a \rrbracket^S - \llbracket a' \rrbracket^S) \cdot 2^{-d}$ and take $\llbracket d \rrbracket^S$ as output.

Table 4: Protocol for truncation [16, Protocol 3.1].

[0090] 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 $a' := b \bmod 2^d$ is precisely the “semantic” low order bits of a . Effectively, shifting a allows for the handling the positive and negative cases simultaneously.

[0091] Given $\llbracket b \rrbracket^S = \llbracket a + 2^{k-1} \rrbracket^S$ the parties approximate $\llbracket b \bmod 2^d \rrbracket^S$ using a mask and open technique. First, the pair $\llbracket r \rrbracket^S, \llbracket r' \rrbracket^S$ are preprocessed where $r \xleftarrow{\$} \mathbb{F}_q$ and $r' := (r \bmod 2^d)$. The masked value $c := b + r$ is then revealed to all parties. It is required that $b + r$ does not wrap around $\mathbb{F}_q$, which will be discussed later. The parties define $c' := c \bmod 2^d = b + r \bmod 2^d$. Observe that $\llbracket a' \rrbracket^S := c' - \llbracket r' \rrbracket^S = (b + r \bmod 2^d) - \llbracket r' \rrbracket^S$ is equal to $(b \bmod 2^d)$ or $(b - 1 \bmod 2^d)$. This -1 term is due to the possibility of $b + r$ generating a carry bit at bit position d which is eliminated by the $\bmod 2^d$ operation. As such, this truncation will result in the same rounding error as in [36, 37]. The final result can then be computed as $\llbracket d \rrbracket^S := (\llbracket a \rrbracket^S - \llbracket a' \rrbracket^S) \cdot 2^{-d}$ which equals $a/2^d$ or $a/2^d - 1$ over the integers.

[0092] In [16], the multiplication and division are given separately. A follow up work [12] gave an optimization which combined the multiplication and division into a single round as shown in Table 5. 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.

$\Pi_{\text{mul-trunc}}(\llbracket x \rrbracket^S, \llbracket y \rrbracket^S)$
Parties hold preprocessed shares $\llbracket r \rrbracket^{S,t}, \llbracket r \rrbracket^{S,2t}, \llbracket r' \rrbracket^{S,t}$, where r is random and $r' = r \bmod 2^d$. Multiple and truncate: On inputs $\llbracket x \rrbracket^S, \llbracket y \rrbracket^S$, each party does the following: 1. Compute $\llbracket z' \rrbracket^{S,2t} = \llbracket x \rrbracket^S, \llbracket y \rrbracket^S + r^{S,2t}$ 2. Send $\llbracket z' \rrbracket^{S,2t}$ to P_1, who reconstructs z' and broadcasts the result to the other parties. 3. Compute $w = (z' + 2^{k-1}) \bmod 2^d$ and $\llbracket w' \rrbracket^{S,t} = w - \llbracket r' \rrbracket^{S,t}$. 4. Output $\llbracket z \rrbracket^{S,t} = (z' - \llbracket r \rrbracket^{S,t} - \llbracket w' \rrbracket^{S,t}) \cdot 2^{-d}$

Table 5: Protocol for multiplication for Shamir secret sharing.

[0093] The Shamir secret sharing method according to embodiments is conceptually similar to the replicated secret sharing method. This generality speaks to the versatility of the methods according to embodiments in that it can be applied to many settings.

[0094] A first step is for the parties to sample a linear secret sharing of a random value r , which does not wrap around the modulus. One method for achieving this is to have the computer P_i for $i \in [t + 1]$ sample $r_i \xleftarrow{\$} \mathbb{S}_{q/(t+1)}$ and generate a sharing of $\llbracket r \rrbracket^S$ and $\llbracket r^i/2^d \rrbracket^S$. Given these, the parties can define $\llbracket r \rrbracket^S := \sum_i \llbracket r_i \rrbracket^S$ and $\llbracket r' \rrbracket^S := \sum_i \llbracket r_i/2^d \rrbracket^S$. By the same analysis as above, $r/2^d = r' + e$ for some error value e with at most $\log_2 t$ bits.

[0095] Then, to truncate an existing sharing $\llbracket x \rrbracket^S$ of a secret value x by 2^d the parties reveal $\llbracket w \rrbracket^S := \llbracket x \rrbracket^S + \llbracket r \rrbracket^S$ to all parties. The output shares are defined as $\llbracket z \rrbracket^S := w/2^d - \llbracket r' \rrbracket^S$. Since the random value r is not uniformly distributed over $\mathbb{F}_q$, this protocol requires that $|x| \ll q$ in order to argue that the secret value x remains hidden, which is a common property to all efficient division protocols. The detailed protocol is presented in Table 13.

C. Additive Secret Sharing

[0096] A value $x \in \mathbb{G}$ can be additive secret shared by uniformly sampling $x_1, \dots, x_n \in \mathbb{G}$ such that $x = \sum_i x_i$. The additive secret share of the i^{th} computer P_i is then $\llbracket x \rrbracket_i^A := x_i$. Observe that unless all the n parties come together, no information about the secret x is learnt. Embodiments

can 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, but multiplication between two secret shared values needs additional processing.

[0097] The aforementioned approaches for replicated and Shamir secret sharing utilize a setting in which an honest majority is present among the parties. This is inherently required by these secret sharing schemes. Nevertheless, it is not a requirement of the efficient sharing and truncation technique according to embodiments. In fact, the truncation methods according to embodiments can be viewed as general techniques that works for any $t < n$.

[0098] For example, in the additive secret sharing setting, to truncate a secret shared value $\llbracket x \rrbracket \in \mathbb{S}_N$ where $x \in \mathbb{S}_{2^{k'}}$ and $\log_2 N \geq k' + \lambda + \log_2 n$, first let the parties sample a linear secret sharing of a random value r which does not wrap round $\mathbb{S}_N$. In particular, each computer P_i for $i \in [t + 1]$ samples $r_i \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$ and generates sharings of $\llbracket r_i \rrbracket$ and $\llbracket r_i/2^d \rrbracket$ in $\mathbb{S}_N$. Given these, the parties can define $\llbracket r \rrbracket := \sum_i \llbracket r_i \rrbracket$ and $\llbracket r' \rrbracket := \sum_i \llbracket r_i/2^d \rrbracket$. By the same analysis as above, embodiments have that $r/2^d = r' + e$ for some error value e with at most $\log_2 t$ bits.

[0099] To illustrate this idea, embodiments show how to incorporate the truncation protocol into additive secret sharing with dishonest majority and present the fixed point multiplication protocol in Table 15.

IV. REPLICATED SECRET SHARING PROTOCOLS FOR MULTI-PARTY FIXED POINT MULTIPLICATION

[0100] Embodiments provide for a new technique for fixed point multiplication with replicated secret sharing in the presence of a semi-honest adversary. It is then shown how to incorporate this method into a general MPC protocol to compute any arithmetic circuit.

[0101] Additionally, embodiments provide for a method of performing fixed point multiplication with replicated secret sharing in the presence of a malicious adversary.

A. Replicated Secret Sharing: Semi-Honest

[0102] For a semi-honest protocol for fixed point multiplication with replicated secret sharing, consider two fixed point values, a first secret value x and a second secret value y , represented in two's-complement form in a ring $\mathbb{S}_{2^{k'}}$ where the bottom d bits denote the decimal. The two fixed point value x and y can be secret values that are not known in the clear to any computer in the system. The computers in the system (e.g., the parties) hold replicated secret shares $\llbracket x \rrbracket^R$ and $\llbracket y \rrbracket^R$ of the first secret value x and the second secret value y , respectively, in an extended ring $\mathbb{S}_{2^k}$ where $k \geq (k' + \lambda + \log_2(m))$.

[0103] The computers in the system can attempt to compute a replicated secret share $\llbracket z \rrbracket^R$ where the output value $z \approx \frac{xy}{2^d}$. The output value z can be the product of the first secret value x and the second secret value y . The method can take place among n computers $P_1, \dots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. The system of computers can include a plurality of sets of computers, where each set can include more than one computer. A number of distinct sets can be denoted by m and can relate to the total number of computers n by $m = \binom{n}{n-t} D_1, \dots, D_m \subset [n]$. The method is described formally in Table 6.

[0104] In a setup phase, every pair of computers P_i and P_j in the system can jointly sample a first pseudo-random key $s_{i,j}$, while the computers in each set D_j sample a second pseudo-random key s_j . Then, in an online phase, for each fixed point multiplication, each of the plurality of computers can jointly generate replicated shares of a random value r by locally generating each share r_j at random 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 random value r lies in $\mathbb{S}_{2^{k'+\lambda+\log_2(m)}}$. The parties then use the second pseudo-random keys $s_{i,j}$ to generate a replicated sharing of zero - represented by the β 's where each β_i lies in $\mathbb{S}_{2^k}$.

[0105] Each computer then generates a masked share z' based on the computer's first secret value share, second secret value share, and random value share. The masked share z' can be determined using $z' = x \cdot y + r$, which can be further masked with the share of zero. The masked share z' can be revealed to all computers in the first set of computers D_1 . Note that the

sharing of zero ensures that no individual share of the masked share z' is revealed in the clear to any computer - 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 the product of the first secret value x and the second secret value y if revealed in the clear.

[0106] Now, when the computers in the first set D_1 obtain the masked share $z' = x \cdot y + r$, observe that although the random value r is not entirely uniform in the ring $\mathbb{S}_{2^k}$, its sampling space is sufficiently larger than the value of the product of the first secret value x and the second secret value y and it is in fact statistically close to a uniform distribution, hence the masked share z' looks sufficiently random to all the computers. Finally, each computer can obtain a replicated secret share of the sharing of $\llbracket z \rrbracket^R$ by determining the first share $z_1 = z'/d - r_1/d$ and the rest of the shares as $z_j = -r_j/d$.

[0107] FIG. 3 illustrates an example signaling process 300 for a replication protocol for multi-party fixed point multiplication. FIG. 3 illustrates the signaling process of a replication protocol as described in Table 6. As shown in FIG. 3, a network includes a plurality of computers including a first computer 302, a second computer 304, and a third computer 306. The computers can perform a replication method for multi-party fixed point multiplication. Although three computers are illustrated in FIG. 3, it is understood that any number of computers can interact using the methods(s) as described herein.

[0108] Prior to step 310, each computer of the plurality of computers can obtain a first secret share ($\llbracket x \rrbracket^R$) of a first value (x) and a second secret share ($\llbracket y \rrbracket^R$) of a second value (y).

[0109] At step 310, each computer in the network of computers, including the first computer 302, the second computer 304, and the third computer 306, can locally sample keys ($s_{i,j}$, s_j). For example, each computer can sample a first pseudo-random key ($s_{i,j}$) and a second pseudo-random key (s_j).

[0110] The first pseudo-random key ($s_{i,j}$) can be generated by a pair of computers. Each computer in the network of computers can be paired with another computer in the network. The two computers in each pair of computers can jointly sample the first pseudo-random key ($s_{i,j}$). In

particular, for each pair of computers $i, j \in [n]$ where $i < j$, the computers P_i and P_j jointly sample the first PRF key $s_{i,j} \leftarrow \{0,1\}^k$.

[0111] The second pseudo-random key (s_j) can be generated by a set of computers. Each computer in the network of computers can be in a set (e.g., subset) of computers in the network. For example, the network can include 10 computers and 2 sets of computers that each comprise 5 computers. Each set of computers can jointly sample the second pseudo-random key (s_j) with the set of computers (D_j) of multiple sets.

[0112] For example, a pair of computers (P_i and P_j) (e.g., such as the first computer 302 and the second computer 304) can jointly sample a first PRF key ($s_{i,j}$), and computers in each set of computers (D_j) can sample a second PRF key (s_j). As such, each computer in the network can obtain two keys, where each key is shared between at least two computers.

[0113] At step 315, the first computer 302, the second computer 304, and the third computer 306 can locally generate replicated shares of a random value (r) by generating each share (r_j) at random using the second pseudo-random key (s_j). Each computer in the network can obtain a share $\llbracket r \rrbracket^R$ of the random value (r). The shares may differ from computer to computer.

[0114] The computers can sample to random values in any suitable manner. As an example, each computer can maintain a counter (cnt) that can be used as input to a pseudo-random function number generator along with the second pseudo-random key (s_j). For example, each computer can sample $\llbracket r \rrbracket^R$ such that $r_j := \text{PRF}_{s_j}(cnt)$.

[0115] At step 320, the first computer 302, the second computer 304, and the third computer 306 can locally use the first pseudo-random keys ($s_{i,j}$) to generate a replicated sharing of zero (e.g., an n-out-of-n secret sharing of zero, which is represented by β). For example, each computer can generate a preprocessing zero value ($\gamma_{i,j}$). One half of the computers can generate positive preprocessing zero values ($\gamma_{i,j}$), while the other half of the computers can generate negative preprocessing zero values ($\gamma_{i,j}$). For example, computers $j \in [1, i)$ can generate a positive preprocessing value ($\gamma_{i,j}$), while computers $j \in (i, n]$ can generate a negative preprocessing value ($\gamma_{i,j}$). Each preprocessing zero value ($\gamma_{i,j}$) can be generated using the first

PRF key ($s_{i,j}$) and the counter (cnt). Each computer can then communicate with the other computers in their set of computers to obtain each preprocessing zero value ($\gamma_{i,j}$) from the set. The replicated sharing of zero (β) can be determined based on the preprocessing zero values ($\gamma_{i,j}$) from the set. For example, the replicated sharing of zero (β) can be equal to the summation of the preprocessing zero values ($\gamma_{i,j}$) from the set (e.g., $\beta_i = \sum_{j \in [n]/i} \gamma_{i,j}$).

[0116] At step 325, the first computer 302, the second computer 304, and the third computer 306 can locally compute a masked share (z'_i). The masked share (z'_i) can be the product of the first secret share ($[[x]]^R$) and the second secret share ($[[y]]^R$) masked with the replicated sharing value (β) and the replicated random shares of the random value ($[[r]]^R$). The first computer 302 can determine a first masked share ($z'_{i=1}$), the second computer 304 can determine a second masked share ($z'_{i=2}$), and the third computer 306 can determine a third masked share ($z'_{i=3}$). In particular, each computer of the plurality of computers can determine the masked share as:

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

[0117] At step 330, the masked shares (z'_i) computed by each computer in a set of computers (D_1) can be shared across the set of computer, such as the first computer 302, the second computer 304, and the third computer 306. Each computer in the set of computers (D_1) can obtain a set shared masked value (z') from the masked shares (z'_i) computed by each computer in the set of computers (D_1). The set shared masked value (z') can be the same value held by each computer in the set of computers (D_1). This can be performed since the masked shares ($z'_1, \dots, z'_n$) can be an n -out-of- n secret sharing of the set shared masked value z' . The set shared masked value z' can be reveal to each computer in the set of computers (e.g., $P_i \in D_1$). The masked shares (z'_i) can appear to be sufficiently random to all the computers in the set of computers.

[0118] At step 335, after receiving the masked shares (z'_i) from each computer within the set of computers (D_1), the first computer 302, the second computer 304, and the third computer 306 can generate replicated secret share $[[z]]^R$. The replicated secret share can be generated based on

the received plurality of masked shares (z'_i) and the replicated random shares of the random value ($[[r]]^R$).

[0119] In some embodiments, the first computer can determine the first replicated share of the output value $[[z]]^R$ based on a masked value (z') created from the masked shares (z'_i), the decimal bit (d), and the first computer's share of the random value (r_1). In particular, the first computer can determine the first replicated share of the output value $[[z]]^R$ by performing $z_1 := z'/d - r_1/d$. The other computers can respectively determine the other replicated shares of the output value $[[z]]^R$ based on the shares of the random value (r_i) held by that computer and the decimal bit (d). In particular, the other computers can determine the replicated shares of the output value $[[z]]^R$ by performing $z_j := -r_j/d$.

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

- The protocol outputs $[[z]]^R$ 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 $\text{PRF}: \{0,1\}^k \times \{0,1\}^* \rightarrow \mathbb{S}_{2^{k-\log_2(m)}}$ and $\widetilde{\text{PRF}}: \{0,1\}^k \times \{0,1\}^* \rightarrow \mathbb{S}_{2^k}$ be pseudorandom functions.
- Let $A_1, \dots, A_n \subset [m]^2$ 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 $B_1, \dots, B_n \subset [m]$ 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 $s_{i,j} \leftarrow \{0,1\}^k$.
- Parties $P_i \in D_j$ jointly sample key $s_j \leftarrow \{0,1\}^k$. All parties keep a counter $\text{cnt} := 0$.

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

1. The parties sample $[[r]]^R$ such that $r_j := \text{PRF}_{s_j}(\text{cnt})$.
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 $\gamma_{i,j} := \widetilde{\text{PRF}}_{s_{j,i}}(\text{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\text{PRF}}_{s_{i,j}}(\text{cnt})$ 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 $\llbracket x \rrbracket_i^R = \{x_j | i \in D_j\}$ (resp. $\llbracket y \rrbracket^R$).

4. Let $(z'_1, \dots, z'_n)$ be an n -out-of- n secret sharing of z' . Reveal z' to $P_i \in D_1$.

5. The parties output $\llbracket z \rrbracket^R$ where $z_1 := z'/d - r_1/d$ and $z_j := -r_j/d$ for $j \in [2, m]$ and increment $\text{cnt} := \text{cnt} + 1$.

Table 6: Protocol for fixed point multiplication with replicated secret shares against semi-honest adversaries.

[0120] A tradeoff between the number of rounds and the number of messages per communication can be made in the protocol illustrated in Table 6. In the multiply phase, the protocol utilizes one round of communication. However, each computer sends its share of the masked share z' to every computer in the set D_1 and so the total number of messages exchanged is $n \cdot (n - t)$ which is quadratic in the number of computers. Alternatively, other embodiments can have a two-round protocol where, in the first round, all computers can send their shares of the masked share z' only to one computer (e.g., a leader computer P_1) in the set of computers D_1 . The leader computer can then forward the masked share z' to the other computers in the set of computers D_1 . This reduces the number of messages to $(n - 1) + (n - t - 1)$, which is linear in the number of computers present.

[0121] In terms of being information theoretic, computational assumptions are used for the PRF. However, the use of a PRF is utilized to reduce communication. In fact, to make the protocol completely information theoretic, embodiments can replace the PRF calls by having one computer pick (e.g., generate or otherwise obtain) the corresponding random string and share it with the others (at the cost of more communication).

B. Replicated Secret Sharing: Malicious

[0122] This section builds on the replicated secret sharing methods in the semi-honest setting from the previous section to describe a protocol for fixed point multiplication with replicated secret sharing in the presence of a malicious adversary. It is then shown how to incorporate this into a general MPC protocol to compute any arithmetic circuit.

[0123] Multiplication can be performed on two values represented using replicated secret sharing for any n -parties where $n \geq 3$. First, a multiplication protocol in such a secret shared setting will be discussed in reference to Tables 10 and 11. Then, a fixed point multiplication with truncation protocol will be described with this multiplication protocol as a sub-routine, in reference to Table 9.

[0124] The fixed point multiplication method can include a first secret value x and a second secret value y , and let a ring be $\mathbb{Z}_{2^k}$. Computers in a system can have replicated secret shares $\llbracket x \rrbracket^R$ and $\llbracket y \rrbracket^R$ of the first secret value x and the second secret value y , respectively. The computers can attempt to compute a replicated secret share $\llbracket z \rrbracket^R$ where the output value z is the product of the first secret value x and the second secret value y (e.g., $z \approx xy$). 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. The replicated multiplication protocol is illustrated in Table 7, while the “promote” protocol is illustrated in Table 8.

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

- Protocol for multiplication with replicated secret shares.
- The protocol outputs $\llbracket z \rrbracket^R$ where $z \approx xy$.
- Let $\widetilde{\text{PRF}}: \{0,1\}^k \times \{0,1\}^* \rightarrow \mathbb{S}_{2^k}$ be a pseudorandom function.
- Let $A_1, \dots, A_n \subset [m]^2$ 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 $s_{i,j} \leftarrow \{0,1\}^k$. All parties keep a counter $\text{cnt} := 0$.

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

1. Each party P_i defines $r'_{i,j} := \widetilde{\text{PRF}}_{s_{ji}}(\text{cnt})$ for $j \in [1, i)$ and $r'_{i,j} := -\widetilde{\text{PRF}}_{s_{i,j}}(\text{cnt})$ 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 $\llbracket x \rrbracket_i^R = \{x_j | i \in D_j\}$ (resp. $\llbracket y \rrbracket^R$).

3. The parties invoke protocol Π_{Promote}^R (in Table 8) where P_i inputs z'_i and receives $\llbracket z \rrbracket_i^R$ where z is defined as $z = \sum_{j \in [n]} z_j$. The parties output $\llbracket z \rrbracket^R$ and increment $\text{cnt} := \text{cnt} + 1$.

Table 7: Protocol for multiplication with replicated secret shares

$$\Pi_{\text{Promote}}^R(\llbracket z' \rrbracket)$$

- The protocol outputs $\llbracket z \rrbracket^R$.
- Let $\text{PRF}: \{0,1\}^k \times \{0,1\}^* \rightarrow \mathbb{S}_{2^k}$ be a pseudorandom function.
- Let $N_1, \dots, N_n \subset (n, m]$ 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 $\{P_i | i \in D_j\}$ jointly sample $s_j \leftarrow \{0,1\}^k$. All parties keep a counter $\text{cnt} := 0$.

Promote: Upon input z_i , from P_i , parties compute the product as follows:

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

Table 8 Protocol for promoting an n -out-of- n sharing into a replicated secret sharing.

[0125] FIG. 4 illustrates a block diagram 400 illustrating an example replication method for multi-party fixed point multiplication. The method illustrated in FIG. 4 will be discussed in reference to Table 9.

[0126] Prior to step 410, each computer can store a first secret value share x_j from a first secret value x of a first replicated secret share $\llbracket x \rrbracket^R$ and a second secret value share y_j from a second secret value y of a second replicated secret share $\llbracket y \rrbracket^R$.

[0127] At step 410, each computer can sample a first pseudorandom key $(s_{i,j})$ and a second pseudorandom key (s_j) . The first pseudorandom key $(s_{i,j})$ can be jointly sampled by a pair of computers (P_i and P_j). The second pseudorandom key (s_j) can be sampled by the set of computers in each set (D_j). The sampling of keys is discussed in a setup phase in Table 9.

[0128] Each computer can locally store a counter that can be incremented, decremented, and reset equally by each computer. The counters can allow the computers to remain in sync in regards to pseudorandom function generators and any other processes that utilize an incremental value.

[0129] At step 420, after generating the first pseudorandom key $(s_{i,j})$ and the second pseudorandom key (s_j) , each computer can generate replicated shares of a first random value (r) and a second random value (r') .

[0130] The first random value (r) can be sampled by each computer to obtain a first random value share (r_j) . The first random value share (r_j) can be generated using a pseudorandom function generator based on the second pseudorandom key (s_j) . The pseudorandom function generator can also accept the counter as input.

[0131] The second random value (r') can be generated by each computer to obtain a second random value share (r'_j) based on the first random value share (r_j) and a decimal bit value d that can denote a decimal value. In particular, the second random value share (r'_j) can be determined by dividing the first random value share (r_j) by the decimal bit value d .

[0132] At step 430, after generating the random value shares, the computers can generate a replicated sharing of zero (or an n -out-of- n secret sharing of zero). The replicated sharing of zero can be created based on the first pseudorandom keys $(s_{i,j})$. The replicated sharing of zero is represented by β , where each β_i lies in the ring Z_{2k} . In particular, the replicated sharing of zero

(β_i) can be generated by performing the following : define $\gamma_{i,j} := \widetilde{\text{PRF}}_{s_{j,i}}(cnt)$ for $j \in [1, i]$ and $\gamma_{i,j} := -\widetilde{\text{PRF}}_{s_{i,j}}(cnt)$ for $j \in (i, n]$, and then set $\beta_i := \sum_{j \in [n]/i} \gamma_{i,j}$.

[0133] At step 440, after generating the replication sharing of zero, the computers can generate a masked share (z'_i) . The masked share (z'_i) can provide the combination of values (x, y) that is masked by the share of zero. The masked share (z'_i) can be generated based on the first secret value share x_j , the second secret value share y_j , the first random value share (r_j) , and the replicated sharing of zero (β_i) . In particular, each computer can compute:

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

[0134] In some embodiments, the computers can jointly sample replicated secret shares of a MAC key $([\alpha]^R)$, where α is randomly sampled. The MAC α can be added with the masked share z' . In instances where a covert adversary may be included as a computer, the masked share z' with an added MAC α can be processed using a batch check process (e.g., at 470) to determine that the MAC $\alpha z'$ was correctly generated, detecting whether a computer cheated or did not entirely follow the protocol.

[0135] At step 450, after generating the masked share (z'_i) , a set shared masked share (z') (which can be a set shared value that is shared by each computer in the set) can be determined by each computer in each set of computers (D_i) , which can be shared across the computers in the set of computers (D_i) . The computers may observe that the masked shares may appear to be sufficiently random to all of the computers in the set of computers. For example, each computer can share their generated masked share (z'_i) with the computer's set of computers. Upon receiving each masked share (z'_i) from the computers in the set of computers, each computer in the set of computers can obtain the set shared masked share (z') from the plurality of masked shares (z'_i) .

[0136] At step 460, after obtaining the set shared masked share (z') , each computer can generate replicated secret shares $[[z]]^R$. This can be performed using the set shared masked shares (z') , the first random value share (r_j) , and the decimal bit value (d) . In some embodiments,

the first computer can determine the first replicated secret share z_1 based on the set shared masked share (z'), the decimal bit value (d) and the first share of the first random value (r_1). For example, the first computer can determine the first replicated secret share z_1 by performing $z_1 := z'/d - r_1/d$. The other computers of the plurality of computers can generate their shares of the replicated secret share $[[z]]^R$ based on their respective shares of the first random value share (r_j) and the decimal bit value (d). For example, the other computers can determine their replicated secret share z_j by performing $z_j := -r_j/d$.

[0137] In some embodiments, the replicated secret shares $[[z]]^R$ can be generated based on removing a MAC key α from a received masked replicated secret share $[[az]]^R$.

[0138] In some embodiments, at step 470, replicated secret shares of the MAC key ($[[\alpha]]^R$) can be checked for consistency using a batch check protocol to ensure that masked shares (z') were correctly opened using ($[[az']]^R$). Executing the batch check protocol can identify any adversarial parties that attempted to provide a masked value z' maliciously, as such an adversarial computer would not know the MAC key (α).

[0139] For example, in some embodiments, can perform a MAC check to further increase security of the system. Consider the 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 computers hold replicated secret shares $[[x]]^R, [[y]]^R$ in an extended ring $\mathbb{S}_{2^k}$ where $k \geq (k' + 2\lambda + \log_2(m))$. The computers can attempt to compute a replicated secret share $[[z]]^R$ where the output value $z \approx \frac{xy}{2^d} \in \mathbb{S}_{2^{k'}}$. The protocol works among n computers $P_1, \dots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \dots, D_m \subset [n]$ be the m distinct subsets of computers of size $(n-t)$. In some embodiments, prior to the execution of the fixed point multiplication protocol (and as part of the overall MPC protocol), the computers can jointly sample replicated secret shares of a MAC key $[[\alpha]]^R$, where α is randomly sampled from $\mathbb{S}_{2^\lambda}$. Then, the computers generate shares of the MAC of one of the inputs $[[\alpha x]]^R$ using the protocol in Table 9.

[0140] Embodiments can instantiate functionality $\mathcal{F}_{\text{MAC.Check}}$ using the “MAC Check” protocol in Figure 10 of [23].

$$\Pi_{\text{RShare-Mul-Mal}}(\llbracket a \rrbracket^R, \llbracket x \rrbracket^R, \llbracket ax \rrbracket^R, \llbracket y \rrbracket^R, d)$$

- The protocol outputs $\llbracket z \rrbracket^R, \llbracket az \rrbracket^R$ where $z \approx xy/d$.
- Let $\text{PRF}: \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \mathbb{S}_{2^{k-\log_2(m)-\lambda}}$ and $\widetilde{\text{PRF}}: \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \mathbb{S}_{2^k}$ be pseudorandom functions.
- Let $A_1, \dots, A_n \subset [m]^2$ 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 $B_1, \dots, B_n \subset [m]$ 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 $\text{cnt} := 0$.

Multiply. Upon input shares $\llbracket a \rrbracket^R, \llbracket x \rrbracket^R, \llbracket ax \rrbracket^R, \llbracket y \rrbracket^R \in \mathbb{S}_{2^k}$, parties compute the product as follows:

1. The parties sample $\llbracket r \rrbracket^R, \llbracket r' \rrbracket^R$ such that $r_j := \text{PRF}_{s_j}(\text{cnt})$ and $r'_j := r_j/d$. Compute $\llbracket ar \rrbracket^R, \llbracket ar' \rrbracket^R$ using the multiplication protocol in Table 7.
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 $\gamma_{i,j} := \widetilde{\text{PRF}}_{s_{j,i}}(\text{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\text{PRF}}_{s_{i,j}}(\text{cnt})$ 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 $\llbracket x \rrbracket_i^R = \{x_j | i \in D_j\}$ (resp. $\llbracket y \rrbracket^R$).

4. The n -out-of- n secret sharing z' is revealed to all parties.
5. Invoke functionality $\mathcal{F}_{\text{MAC.Check}}$ (Table 10). Each P_i inputs $(z', (\alpha z')_i, \alpha_i^*)$ where α_i^* is the

share of $\llbracket \alpha \rrbracket^R$ held by P_i corresponding to set D_i. Abort the protocol if it returns “Abort”. 6. The parties increment $\text{cnt} := \text{cnt} + 1$ and output $\llbracket \alpha z \rrbracket^R, \llbracket z \rrbracket^R$ where $z_1 := z'/d - r_1/d \text{ and } z_j := -r_j/d \text{ for } j \in [2, m]$ $\llbracket \alpha z \rrbracket^R := \llbracket \alpha \rrbracket^R (z'/d) - \llbracket \alpha r' \rrbracket^R$
--

Table 9: Protocol for fixed point multiplication with replicated secret shares against malicious adversaries

$\mathcal{F}_{\text{MAC.Check}}$ 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 $\sigma \neq (\alpha \cdot z')$.
--

Table 10: Ideal functionality for checking the MAC on a secret shared value.

$\Pi_{\text{RShare-MPC-Mal}}$ - The protocol works among n parties $P_1, \dots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m := \binom{n}{n-t}$ and $D_1, \dots, D_m \subset [n]$ 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 $\llbracket \alpha \rrbracket^R$. Each party P_i gets $\llbracket \alpha \rrbracket_i^R := \{\alpha_i \mid i \in D_j\}$. Then, run the setup of protocol $\Pi_{\text{RShare-Mul}}$ in Table 6. Share: Every party $P \in \{P_1, \dots, P_n\}$ shares an input value $x \in \mathbb{S}_{2^{k'}}$ as follows: 1. Sample $x_1, \dots, x_m \leftarrow \mathbb{S}_{2^k}$ 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 $\llbracket x \rrbracket_i^R := \{x_i \mid i \in D_j\}$. 3. Run any MPC protocol (e.g. [28]) to compute functionality $\mathcal{F}_{\text{Range}}$ (Table 12) where P_i's input is $\llbracket x \rrbracket_i^R$. 4. Run the multiplication protocol $\Pi_{\text{RShare-Mul}}$ (Fig 10) where each P_i's input is
--

$(\llbracket x \rrbracket_i^R, \llbracket \alpha \rrbracket_i^R)$ and output is $\llbracket \alpha x \rrbracket_i^R$. Linear operations: For any constant c and share $(\llbracket y \rrbracket^R, \llbracket \alpha y \rrbracket^R)$, parties perform linear operations on share $(\llbracket x \rrbracket^R, \llbracket \alpha x \rrbracket^R)$ to compute the output $(\llbracket z \rrbracket^R, \llbracket \alpha z \rrbracket^R)$ as follows: 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 $j \in [m]/1$ Addition of shares: $z_j = x_j + y_j, (\alpha z)_j = (\alpha x)_j + (\alpha y)_j$ for $j \in [m]$ Multiplication by a constant c: $z_j = cx_j, (\alpha z)_j = c \cdot (\alpha x)_j$ for $j \in [m]$ Fixed point multiplication: For any two shares $\llbracket x \rrbracket^R, \llbracket y \rrbracket^R$, parties perform fixed point multiplication by running the multiply phase of $\Pi_{\text{RShare-Mul}}$ in Table 9. . Output reconstruction: Each party P_i holds $\llbracket z \rrbracket_i^R$ and publicly reconstructs z as follows: - For $j \in [m]$, $(t + 1)$ of the parties in set D_j reveal z_j to everyone (can order each set and the first $(t + 1)$ parties speak). - Abort if there is inconsistency in any z_j. Else, compute output $z = \sum_{i \in [m]} z_i$.

Table 11: Protocol for MPC with replicated secret sharing against malicious adversaries.

[0141] In addition to the protocol for semi-honest security, embodiments can 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, there is a guarantee that the term z' was correctly computed by the adversary. Finally, after computing $\llbracket z \rrbracket^R$ as in the semi-honest protocol, embodiments can now also compute the MAC $\llbracket \alpha z \rrbracket^R$.

V. SHAMIR SECRET SHARING PROTOCOL FOR MULTI-PARTY FIXED POINT MULTIPLICATION

[0142] Some embodiments can provide for a semi-honest protocol for fixed point multiplication with Shamir secret sharing.

[0143] FIG. 5 illustrates a block diagram 500 of an example Shamir sharing protocol for multi-party fixed point multiplication.

[0144] Prior to step 510, each computer can obtain two fixed point values, a first secret value x and a second secret value y , where $x, y \in \mathbb{S}_{2^{k'}}$. The first secret value x and the second secret value y can be held by computers in a network comprising a plurality of computers via Shamir secret sharing $[[x]]^S, [[y]]^S$ on a field $\mathbb{F}_q$, where $q \geq 2^{k'+\lambda} \cdot (t+1)$. As such, each computer can hold one share of the first secret value and one share of the second secret value. For each multiplication that occurs between secret values (e.g., when multiplying the first secret value x and the second secret value y), the computers can attempt to learn a Shamir secret share of the product $[[z]]^S$, where $z \approx xy/d$. This method will be described in reference to Table 13.

[0145] For each fixed point multiplication, each computer of the plurality of computers can first perform preprocessing where the computer obtains a $2t$ -out-of- n Shamir secret sharing of a first random value $[[r]]^{S, 2t}$ in the field $\mathbb{F}_q$ and a t -out-of- n Shamir secret sharing of a second random value $[[r']]^S$ in the field $\mathbb{F}_q$. The first random value r is mathematically related to the second random value r' , in particular where $r' = r/d$. The first random value r and the second random value r' are generated 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 computers in the network. There is no overflow in the summation due to sampling r_i from $\mathbb{S}_{2^{k'+\lambda}}$. Once the double sharing with truncation is generated, the rest of the protocol follows from the multiplication techniques for Shamir secret sharing.

[0146] In particular, at step 510, a computer (P_i) can sample a random value (r_i) and generate a Shamir share of a first random value (r_i) to obtain a first random share ($[[r_i]]^{S, 2t}$). Each computer in the network of computers can generate the first random share ($[[r_i]]^{S, 2t}$) of the random value (r_i) and a second random share ($[[r_i/d]]^S$), where the first random share ($[[r_i]]^{S, 2t}$) is determined using $2t$ -out-of- n sharing. The second random share ($[[r_i/d]]^S$) can be generated based on the first random share and a decimal bit value (d).

[0147] At step 520, after generating the first random share ($[[r_i]]^{S, 2t}$), each computer can send the first random share ($[[r_i]]^{S, 2t}$) and the second random share ($[[r_i/d]]^S$) to the other computers in the computer network.

[0148] At step 530, after receiving each share of the first random share ($[[r_i]]^{S, 2t}$) and each share of the second random share ($[[r_i/d]]^S$), each computer can obtain a t -out-of- n Shamir share of a third random share ($[[r]]^{S, 2t}$) and a fourth random share ($[[r']]^S$).

[0149] The third random share ($[[r]]^{S, 2t}$) can be generated based on each share of the first random share ($[[r_i]]^{S, 2t}$). For example, the third random share ($[[r]]^{S, 2t}$) can be equal to a summation of each share of the first random share ($[[r_i]]^{S, 2t}$).

[0150] The fourth random share ($[[r']]^S$) can be generated based on each share of the second random share ($[[r_i/d]]^S$). For example, the fourth random share ($[[r']]^S$) can be equal to a summation of each share of the second random share ($[[r_i/d]]^S$).

[0151] At step 540, each computer can compute masked shares ($[[z']]^{S, 2t}$). The masked shares can provide a fixed-point multiplication (of x, y) masked with the third random share ($[[r]]^{S, 2t}$). Each computer can determine the masked share ($[[z']]^{S, 2t}$) based on the first secret share ($[[x]]^S$), the second secret share ($[[y]]^S$), and the third random share ($[[r]]^{S, 2t}$). For example, each computer can determine the masked share ($[[z']]^{S, 2t}$) by multiplying the first secret share ($[[x]]^S$) and the second secret share ($[[y]]^S$) and then adding in the third random share ($[[r]]^{S, 2t}$).

[0152] At step 550, each computer in the network of computers can send the masked share ($[[z']]^{S, 2t}$) to a leader computer (e.g., P_1), which can, in turn, determine the masked value (z') from the masked shares ($[[z']]^{S, 2t}$). The leader computer can then send the masked value (z') to all other computers in the network. The masked value z' may appear sufficiently random to other receiving computers.

[0153] Specifically, a first computer P_1 can learn a masked value $z' = x \cdot y + r$ and send it to all the other computers of the plurality of computers. Although the first random value r is not entirely uniform in the field $\mathbb{F}_q$, 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 masked value z' looks sufficiently random to all the computers.

[0154] At step 560, each computer can then output a Shamir secret share $[[z]]^S$ using the received value z' and Shamir share $[[r']]^S$. By the correctness of double sharing with truncation

for the first random value r and the second random value r' , all of the computers can obtain the sharing of $\llbracket z \rrbracket^S$ by taking $z'/d - \llbracket r' \rrbracket^S$.

$\mathcal{F}_{\text{Range}}$
Parameters: Ring $\mathbb{S}_{2^k}$ and a value $k' < k$. Input: Each party P_i has input $\llbracket x \rrbracket_i^R$. Computation: Reconstruct x from the shares $\{\llbracket x \rrbracket_i^R\}_{i \in [n]}$. Output “Abort” to all parties if $x > 2^{k'}$.

Table 12: Ideal functionality for checking the range of a secret shared value

$\Pi_{\text{SShare-Mul}}(\llbracket x \rrbracket^R, \llbracket y \rrbracket^R, d)$
- The protocol outputs $\llbracket z \rrbracket^S$ 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 $\llbracket r_i \rrbracket^{S,2t}, \llbracket r_i/d \rrbracket^S$ in $\mathbb{F}_q$, and sends the shares to the other parties. 2. All parties locally compute $r^{S,2t} := \sum_{i \in [t+1]} \llbracket r_i \rrbracket^{S,2t}, \llbracket r' \rrbracket^S := \sum_{i \in [t+1]} \llbracket r_i/d \rrbracket^S$. Multiply. Upon input shares $\llbracket x \rrbracket^S, \llbracket y \rrbracket^S \in \mathbb{F}_q$, parties compute the product as follows: 1. All parties locally compute $\llbracket z' \rrbracket^{S,2t} := \llbracket x \rrbracket^S \cdot \llbracket y \rrbracket^S + \llbracket r \rrbracket^{S,2t}$ and reveal z' to P_1. 2. P_1 sends z' to all the parties. 3. All parties output $\llbracket z \rrbracket^S := z'/d - \llbracket r' \rrbracket^S$.

Table 13: Semi-honest protocol for fixed point multiplication with Shamir secret shares.

[0155] The truncation technique, according to embodiments, 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 $t < n$.

[0156] In some embodiments, the computers can implement a semi-honest protocol for fixed point multiplication with Shamir secret sharing. The Shamir sharing protocol can include fixed point values (x, y) in two-complement form, where the computers hold Shamir secret shares $[[x]]^R$ and $[[y]]^R$. For each fixed point multiplication, the computers may perform the protocol to obtain a Shamir secret share $[[z]]^S$.

[0157] The MPC protocol with Shamir secret sharing leveraging the aforementioned fixed point multiplication protocol, presented in Table 14, can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation.

$\Pi_{\text{SShare-MPC}}$	
- The protocol works among n parties $P_1, \dots, P_n$ 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 $[[x]]^S$ in the field $\mathbb{F}_q$ and sends the shares to all the parties.
Linear operations:	For any constant c and shares $[[x]]^S, [[y]]^S$, parties perform linear operations on share $[[x]]^S$ as follows: 1. Addition by a constant: $[[x + c]]^S = [[x]]^S + c$ 2. Addition of shares: $[[x + y]]^S = [[x]]^S + [[y]]^S$ 3. Multiplication by a constant: $[[cx]]^S = c \cdot [[x]]^S$
Fixed point multiplication:	For any two shares $[[x]]^S, [[y]]^S$, parties perform fixed point multiplication as in Table 13
Reconstruct the output:	Each party P_i holds $[[z]]_j^S$ and publicly reconstructs z as follows: 1. For $j \in [2, \dots, t + 1]$, party P_j sends $[[z]]_j^S$ 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.

Table 14: Protocol for MPC with Shamir secret sharing.

VI. ADDITIVE SECRET SHARING FOR MULTI-PARTY FIXED POINT MULTIPLICATION

[0158] Embodiments provide for a semi-honest protocol for fixed point multiplication with additive secret sharing. Consider two fixed point values including a first secret value x and a second secret value y represented in twos-complement form in $\mathbb{S}_{2^{k'}}$. Computers in a network of computers hold first additive secret shares $\llbracket x \rrbracket^A$ and second additive secret shares $\llbracket y \rrbracket^A$, of the first secret value x and the second secret value y , respectively, in an extended ring $\mathbb{S}_{2^k}$ where $k \geq k' + \lambda + \log_2 n$. Further, it can be assumed that the computers hold preprocessed Beaver triples of the form $\llbracket \alpha \rrbracket^A, \llbracket \beta \rrbracket^A, \llbracket \gamma \rrbracket^A$ in $\mathbb{S}_{2^k}$ where $\alpha, \beta \xleftarrow{\$} \mathbb{S}_{2^k}$ and $\gamma = \alpha \cdot \beta$. The Beaver triples can include a secret share ($\llbracket \alpha \rrbracket^A$) of a third value (α), a secret share ($\llbracket \beta \rrbracket^A$) of a fourth value (β), and a secret share ($\llbracket \gamma \rrbracket^A$) of a fifth value (γ). For each fixed point multiplication, the computers can attempt to determine an additive secret share $\llbracket z \rrbracket^A$ where $z \approx xy/d$, which is formally described with respect to Table 15.

[0159] FIG. 6 illustrates a block diagram of an example additive sharing protocol for multi-party fixed point multiplication according to an embodiment of the present disclosure. The method illustrated in FIG. 6 can be performed by a first computer in a network comprising a plurality of computers.

[0160] Prior to step 602, each computer of the plurality of computers can obtain secret shares of a plurality of secret values and a decimal bit value (d). The secret shares can include a secret share ($\llbracket x \rrbracket^A$) of a first value (x), a secret share ($\llbracket y \rrbracket^A$) of a second value (y), a secret share ($\llbracket \alpha \rrbracket^A$) of a third value (α), a secret share ($\llbracket \beta \rrbracket^A$) of a fourth value (β), and a secret share ($\llbracket \gamma \rrbracket^A$) of a fifth value (γ). The computers of the plurality of computers can attempt to multiply the first value (x) and the second value (y). However, since each computer only stores the secret share ($\llbracket x \rrbracket^A$) of the first value (x) and the secret share ($\llbracket y \rrbracket^A$) of the second value (y), the computers can communicate with one another to perform the additive sharing protocol for multiparty fixed point multiplication in a secret sharing manner according to steps 602-620.

[0161] The shares held (e.g., stored) by each computer can be unique shares that differ from computer to computer. Therefore, when a share of a value is referred to, it is understood that the share of the value is different at different computers.

[0162] The secret share of the third value (α), the secret share of the fourth value (β), and the secret share of the fifth value (γ) make up a preprocessed Beaver triple. The first value (x) and the second value (y) are fixed point values represented in twos-complement form in a ring ($\mathbb{S}_{2^{k'}}$).

[0163] At step 602, each computer of the plurality of computers can determine a share of a first shared random value ($\llbracket r' \rrbracket^A$) and a share of a second shared random value ($\llbracket r \rrbracket^A$). In some embodiments, the determination of the share of the first shared random value ($\llbracket r' \rrbracket^A$) and the share of the second shared random value ($\llbracket r \rrbracket^A$) can be performed in a preprocessing step.

[0164] The computers can determine the first shared random value ($\llbracket r' \rrbracket^A$) and the share of the second shared random value ($\llbracket r \rrbracket^A$) as follows. Each computer can sample an extended ring ($\mathbb{S}_{2^{k'+\lambda}}$) to generate shares of a first initial random value ($\llbracket r_i \rrbracket^A$) and shares of a second initial random value ($\llbracket r_i/d \rrbracket^A$). The shares of the second initial random value ($\llbracket r_i/d \rrbracket^A$) can be created based on the shares of the first initial random value ($\llbracket r_i \rrbracket^A$) and the decimal bit value (d). Each computer can then send the shares of the first initial random value ($\llbracket r_i \rrbracket^A$) and the shares of the second initial random value ($\llbracket r_i/d \rrbracket^A$) to the plurality of computers. After receiving the shares of the first initial random value ($\llbracket r_i \rrbracket^A$) and the shares of the second initial random value ($\llbracket r_i/d \rrbracket^A$), each computer can determine shares of the first shared random value ($\llbracket r' \rrbracket^A$) based on the shares of the second initial random value ($\llbracket r_i/d \rrbracket^A$) (e.g., $\llbracket r' \rrbracket^A = \sum_{i \in [t+1]} \llbracket r_i/d \rrbracket^A$). Each computer can determine a share of the second shared random value ($\llbracket r \rrbracket^A$) based on the shares of the first shared random value ($\llbracket r' \rrbracket^A$) (e.g., $\llbracket r \rrbracket^A = \sum_{i \in [t+1]} \llbracket r' \rrbracket^A$).

[0165] At step 604, after determining the shares of the first shared random value ($\llbracket r' \rrbracket^A$) and the second shared random value ($\llbracket r \rrbracket^A$), each computer of the plurality of computers can determine a first intermediate share ($\llbracket x' \rrbracket^A$) based on the secret share ($\llbracket x \rrbracket^A$) of the first value (x) and the secret share ($\llbracket \alpha \rrbracket^A$) of the third value (α). For example, each computer can determine the first intermediate share ($\llbracket x' \rrbracket^A$), held by that computer, by adding the secret share ($\llbracket x \rrbracket^A$) of the

first value (x) and the secret share ($\llbracket \alpha \rrbracket^A$) of the third value (α) (e.g., $\llbracket x' \rrbracket^A := \llbracket x \rrbracket^A + \llbracket \alpha \rrbracket^A$). The addition of the secret share ($\llbracket \alpha \rrbracket^A$) of the third value (α) can mask the secret share ($\llbracket x \rrbracket^A$) of the first value (x).

[0166] At step 606, after determining the first intermediate share ($\llbracket x' \rrbracket^A$), each computer of the plurality of computers can determine a second intermediate share ($\llbracket y' \rrbracket^A$) based on the secret share ($\llbracket y \rrbracket^A$) of the second value (y) and the secret share ($\llbracket \beta \rrbracket^A$) of the fourth value (β). For example, each computer can determine the second intermediate share ($\llbracket y' \rrbracket^A$), held by that computer, by adding the secret share ($\llbracket y \rrbracket^A$) of the second value (y) and the secret share ($\llbracket \beta \rrbracket^A$) of the fourth value (β) (e.g., $\llbracket y' \rrbracket^A := \llbracket y \rrbracket^A + \llbracket \beta \rrbracket^A$). The addition of the secret share ($\llbracket \beta \rrbracket^A$) of the fourth value (β) can mask the secret share ($\llbracket y \rrbracket^A$) of the second value (y).

[0167] At step 608, after each computer of the plurality of computers determines the first intermediate share ($\llbracket x' \rrbracket^A$) and the second intermediate share ($\llbracket y' \rrbracket^A$), each computer can send the first intermediate share ($\llbracket x' \rrbracket^A$) and the second intermediate share ($\llbracket y' \rrbracket^A$) to the first computer. In some embodiments, the first computer can be referred to as a leader computer.

[0168] The first computer can receive each of the first intermediate shares ($\llbracket x' \rrbracket^A$) of the first intermediate value (x') from the network of computers, thus revealing the first intermediate value (x') in the clear to the first computer. The first computer can receive each of the second intermediate shares ($\llbracket y' \rrbracket^A$) of the second intermediate value (y') from the network of computers, thus revealing the second intermediate value (y') in the clear to the first computer.

[0169] The first computer can reconstruct the first intermediate value (x') and the second intermediate value (y') as described in detail herein according to additive secret sharing techniques.

[0170] At step 610, after receiving each of the first intermediate shares ($\llbracket x' \rrbracket^A$) and each of the second intermediate shares ($\llbracket y' \rrbracket^A$), the first computer can reveal the first intermediate value (x') and the second intermediate value (y') to the other computers of the plurality of computers.

[0171] At step 612, after each computer of the plurality of computers receives the first intermediate value (x') and the second intermediate value (y'), each computer can determine a

third intermediate share ($\llbracket z' \rrbracket^A$). Each computer can determine the third intermediate share ($\llbracket z' \rrbracket^A$) based on the first intermediate value (x'), the second intermediate value (y'), the secret share ($\llbracket x \rrbracket^A$) of the first value (x), the secret share ($\llbracket y \rrbracket^A$) of the second value (y), the secret share ($\llbracket \gamma \rrbracket^A$) of the fifth value (γ) and the share of the second shared random value ($\llbracket r \rrbracket^A$). For example, each computer can determine the third intermediate share ($\llbracket z' \rrbracket^A$) as follows:

$$\llbracket z' \rrbracket^A := x' \cdot \llbracket y \rrbracket^A + y' \cdot \llbracket x \rrbracket^A + \llbracket \gamma \rrbracket^A - x' \cdot y' + \llbracket r \rrbracket^A$$

[0172] At step 614, after each computer determines the third intermediate share ($\llbracket z' \rrbracket^A$), each computer can send the third intermediate share ($\llbracket z' \rrbracket^A$) to the first computer. The first computer can receive each of the third intermediate shares ($\llbracket z' \rrbracket^A$) of the third intermediate value (z') from the network of computers, thus revealing the third intermediate value (z') in the clear to the first computer. The first computer can reconstruct the third intermediate value (z') as described in detail herein according to additive secret sharing techniques.

[0173] At step 616, after receiving each of the third intermediate shares ($\llbracket z' \rrbracket^A$), the first computer can reveal the third intermediate value (z') to the other computers of the plurality of computers. The first computer can send the third intermediate value (z') to the other computers in any suitable manner.

[0174] At step 618, after each computer of the plurality of computers receives the third intermediate value (z'), each computer can determine an output secret share ($\llbracket z \rrbracket^A$) based on the third intermediate value (z'), the decimal bit value (d), and the share of the first shared random value ($\llbracket r' \rrbracket^A$). For example, each computer can divide the third intermediate value (z') by the decimal bit value (d) then subtract off the share of the first shared random value ($\llbracket r' \rrbracket^A$). In particular, each computer can perform $\llbracket z \rrbracket^A := z'/d - \llbracket r' \rrbracket^A$.

[0175] The output secret share ($\llbracket z \rrbracket^A$), which is a secret sharing of an output value (z), can be a secret share that, when combined into the output value (z), is equal to the first value (x) multiplied by the second value (y).

$$\Pi_{\text{AShare-Mul}}(\llbracket x \rrbracket^A, \llbracket y \rrbracket^A, \llbracket \alpha \rrbracket^A, \llbracket \beta \rrbracket^A, \llbracket \gamma \rrbracket^A, d)$$

- The protocol outputs $\llbracket z \rrbracket^A$ 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 $\llbracket r_i \rrbracket^A, \llbracket r_i/d \rrbracket^A$, and sends the shares to the other parties.
2. All parties locally compute $\llbracket r \rrbracket^A = \sum_{i \in [t+1]} \llbracket r_i \rrbracket^A, \llbracket r' \rrbracket^A = \sum_{i \in [t+1]} \llbracket r_i/d \rrbracket^A$.

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

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

Table 15: Semi-honest protocol for fixed point multiplication with additive secret shares.

[0176] The multiplication protocol is multiplication with Beaver triples followed by the 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.

[0177] An MPC protocol with additive secret sharing leveraging the fixed point multiplication protocol, according to embodiments, is presented in Table 16. Such a method can be used to compute any arithmetic circuit where multiplications are fixed point multiplication with truncation.

$\Pi_{\text{AShare-MPC}}$	
- The protocol works among n parties $P_1, \dots, P_n$ 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 $\llbracket x \rrbracket^A$ and sending the shares to the other parties.	
Linear operations: For any constant c and shares $\llbracket x \rrbracket^A, \llbracket y \rrbracket^A$, parties perform linear operations on share $\llbracket x \rrbracket^A$ as follows: 1. Addition by a constant: $\llbracket x + c \rrbracket^A = \llbracket x \rrbracket^A + c$ (only P_1 adds c on its share) 2. Addition of shares: $\llbracket x + y \rrbracket^A = \llbracket x \rrbracket^A + \llbracket y \rrbracket^A$ 3. Multiplication by a constant: $\llbracket cx \rrbracket^A = c \cdot \llbracket x \rrbracket^A$	
Fixed point multiplication: For any two shares $\llbracket x \rrbracket^A, \llbracket y \rrbracket^A$, parties perform fixed point multiplication as in Table 15.	
Reconstruct the output: Each party P_i holds $\llbracket z \rrbracket_i^A$ and publicly reconstructs z as follows: 1. Each party P_i sends its share $\llbracket z \rrbracket_i^A$ to P_1. 2. P_1 recovers z from the shares and sends z to all parties.	

Table 16: Protocol for MPC with additive secret sharing

VII. PERFORMANCE COMPARISON

[0178] The performance of the replicated and Shamir based fixed point multiplication methods according to embodiments will be compared with that of existing work. In particular, embodiments will be compared with [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].

Protocol	Operation	Online Rounds	Online Comm	Offline Comm
Embodiments, Table 6	$\llbracket x \rrbracket^R \cdot \llbracket y \rrbracket^R$	1 or 2	$(n^2 - nt)k$ or $2nk$	0
[36]	$\llbracket x \rrbracket^R \cdot \llbracket y \rrbracket^R$	1	$(n^2 - nt)k$	$(n^2 - nt)tk$

Embodiments, Table 13	$\llbracket x \rrbracket^S \cdot \llbracket y \rrbracket^S$	2	$2nk$	$(n - t - 1)(t + 1)k$
[16]	$\llbracket x \rrbracket^S \cdot \llbracket y \rrbracket^S$	2	$2nk$	$2k^2n$

Table 14: Communication (bits) and round complexity of the method according to embodiments compared to [16, 36]. For Shamir, embodiments do not include the cost of generating beaver triples, if applicable.

A. *Replicated Secret Sharing*

[0179] Mohassel and Zhang [37] study two party secret sharing over $\mathbb{S}_{2k}$ and perform stand-alone multiplication and truncation operations in a single round of communication. Methods according to embodiments (if seen in a limiting two-party setting) are similarly performative to theirs.

[0180] 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.

[0181] Methods according to embodiments eliminate the need to emulate the two party protocol within a binary protocol. As suggested in Table 14, methods according to embodiments, send approximately t or $(n - t)t/2$ times less data, depending on how the masked value z' is revealed. In particular, all the shares of the masked value z' can either be sent to all $(n - t)$ parties in a set D_1 resulting in a single round protocol or a single party can receive them and then send them to the remaining parties in the set D_1 at the cost of an extra round. Regardless, methods according to embodiments requires significantly less communication than [36],

especially in the offline phase where methods according to embodiments are completely non-interactive.

B. Shamir Secret Sharing

[0182] Methods according to embodiments that are utilized for Shamir secret sharing are also a significant improvement over the work of Catrina and Saxena [16]. The primary difference is that the protocol of [16] requires preprocessing secret shares of k bits, $\llbracket r_1 \rrbracket^S, \dots, \llbracket r_k \rrbracket^S$ 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.

[0183] Methods according to embodiments 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 (e.g., methods according to embodiments 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, see Table 14. In practice, embodiments would typically expect $n, t \ll k$ and so, this represents a significant reduction in communication.

VIII. FURTHER COMPUTER SYSTEM DETAILS

[0184] Any of the computer systems mentioned herein may utilize any suitable number of subsystems. Examples of such subsystems are shown in FIG. 7 in computer system 700. In some embodiments, a computer system includes a single computer apparatus, where the subsystems can be the components of the computer apparatus. In other embodiments, a computer system can include multiple computer apparatuses, each being a subsystem, with internal components. A computer system can include desktop and laptop computers, tablets, mobile phones and other mobile devices.

[0185] The subsystems shown in FIG. 7 are interconnected via a system bus 701. Additional subsystems such as a printer 708, keyboard 716, storage device(s) 718, monitor 722 (e.g., a display screen, such as an LED), which is coupled to display adapter 712, and others are shown.

Peripherals and input/output (I/O) devices, which couple to I/O controller 702, can be connected to the computer system by any number of means known in the art such as input/output (I/O) port 714 (e.g., USB, FireWire[®]). For example, I/O port 714 or external interface 720 (e.g. Ethernet, Wi-Fi, etc.) can be used to connect computer system 700 to a wide area network such as the Internet, a mouse input device, or a scanner. The interconnection via system bus 701 allows the central processor 706 to communicate with each subsystem and to control the execution of a plurality of instructions from system memory 704 or the storage device(s) 718 (e.g., a fixed disk, such as a hard drive, or optical disk), as well as the exchange of information between subsystems. The system memory 704 and/or the storage device(s) 718 may embody a computer readable medium. Another subsystem is a data collection device 710, such as a camera, microphone, accelerometer, and the like. Any of the data mentioned herein can be output from one component to another component and can be output to the user.

[0186] A computer system can include a plurality of the same components or subsystems, e.g., connected together by external interface 720, by an internal interface, or via removable storage devices that can be connected and removed from one component to another component. In some embodiments, computer systems, subsystem, or apparatuses can communicate over a network. In such instances, one computer can be considered a client and another computer a server, where each can be part of a same computer system. A client and a server can each include multiple systems, subsystems, or components.

IX. PROOFS

[0187] Security proofs will now be provided for various methods described herein. The proofs are provided to illustrate the improvements provided embodiments as well as the security of the methods and systems. However, it is understood that embodiments are not limited by the following proofs.

A. Replicated Semi-Honest: Proof

[0188] Embodiments now prove that the protocol $\Pi_{\text{RShare-MPC-SH}}$ securely realizes the ideal functionality $\mathcal{F}_{\text{Cd}}$ 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, embodiments prove the following theorem:

[0189] Theorem 5.1 Assuming one way functions exist, protocol $\Pi_{\text{RShare-MPC-SH}}$ presented in Table 15 securely computes $\mathcal{F}_{\text{Cd}}$ (Table 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.

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

The protocol works among n parties $P_1, \dots, P_n$ over an extended ring $\mathbb{S}_{2^k}$. Let $m = \binom{n}{n-1}$ and $D_1, \dots, D_m \subset [n]$ 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 $\Pi_{\text{RShare-Mul}}$ in Table 6.

Share: Every party $P \in \{P_1, \dots, P_n\}$ shares an input value $x \in \mathbb{S}_{2^{k'}}$ as follows:

1. Sample $x_1, \dots, x_m \leftarrow \mathbb{S}_{2^k}$ 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 $\llbracket x \rrbracket_i^R = \{x_i \mid i \in D_j\}$

Linear operations: For any constant c and share $\llbracket y \rrbracket^R$, parties perform linear operations on share $\llbracket x \rrbracket^R$ to compute the output $\llbracket z \rrbracket^R$ 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 $\llbracket x \rrbracket^R, \llbracket y \rrbracket^R$, parties perform fixed point multiplication by running the multiply phase of $\Pi_{\text{RShare-Mul}}$ in Table 6.

Output reconstruction: Each party P_i holds $\llbracket z \rrbracket_i^R$ 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.

Table 15: Protocol for MPC with replicated secret sharing against semi-honest adversaries.

[0190] 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 $P_{H_1}, \dots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{\text{Cd}}$ and the set $(\text{inp}_{i^*}, r_{i^*})_{P_{i^*} \in S^*}$ indicating the corrupt parties' inputs and randomness. The strategy of $\mathcal{S}$ is described below.

[0191] Share: On behalf of each honest party P_i , do:

[0192] 1. Sample $x_1, \dots, x_m \leftarrow \mathbb{S}_{2^k}$ such that $0 = \sum_{i \in [m]} x_i$.

[0193] 2. Send x_j to $\mathcal{A}$ for each corrupt party P_{i^*} where $i^* \in D_j$.

[0194] Linear Operations: These are local operations that don't need to be simulated.

[0195] Fixed point multiplication: $\mathcal{S}$ participates in the setup phase as in Table 6. For each multiplication, for any two shares $\llbracket x^R \rrbracket, \llbracket y^R \rrbracket$, $\mathcal{S}$ does:

[0196] 1. Recall that S_j is the set of parties indexed by D_j . Sample $\llbracket r^R \rrbracket$ 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 $r_j := \text{PRF}_{S_j}(\text{cnt})$ as in Table 6.

[0197] 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:

[0198] - If $P_j \in S^*$, define $\gamma_{i,j} := \widetilde{\text{PRF}}_{S_{j,i}}(\text{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\text{PRF}}_{S_{i,j}}(\text{cnt})$ for $j \in (i, n]$ as in Table 6.

[0199] - 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 Γ and set $\gamma_{i,j} = -\Gamma[i][j]$.

[0200] 3. Run Step 3 to Step 5.

[0201] **Output reconstruction:** S reconstructs output out as follows.

[0202] **If $P_1 \notin S^*$:**

[0203] 1. Receive a set of messages from $\mathcal{A}$.

[0204] 2. On behalf of P_1 , send output out to all other parties.

[0205] **If $P_1 \in S^*$:**

[0206] 1. Set $z^* = 0$.

[0207] 2. From $\mathcal{A}$'s inputs and randomness $(\text{inp}_{i^*}, r_{i^*})_{P_{i^*} \in S^*}$, 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 $\mathcal{A}$.

[0208] 3. Let $j_1, \dots, j_\ell \in [m]$ denote the indices for which $S_j \cap S^* = \emptyset$.

[0209] 4. Pick $z_{j_1}, \dots, z_{j_{\ell-1}} \leftarrow \mathbb{S}_{2^k}$. Compute $z_{j_\ell} \in \mathbb{S}_{2^k}$ as $z_{j_\ell} = \text{out} - \sum_{i \in [\ell-1]} z_{j_i} - z^*$.

[0210] 5. On behalf of the honest parties, send $\{z_{j_1}, \dots, z_{j_\ell}\}$ to $\mathcal{A}$.

[0211] Embodiments show that the above simulation strategy is successful via a hybrid argument.

[0212] Hyb_0 : Consider a simulator SimHyb that plays the role of the honest parties as in Table 15. This is the real world.

[0213] Hyb_1 : **Switching output.** This is similar to Hyb_0 except that SimHyb now runs the output reconstruction step as done by S in the ideal world by using the output out from the ideal functionality $\mathcal{F}_{\text{Cd}}$. Honest parties also get their output from $\mathcal{F}_{\text{Cd}}$ in this hybrid.

[0214] 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 $\{z_{j_1}, \dots, z_{j_\ell}\}$ 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.

[0215] Hyb₂: 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.

[0216] 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, embodiments can build a reduction $\mathcal{A}_{\text{PRF}}$ that breaks the security of the pseudorandom function PRF which is a contradiction. Thus, the two hybrids are computationally indistinguishable.

[0217] Hyb₃: 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 $\widetilde{\text{PRF}}$.

[0218] As in the previous case, the computational indistinguishability between Hyb_2 and Hyb_3 follows from the security of the pseudorandom function $\widetilde{\text{PRF}}$.

[0219] Hyb₄: 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.

[0220] 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 embodiments additionally include the linear operations.

[0221] It can be argued 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 $(z'_1, \dots, z'_n)$. For each honest party P_i , the z'_i it generates consists of a term of the product $(\sum_{(j,j') \in A_i} x_j y_{j'})$ 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, it can be argued 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

[0222] In this section, embodiments prove that the protocol $\Pi_{\text{RShare-MPC-Mal}}$ securely realizes the ideal functionality $\mathcal{F}_{\text{Cd}}$ 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, embodiments prove the following theorem:

[0223] Theorem 3. Assuming one way functions exist, protocol $\Pi_{\text{RShare-MPC-Mal}}$ presented in Table 11 securely computes $\mathcal{F}_{\text{Cd}}$ (Table 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.

[0224] 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 $P_{H_1}, \dots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the

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

[0225] 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 $\{\llbracket \alpha \rrbracket_i^R\}_{P_i \in H}$ on behalf of the honest parties. Then, on behalf of each honest party, run the setup of protocol $\Pi_{\text{RShare-Mul}}$ in Table 6 as in the real world.

[0226] Share: On behalf of each honest party P :

[0227] 1. Sample $x_1, \dots, x_m \leftarrow \mathbb{S}_{2k}$ such that $0 = \sum_{i \in [m]} x_i$. Send x_j to $\mathcal{A}$ for each corrupt party P_{i^*} where $i^* \in D_j$.

[0228] 2. Simulate the MPC protocol used to compute functionality $\mathcal{F}_{\text{Range}}$ (Table 12).

[0229] 3. **Switch PRF to random:** Run the multiplication protocol $\Pi_{\text{RShare-Mul}}$ (Table 7, Table 8) 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,

[0230] - In Step 1 of Table 7, on behalf of each honest party P_i , compute terms r'_j as follows: If $P_j \in S^*$, define $r'_{i,j} := \widetilde{\text{PRF}}_{S_{j,i}}(\text{cnt})$ for $j \in [1, i)$ and $r'_{i,j} := -\widetilde{\text{PRF}}_{S_{i,j}}(\text{cnt})$ for $j \in (i, n]$ as in Table 7. If $P_j \notin S^*$, and $j > i$, sample $r'_{i,j} \leftarrow \mathbb{S}_{2k}$ and store the value in $R'[i][j]$. If $P_j \notin S^*$, and $j < i$, look up $R'[i][j]$ and set $r'_{i,j} = -R'[i][j]$.

[0231] - In Step 1 of Table 8, for each $j \in (n, m]$, if $S_j \cap S^* = \emptyset$, sample $z_j \leftarrow \mathbb{S}_{2k}$. Else, set $z_j := \text{PRF}_{S_j}(\text{cnt})$ as in Table 8.

[0232] For each corrupt party P_{i^*} 's input sharing:

[0233] 1. Using the honest parties' shares $\{\llbracket x_{i^*} \rrbracket_i^R\}_{P_i \in H}$ and the simulator of the MPC protocol for $\mathcal{F}_{\text{Range}}$, extract x_{i^*} . If the extraction is unsuccessful, output "Extraction Abort".

[0234] 2. Query the ideal functionality $\mathcal{F}_{\text{Cd}}$ with inputs $\{x_{i^*}\}_{P_{i^*} \in S^*}$ and receive output out.

[0235] Linear Operations: These are local operations that don't need to be simulated.

[0236] Fixed point multiplication: For each multiplication, for any two input shares $\llbracket x \rrbracket^R, \llbracket \alpha x \rrbracket^R, \llbracket y \rrbracket^R$, 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:

[0237] 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 $r_j := \text{PRF}_{s_j}(\text{cnt})$ as in Table 9. Compute $r'_j = r_j/d$ and run the multiplication protocol (Table 7, Table 11) to compute $\llbracket ar \rrbracket^R, \llbracket ar' \rrbracket^R$ with the same changes as listed above in Step 3.

[0238] 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:

[0239] - If $P_j \in S^*$, define $\gamma_{i,j} := \widetilde{\text{PRF}}_{s_{j,i}}(\text{cnt})$ for $j \in [1, i)$ and $\gamma_{i,j} := -\widetilde{\text{PRF}}_{s_{i,j}}(\text{cnt})$ for $j \in (i, n]$ as in Table 9.

[0240] - 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 $\gamma_{i,j} = -\Gamma[i][j]$.

[0241] 3. Run Step 3 to Step 6 as in Table 9.

[0242] 4. **MAC Check Failure:** From the knowledge of all shares of $\llbracket \alpha \rrbracket^R, \llbracket x \rrbracket^R, \llbracket y \rrbracket^R, \llbracket r \rrbracket^R$ 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, $\mathcal{F}_{\text{MAC.Check}}$ does not abort but the values sent by $\mathcal{A}$ are inconsistent with those calculated above $\{z'_{i^*}\}_{P_{i^*} \in S^*}$.

[0243] Output reconstruction:

[0244] 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 $S_j \cap S^* \neq \emptyset$.

[0245] 2. Let $j_1, \dots, j_\ell \in [m]$ denote the indices for which $S_j \cap S^* = \emptyset$.

[0246] 3. Pick $z_{j_1}, \dots, z_{j_{\ell-1}} \leftarrow \mathbb{S}_{2^k}$. Compute $z_{j_\ell} \in \mathbb{S}_{2^k}$ as $z_{j_\ell} = \text{out} - \sum_{i \in [\ell-1]} z_{j_i} - z^*$.

- [0247] 4. On behalf of the honest parties, send their shares of z as in the real protocol.
- [0248] 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 $\mathcal{F}_{C_d}$ to deliver output “Abort” to that honest party. Else, instruct $\mathcal{F}_{C_d}$ to deliver the correct output out.
- [0249] Embodiments now show that the above simulation strategy is successful via a hybrid argument.
- [0250] Hyb₀: Consider a simulator SimHyb that plays the role of the honest parties as in Table 11. This is the real world.
- [0251] Hyb₁: **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 $\alpha = \alpha^*$.
- [0252] Indistinguishability of Hyb₀ and Hyb₁ is implied by the security of the MPC protocol used to jointly generate α .
- [0253] Hyb₂: **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 $\{\llbracket x_{i^*} \rrbracket_i^R\}_{P_i \in H}$ and the simulator of the MPC protocol for $\mathcal{F}_{\text{Range}}$, extract $\{x_{i^*}\}_{i \in S^*}$ and query $\mathcal{F}_{C_d}$ to learn output out. If the extraction is unsuccessful, output “Extraction Abort”.
- [0254] From the security of the MPC protocol for $\mathcal{F}_{\text{Range}}$, the probability that the Simulator of MPC protocol for $\mathcal{F}_{\text{Range}}$ 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 Hyb₂ is negligible. Apart from that, notice that the distributions produced by both Hyb₁ and Hyb₂ are the same. Hence, the two hybrids are indistinguishable.
- [0255] Hyb₃: **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 $\mathcal{F}_{C_d}$.
- [0256] Honest parties also get their output from $\mathcal{F}_{C_d}$ in this hybrid.

[0257] 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 $\{z_{j_1}, \dots, z_{j_\ell}\}$ 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 $\mathcal{F}_{\text{Cd}}$ 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.

[0258] **Hyb₄: 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.

[0259] 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 $\widetilde{\text{PRF}}$.

[0260] **Hyb₅: 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 $\mathcal{F}_{\text{MAC.Check}}$ succeeds.

[0261] Since the MAC key α appears uniformly random to $\mathcal{A}$, the security of $\mathcal{F}_{\text{MAC.Check}}$ guarantees that the probability that $\mathcal{F}_{\text{MAC.Check}}$ 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.

[0262] **Hyb₆: Simulate $\mathcal{F}_{\text{Range}}$.** In the input sharing step, simulate the MPC protocol used to compute functionality $\mathcal{F}_{\text{Range}}$.

[0263] Indistinguishability of Hyb_6 and Hyb_7 follows from the security of the MPC protocol used to compute $\mathcal{F}_{\text{Range}}$.

[0264] **Hyb₇: 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.

[0265] 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 $\mathcal{F}_{\text{MAC.Check}}$) 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 $\llbracket x \rrbracket^R$ computed as part of the circuit evaluation, every share of $\llbracket x \rrbracket^R$ 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

[0266] In this section, embodiments prove that the protocol $\Pi_{\text{SShare-MPC}}$ securely realizes the ideal functionality $\mathcal{F}_{\text{Cd}}$ 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, embodiments prove the following theorem:

[0267] **Theorem 7.1** The protocol $\Pi_{\text{SShare-MPC}}$ presented in Table 14 statistically securely realizes the functionality $\mathcal{F}_{\text{Cd}}$ (defined in Table 2) with rounding error distribution $\mathcal{E}$ in the presence of a semi-honest adversary that corrupts $t < n/2$ parties.

[0268] *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.

[0269] For security, consider an adversary $\mathcal{A}$ that corrupts a set C of t parties, denoted as $P_{C_1}, \dots, P_{C_t}$. Let the honest parties be denoted by set H comprising $P_{H_1}, \dots, P_{H_{n-t}}$. The simulator $\mathcal{S}$ has as input the following: the output out of the functionality $\mathcal{F}_{C_d}$ and the set $\{(\text{inp}_{C_i}, r_{C_i})\}_{i \in [t]}$ indicating the corrupt parties' inputs and randomness. Embodiments 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:

[0270] Share: On behalf of each honest party P_{H_i} :

[0271] 1. Sample $s_1, \dots, s_t \xleftarrow{\$} \mathbb{F}_d$.

[0272] 2. Send s_j to $\mathcal{A}$ for each corrupt party P_{C_j} .

[0273] Linear operations: These are local operations that don't need to be simulated.

[0274] Fixed point multiplication: For each fixed point multiplication:

[0275] - Preprocessing:

[0276] 1. For each corrupt party P_{C_i} where $C_i \in [t + 1]$, sample $r_{C_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$.

[0277] 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 $\llbracket r_{H_i} \rrbracket^{S, 2t}, \llbracket r_{H_i}/d \rrbracket^S$, and send the shares to the corrupted parties.

[0278] 3. Let $r := \sum_{i \in [t+1]} r_i$.

[0279] • Online multiplication:

[0280] • If P_1 is an honest party, let $z' := r$ and send it to $\mathcal{A}$ on behalf of P_1 .

[0281] • If P_1 is a corrupt party, compute the shares of $\llbracket z' \rrbracket^{S, 2t}$ 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.

[0282] Reconstruct the output:

[0283] • If P_1 is an honest party, then let $z := \text{out}$ and send it to $\mathcal{A}$ on behalf of P_1 .

[0284] • If P_1 is a corrupt party, compute the shares of $\llbracket z \rrbracket^S$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z = \text{out}$, and send these shares to P_1 on behalf of the honest parties.

[0285] Embodiments now show that the above simulation strategy is successful via a hybrid argument.

[0286] Hyb_0 : $\mathcal{A}$'s view and the honest parties' output in the real world.

[0287] 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 $z := \text{out}$ 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 $z = \text{out}$, 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.

[0288] 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.

[0289] 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 $z' = x \cdot y + r$.

[0290] 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' .

[0291] 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 $x \cdot y + r$.

[0292] 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 .

[0293] 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.

[0294] 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

[0295] In this section, embodiments prove that the protocol $\Pi_{\text{AShare-MPC}}$ securely realizes the ideal functionality $\mathcal{F}_{\text{Cd}}$ 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, embodiments prove the following theorem:

[0296] **Theorem 12.1** Given preprocessed Beaver triples, the protocol $\Pi_{\text{AShare-MPC}}$ presented in Table 16 statistically securely realizes the functionality $\mathcal{F}_{\text{Cd}}$ (defined in Table 2) with rounding error distribution $\mathcal{E}$ in the presence of a semi-honest adversary that corrupts $t < n$ parties.

[0297] *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, the method according to embodiments can include multiplication with Beaver triples followed by a truncation using $\llbracket r \rrbracket^A$ and $\llbracket r' \rrbracket^A$. 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.

[0298] For security, consider an adversary $\mathcal{A}$ that corrupts a set C of t parties, denoted as $P_{C_1}, \dots, P_{C_t}$. Let the honest parties be denoted by set H comprising $P_{H_1}, \dots, P_{H_{n-t}}$. The simulator $\mathcal{S}$

has as input the following: the output out of the functionality $\mathcal{F}_{C_d}$ and the set $\{(\text{inp}_{C_i}, r_{C_i})\}_{i \in [t]}$ indicating the corrupt parties' inputs and randomness. Embodiments 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:

[0299] Share: On behalf of each honest party P_{H_i} :

[0300] 1. Sample $s_1, \dots, s_t \xleftarrow{\$} \mathbb{S}_{2^k}$.

[0301] 2. Send s_j to $\mathcal{A}$ for each corrupt party P_{C_j} .

[0302] Linear operations: These are local operations that don't need to be simulated.

[0303] Fixed point multiplication: For each fixed point multiplication with truncation:

[0304] - Preprocessing:

[0305] 1. For each corrupt party P_{C_i} where $C_i \in [t + 1]$, sample $r_{C_i} \xleftarrow{\$} \mathbb{S}_{2^{k'+\lambda}}$.

[0306] 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 $\llbracket r_{H_i} \rrbracket^A / \llbracket r_{H_i} / d \rrbracket^A$, and send the shares to the corrupted parties.

[0307] 3. Let $r := \sum_{i \in [t+1]} r_i$.

[0308] Online multiplication:

[0309] 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 .

[0310] If P_1 is a corrupt party, then let the shares of $\llbracket x' \rrbracket^A$ and $\llbracket y' \rrbracket^A$ of the honest parties be random and send them to P_1 on behalf of them. Next, compute the shares of $\llbracket z' \rrbracket^A$ 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.

[0311] Reconstruct the output:

[0312] If P_1 is an honest party, then let $z := \text{out}$ and send it to $\mathcal{A}$ on behalf of P_1 .

[0313] If P_1 is a corrupt party, then compute the shares of $[[z]]^A$ for all the corrupt parties. Let the shares of the honest parties be random such that the reconstructed $z = \text{out}$, and send these shares to P_1 on behalf of the honest parties.

[0314] Embodiments 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.

[0315] Hyb_0 : $\mathcal{A}$'s view and the honest parties' output in the real world.

[0316] 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 $z := \text{out}$ 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 $z = \text{out}$, 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.

[0317] 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.

[0318] 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 $z' = x \cdot y + r$.

[0319] 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' .

[0320] 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 $x \cdot y + r$.

[0321] Since $x \cdot y \in \mathbb{S}_{2k'}$, and there is at least one r_i (sampled from $\mathbb{S}_{2k'+\lambda}$) 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 .

[0322] Hyb_4 : Same as Hyb_2 but for fixed point multiplication by truncation, the simulator manipulates the honest parties' shares of x' and y' to be random.

[0323] First, $x' = x + \alpha$ is uniformly random in $\mathbb{S}_{2k}$ as $\alpha \xleftarrow{\$} \mathbb{S}_{2k}$. 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 .

[0324] 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.

[0325] 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.

X. REFERENCES

1. Google cloud ai. cloud.google.com/products/machine-learning/.
2. Machine learning on aws. aws.amazon.com/machine-learning/.
3. Microsoft azure. azure.microsoft.com/en-us/services/machine-learning-studio/.
4. Watson machine learning. 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. Abdelrahman 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. Aby3: 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.

[0326] Aspects of embodiments can be implemented in the form of control logic using hardware circuitry (e.g. an application specific integrated circuit or field programmable gate array) and/or using computer software stored in a memory with a generally programmable processor in a modular or integrated manner, and thus a processor can include memory storing software instructions that configure hardware circuitry, as well as an FPGA with configuration instructions or an ASIC. As used herein, a processor can include a single-core processor, multi-core processor on a same integrated chip, or multiple processing units on a single circuit board or networked, as well as dedicated hardware. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will know and appreciate other ways and/or methods to implement embodiments of the present disclosure using hardware and a combination of hardware and software.

[0327] Any of the software components or functions described in this application may be implemented as software code to be executed by a processor using any suitable computer language such as, for example, Java, C, C++, C#, Objective-C, Swift, or scripting language such as Perl or Python using, for example, conventional or object-oriented techniques. The software code may be stored as a series of instructions or commands on a computer readable medium for storage and/or transmission. A suitable non-transitory computer readable medium can include random access memory (RAM), a read only memory (ROM), a magnetic medium such as a hard-drive or a floppy disk, or an optical medium such as a compact disk (CD) or DVD (digital versatile disk) or Blu-ray disk, flash memory, and the like. The computer readable medium may be any combination of such devices. In addition, the order of operations may be re-arranged. A process can be terminated when its operations are completed, but could have additional steps not included in a figure. A process may correspond to a method, a function, a procedure, a

subroutine, a subprogram, etc. When a process corresponds to a function, its termination may correspond to a return of the function to the calling function or the main function

[0328] Such programs may also be encoded and transmitted using carrier signals adapted for transmission via wired, optical, and/or wireless networks conforming to a variety of protocols, including the Internet. As such, a computer readable medium may be created using a data signal encoded with such programs. Computer readable media encoded with the program code may be packaged with a compatible device or provided separately from other devices (e.g., via Internet download). Any such computer readable medium may reside on or within a single computer product (e.g. a hard drive, a CD, or an entire computer system), and may be present on or within different computer products within a system or network. A computer system may include a monitor, printer, or other suitable display for providing any of the results mentioned herein to a user.

[0329] Any of the methods described herein may be totally or partially performed with a computer system including one or more processors, which can be configured to perform the steps. Thus, embodiments can be directed to computer systems configured to perform the steps of any of the methods described herein, potentially with different components performing a respective step or a respective group of steps. Although presented as numbered steps, steps of methods herein can be performed at a same time or at different times or in a different order. Additionally, portions of these steps may be used with portions of other steps from other methods. Also, all or portions of a step may be optional. Additionally, any of the steps of any of the methods can be performed with modules, units, circuits, or other means of a system for performing these steps.

[0330] The specific details of particular embodiments may be combined in any suitable manner without departing from the spirit and scope of embodiments of the disclosure. However, other embodiments of the disclosure may be directed to specific embodiments relating to each individual aspect, or specific combinations of these individual aspects.

[0331] The above description of example embodiments of the present disclosure has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to

limit the disclosure to the precise form described, and many modifications and variations are possible in light of the teaching above.

[0332] A recitation of "a", "an" or "the" is intended to mean "one or more" unless specifically indicated to the contrary. The use of "or" is intended to mean an "inclusive or," and not an "exclusive or" unless specifically indicated to the contrary. Reference to a "first" component does not necessarily require that a second component be provided. Moreover, reference to a "first" or a "second" component does not limit the referenced component to a particular location unless expressly stated. The term "based on" is intended to mean "based at least in part on."

[0333] All patents, patent applications, publications, and descriptions mentioned herein are incorporated by reference in their entirety for all purposes. None is admitted to be prior art. Where a conflict exists between the instant application and a reference provided herein, the instant application shall dominate.

WHAT IS CLAIMED IS:

- 1 1. A method performed by a first computer of a plurality of computers for
2 implementing a replication protocol for multi-party fixed point multiplication, the method
3 comprising:
4 obtaining a first secret share of a first secret value (x) and a second secret share of
5 a second secret value (y);
6 jointly sampling a first pseudo-random key ($s_{i,j}$) with another computer paired
7 with the first computer and a second pseudo-random key (s_j) with a first set (D_j) of multiple sets
8 in the plurality of computers;
9 generating a replicated random share of a random value ($[[r]]^R$) using the second
10 pseudo-random key (s_j);
11 generating a replicated sharing value (β) using the first pseudo-random key ($s_{i,j}$);
12 computing a masked share ($z'_{i=1}$) by providing a product of the first secret share
13 and the second secret share masked with the replicated sharing value (β) and the replicated
14 random shares of the random value ($[[r]]^R$);
15 receiving, by the first computer from computers in the first set (D_j), a plurality of
16 masked shares (z'_i), thereby revealing a set shared value z' to the first set (D_j); and
17 generating a replicated secret share ($[[z]]^R$) using the received plurality of masked
18 shares (z'_i) and the replicated random shares of the random value ($[[r]]^R$).
- 1 2. The method of claim 1, wherein generating the one or more replicated
2 random shares of the random values ($[[r]]^R$) using the second pseudo-random key (s_j) further
3 includes adding a MAC key (α) to the one or more replicated random shares of the value,
4 wherein the MAC key (α) is randomly sampled.
- 1 3. The method of claim 2, wherein the masked share ($z'_{i=1}$) includes the
2 MAC key (α).
- 1 4. The method of claim 3, wherein the method further comprises:

2 executing a batch check consistency protocol to determine whether each computer
3 in the plurality of computers generated the replicated secret share ($[[z]]^R$) from the masked share
4 ($z'_{i=1}$) that included the MAC key (α).

1 5. The method of claim 1, wherein the replicated sharing value (β) from each
2 of the computers of the plurality of computers sum to zero.

1 6. The method of claim 1, wherein after receiving the plurality of masked
2 shares (z'_i) from the plurality of computers in the first set (D_j) the method further comprises:
3 obtaining the set shared value (z') from the plurality of masked shares and the
4 masked share (z'_i), wherein the plurality of masked shares and the masked share (z'_i) are an n-
5 out-of-n secret sharing of the set shared value (z').

1 7. The method of claim 1, wherein computers of the plurality of computers in
2 a second set of the multiple sets determine a respective value for the replicated secret share
3 ($[[z]]^R$) using respective random shares of the random value ($[[r]]^R$).

1 8. The method of claim 1 further comprising:
2 obtaining a decimal bit value (d).

1 9. The method of claim 8, wherein generating the replicated secret share
2 ($[[z]]^R$) further comprises:
3 generating, by the first computer, a first replicated secret share (z_1) of the
4 replicated secret share ($[[z]]^R$) by dividing the set shared value (z') by the decimal bit value (d)
5 and then subtracting the replicated random share of a value ($[[r]]^R$) divided by the decimal bit
6 value (d).

1 10. A method performed by a first computer of a plurality of computers for
2 implementing a sharing protocol for multi-party fixed point multiplication, the method
3 comprising:
4 obtaining a secret share of a first value (x), a secret share of a second value (y);

5 sampling a first random value (r_i) to obtain a first random share ($[[r_i]]^{S, 2t}$) of the
 6 first random value (r_i) and a second random share ($[[r_i/d]]^S$);
 7 sending the first random share ($[[r_i]]^{S, 2t}$) and the second random share ($[[r_i/d]]^S$) to
 8 the plurality of computers;
 9 obtaining each share of the first random share ($[[r]]^{S, 2t}$) and each share of the
 10 second random share ($[[r/d]]^S$) from the plurality of computers;
 11 computing a third random share ($[[r]]^{S, 2t}$) and a fourth random share ($[[r']]^S$)
 12 based on each share of the first random share ($[[r]]^{S, 2t}$) and each share of the second random
 13 share ($[[r/d]]^S$), respectively;
 14 computing a masked share ($[[z']]^{S, 2t}$) that provides a product of the secret share of
 15 the first value (x) and the secret share of the second value (y) masked with the third random
 16 share ($[[r]]^{S, 2t}$);
 17 obtaining a masked value (z') based on a plurality of masked shares ($[[z']]^{S, 2t}$)
 18 determined by the plurality of computers; and
 19 generating an output secret share ($[[z]]^S$) using the masked value (z') and the
 20 fourth random share ($[[r']]^S$).

1 11. The method of claim 10, wherein the second random share ($[[r_i/d]]^S$) is
 2 equal to the first random share ($[[r_i]]^S$) divided by a decimal bit value (d).

1 12. The method of claim 10, wherein the output secret share ($[[z]]^S$) represents
 2 a shared output value (z) that is approximately equal to the first value (x) multiplied by the
 3 second value (y) divided by a decimal bit value (d).

1 13. The method of claim 10, wherein obtaining the masked value (z') further
 2 comprises:
 3 obtaining, by the first computer, a plurality of masked shares ($[[z']]^{S, 2t}$) from the
 4 plurality of computers;
 5 generating the masked value (z') based on the plurality of masked shares ($[[z']]^{S, 2t}$); and
 6 providing the masked value (z') to the plurality of computers.

1 14. The method of claim 10, wherein generating the output secret share
2 ($[[z]]^S$), further comprises:
3 generating the output secret share ($[[z]]^S$) by dividing the masked value (z') by a
4 decimal bit value (d) and subtracting off the fourth random share ($[[r']^S$).

1 15. The method of claim 10, wherein the first random share ($[[r_i]]^{S, 2t}$) is
2 determined using $2t$ -out-of- n sharing.

1 16. A method performed by a first computer of a plurality of computers for
2 implementing an additive sharing protocol for multi-party fixed point multiplication, the method
3 comprising:

4 obtaining a secret share ($[[x]]^A$) of a first value (x), a secret share ($[[y]]^A$) of a
5 second value (y), a decimal bit value (d), a secret share ($[[\alpha]]^A$) of a third value (α), a secret share
6 ($[[\beta]]^A$) of a fourth value (β), and a secret share ($[[\gamma]]^A$) of a fifth value (γ);

7 determining a share of a first shared random value ($[[r']]^A$) and a share of a second
8 shared random value ($[[r]]^A$);

9 determining a first intermediate share ($[[x']]^A$) based on the secret share ($[[x]]^A$) of
10 the first value (x) and the secret share ($[[\alpha]]^A$) of the third value (α);

11 determining a second intermediate share ($[[y']]^A$) based on the secret share ($[[y]]^A$)
12 of the second value (y) and the secret share ($[[\beta]]^A$) of the fourth value (β);

13 receiving, by the first computer, a plurality of first intermediate shares and a
14 plurality of second intermediate shares from the plurality of computers to reveal a first
15 intermediate value (x') and a second intermediate value (y');

16 sending, by the first computer, the first intermediate value (x') and the second
17 intermediate value (y') to the plurality of computers;

18 determining a third intermediate share ($[[z']]^A$) based on the first intermediate
19 value (x'), the second intermediate value (y'), the secret share ($[[x]]^A$) of the first value (x), the
20 secret share ($[[y]]^A$) of the second value (y), the secret share ($[[\gamma]]^A$) of the fifth value (γ) and the
21 share of the second shared random value ($[[r]]^A$);

22 receiving, by the first computer, a plurality of third intermediate shares from the
 23 plurality of computers to reveal a third intermediate value (z');
 24 sending, by the first computer, the third intermediate value (z') to the plurality of
 25 computers; and
 26 determining an output secret share ($\llbracket z \rrbracket^A$) based on the third intermediate value
 27 (z'), the decimal bit value (d), and the share of the first shared random value ($\llbracket r' \rrbracket^A$).

1 17. The method of claim 16, wherein the secret share of the third value (α),
 2 the secret share of the fourth value (β), and the secret share of the fifth value (γ) make up a
 3 preprocessed Beaver triple, and wherein the first value (x) and the second value (y) are fixed
 4 point values represented in twos-complement form in a ring ($\mathbb{S}_{2^{k'}}$).

1 18. The method of claim 16, and wherein determining the share of the first
 2 shared random value ($\llbracket r' \rrbracket^A$) and the share of the second shared random value ($\llbracket r \rrbracket^A$) further
 3 comprises:
 4 sampling an extended ring ($\mathbb{S}_{2^{k'+\lambda}}$) to generate shares of a first initial random
 5 value ($\llbracket r_i \rrbracket^A$) and shares of a second initial random value ($\llbracket r_i/d \rrbracket^A$), where the shares of the
 6 second initial random value ($\llbracket r_i/d \rrbracket^A$) are created based on the shares of the first initial random
 7 value ($\llbracket r_i \rrbracket^A$) and the decimal bit value (d);
 8 sending the shares of the first initial random value ($\llbracket r_i \rrbracket^A$) and the shares of the
 9 second initial random value ($\llbracket r_i/d \rrbracket^A$) to the plurality of computers;
 10 determining shares of the first shared random value ($\llbracket r' \rrbracket^A$) based on the shares of
 11 the second initial random value ($\llbracket r_i/d \rrbracket^A$); and
 12 determining a share of the second shared random value ($\llbracket r \rrbracket^A$) based on the shares
 13 of the first shared random value ($\llbracket r' \rrbracket^A$).

1 19. The method of claim 16, wherein the output secret share ($\llbracket z \rrbracket^A$) is a share
 2 of an additive output value (z) that is approximately equal to the first value (x) multiplied by the
 3 second value (y) divided by the decimal bit value (d).

1 20. The method of claim 16, wherein the first value (x) and the second value
2 (y) are fixed point vales represented in twos-complement form.

1 21. A computer comprising:
2 a processor;
3 a network interface; and
4 a non-transitory computer-readable medium comprising code for instructing the
5 processor to implement the method of any one of claims 1-20.