

TITLE OF THE INVENTION

Arithmetic for Secure Multi-Party Computation with Modular Integers

RELATED APPLICATIONS

[0001] The subject matter of this application is related to U.S. Provisional Application No. 62809543, filed on 2019-02-22, which is hereby incorporated by reference in its entirety. The subject matter of this application is also related to Patent Cooperation Treaty Application No. PCT/US2018/048963, filed on 2018-08-30 and published on 2019-03-07 as World Intellectual Property Organization International Publication Number WO2019046651A2, which is hereby incorporated by reference in its entirety.

BACKGROUND OF THE INVENTION

[0002] Privacy-preserving multi-party computation (MPC) techniques enable multiple parties to collaboratively evaluate a function to produce a shared or revealed output while keeping the inputs private. Such computations are used, for example, in medicine and finance, when the input data comes from distinct private data sources that cannot disclose their data, but a public result based on the confidential data is needed.

[0003] A MPC is generally split both temporally and geographically across multiple participants. The participants, each of which represents a separate computing system, typically include k parties and one trusted dealer. As used herein, the terms party and player are used interchangeably and refer to individual party computer systems participating in a multi-party computation.

[0004] After compilation of code to implement the computation, the trusted dealer first executes an offline phase of the MPC. In the offline phase, the trusted dealer produces masks (masking data, also referred to as triplets), and distributes shares of these masks to the parties such that each party knows only its share of the mask and none of them know the plaintext mask value represented by a sum of the shares. The determination of the masks typically depends on the data expected to be operated upon from statistical analysis perspective so that the masks are appropriately configured in relation to the data.

[0005] The k parties then collaboratively execute an online phase of the MPC, with synchronization steps where parties can exchange or broadcast messages according to a

defined MPC protocol. The online phase can be run in a firewalled environment to which the trusted dealer has no access.

[0006] A MPC can be the distributed equivalent of a plaintext pseudocode, which we can describe as a single static assignment (SSA) graph of MPC-friendly elementary operations. The nodes of the SSA graph are plaintext variables, and each party gets a local view (or secret share) of the variables. We denote this local view as an MPC container. The MPC-friendly elementary operations are referred to as builtins that take MPC containers and optionally some static parameters, as input and produce MPC containers as output.

[0007] Figure 1 illustrates a schematic of MPC containers for k parties of a multi-party computation. Globally, a MPC container holds all the information about one variable in the SSA, namely, a plaintext value x that can be either public (known by all parties, but not the trusted dealer) or a secret-shared $[[x]]$ (each party knows its share only), one mask $[[\lambda]]$ (known by the dealer, and secret shared among all parties), and the optional masked value $a = x + \lambda$ (known by all parties, but typically not the dealer). Notation note: the double square bracket notation $[[\]]$ is used herein to denote a secret shared value.

[0008] Locally, each party has a trace of the MPC container, which can be a structure with fields as follows:

- the public value x (if the container is publicly revealed)
- one share x_j of the public value
- one share λ_j of the container's mask
- the masked value a (if the container is masked and revealed).

[0009] Analyzing the union of the k containers, the following holds: the plaintext value of the container is by definition the sum of all shares $\sum x_j$, if the container is public, all parties know the plaintext value, and each party has this value x populated. In this case, none of the other fields need be used in the MPC protocol. The mask of the container is $\lambda = \sum \lambda_j$. The value of the mask is known only by the trusted dealer (during the offline phase). None of the parties knows or learns the actual mask during the online phase. The masked value a is equal to $\lambda + x$. The special mask-and-reveal operation instructs each party to broadcast its $x_j + \lambda_j$, which allows them to jointly reconstruct and store the same field $a =$

$x + \lambda$. All other technical or local variable that appears in the builtins are called ephemeral variables.

[0010] Two prior works have proposed the use of fixed-point integer representations, as opposed to floating-point representations, for representing real numbers in the context of MPC. The first of the prior works will be referred to hereinafter as “SecureML”, by P. Mohassel and Y. Zhang, and titled “SecureML: A system for scalable privacy-preserving machine learning”, in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, pages 19–38, 2017. The second of the prior works will be referred to hereinafter as “ABY³”, by P. Mohassel and P. Rindal, and titled “ABY³: A mixed protocol framework for machine learning”, in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pages 35–52, 2018.

[0011] The modReal Representation. In these works, real number arithmetic is implemented through the evaluation of additions, subtractions and rounded divisions on fixed-size integer types (such as *int64*, that is, integers modulo 2^{64}). For instance, to multiply two 30-bit integers x and y and retain the 30 most significant bits of the result (analogously to floating point multiplication), one can first compute xy (by fitting the product into a 64-bit integer without any overflows) and then right-shift the result by 30 bits. The two works adapt these operations to the MPC setting where a plaintext $x \in \mathbb{Z}$ is secret-shared among k parties ($k = 2$ in SecureML and $k \geq 3$ in ABY³) as $(x_1, \dots, x_k) \in (\mathbb{Z}/q\mathbb{Z})^k$ (for some integer q), such that $x = \text{centermod}_q(x_1 + \dots + x_k)$. The signed mathematical lift $\text{centermod}_q(x)$ is defined to be the unique integer $\tilde{x} \in [-q/2, q/2)$ such that $\tilde{x} \equiv x \pmod{q}$. We denote this particular representation of real numbers on arithmetic shares as the modReal representation.

[0012] SecureML, Section 4.1, subsection titled “Arithmetic Operations on Shared Decimal Numbers” suggests as follows. Consider a fixed-point product of two binary decimal numbers x and y stored in a common integer representation such that ℓ_D bits of each number are defined to represent the fractional part of the number (binary digits after the decimal point). In this case, $x' = 2^{\ell_D}x$ and $y' = 2^{\ell_D}y$ will be integers that can be natively represented using an integer back-end. The product $z = x'y'$, with the

multiplication taking place in $\mathbb{Z}/2^\ell\mathbb{Z}$ for some integer ℓ , however, will have up to $2\ell_D$ bits in the fractional part of the representation. It may not be practical, however, to maintain the additional binary digits after the decimal, particularly in the context of a MPC. SecureML suggests to decrease the number of bits representing the fractional part in the result by ignoring or truncating the last ℓ_D bits. For example, the product can be represented as: $z = 2^{\ell_D}z_1 + z_0$ where $0 \leq z_0 < 2^{\ell_D}$, and consider the result to include only $z_1 = \lfloor z/2^{\ell_D} \rfloor$.

[0013] SecureML then extends above truncation technique to a secret shared value $\llbracket z \rrbracket$. In effect, two parties holding respective secret shares of $\llbracket z \rrbracket$ can truncate their individual shares. It is then shown in Theorem 1 that with high probability, the two truncated shares can reconstruct the truncation $\lfloor z/2^{\ell_D} \rfloor$ of z . This solution relies on the following premise: if x_1 and x_2 are two secret shares of a very small x , then with very high probability, the two shares x_1 and x_2 are in the two opposite half circles mod 2^ℓ . The solution, which only works with two parties, relies on a local procedure to truncate the shares, but has a non-zero probability that the resulting secret shared value will be incorrect.

[0014] ABY³ addresses the limitations of the SecureML two party truncation technique in §5.1.1 and §5.2.2 presents additional techniques for truncation that extend to three or more parties in a MPC setting. Again, however, these additional techniques have a non-negligible probability of failure in practice.

[0015] In the two prior works, multiplication is performed using Beaver triplets modulo q , rounding can be performed locally on each share (at a price of a small error on the plaintext value bounded by $\sqrt{k}/2$) and addition can be done locally at each share. Two variants for division are proposed: the first one, in SecureML, is a two party solution without communication between the two parties and the second one, in ABY³ is a Beaver triplet-based solution for k parties. In both cases, the result of the division is correct most of the time; yet, a huge undetectable overflow occurs with some small, but non-zero probability. The overflow probability for a plaintext range of M integer elements and shares modulo q is M/q . In the above-mentioned example of multiplying two 30-bit integers, the overflow probability is $2^{61}/2^{64} = 1/8$ and drops down to $2^{41}/2^{64}$ if we reduce the precision to 20 bits. When it occurs, the overflow is large (the overflow amplitude has been mitigated since SecureML, yet, it still changes the most significant bits of the result in the new ABY³

revision). Thus, any aggregate function that includes these wrong values gets affected. The overflow probability is per-coefficient, so if we do a computation on a vector or a matrix of N coefficients (as is the case for most ML use-cases such as logistic regression, linear regression, neural nets), the presence of at least one destructive overflow gets multiplied by N and becomes non-negligible.

[0016] The two possible counter-measures proposed by the two prior works to deal with millions of coefficients are either to reduce the precision (provide at most 10 bits of precision on a 64-bit backend), or to increase the backend-size to e.g. 128-bits (4 to 16 times slower). Note that 10 bits of fixed point precision can be enough in numerically stable use-cases, where we know the exact distribution of the plaintext (e.g. feature-scaled input, output of a sigmoid etc.). However, we can also encounter major underflow situations if the fixed point exponent is estimated 4 or 5 bits higher than the actual values, because of an uncertainty on the secret plaintext distribution.

SUMMARY OF THE INVENTION

[0017] In the context of a secure multi-party computation, a secret shared value jointly stored by multiple parties in a first modular representation is cast into a second modular representation having a larger most significant bit. The cast includes a trusted dealer secret sharing with the parties a masking value the range of which is divided into two halves. The parties use the masking value to mask and reveal a sum of the secret shared value and the masking value. The trusted dealer also secret shares with the parties a bit encoding the half of the range that contains the masking value. The parties use the secret shared bit along with the sum to collaboratively reconstruct a set of secret shares representing the secret shared value in the second modular format. In contrast with previous work, the disclosed solution eliminates a non-zero probability of error without sacrificing efficiency or security. This method is described in additional detail in section 3.1 of the detailed description below.

[0018] A method can be performed by a secure multi-party computing system (SMPCS) configured for performing multi-party computations on secret shared values, where the SMPCS includes a trusted dealer computing system and a plurality of party computing systems in secure networked communication. The method is for casting a secret shared value from a first modular representation to a second modular representation, wherein the

first modular representation has a range defined by a most significant bit position and a least significant bit position of the first modular representation, and wherein the second modular representation has a range defined by a most significant bit position and a least significant bit position of the second modular representation, and wherein the most significant bit position of the second modular representation is greater than the most significant bit position of the first modular representation.

[0019] In accordance with one embodiment, each of the party computing systems stores a respective secret share of the secret shared value in the first modular representation. The trusted dealer computing system stores a masking value and communicates to each of the party computing systems a respective secret share of the masking value in the first modular representation so that the masking value is secret shared between the party computing systems in the first modular representation. The trusted dealer computing system: determines a subrange in which the masking value resides, wherein the subrange is selected from a plurality of predetermined non-overlapping subranges representing the range of the first modular representation; encodes the determined subrange into a numerical subrange identifier, and communicates to each of the party computing systems a respective secret share of the numerical subrange identifier so that the numerical subrange identifier is secret shared between the party computing systems. The trusted dealer computing system transforms the masking value, in a first transformation, into a transformed masking value by at least performing a first mathematical lift on the masking value. The trusted dealer computing system communicates to each of the party computing systems a respective secret share of the transformed masking value in the second modular representation so that the transformed masking value is secret shared between the party computing systems in the second modular representation. The party computing systems collaboratively compute and reveal a sum of the secret shared value and the masking value as a masked value without revealing the secret shares of the secret shared value or the secret shares of the masking value.

[0020] The SMPCS transforms the masked value, in a second transformation, into a first transformed masked value by at least performing a second mathematical lift on the masked value. The SMPCS transforms the masked value, in a third transformation, into a second transformed masked value by at least performing a third mathematical lift on the masked

value, wherein the first mathematical lift, the second mathematical lift and the third mathematical lift are each different from one another. The party computing systems collaboratively compute the secret shared value in the second modular representation such that each of the party computing systems computes a respective secret share of the secret shared value in the second modular representation based on: the first transformed masked value, the second transformed masked value, for each party computing system, the respective secret share of the numerical subrange identifier, and for each party computing system, the respective secret share of the transformed masking value.

[0021] The method can be performed wherein the first mathematical lift and the second mathematical lift produce values relative to each other that are offset by one quarter of the range of the first modular representation, and wherein the second mathematical lift and the third mathematical lift produce values relative to each other that are offset by one half of the range of the first modular representation.

[0022] The method can be performed wherein the first mathematical lift, the second mathematical lift, and the third mathematical lift are each selected from the group consisting of: a quartermod lift parameterized by $2^{M_{\text{msb}}}$, where $(M_{\text{msb}} - 1)$ is the most significant bit position of the first modular representation, a centermod lift parameterized by $2^{M_{\text{msb}}}$, and a posmod lift parameterized by $2^{M_{\text{msb}}}$.

[0023] The method can be performed wherein the first mathematical lift is a quartermod lift on the masking value parameterized by $2^{M_{\text{msb}}}$, wherein the second mathematical lift is a centermod lift on the masked value parameterized by $2^{M_{\text{msb}}}$, wherein the third mathematical lift is a posmod lift on the masked value parameterized by $2^{M_{\text{msb}}}$, and where $(M_{\text{msb}} - 1)$ is the most significant bit position of the first modular representation.

[0024] The method can be performed wherein a quartermod lift on x parameterized by 2^M is defined as a unique real $\tilde{x} \in \left[-\frac{2^M}{4}, \frac{3 \cdot 2^M}{4}\right)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$, wherein a centermod lift on x parameterized by 2^M is defined as a unique real $\tilde{x} \in \left[-\frac{2^M}{2}, \frac{2^M}{2}\right)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$, and wherein a posmod lift on x parameterized by 2^M is defined as a unique real $\tilde{x} \in [0, 2^M)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$.

[0025] The method can be performed wherein the first transformation further includes: rounding a result of the first mathematical lift to a nearest multiple of $2^{p'_{lsb}}$, where p'_{lsb} is the least significant bit position of the second modular representation, and taking a modulo $2^{M'_{msb}}$ of the rounding, where $(M'_{msb} - 1)$ is the most significant bit position of the second modular representation; wherein the second transformation further includes: rounding a result of the second mathematical lift to a nearest multiple of $2^{p'_{lsb}}$, and taking a modulo $2^{M'_{msb}}$ of the rounding; and wherein the third transformation further includes: rounding a result of the third mathematical lift to a nearest multiple of $2^{p'_{lsb}}$, and taking a modulo $2^{M'_{msb}}$ of the rounding.

[0026] The method can be performed wherein the first transformation, the second transformation and the third transformation each includes projecting a result of the associated mathematical lift onto $\mathbb{R}/2^{M'_{msb}}\mathbb{Z}$, where $(M'_{msb} - 1)$ is the most significant bit position of the second modular representation.

[0027] The method can be performed wherein the “plurality of predetermined non-overlapping subranges” consists of two equally sized subranges, and wherein the numerical subrange identifier is selected from a group consisting of the numbers 0 and 1.

[0028] The method can be performed wherein the secret shared numerical subrange identifier is used by the party computing systems collaboratively to select between the first transformed masked value and the second transformed masked value for use in calculating the secret shared value in the second modular representation.

[0029] The method can be performed wherein the first modular representation and the second modular representation are fixed point integer representations of real numbers.

[0030] A method can be performed by a SMPCS configured for performing multi-party computations on secret shared values, where the SMPCS includes a plurality of party computing systems in secure networked communication. The method is for casting a secret shared value from a first modular representation to a second modular representation, wherein the first modular representation has a range defined by a most significant bit position and a least significant bit position of the first modular representation, and wherein the second modular representation has a range defined by a most significant bit position and a least significant bit position of the second modular representation, and wherein the

most significant bit position of the second modular representation is greater than the most significant bit position of the first modular representation.

[0031] In accordance with one embodiment, each of the party computing systems stores a respective secret share of the secret shared value in the first modular representation. The SMPCS communicates to each of the party computing systems a respective secret share of a masking value so that the masking value is secret shared between the party computing systems. The SMPCS communicates to each of the party computing systems a respective secret share of an indication of a subrange in which the masking value resides, wherein the subrange is selected from a plurality of predetermined non-overlapping subranges representing the range of the first modular representation, so that the subrange indication is secret shared between the party computing systems. The SMPCS transforms the masking value, in a first transformation, into a transformed masking value by at least performing a first mathematical lift on the masking value. The SMPCS communicates to each of the party computing systems a respective secret share of the transformed masking value in the second modular representation so that the transformed masking value is secret shared between the party computing systems in the second modular representation. The party computing systems collaboratively compute and reveal a sum of the secret shared value and the masking value as a masked value without revealing the secret shares of the secret shared value or the secret shares of the masking value. The SMPCS transforms the masked value, in a second transformation, into a first transformed masked value by at least performing a second mathematical lift on the masked value. The SMPCS transforms the masked value, in a third transformation, into a second transformed masked value by at least performing a third mathematical lift on the masked value, wherein the first mathematical lift, the second mathematical lift and the third mathematical lift are each different from one another. The party computing systems collaboratively compute the secret shared value in the second modular representation such that each of the party computing systems computes a respective secret share of the secret shared value in the second modular representation based on: the first transformed masked value, the second transformed masked value, for each party computing system, the respective secret share of the subrange indication, and for each party computing system, the respective secret share of the transformed masking value.

[0032] The method can be performed wherein the first transformation, the second transformation and the third transformation each includes projecting a result of the associated mathematical lift onto $\mathbb{R}/2^{M'_{\text{msb}}}\mathbb{Z}$, where $(M'_{\text{msb}} - 1)$ is the most significant bit position of the second modular representation.

[0033] The method can be performed wherein the “plurality of predetermined non-overlapping subranges” consists of two equally sized subranges, and wherein the numerical subrange identifier is selected from a group consisting of the numbers 0 and 1.

[0034] The method can be performed wherein the secret shared numerical subrange identifier is used by the party computing systems collaboratively to select between the first transformed masked value and the second transformed masked value for use in calculating the secret shared value in the second modular representation.

[0035] The method can be performed wherein the first modular representation and the second modular representation are fixed point integer representations of real numbers.

[0036] A method for performing a multiplication of a first secret shared value stored in a respective initial modular representation and a second secret shared value stored in a respective initial modular representation, can include: casting, in accordance with any one of the foregoing methods for casting, at least one of the first secret shared value and the second secret shared value into a common modular representation such that both the first secret shared value and the second secret shared value are stored in the common modular representation; and performing a beaver multiplication on the first secret shared value and the second secret shared value in the common modular representation.

[0037] A method for combining a plurality of secret shared values, each being stored in a respective initial modular representation, and a corresponding plurality of static coefficients can include: casting, in accordance with any one of the foregoing methods for casting, at least one of the plurality of secret shared values into a common modular representation such that all of the plurality of secret shared values are stored in the common modular representation; and performing a linear combination on the plurality of secret shared values stored in the common modular representation and the corresponding plurality of static coefficients.

[0038] A method of evaluating a continuous function on a secret shared value can include: selecting a Fourier series, based on the secret shared value, from a plurality of determined Fourier series, wherein each of the plurality of determined Fourier series is configured to approximate the continuous function on an associated subinterval of a domain of the continuous function; based on the selection, casting, in accordance with any one of the foregoing methods for casting, the secret shared value from a first modular representation to a second modular representation; and approximating a value of the continuous function on the secret shared value in the second modular representation using the selected Fourier series.

[0039] A system can include a plurality of computer systems configured to perform any one of the foregoing methods.

[0040] A non-transitory computer-readable medium can be encoded with computer code that, when executed by a plurality of computer systems, causes the plurality of computer systems to perform any one of the foregoing methods.

BRIEF DESCRIPTION OF THE DRAWINGS

[0041] Figure 1 illustrates a schematic of MPC containers.

[0042] Figure 2 illustrates the relationship between bits in a fixed point representation.

[0043] Figure 3 illustrates four casting operations for casting representations between classes.

[0044] Figure 4 illustrates an example application of each of the four casting operations on an example input value.

[0045] Figure 5 illustrates a pseudocode implementation of an offline phase of the Lift operation.

[0046] Figure 6 illustrates a pseudocode implementation of an online phase of the Lift operation.

[0047] Figure 7 illustrates a pseudocode implementation of an offline phase of the Share Refresh operation.

[0048] Figure 8 illustrates a pseudocode implementation of an online phase of the Share Refresh operation.

[0049] Figure 9 illustrates a pseudocode implementation of a Beaver multiplication operation on two secret shared values in modular representation.

[0050] Figure 10 illustrates a schematic of the Beaver multiplication operation.

[0051] Figure 11 illustrates a pseudocode implementation of a Beaver multiplication operation on one secret shared value in modular representation and one public value.

[0052] Figure 12 illustrates a schematic of the Beaver multiplication operation.

[0053] Figure 13 illustrates an example input upon which a cleaning step can be performed.

[0054] Figure 14 illustrates a pseudocode implementation of a Linear combination operation with ModReal representation.

[0055] Figure 15 illustrates a schematic of the Linear combination operation.

[0056] Figure 16A shows the communication channels for a trusted dealer model.

[0057] Figure 16B communication channels for the honest but curious model.

[0058] Figure 17 illustrates a schematic of communication channels between parties during an online phase.

[0059] Figure 18 illustrates a general computer architecture that can be appropriately configured to implement components disclosed herein.

DETAILED DESCRIPTION

[0060] In the following description, references are made to various embodiments in accordance with which the disclosed subject matter can be practiced. Some embodiments may be described using the expressions one/an/another embodiment or the like, multiple instances of which do not necessarily refer to the same embodiment. Particular features, structures or characteristics associated with such instances can be combined in any suitable manner in various embodiments unless otherwise noted. By way of example, this disclosure may set out a set or list of a number of options or possibilities for an embodiment, and in

such case, this disclosure specifically contemplates all clearly feasible combinations and/or permutations of items in the set or list.

[0061] 1 Overview

[0062] In this disclosure, we present novel techniques for performing real number arithmetic with high numerical precision using modular integer representations in a multi-party computation (MPC) setting. In particular, we disclose a method for casting a secret shared value stored in a first modular representation to a second modular representation. In contrast with the previous work, the disclosed solution advantageously eliminates errors without sacrificing efficiency and security, which is important for machine learning techniques such as linear and logistic regressions.

[0063] We adopt a full-threshold security model for MPC and split computations into offline and online phases. The offline phase (independent of the input data) can be performed either by a trusted dealer or by an honest-but-curious dealer. Stronger models and verifiability of this phase can be achieved via standard techniques such as oblivious transfer and cut-and-choose. The online phase computes additive shares of the result.

[0064] In our work, we keep the same definition of the shares as in the prior work, but we modify the triplet (masking value) definition by adding a small term that eliminates completely the overflow (as long as $M/q \leq 1/2$). The method keeps a single round of communication during the online phase and adds only a few binary operations that preserve the overall running time. The new method is always correct, the security is unconditional and it works for any number of parties.

[0065] To briefly summarize, the overflow in ABY³ comes from the fact that during the division operation, for example, one has to reconstruct the small plaintext $x \in \mathbb{Z}$ from a random mask $r' \bmod q$ (chosen during the offline phase) and a masked value $a = x' - r' \bmod q$. The latter is revealed during the online phase before applying a division operation that is incompatible with the classes mod q . In order to do that, both r' and a are implicitly lifted to integers in $[-q/2, q/2)$. More precisely, in ABY³ §5.1.1, 5.1.2, these two implicit centered lifts occur during two operations: 1) in the definition of the triplet $r = r'/2^d$ where $r' \in \mathbb{Z}/q\mathbb{Z}$ is random and 2) in the reveal $x' - r' \in \mathbb{Z}/q\mathbb{Z}$ and locally compute $(x' - r')/2^d$ operation. These two lifts are not always compatible with the further assumption that $a - r'$ is equal to x' over

the reals (the equality may differ by $+q$ or $-q$ with small probability). In our new setting, we preserve the full range of r' , which ensures unconditional security. Yet, we split the group $\mathbb{Z}/q\mathbb{Z}$ in two halves, and during the offline phase, we secret share an additional bit encoding the half group that contains r' and add this secret shared bit to the same triplet. Each half-group corresponds to a different lift for $a - r'$ so the parties can compute both lifts and then use the shared bit to obviously select the right one during the online phase.

[0066] 2 Representations of Plaintext Values and Secret Shares

[0067] 2.1 Floating and fixed-point representations

[0068] A plaintext real value x can be represented in floating-point form as $x = m \cdot 2^e \in \mathbb{R}$ where

- The mantissa m is normalized so that $1/2 \leq |m| < 1$, and
- The exponent $e \in \mathbb{Z}$ (data dependent).

This representation is clearly data-dependent in the sense that both m and e depend on the plaintext value x . In particular, the value of the exponent e is precisely $\lceil \log_2 |x| \rceil$. Since the floating-point representation needs information about the size of x to determine the exponent e , it is data dependent and not MPC friendly, as this value e is not known by the compiler prior to a multi-party computation.

[0069] A plaintext real value x can also be represented in fixed-point (integer) form as $x = m \cdot 2^e \in \mathbb{R}$ where

- The mantissa m is normalized so that $1/2 \leq |m| < 1$, and
- The exponent $e \in \mathbb{Z}$ (public value).

[0070] From the compiler's perspective prior to commencing a multi-party computation, the exponent can be regarded or treated as public whereas the mantissa can be treated as private. Since the compiler does not know the secret number x before the computation, it is impossible for the compiler to determine the exact e that guarantees the above normalization for m . This representation is not suitable either since it can easily overflow or underflow as shown in the following two examples:

[0071] Overflow. Suppose that we are multiplying the following two number (in binary): $x = 10.1_{(2)}$ and $y = 10_{(2)}$. Both of these have exponent 2. Yet, if the precomputed public

exponent of the result is still two, the computation will overflow since: $x \times y = 101_{(2)}$. Thus, we need an exponent of at least 3 to avoid the overflow.

[0072] Underflow. Suppose that we are in a scenario where the number of bits in the mantissa is bounded (say 4 bits) and we would like to compute $x^2 = 1.1_{(2)} \times 1.1_{(2)} = 10.01_{(2)}$. If our pre-computed exponent were 3 instead of 2, with this limitation, we will lose precision, namely, the result will be $010.0_{(2)}$ instead of $10.01_{(2)}$.

[0073] We thus adopt an intermediate method where we book-keep a sufficiently good bound on the exponent for each value in the program. This bound can be estimated statically, by certain statistical analysis, via a special purpose compiler. This method keeps track of public bounds, as precisely as possible, on the secret value, without revealing the secret value itself. From that point of view, our real number representation is hybrid by combining features of both fixed point and floating point numbers.

[0074] From the compiler's perspective, the exponent can be considered to be public whereas the mantissa can be regarded private. Yet, as the compiler certainly does not know the secret number x , it is impossible for the compiler to determine the exact exponent e that guarantees the above normalization for m , but only an upper bound.

[0075] 2.2 Hybrid plaintext representation of reals

[0076] We introduce the following parameters for representing real numbers in a fixed-point (integer) representation:

- p_{lsb} denotes the bit position of the least-significant bit;
- $(p_{msb} - 1)$ denotes the bit position of the most-significant bit;
- $\rho = p_{msb} - p_{lsb}$ denotes the plaintext numerical window, i.e. the fixed-point precision.

The parameters p_{msb} and p_{lsb} can be determined by the compiler based on statistical analysis of data expected to be operated upon. In practice, the limiting factor is the size of the numerical window ρ (that is, the number of bits of the mantissa m ; equivalently, the difference between the exponent bound kept by the compiler and the number of binary digits in the fractional part to be kept). The smaller the ρ is, the more efficient the arithmetic is on the backend. Thus, the static analysis needs to ensure the following two conditions:

- The statically determined bound is sufficient to avoid overflows.
- The bound is as precise as possible to avoid loss in precision (given the limitation on ρ).

[0077] Definition 1 (integer representation of real numbers). Given parameters p_{msb} and p_{lsb} , the class of plaintext values represented using these parameters is denoted by $\mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ and is defined to be:

$$\mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}} = \{x \in 2^{p_{\text{lsb}}} \cdot \mathbb{Z}, |x| < 2^{p_{\text{msb}}}\} \subset \mathbb{Q}$$

[0078] For example, $x = 1$ will have $p_{\text{msb}} = 1$, $x = 3$ has $p_{\text{msb}} = 2$. Note that the class $\mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ represents all real (in this case, rational) numbers strictly between $-2^{p_{\text{msb}}}$ and $2^{p_{\text{msb}}}$ with step $2^{p_{\text{lsb}}}$. Note that p_{lsb} is often negative (when we want to compute exact integral parts and a few bits of precision from the fractional parts), but in general, this need not be the case. Negative p_{lsb} means that we are considering bits for the fractional part. The $\rho = p_{\text{msb}} - p_{\text{lsb}}$ essentially tells us how many bits we need to represent the number. The more negative p_{lsb} , the higher precision you have on the float and the more bits you need to represent it.

Example 1. For $p_{\text{msb}} = 40$ and $p_{\text{lsb}} = -10$, we have $\rho = 40 - (-10) = 50$. In this case, the least significant bit would be in position -10 and the most significant bit would be in position 39, with the least significant non-fractional bit in position 0.

Example 2. For $p_{\text{lsb}} = -2$ and $p_{\text{msb}} = 5$, the number is represented as binary as $1011.1_{(2)}$ and in $\mathcal{P}_{5, -2}$ as $01011.10_{(2)}$. In this case, two fractional bits would occupy the bit positions -2 and -1, while the non-fractional bits would occupy positions 0 through 4. Notice that a given real number can belong to many different plaintext classes (i.e., 11.5 in the above example belongs to $\mathcal{P}_{5, -2}$, but it also belongs to $\mathcal{P}_{5, -10}$ as well as $\mathcal{P}_{10, -2}$).

[0079] Figure 2 illustrates the relationship between the bits p_{msb} and p_{lsb} in a fixed point representation and the associated bits that would be affected in underflow and overflow conditions.

[0080] Plaintext overflow. The set $\mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ is neither a group nor a ring; in particular, it is not stable under the real arithmetic operations such as addition, subtraction, multiplication, etc. For example, the number 3 belongs to $\mathcal{P}_{2, 0}$, but $3 + 3$, $-3 - 3$ and $3 \times$

3 all exceed the bound $2^{p_{\text{msb}}}$. More generally, we call plaintext overflow any case where the result of an expression exceeds the $2^{p_{\text{msb}}}$ of the output plaintext set.

[0081] To avoid the risk of plaintext overflow, we ensure p_{msb} is at least $\log_2(x)$. But the parameters p_{msb} and p_{lsb} of a variable have to be specified at compile time (before knowing the actual value of the value x), so to mitigate the risk, we need to set p_{msb} to be an upper bound for all possible values. This ensures that no overflow occurs when assigning the actual value to the variable. This also means that the actual (or floating-point) precision is always smaller than the fixed-point precision $p_{\text{msb}} - p_{\text{lsb}}$.

[0082] Plaintext underflow. If the fixed exponent is chosen to be too large, it is possible to completely lose the plaintext information. For example, if $x = 1 \times 10^{-3}$ and $y = 6 \times 10^{-6}$, $xy = 6 \times 10^{-6}$ then for the floating point representation one decimal digit is enough to represent the mantissa of the product and the computed exponent is $\lceil \log_2(x) \rceil = -18$.

[0083] On the other hand, in the fixed-point world, if the plaintext parameters of the inputs have been set to $(p_{\text{msb}}, p_{\text{lsb}}) = (0, -14)$ which corresponds to 4 decimal digits of precision, a common mistake would be to assume that the product p_{msb} is the sum of the two inputs p_{msb} , and here, expect a result of the same 14 bits of fixed-point precision, so with parameters $(p_{\text{msb}}, p_{\text{lsb}}) = (0, -14)$: in this case, the result is computed as: $0.0010 \times 0.0060 = 0.0000$ which completely loses the result. Such cases are called plaintext underflow, and although we effectively have 14 bits of fixed point precision in this example, the computed result is completely wrong from a floating point perspective.

[0084] 2.3 ModReal - modular representation of real numbers

[0085] The ModReal representation refers to a modular integer representation real arithmetic, which can be based on 64-bit or 128-bit integers, for example. This representation can use the fixed point unit of a computer processor with a fixed-point representation of real numbers, having a fixed number of binary digits after the decimal point.

[0086] To achieve security in multi-party computations, plaintext values of a certain size are secret shared to share values that typically belong to a class of values with larger parameters. In addition, secret shares are masked by values from the larger class. These larger secret shares classes are parametrized by two integers M_{msb} and p_{lsb} . We voluntarily

use the same symbol p_{lsb} , because in all cases, it matches the plaintext least significant position; however, the parameter M_{msb} used for secret shared data will typically be larger than the p_{msb} used for the corresponding plaintext.

- p_{lsb} denotes the bit position of the least-significant bit of the masks / shares;
- $(M_{\text{msb}} - 1)$ denotes the bit position of the most significant bit of the masks / shares.

[0087] Definition 2 (modular representation of real numbers). Given parameters M_{msb} and p_{lsb} , the class used for modular representation of secret shares of real numbers is defined as the finite abelian group

$$\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}} = 2^{p_{\text{lsb}}} \mathbb{Z} / 2^{M_{\text{msb}}} \mathbb{Z}$$

Note that $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ is isomorphic (as an abelian group) to $(\mathbb{Z} / 2^{M_{\text{msb}} - p_{\text{lsb}}} \mathbb{Z}, +)$ and has a natural $\mathbb{Z}$ -module structure that we will use.

[0088] The important property of the class $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ (unlike the set of all real numbers $\mathbb{R}$) that will ultimately allow us to achieve information-theoretic security properties of the MPC computations is that it admits a uniform distribution.

[0089] Example. For $p_{\text{lsb}} = -2$ and $p_{\text{msb}} = 6$, the number $x = 11.5 = 12 + 1 + 2 + 8$ is represented as binary as 1011.1 and in $\mathcal{M}_{6, -2}$ as 001011.10. The opposite number $x' = -11.5 = 52.5 = 1/2 + 4 + 16 + 32 \pmod{64}$ is represented as binary as 110100.1 and in $\mathcal{M}_{6, -2}$ as 110100.10. Note that the bits of the integer parts of x and x' are opposite. The first bit gives the sign of the number: if the first bit is 0 then the number is positive, else it is negative.

[0090] 2.4 ModReal mathematical lifts

[0091] Before we define the notion of secret shares, we present several natural lifts of the torus $\mathbb{R} / 2^M \mathbb{Z}$ (for a positive integer M) to $\mathbb{R}$ (the real numbers) that will be used throughout. Here, a lift means a section of the natural surjection $\mathbb{R} \rightarrow \mathbb{R} / 2^M \mathbb{Z}$. We define four mathematical lifts that go from $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ to $\mathbb{R}$ that are parametrized by an integer 2^M as follows:

- $\text{centermod}_{2^M}(x)$ is defined as the unique real number $\tilde{x} \in \left[-\frac{2^M}{2}, \frac{2^M}{2}\right)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$
- $\text{posmod}_{2^M}(x)$ is defined as the unique real $\tilde{x} \in [0, 2^M)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$

- $\text{quartermod}_{2^M}(x)$ is defined as the unique real $\tilde{x} \in \left[-\frac{2^M}{4}, \frac{3 \cdot 2^M}{4}\right)$ such that $\tilde{x} - x \in 2^M \mathbb{Z}$
- $\text{diffmod}_{2^M}(x)$ is defined as $\text{posmod}_{2^M}(x) - \text{centermod}_{2^M}(x)$. This difference is either 0 or 2^M depending on which half space x belongs to.

[0092] 2.5 Modular representation of shares

[0093] The modular representation can be used advantageously to represent shares of a secret-shared real number.

[0094] Definition 3 (modular representation of secret shares). In one embodiment, a number $x \in \mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ is secret shared in the class $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ as $\llbracket x \rrbracket_{M_{\text{msb}}, p_{\text{lsb}}} = (x_1, \dots, x_k) \in \mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}^k$ if

$$x = \text{centermod}_{2^{M_{\text{msb}}}} \left(\sum_{j=1}^k x_j \right)$$

The share is correct and unconditionally secure if $\log_2|x| < M_{\text{msb}} - 1$, which condition is met when $M_{\text{msb}} \geq p_{\text{msb}} + 1$.

[0095] Advantages of a modular representation over a non-modular representation of MPC shares can be shown by example.

[0096] Example 1. We first consider the case of a non-modular representation of each share using the above non-modular class $\mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$. The plaintext integer 42 can be secret-shared among three parties as [33,97,-88] in $\mathcal{P}_{7,0}$, or as [164301,-806845,642586] in $\mathcal{P}_{20,0}$. In both cases, we need all the three shares to recompute the exact plaintext integer. In the first scenario, however, the sum of the first two shares is quite large compared to $2^7 = 128$, and gives a strong indication that the plaintext integer is positive. The problem is mitigated in the second example, which uses a larger p_{msb} but at the cost of a larger representation.

[0097] Example 2: We next consider the same integer 42 as secret-shared among three parties as [33,97,40] in $\mathcal{M}_{7,0}$. We need all the three shares to recompute the exact integer plaintext, but this time, since $\mathcal{M}_{7,0}$ is a group and has a uniform distribution, the first two shares give no information at all on the plaintext value. The security of such shares, for a coalition of $k - 1$ parties out of k is information theoretic.

[0098] 3 Casting Operations

[0099] In order to support basic arithmetic operations such as addition and multiplication on secret shared real numbers represented in modular integer form, we need to first ensure that the two operands can be cast in compatible classes (plaintext and secret shares classes). It is thus important to enable casting operations to efficiently transform a representation with respect to a given set of parameters to a representation with respect to another set of parameters. In this section, we explain how to transform a set of modular shares from a given class $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ into another set of modular shares from another class $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$ with different parameters $M'_{\text{msb}}, p'_{\text{lsb}}$. The casting operations may or may not require communication depending on whether we decrease or increase the parameters p_{lsb} and M_{msb} .

[0100] Figure 3 illustrates four casting operations for casting representations between classes depending on whether we increase or decrease either of the parameters p_{lsb} and M_{msb} . These four casting operations include Extend, Round, Project and Lift (with Lift being distinct from the four natural lifts referenced above).

[0101] Figure 4 illustrates an example application of each of the four casting operations on an example input value. In the illustrated example, modular shares for the plaintext value $x = 1011.1_{(2)}$ are secret shared among two parties with modular shares $x_1 = 001000.10$ and $x_2 = 000011.00$ in $\mathcal{M}_{6,-2}$. The example shows how the share x_1 is cast to another modular number with a different parameter p_{lsb} or M_{msb} . For simplicity, the operations are presented on scalars, but more generally, all these operations are to be performed independently on each coefficient of any vector/matrix/tensor.

[0102] Extend ($p'_{\text{lsb}} < p_{\text{lsb}}$): The Extend casting operation adds additional least significant bits to each input share and sets the new least significant bits to 0 in the new representation to fill in the new bits in the representation. The Extend operation does not require communication between the parties.

[0103] Round ($p'_{\text{lsb}} > p_{\text{lsb}}$): The Round casting operation deletes least significant bits from each input share. The Round casting operation can be implemented either with or without communication between the parties holding the shares. In the case that the Round casting operation is implemented without communication, each party rounds to remove the

least significant bits locally from their share. This implementation does not guarantee that the p'_{lsb} least significant bits of the plaintext are correct due to carry-related errors. The Round casting operation can, however, be implemented with communication between the parties to ensure that secret shares add up to a plaintext value whose p'_{lsb} are correct. Such an implementation of the Round casting operation that preserves precision, called ShareRefresh, is presented below, but it is more expensive in terms of run-time, memory and communication.

[0104] Project ($M'_{\text{msb}} < M_{\text{msb}}$): The Project casting operation, which also does not require communication between the parties, transforms shares according to the following natural surjective homomorphism:

$$\mathbb{Z}/2^{M_{\text{msb}}-p_{\text{lsb}}}\mathbb{Z} \rightarrow \mathbb{Z}/2^{M'_{\text{msb}}-p_{\text{lsb}}}\mathbb{Z}$$

[0105] Lift ($M'_{\text{msb}} > M_{\text{msb}}$): The Lift casting operation requires communication between the parties as well as offline and online phases. An implementation of the Lift casting operation that computes the results correctly 100% of the time is presented below. This implementation improves on the prior work where the operations are probabilistic and have a non-zero probability of failure.

[0106] Below, we explain an implementation of the Lift casting operation with its online and offline phases in accordance with one embodiment. We do this in a slightly more general way by not necessarily assuming that $p'_{\text{lsb}} = p_{\text{lsb}}$. This has the practical advantage that one can often merge the cast operations Lift and ShareRefresh into a single operation that requires communication instead of two separate ones.

[0107] 3.1 Implementation of the Lift casting operation on secret shares in integer modular representation

[0108] In this section we explain how to transform or cast a set of shares from a first modular class $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ into another set of shares of a second modular class $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$, where the M'_{msb} of the second class is greater than the M_{msb} of the first class, and where the p'_{lsb} of the second class can be less than, equal to or greater than the p_{lsb} of the first class. We will refer to this particular cast operation on secret shares in integer modular representation as Lift (capitalized to distinguish from more general natural lifts described above). In one embodiment, the Lift operation is executed in part offline

(without communication between the parties) and in part online (with communication between the parties) in a system for multi-party computation including a trusted dealer.

[0109] In accordance with one embodiment, the Lift operation operates on a secret shared MPC container (recall, as introduced in the Background section above), which is masked-and-revealed as follows:

- secret shares $\llbracket x \rrbracket_{M_{\text{msb}}, p_{\text{lsb}}}$ of a plaintext $x \in \mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ with $p_{\text{msb}} \leq M_{\text{msb}} - 2$
- mask shares $\llbracket \lambda \rrbracket_{M_{\text{msb}}, p_{\text{lsb}}}$ whose sum λ is uniformly random in $\mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ and unknown by the parties.
- the masked value $a = x + \lambda \in \mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$, revealed and known by all parties.

[0110] The goal of the Lift operation is to compute secret shares $\llbracket z \rrbracket_{M'_{\text{msb}}, p'_{\text{lsb}}}$ of the same plaintext x , where M'_{msb} is larger than M_{msb} .

[0111] Definition 4.

Let $\lambda \in \mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$ and $N := 2^{M_{\text{msb}}}$.

We partition the range $I = \left[-\frac{N}{4}, \frac{3N}{4}\right) \subset \mathbb{R}$ as the disjoint union $I_0 \sqcup I_1$ where

$$I_0 := \left[-\frac{N}{4}, \frac{N}{4}\right) \text{ and}$$

$$I_1 := \left[\frac{N}{4}, \frac{3N}{4}\right).$$

We define a selection variable or bit as follows

$$b_\lambda = \begin{cases} 0 & \text{if } \text{quartermod}_N(\lambda) \in I_0 \Leftrightarrow \text{quartermod}_N(\lambda) < \frac{N}{4} \\ 1 & \text{if } \text{quartermod}_N(\lambda) \in I_1 \Leftrightarrow \text{quartermod}_N(\lambda) \geq \frac{N}{4} \end{cases}$$

In particular, $b_\lambda = 0$ is equivalent to $\lambda \in I_0 + 2^{M_{\text{msb}}} \mathbb{Z}$, which can be rewritten as $\lambda \in [-2^{M_{\text{msb}}-2}, 2^{M_{\text{msb}}-2})$. Similarly $b_\lambda = 1$ is equivalent to $\lambda \in I_1 + 2^{M_{\text{msb}}} \mathbb{Z}$, which can be rewritten as $\lambda \in [2^{M_{\text{msb}}-2}, 3 \times 2^{M_{\text{msb}}-2})$.

[0112] The cast operation leverages these two possibilities, which can be encoded into the bit b_λ , for $\text{quartermod}_N(\lambda)$: $\lambda \in I_0$ or $\lambda \in I_1$. Letting $a = x + \lambda \in 2^{p_{\text{lsb}}} \mathbb{Z} / 2^{M_{\text{msb}}} \mathbb{Z}$, one can consider the two mathematical lifts (both in $\mathbb{R}$)

$$\tilde{a}_0 = \text{centermod}_N(a) \text{ and } \tilde{a}_1 = \text{posmod}_N(a).$$

We can require that the secret shared value x be within the range $x \in [-N/4, N/4)$, and with this condition being met, we observe the following equality:

$$\text{centermod}_N(x) + \text{quartermod}_N(\lambda) = \begin{cases} \tilde{a}_0 = \text{centermod}_N(a) & \text{if } b_\lambda = 0 \\ \tilde{a}_1 = \text{posmod}_N(a) & \text{if } b_\lambda = 1 \end{cases}$$

or more concisely

$$\text{centermod}_N(x) + \text{quartermod}_N(\lambda) = \begin{cases} \tilde{a}_0 & \text{if } b_\lambda = 0 \\ \tilde{a}_1 & \text{if } b_\lambda = 1 \end{cases}.$$

One can logically rewrite this equation as

$$\text{centermod}_N(x) + \text{quartermod}_N(\lambda) = \tilde{a}_0 + b_\lambda(\tilde{a}_1 - \tilde{a}_0) \in \mathbb{R} \quad (1)$$

in order to effectively select between the values $\tilde{a}_0$ and $\tilde{a}_1$ depending on the value of b_λ .

[0113] To obtain shares modulo $2^{M'_{\text{msb}}}$ for x (the unmasking of x into the second class), one can project the above equality to the torus $\mathbb{R}/2^{M'_{\text{msb}}}\mathbb{Z}$ via the natural surjection $\mathbb{R} \twoheadrightarrow \mathbb{R}/2^{M'_{\text{msb}}}\mathbb{Z}$. Since the trusted dealer can precompute and secret share the image v of $\text{quartermod}_N(\lambda)$ (in $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}} = 2^{p'_{\text{lsb}}}\mathbb{Z}/2^{M'_{\text{msb}}}\mathbb{Z}$), the parties can recover $[[x]]_{M'_{\text{msb}}, p'_{\text{lsb}}}$ out of knowledge of

$$\tilde{a}_0 + b_\lambda(\tilde{a}_1 - \tilde{a}_0) \bmod 2^{M'_{\text{msb}}}$$

in the online phase in conjunction with v based on the above equation (1).

[0114] This can be done more particularly as follows. Let $a_j = \tilde{a}_j \bmod 2^{M'_{\text{msb}}}$ for $j \in \{0, 1\}$. The parties can determine $(a_1 - a_0)$ based on the revealed masked value a . The trusted dealer precomputes b_λ and secret shares it as $[[b_\lambda]]$ amongst the parties. Although the secret sharing of b_λ can be done in $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$, we note that all but the top $M'_{\text{msb}} - M_{\text{msb}}$ bits of the binary representation of $(a_1 - a_0)$ will always be zero. It thus suffices for the trusted dealer to secret share the bit b_λ using the smaller modular class $\mathcal{M}_{M'_{\text{msb}} - M_{\text{msb}}, 0}$ in order for the parties to perform multiplications on the shares of $[[b_\lambda]]$ and $(a_1 - a_0)$.

[0115] To summarize, the trusted dealer needs to:

1. Generate a random mask $\lambda \in 2^{p_{\text{lsb}}}\mathbb{Z}/2^{M_{\text{msb}}}\mathbb{Z}$
2. Compute the selection bit b_λ and secret share it in $\mathcal{M}_{M'_{\text{msb}} - M_{\text{msb}}, 0}$
3. Compute a particular lift v of that λ to $2^{p'_{\text{lsb}}}\mathbb{Z}/2^{M'_{\text{msb}}}\mathbb{Z}$ and secret share it in $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$. More precisely,

$$v := \text{roundTo}(\text{quartermod}_{2^{M_{\text{msb}}}}(\lambda), 2^{p'_{\text{lsb}}}) \bmod 2^{M'_{\text{msb}}}$$

where the function $\text{roundTo}(z, 2^\ell)$ takes a real number z and an integer ℓ and rounds z to the nearest integer multiple of 2^ℓ . v can then be secret shared as $[[v]]$ in $\mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$.

[0116] In the online phase of the computation, the parties use the lifts a_0 and a_1 to $2^{p'_{\text{lsb}}}\mathbb{Z}/2^{M'_{\text{msb}}}\mathbb{Z}$ and only extract the top $M'_{\text{msb}} - M_{\text{msb}}$ bits for the oblivious selection in the computation of the above $\text{centermod}_N(x) + \text{quartermod}_N(\lambda)$.

[0117] 3.1.1 Pseudocode Implementation of the Lift operation

[0118] Figure 5 illustrates a pseudocode implementation of an offline phase of the Lift operation in accordance with one embodiment. Figure 6 illustrates a pseudocode implementation of an online phase of the Lift operation in accordance with this embodiment.

[0119] In addition to the `roundTo` function described above, the Lift operation as illustrated in Figures 5 and 6 leverages the `secretShares` function, which takes as inputs a value and a specification of a class, and outputs a plurality of secrets shares summing to the input value in the representation of the supplied specification of the class.

[0120] With respect to the online phase of the operation illustrated in Figure 6, we define the operation $\boxtimes$ (for $x \in 2^{p'_{\text{lsb}}}\mathbb{Z}/2^{M'_{\text{msb}}}\mathbb{Z}$ and $y \in \mathbb{Z}/2^{M'_{\text{msb}} - M_{\text{msb}}}\mathbb{Z}$) as — $x \boxtimes y := \text{msb}_{M'_{\text{msb}} - M_{\text{msb}}}(x) \cdot y$ extended with $M_{\text{msb}} - p'_{\text{lsb}}$ zeroes at the end. In this definition, the function $\text{msb}_N(x)$ outputs the N most significant bits of x .

[0121] 3.2 The ShareRefresh operation

[0122] Definition 5. Let $(M_{\text{msb}}, p_{\text{lsb}})$ and $(M'_{\text{msb}}, p'_{\text{lsb}})$ be two pairs of parameters such that $M'_{\text{msb}} \leq M_{\text{msb}}$. The map

$$\text{projectAndRound}: \mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}} \rightarrow \mathcal{M}_{M'_{\text{msb}}, p'_{\text{lsb}}}$$

is defined as follows: for $z \in \mathcal{M}_{\text{params}}$, let z_0 be any of its representatives in $\mathbb{R}$ and set

$$\text{projectAndRound}_{M'_{\text{msb}}, p'_{\text{lsb}}}(z) := \argmin_{q \in 2^{p'_{\text{lsb}}}\mathbb{Z}} |z_0 - q| \pmod{2^{M'_{\text{msb}}}}$$

One can observe that this can be rewritten as:

$$\text{projectAndRound}_{M'_{\text{msb}}, p'_{\text{lsb}}}(z) := 2^{p'_{\text{lsb}}} \lfloor z_0 2^{-p'_{\text{lsb}}} \rfloor \pmod{2^{M'_{\text{msb}}}}$$

[0123] Notation note: the notation $\lfloor x \rfloor$ is used herein to denote rounding of the number x to the nearest integer.

[0124] Proposition 1. The Definition 5 is well-defined, i.e. it does not depend on the choice of the representative z_0 .

[0125] Proof. Let $z \in M_{\text{params}}$ and let z_1, z_2 be two representatives of z in $\mathbb{R}$. By definition, there exists $\omega \in \mathbb{Z}$ such that $z_1 - z_2 = 2^{M_{\text{msb}}} \omega$. Since $p'_{\text{lsb}} \leq M'_{\text{msb}} \leq M_{\text{msb}}$, we have $2^{M_{\text{msb}} - p'_{\text{lsb}}} \omega \in \mathbb{Z}$. In particular, $\lfloor z_1 2^{-p'_{\text{lsb}}} \rfloor = \lfloor z_2 2^{-p'_{\text{lsb}}} \rfloor + 2^{M_{\text{msb}} - p'_{\text{lsb}}} \omega$. Since $M'_{\text{msb}} \leq M_{\text{msb}}$, we get $2^{p'_{\text{lsb}}} \lfloor z_1 2^{-p'_{\text{lsb}}} \rfloor \equiv 2^{p'_{\text{lsb}}} \lfloor z_2 2^{-p'_{\text{lsb}}} \rfloor \pmod{2^{M'_{\text{msb}}}}$.

[0126] From this point on, we assume that params and params' satisfy the assumptions of Proposition 1.

[0127] In one embodiment, a Share Refresh operation is executed in part offline (without communication between the parties) and in part online (with communication between the parties). Figure 7 illustrates a pseudocode implementation of an offline phase of the Share Refresh operation in accordance with one embodiment. Figure 8 illustrates a pseudocode implementation of an online phase of the Share Refresh operation in accordance with this embodiment.

[0128] 4 Beaver Multiplication

[0129] 4.1 Multiplication on ModReal backend

[0130] The multiplication between two secret shares $\llbracket x \rrbracket_{M_{\text{msb}}^x, p_{\text{lsb}}^x}$ and $\llbracket y \rrbracket_{M_{\text{msb}}^y, p_{\text{lsb}}^y}$ is defined in:

$$\mathcal{M}_{M_{\text{msb}}^x, p_{\text{lsb}}^x} \times \mathcal{M}_{M_{\text{msb}}^y, p_{\text{lsb}}^y} \rightarrow \mathcal{M}_{M_{\text{msb}}^z, p_{\text{lsb}}^z}$$

where M_{msb}^z and p_{lsb}^z are referred to as the output parameters. These parameters are typically determined by the compiler during a static analysis.

[0131] 4.2 Parameter computation expressions

[0132] The following expressions show the computation of various parameters that are used in conjunction with the figures and descriptions of the beaver multiplication operations that follow:

- $p_{\text{lsb}}^{Aw} = \max(p_{\text{lsb}}^A, p_{\text{lsb}}^z - p_{\text{msb}}^B)$
- $p_{\text{lsb}}^{Bw} = \max(p_{\text{lsb}}^B, p_{\text{lsb}}^z - p_{\text{msb}}^A)$
- $p_{\text{lsb}}^W = p_{\text{lsb}}^{Aw} + p_{\text{lsb}}^{Bw}$
- $p_{\text{msb}}^W = \min(p_{\text{msb}}^A + p_{\text{msb}}^B, p_{\text{msb}}^z)$

- $M_{\text{msb}}^W = p_{\text{msb}}^W + 2$
- $\eta = M_{\text{msb}}^W - p_{\text{lsb}}^W$
- $M_{\text{msb}}^{Aw} = p_{\text{lsb}}^{Aw} + \eta$
- $M_{\text{msb}}^{Bw} = p_{\text{lsb}}^{Bw} + \eta$

[0133] 4.3 Beaver multiplication on modular backend (Secret-Secret)

[0134] Figure 9 illustrates a pseudocode implementation of a Beaver multiplication operation on two secret shared values in modular representation in accordance with one embodiment. Figure 10 illustrates a schematic of the Beaver multiplication operation.

[0135] 4.4 Beaver multiplication on modular backend (Secret-Public)

[0136] Figure 11 illustrates a pseudocode implementation of a Beaver multiplication operation on one secret shared value in modular representation and one public value in accordance with one embodiment. Figure 12 illustrates a schematic of the Beaver multiplication operation.

[0137] 5 Linear Combination

[0138] Let $c^{(1)}, \dots, c^{(n)}$ be public real numbers given as double-precision floating point numbers, and let $\llbracket x^{(1)} \rrbracket_*, \dots, \llbracket x^{(n)} \rrbracket_*$ be secret shares with corresponding parameters $M_{\text{msb}}^{(i)}, p_{\text{msb}}^{(i)}, p_{\text{lsb}}^{(i)}$. Let $\beta^{(i)} := \lfloor 2^{24} \cdot c^{(i)} \rfloor$, $\alpha^{(i)} := 2^{-v_2(\beta^{(i)})} \beta^{(i)}$, and $k^{(i)} := 24 - v_2(\beta^{(i)})$, where for a prime p , $v_p(m)$ denotes p -adic valuation of an integer $m \in \mathbb{Z}$. In practice, these integers can be represented as 64-bit signed integers. The public vector

$$\left(\frac{\alpha^{(1)}}{2^{k^{(1)}}}, \dots, \frac{\alpha^{(n)}}{2^{k^{(n)}}} \right) \neq 0$$

will be our approximation of the coefficients. The choice of 24 above is in order to resemble the plaintext of type `float32`, whereby the mantissa is 24 bits.

[0139] The goal is then to compute a secret share of

$$z = \sum_{i=1}^n \frac{\alpha^{(i)}}{2^{k^{(i)}}} \cdot x^{(i)}$$

with corresponding parameters M_{msb}^z and p_{lsb}^z . We assume that the input shares are either simultaneously in floating point representation or in modular representation. In one

embodiment, mixed inputs are not supported and require an explicit cast. The output backend, however, can be different from the input backend (e.g., having inputs in floating point representations and output in modular representation is fine).

[0140] 5.1 Default output parameters

When no plaintext parameters are available for the output, the statistical analyzer of the compiler may use the following propagation formula to estimate the stats of the result: if all plaintexts are independent variables of mean and variance $\mathbb{E}(x^{(i)})$, $\text{Var}(x^{(i)})$, then the mean and variance of the output

$$z = \sum_{i=1}^n c^{(i)} x^{(i)}$$

are:

$$\mathbb{E}(z) = \sum_{i=1}^n c^{(i)} \mathbb{E}(x^{(i)})$$

$$\text{Var}(z) = \sum_{i=1}^n (c^{(i)})^2 \text{Var}(x^{(i)})$$

Note that the last formula is only true for independent variables. In one embodiment, we use the following estimate for the variance of a sum:

$$\begin{aligned} \text{Var}(X + Y) &= \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y) \leq \\ &\leq \text{Var}(X) + \text{Var}(Y) + 2\sqrt{\text{Var}(X) \cdot \text{Var}(Y)} = (\sigma_X + \sigma_Y)^2. \end{aligned}$$

[0141] Based on this propagation formula, the compiler can already infer a reasonable p_{msb} , p_{lsb} for the output container, and then assign a $M_{\text{msb}} > p_{\text{msb}}$ to ensure the correctness and security of the computation. If the compiler cannot make the independence assumption, other techniques should be used to estimate the stats of the result, like simulation on fake values for instance.

[0142] 5.2 Cleaning step

[0143] In one embodiment, before running a linear combination, the compiler preprocesses the linear combination input as follows: for all $1 \leq i \leq n$,

- Remove all indices i for which $\alpha^{(i)} = 0$.
- Remove all indices i for which:

$\log_2|\alpha^{(i)}| - k^{(i)} + p_{\text{msb}}^{(i)} \leq \max_j (\log_2|\alpha^{(j)}| - k^{(j)} + p_{\text{lsb}}^{(j)})$. This means that the term is negligible and can be discarded. Note that if we are dynamically removing terms one-by-one, the right-hand side of the above inequality will not change (a term that is removed cannot have a p_{lsb} that equals the maximum).

[0144] Figure 13 illustrates an example input upon which the cleaning step can be performed. In the example, the interval with label 1 should be discarded as negligible since the p_{msb} position is left of the maximal p_{lsb} position of the remaining intervals.

[0145] Furthermore, letting $\mathcal{I}$ be the remaining index set, we can test according to the following criteria (the output parameters should intersect the input parameters windows):

- Ensure the inequality

$$p_{\text{lsb}}^{(z)} \leq \max_{i \in \mathcal{I}} (\log_2|\alpha^{(i)}| - k^{(i)} + p_{\text{msb}}^{(i)})$$

Otherwise, declare the result negligible or zero.

- Ensure the following inequality

$$p_{\text{msb}}^{(z)} \geq \max_{i \in \mathcal{I}} (\log_2|\alpha^{(i)}| - k^{(i)} + p_{\text{lsb}}^{(i)})$$

Otherwise, one gets undefined behavior.

[0146] 5.3 Linear combination builtin: ModReal case

[0147] In one embodiment, the compiler provides two working parameters for the linear combination: M_{msb}^w and p_{lsb}^w . These are selected in such a way that they satisfy the following properties:

- The lift to the ModReal shares class $\mathcal{M}_{M_{\text{msb}}^w, p_{\text{lsb}}^w}$ of the inputs ensures that the individual scalar products $\frac{\alpha^{(i)}}{2^{k^{(i)}}} \cdot x^i$ can be computed without overflow and keeping sufficient precision (the latter to be defined precisely).
- The sum of ALL secret shares (there are k of them) of ALL individual terms (scalar products; there are n of them) can be computed in this class without an overflow (the latter to be defined precisely).

This makes the following default choice of parameters reasonable:

- $p_{\text{lsb}}^w(1) = p_{\text{lsb}}^z - \log_2(nk)/2,$
- $p_{\text{lsb}}^w(2) = \min_{1 \leq i \leq k} (p_{\text{lsb}}^{(i)} - k^{(i)}),$

- $M_{\text{msb}}^w = M_{\text{msb}}^z,$
- $p_{\text{lsb}}^w = \max(p_{\text{lsb}}^w(1), p_{\text{lsb}}^w(2)).$

[0148] For each $1 \leq i \leq n$, the i th input container is first casted to a ModReal number with parameters $M_{\text{msb}}^w + k^{(i)}$ and $Mp_{\text{lsb}}^w + k^{(i)}$. When $M_{\text{msb}}^w + k^{(i)} > M_{\text{msb}}^{(i)}$, this requires a lift triplet (involving the container's mask λ , and two ephemeral precomputed data b, v). Then, the multiplication by $\alpha^{(i)}/2^{k^{(i)}}$ yields a serializable value of parameters M_{msb}^w and p_{lsb}^w , which gets accumulated with the other values. Finally, the sum is cast to the output parameters.

[0149] Figure 14 illustrates a pseudocode implementation of a Linear combination operation with ModReal representation in accordance with one embodiment. Figure 15 illustrates a schematic of the Linear combination operation.

[0150] 6 Fourier Approximation of Real-Valued Functions

[0151] Normally, a Fourier approximation of a complex-valued function $f(x)$ will be given in the form

$$f(x) \sim \sum_{n=-N}^N a_n e^{2\pi i n x / T}$$

where T is the period and $a_n \in \mathbb{C}$. The coefficients a_n are in general complex numbers, but if $f(x)$ is a real-valued function then $a_{-n} = \overline{a_n}$ for $n > 0$ and $a_0 \in \mathbb{R}$. We then have

$$f(x) \sim \text{Re} \left(\sum_{n=0}^N c_n e^{2\pi i n x / T - i \varphi_n} \right).$$

This yields the transform

$$f(x) \sim \sum_{n=0}^N c_n \cos \left(\frac{2\pi n x}{T} - \varphi_n \right) \quad (2)$$

A Fourier term will be a triple (c_n, n, φ_n) for $n = 0, 1, \dots, N$ where $c_n \in \mathbb{R}$ is the real coefficient and φ_n is the phase.

[0152] As an example, consider the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. The real-valued function $f(x) = \sigma(x) - 1/2$ is then odd, which implies that the phases φ_n are all equal to $\pi/2$.

[0153] 6.1 Fourier series evaluation: ModReal case

[0154] We now explain how to evaluate the expression (2) for $x \in \mathcal{P}_{p_{\text{msb}}, p_{\text{lsb}}}$ and secret shares $[[x]] \in \mathcal{M}_{M_{\text{msb}}, p_{\text{lsb}}}$.

[0155] Recall that the mask $\lambda \in \left[-\frac{1}{4} \cdot 2^{M_{\text{msb}}}, \frac{3}{4} \cdot 2^{M_{\text{msb}}}\right)$.

[0156] The trusted dealer first computes the mask λ , the corresponding bit b_λ and then computes and secret shares (in the offline phase) the vector $c_n e^{-2\pi n \lambda / T - i \varphi_n}$ for $n = 0, 1, \dots, N$.

[0157] The online computation is then performed as follows:

1. Compute $\bar{a} = x + \lambda$ as an element of $\mathbb{Z}/2^{M_{\text{msb}}}\mathbb{Z}$.
2. Compute the correct lift $a = x + \lambda \in \mathbb{R}$ by using the secret shares of $\bar{a}$ and of b_λ . This is possible since $a = \text{centermod}(\bar{a}) + 2^{M_{\text{msb}}} \cdot b_\lambda \cdot \text{msb}(\bar{a})$.
3. Compute $e^{2\pi n a / T}$ for $n = 0, 1, \dots, N$ (this requires the lift a and not just the original $\bar{a}$).
4. Compute secret shares of $c_n e^{2\pi n x / T} = e^{2\pi n a / T} \cdot (c_n e^{-2\pi n \lambda / T - i \varphi_n})$. This requires local multiplications in a certain plaintext class to be determined.

[0158] To determine the plaintext parameters of the result, we impose the constraint that the entire computation occurs within a maximal numerical window of η bits (e.g., $\eta = 64$ if we would like to use only 64-bit integers on the backend).

[0159] First, p_{msb}^z can be estimated using the trivial bound, namely

$$p_{\text{msb}}^z = \log_2 \left(\sum_{n=0}^N |c_n| \right).$$

To get the optimal p_{lsb}^z , let p be the numerical window, that is $p = p_{\text{msb}}^z - p_{\text{lsb}}^z$. Letting $\varepsilon = \lceil \log_2 N \rceil$, our constraint is given by the inequality

$$2p + \varepsilon + 2 \leq \eta,$$

i.e.,

$$p \leq \frac{\eta - \varepsilon - 2}{2}$$

[0160] That means that we need to determine the maximal numerical window for the output of the computation. To see why this inequality should hold, note that if one requires p bits of precision for each of the multipliers in the products

$$c_n e^{2\pi i n x/T} = e^{2\pi i n a/T} \cdot \left(c_n e^{\frac{2\pi i n \lambda}{T} - \varphi_n} \right),$$

hence, the $2p$. We then have to take the sum of ε of these terms and add the two additional bits for the modular shares class. Hence, we can set

$$p_{\text{lsb}}^z = p_{\text{msb}}^z - \left\lfloor \frac{\eta - \varepsilon - 2}{2} \right\rfloor$$

In addition, the compiler needs to provide working parameters for the Fourier series builtin:

- $(M_{\text{msb}}^{aw}, p_{\text{lsb}}^{aw})$ - lifting parameters for the masked value $a = x + \lambda$,
- $(M_{\text{msb}}^{lw}, p_{\text{lsb}}^{lw})$ - lifting parameters for the mask λ .

[0161] To determine the working parameters, we start by determining the least significant bit positions (we do this in order to guarantee sufficient precision). The following formulas are reasonable:

- $p_{\text{lsb}}^{aw} = -p$ and $M_{\text{msb}}^{aw} = p_{\text{lsb}}^{aw} + \eta$,
- $p_{\text{lsb}}^{lw} = p_{\text{lsb}}^z - \varepsilon$ and $M_{\text{msb}}^{lw} = p_{\text{lsb}}^{lw} + \eta$.

This choice requires some justification: the parameter $p_{\text{lsb}}^{aw} = -p$ ensures that we have a numerical window of size p in the computation of $e^{2\pi i n a/T}$. Indeed, the most significant bit position for that quantity is 0, so we need at least p binary digits in the fractional part. Yet, the computation assumes that both inputs to Beaver require the same numerical window, that is, we have a multiplication that can be viewed as a map

$$\mathcal{M}_{M_{\text{msb}}^{aw}, p_{\text{lsb}}^{aw}} \times \mathcal{M}_{M_{\text{msb}}^{lw}, p_{\text{lsb}}^{lw}} \rightarrow \mathcal{M}_{p_{\text{lsb}}^{lw} + p_{\text{lsb}}^{aw} + \eta, p_{\text{lsb}}^{lw} + p_{\text{lsb}}^{aw}}.$$

[0162] 7 Communication model

[0163] Figures 16A and 16B illustrate schematics of communication channels between computing systems during an offline phase of a multi-party computation (MPC). Figure 16A shows the communication channels for a trusted dealer model while Figure 16B shows the channels for the honest but curious model. In the trusted dealer model, the dealer sends numerical masking data (also referred to as triplets) to each party via a private channel. In the honest but curious model, the parties have access to a private broadcast channel shared between all of them, and each party shares an additional private channel with the dealer. The private channels are denoted with dashed lines. In the trusted dealer model, the dealer generates the masking data and uses private channels to send to each party a respective share of the data (one-way arrows). In the honest but curious model, the parties

collaborate for the generation of the numerical masking data, and to do this, they need an additional private broadcast channel between them that is not accessible to the dealer. Although only three parties are shown, the model can be extended to an arbitrary number of parties.

[0164] Figure 17 illustrates a schematic of communication channels between parties during an online phase in accordance with one embodiment. The parties send and receive masked values via a public broadcast channel denoted with solid lines. The online phase is the same in both the trusted dealer and honest but curious models and the dealer is not present.

[0165] Although the methods disclosed herein are in some cases described with respect to the trusted dealer model, these methods can also be used with the honest but curious model.

[0166] 8 Computer Implementation

[0167] Components of the embodiments disclosed herein, which may be referred to as methods, processes, applications, programs, modules, engines, functions or the like, can be implemented by configuring one or more computers or computer systems using special purpose software embodied as instructions on a non-transitory computer readable medium. The one or more computers or computer systems can be or include one or more standalone, client and/or server computers, which can be optionally networked through wired and/or wireless networks as a networked computer system.

[0168] The special purpose software can include one or more instances thereof, each of which can include, for example, one or more of client software, server software, desktop application software, app software, database software, operating system software, and driver software. Client software be configured to operate a system as a client that sends requests for and receives information from one or more servers and/or databases. Server software can be configured to operate a system as one or more servers that receive requests for and send information to one or more clients. Desktop application software and/or app software can operate a desktop application or app on desktop and/or portable computers. Database software can be configured to operate one or more databases on a system to store data and/or information and respond to requests by client software to

retrieve, store, and/or update data. Operating system software and driver software can be configured to provide an operating system as a platform and/or drivers as interfaces to hardware or processes for use by other software of a computer or computer system. By way of example, any data created, used or operated upon by the embodiments disclosed herein can be stored in, accessed from, and/or modified in a database operating on a computer system.

[0169] Figure 18 illustrates a general computer architecture 1800 that can be appropriately configured to implement components disclosed in accordance with various embodiments. The computing architecture 1800 can include various common computing elements, such as a computer 1801, a network 1818, and one or more remote computers 1830. The embodiments disclosed herein, however, are not limited to implementation by the general computing architecture 1800.

[0170] Referring to Figure 18, the computer 1801 can be any of a variety of general purpose computers such as, for example, a server, a desktop computer, a laptop computer, a tablet computer or a mobile computing device. The computer 1801 can include a processing unit 1802, a system memory 1804 and a system bus 1806.

[0171] The processing unit 1802 can be or include one or more of any of various commercially available computer processors, which can each include one or more processing cores that can operate independently of each other. Additional co-processing units, such as a graphics processing unit 1803, also can be present in the computer.

[0172] The system memory 1804 can include volatile devices, such as dynamic random access memory (DRAM) or other random access memory devices. The system memory 1804 can also or alternatively include non-volatile devices, such as a read-only memory or flash memory.

[0173] The computer 1801 can include local non-volatile secondary storage 1808 such as a disk drive, solid state disk, or removable memory card. The local storage 1808 can include one or more removable and/or non-removable storage units. The local storage 1808 can be used to store an operating system that initiates and manages various applications that execute on the computer. The local storage 1808 can also be used to store special purpose software configured to implement the components of the embodiments

disclosed herein and that can be executed as one or more applications under the operating system.

[0174] The computer 1801 can also include communication device(s) 1812 through which the computer communicates with other devices, such as one or more remote computers 1830, over wired and/or wireless computer networks 1818. Communications device(s) 1812 can include, for example, a network interface for communicating data over a wired computer network. The communication device(s) 1812 can include, for example, one or more radio transmitters for communications over Wi-Fi, Bluetooth, and/or mobile telephone networks.

[0175] The computer 1801 can also access network storage 1820 through the computer network 1818. The network storage can include, for example, a network attached storage device located on a local network, or cloud-based storage hosted at one or more remote data centers. The operating system and/or special purpose software can alternatively be stored in the network storage 1820.

[0176] The computer 1801 can have various input device(s) 1814 such as a keyboard, mouse, touchscreen, camera, microphone, accelerometer, thermometer, magnetometer, or any other sensor. Output device(s) 1816 such as a display, speakers, printer, or eccentric rotating mass vibration motor can also be included.

[0177] The various storage 1808, communication device(s) 1812, output devices 1816 and input devices 1814 can be integrated within a housing of the computer, or can be connected through various input/output interface devices on the computer, in which case the reference numbers 1808, 1812, 1814 and 1816 can indicate either the interface for connection to a device or the device itself as the case may be.

[0178] Any of the foregoing aspects may be embodied in one or more instances as a computer system, as a process performed by such a computer system, as any individual component of such a computer system, or as an article of manufacture including computer storage in which computer program instructions are stored and which, when processed by one or more computers, configure the one or more computers to provide such a computer system or any individual component of such a computer system. A server, computer server, a host or a client device can each be embodied as a computer or a computer system. A

computer system may be practiced in distributed computing environments where operations are performed by multiple computers that are linked through a communications network. In a distributed computing environment, computer programs can be located in both local and remote computer storage media.

[0179] Each component of a computer system such as described herein, and which operates on one or more computers, can be implemented using the one or more processing units of the computer and one or more computer programs processed by the one or more processing units. A computer program includes computer-executable instructions and/or computer-interpreted instructions, such as program modules, which instructions are processed by one or more processing units in the computer. Generally, such instructions define routines, programs, objects, components, data structures, and so on, that, when processed by a processing unit, instruct the processing unit to perform operations on data or configure the processor or computer to implement various components or data structures.

[0180] Components of the embodiments disclosed herein, which may be referred to as modules, engines, processes, functions or the like, can be implemented in hardware, such as by using special purpose hardware logic components, by configuring general purpose computing resources using special purpose software, or by a combination of special purpose hardware and configured general purpose computing resources. Illustrative types of hardware logic components that can be used include, for example, Field-programmable Gate Arrays (FPGAs), Application-specific Integrated Circuits (ASICs), Application-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), and Complex Programmable Logic Devices (CPLDs).

[0181] 9 Concluding comments

[0182] Although the subject matter has been described in terms of certain embodiments, other embodiments that may or may not provide various features and aspects set forth herein shall be understood to be contemplated by this disclosure. The specific embodiments described above are disclosed as examples only, and the scope of the patented subject matter is defined by the claims that follow. In the claims, the term “based upon” shall include situations in which a factor is taken into account directly and/or

indirectly, and possibly in conjunction with other factors, in producing a result or effect. In the claims, a portion shall include greater than none and up to the whole of a thing; encryption of a thing shall include encryption of a portion of the thing. In method claims, any reference characters are used for convenience of description only, and do not indicate a particular order for performing a method.