

**METHODS AND SYSTEMS FOR ADDITION, MULTIPLICATION, SUBTRACTION,
AND DIVISION OF RATIONAL NUMBERS ENCODED IN THE DOMAIN OF FAREY
RATIONALS FOR MPC SYSTEMS**

Cross Reference to Related Applications

[0001] This application is a continuation-in-part of U.S. patent application Serial No. 18/535,875, filed December 11, 2023, entitled “Methods and Systems for P -Adic Encoding and Decoding of Rational Data for FHE Systems,” which was based upon and claimed the benefit of U.S. provisional application Serial No. 63/386,700, filed December 9, 2022, entitled “PIE p -adic Encoding for High-Precision Arithmetic in Homomorphic Encryption,” all of which are specifically incorporated herein by reference for all that they disclose and teach. This application is further based upon and claims the benefit of U.S. Provisional Application Serial No. 63/471,940, filed June 8, 2023, entitled “Mercury: Constant-Round Protocols for Multi-Party Computation with Rationals,” all of which is also specifically incorporated herein by reference for all that it discloses and teaches.

Background of the Invention

[0002] The advancement of science is possible when knowledge is shared and information is exchanged in a seamless manner. In a world where many businesses rely on information as their main assets, analysis over data is a crucial competitive advantage. Consequently, the amount of data processed and stored will continue to increase, creating a demand for virtualized services. To this end, some applications can be provided as cloud computing resources including Internet of Things (IoT), machine learning, virtual reality (VR) and blockchain. As a result, concerns about custody and privacy of data are on the rise.

[0003] Modern concealment/encryption employs mathematical techniques that manipulate positive integers or binary bits. Asymmetric concealment/encryption, such as RSA (Rivest-Shamir-Adleman), relies on number theoretic one-way functions that are predictably difficult to factor and can be made more difficult with an ever-increasing size of the encryption keys. Symmetric encryption, such as DES (Data Encryption Standard) and AES (Advanced Encryption Standard), uses bit manipulations within registers to shuffle the concealed text/cryptotext/ciphertext to increase “diffusion” as well as register-based operations with a shared key to increase “confusion.” Diffusion and confusion are measures for the increase in statistical entropy on the data payload being transmitted. The concepts of diffusion and confusion in encryption are normally attributed as first being identified by Claude Shannon in the 1940s. Diffusion is generally thought of as complicating the mathematical process of

generating unencrypted (plain text) data from the encrypted (cryptotext/ciphertext) data, thus, making it difficult to discover the encryption key of the concealment/encryption process by spreading the influence of each piece of the unencrypted (plain) data across several pieces of the concealed/encrypted (cryptotext) data. Consequently, an encryption system that has a high degree of diffusion will typically change several characters of the concealed/encrypted (cryptotext/ciphertext) data for the change of a single character in the unencrypted (plain) data making it difficult for an attacker to identify changes in the unencrypted (plain) data. Confusion is generally thought of as obscuring the relationship between the unencrypted (plain) data and the concealed/encrypted (cryptotext) data. Accordingly, a concealment/encryption system that has a high degree of confusion would entail a process that drastically changes the unencrypted (plain) data into the concealed/encrypted (cryptotext/ciphertext) data in a way that, even when an attacker knows the operation of the concealment/encryption method (such as the public standards of RSA, DES, and/or AES), it is still difficult to deduce the encryption key.

[0004] Homomorphic Encryption is a form of encryption that allows computations to be carried out on concealed ciphertext as it is concealed/encrypted without decrypting the ciphertext that generates a concealed/encrypted result which, when decrypted, matches the result of operations performed on the unencrypted plaintext.

[0005] The word homomorphism comes from the ancient Greek language: ὁμός (homos) meaning “same” and μορφή (morphe) meaning “form” or “shape.” Homomorphism may have different definitions depending on the field of use. In mathematics, for example, homomorphism may be considered a transformation of a first set into a second set where the relationship between the elements of the first set are preserved in the relationship of the elements of the second set.

[0006] For instance, a map f between sets A and B is a homomorphism of A into B if

$$f(a_1 \text{ op } a_2) = f(a_1) \text{ op } f(a_2) \mid a_1, a_2 \in A$$

where “op” is the respective group operation defining the relationship between A and B .

[0007] More specifically, for abstract algebra, the term homomorphism may be a structure-preserving map between two algebraic structures such as groups, rings, or vector spaces. Isomorphisms, automorphisms, and endomorphisms are typically considered special types of homomorphisms. Among other more specific definitions of homomorphism, algebra homomorphism may be considered a homomorphism that preserves the algebra structure between two sets.

[0008] Multi-Party Computation (MPC) is a cryptographic technique that allows multiple parties to jointly compute a function over their private data without revealing it to each other. MPC works by using complex encryption to distribute computation between multiple parties.

MPC enables parties to share data for computing tasks and obtain the output, but no party learns anything about others' data or secrets.

Summary of the Invention

[0009] An embodiment of the present invention may comprise a method for Multi-Party Computation (MPC) on a Multi-Party Computation (MPC) system of rational data encoded in a domain of Farey rationals, the method comprising: encoding by a first data server computing device a first rational number x_0/y_0 into a first encoded integer corresponding to the first rational number x_0/y_0 as an encode function of p -adic arithmetic in the domain of Farey rationals that is induced by a ring isomorphism performed on the first rational number x_0/y_0 , such that the first encoded integer preserves inverses, and additive and multiplicative homomorphic properties; sending by the first data server computing device the first encoded integer to a third data server computing device; encoding by a second data server computing device a second rational number x_1/y_1 into a second encoded integer corresponding to the second rational number x_1/y_1 as the encode function of p -adic arithmetic in the domain of Farey rationals that is induced by a ring isomorphism performed on the second rational number x_1/y_1 , such that the second encoded integer preserves inverses, and additive and multiplicative homomorphic properties; sending by the second first data server computing device the second encoded integer to the third data server computing device; computing by the third data server computing device an arithmetic function as part of the MPC system with the first encoded integer and the second encoded integer to obtain a result encoded integer; sending by the third data server computing device the result encoded integer to a fourth data server computing device; and decoding by the fourth data server computing device the result encoded integer into a result rational number x_r/y_r corresponding to the result encoded integer as a decode inverse function of p -adic arithmetic in the domain of Farey rationals performed on the result encoded integer.

[0010] An embodiment of the present invention may further comprise a Farey rational encoding system for Multi-Party Computation (MPC) on a Multi-Party Computation (MPC) system of rational data encoded in a domain of Farey rationals, the Farey rational encoding system comprising: a first data server computing system that encodes a first rational number x_0/y_0 into a first encoded integer corresponding to the first rational number x_0/y_0 as an encode function of p -adic arithmetic in the domain of Farey rationals that is induced by a ring isomorphism performed on the first rational number x_0/y_0 , such that the first encoded integer preserves inverses, and additive and multiplicative homomorphic properties and sends the first encoded integer to a third data server computing device; a second data server computing device that encodes a second rational number x_1/y_1 into a second encoded integer corresponding to the

second rational number x_l/y_l as the encode function of p -adic arithmetic in the domain of Farey rationals that is induced by a ring isomorphism performed on the second rational number x_l/y_l , such that the second encoded integer preserves inverses, and additive and multiplicative homomorphic properties and sends the second encoded integer to the third data server computing device; the third data server computing device that computes an arithmetic function as part of the MPC system with the first encoded integer and the second encoded integer to obtain a result encoded integer and sends the result encoded integer to a fourth data server computing device; and the fourth data server computing device that decodes the result encoded integer into a result rational number x_r/y_r corresponding to the result encoded integer as a decode inverse function of p -adic arithmetic in the domain of Farey rationals performed on the result encoded integer.

Brief Description of the Drawings

- [0011] In the drawings,
- [0012] FIG. 1 is a block diagram of the hardware implementation for an embodiment.
- [0013] FIG. 2 is a flow chart of operations for an embodiment.

Detailed Description of the Embodiments

[0014] Secure computation has become a necessity in the modern world. Its applications are widespread: from allowing medical researchers to compute statistics over private patient data without violating HIPAA (Health Insurance Portability and Accountability Act) to helping large companies like Meta and Google avoid GDPR (General Data Protection Regulation) fines. A ubiquitous and popular choice for secure computation is Multi-Party Computation (MPC). Most MPC protocols work over finite fields or rings, which means that encoding techniques are required to map rational-valued data into the algebraic structure being used. The various embodiments herein leverage an encoding technique introduced in U.S. patent application Serial No. 18/535,875, filed December 11, 2023, entitled “Methods and Systems for P -Adic Encoding and Decoding of Rational Data for FHE Systems,” and are presented herein as **Mercury**—a family of protocols for addition, multiplication, subtraction, and division of rational numbers. Notably, the output of an embodiment’s division protocol is exact (i.e., it does not use iterative methods). The protocols of the various embodiments offer significant improvements in both round complexity and communication complexity when compared with prior art, and are secure for a dishonest minority of semi-honest parties. To reiterate, the encoding technique of the various embodiments’ protocols are based on is composable, so it can be paired with any MPC protocol over a prime-order field.

[0015] 1 Introduction

[0016] Secure computation is a tool which allows functions of private data to be evaluated without revealing those data. One such tool is homomorphic encryption (HE), which allows, e.g., a data owner to encrypt their data and outsource computations on those data. More specifically, anyone can compute reasonable functions of the encrypted data and never learn more about the data than what the output of the function reveals. However, most HE schemes are prohibitively slow, especially when computing products. Further, most HE schemes, with the exception of CKKS (Cheon, Kim, Kim, and Song), do not support division. The CKKS scheme can perform division by approximating the inverse function with a Taylor polynomial $T(x) \approx x^{-1}$ and then computing $c_0/c_1 \approx c_0 T(c_1)$, for ciphertexts c_0, c_1 .

[0017] Another well-studied form of secure computation is Multi-Party Computation (MPC). In the classic setting, n mutually distrusting parties P_i possess private data d_i and wish to jointly compute a function $F(d_1, \dots, d_n)$ without revealing d_i to P_j for $i \neq j$. Much work has been done extending these results, developing new tools for MPC, and implementing these tools in the real world. Many of these protocols rely on *secret sharing*. In secret sharing, each P_i is provided masked pieces (called *shares*) of private data (henceforth, *secrets*). These shares are chosen such that only authorized sets of parties can determine a secret if they pool their shares. The parties use their shares to perform computations on the secrets by communicating (e.g., sending/receiving shares, creating and sending new shares, etc.) with one another as needed. A common primitive for secret sharing is Shamir's scheme, which is based on polynomial interpolation over a finite field. An advantage of the Shamir scheme is that it is *additively homomorphic* so that shares of two secrets can be added locally to give a sharing of the sum of those secrets. Additively homomorphic secret sharing is also used by the well-known MP-SPDZ framework.

[0018] Most MPC protocols are defined over finite rings or fields such as $\mathbb{Z}/n\mathbb{Z}$ or $GF(2^f)$, and as such require real data (often in the form of fixed-point or floating-point numbers) to be encoded as elements of the ring (field). Further, since the goal is evaluating certain functions (e.g., polynomials) over secret shared values, the encoding method must be homomorphic with respect to the operations which compose the function (e.g., addition and multiplication). Some works encode a fixed-point or floating-point number a with precision f as the integer $a \cdot 2^f$. Other approaches work with floating-point numbers by separately encoding the sign, exponent, and significand, along with an extra bit that is set to 0 if, and only if, the number is 0.

[0019] The various embodiments differ significantly from all of these approaches. Instead of choosing a set of fixed-point numbers and designing the protocols around them, the various embodiments start with a set of rationals (with bounded numerators and denominators) which

can be constructed to contain a desired set of fixed-point numbers. This set of rationals, paired with an encoding that is homomorphic with respect to both addition and multiplication, forms the basis of the protocols of the various embodiments. The range of the encoding may be any ring/field of the form $\mathbb{Z}/n\mathbb{Z}$. This means that, for the most part, the protocols of the various embodiments are obtained by simply “attaching” the encoder to existing protocols. The exception is division which relies heavily on the structure of the aforementioned set of rationals.

[0020] The protocols of the various embodiments require only a constant number of rounds and have communication complexity $O(tn)$, where n is the number of parties and $t - 1$ is a bound on the number of corrupted parties. The protocols of the various embodiments are also generic, by which it is meant that the protocols do not depend on the underlying primitive being used. For example, even though the description of the various embodiments presented herein use Shamir’s scheme as the example foundation, the protocols of the various embodiments for rational arithmetic could easily be translated to use additive secret sharing (e.g., implemented with MP-SPDZ). Moreover, the encoding technique on which the protocols of the various embodiments are based is composable, meaning that the encoding works with any MPC protocol based on secret sharing over a prime-order field.

[0021] 2 Preliminaries

[0022] 2.1 Notation

[0023] For a positive integer a , $\mathbb{Z}/a\mathbb{Z}$ denotes the ring of integers modulo a . In case a is prime, we write $\mathbb{F}_a$. The elements of $\mathbb{Z}/a\mathbb{Z}$ will be represented by integers $0, 1, \dots, a - 1$. For a ring R , $R[x]$ will denote the ring of polynomials with coefficients in R . We use $y \leftarrow A(x)$ to denote that a randomized algorithm A on input x outputs y . If A is deterministic, we simply write $y = A(x)$. All circuits we consider are arithmetic over a field $\mathbb{F}_a$, and have fan-in 2.

[0024] Shamir’s scheme. Here is a brief overview of Shamir Secret Sharing (SSS), and the notation used therein. Suppose we have n parties and wish for any set of $t \leq n$ parties to be able to reconstruct a secret by pooling their shares. This is called t -out-of- n threshold scheme. One creates *Shamir shares* of a secret $s \in \mathbb{F}_p$ by generating a random polynomial $f(x) \in \mathbb{F}_p[x]$ of degree at most $t - 1$ whose constant term is s (i.e., $f(0) = s$ and whose remaining coefficients are chosen uniformly from $\mathbb{F}_p$). Shares of s are the field elements $f(i)$, $i \in \mathbb{F}_p \setminus \{0\}$. We assume the i^{th} party receives the share $f(i)$. We use $[x]_i$ to denote the i^{th} party’s share of $x \in \mathbb{F}_p$, and $[x]$ to denote a sharing of x among all parties. Then, for example, $[x] + [y]$ will mean “each party adds their share of x to their share of y ,” and $c[x]$ will mean “each party multiplies their share of x by c .” Any collection of t parties can pool their shares $[s]$ and reconstruct the polynomial f using Lagrange interpolation, thereby obtaining the secret $s = f(0)$.

[0025] 2.1 Framework

[0026] As mentioned above, we use t -out-of- n SSS over $\mathbb{F}_p$ for all protocols. The various embodiment's protocols are built on the basic Shamir addition, and the multiplication introduced by Gennaro, Rabin and Rabin, which is itself an optimization of the multiplication described by Ben-Or, Goldwasser and Wigderson. To ensure the multiplication protocol works, the various embodiments require $t - 1 < n/2$. The various embodiments also tolerate up to $t - 1$ semi-honest (honest-but-curious) adversaries. The communication complexity of a protocol is measured by the number of field elements sent by all parties throughout the protocol. When comparing with protocols that measure communication complexity in bits, the embodiments simply multiply its communication complexities by $\log_2(p)$.

[0027] 3 Protocols for Rational Numbers

[0028] A family of efficient MPC protocols for performing computations with rational numbers is proposed herein. These protocols are obtained by pairing an encoder that maps certain rational numbers to integers with the familiar SSS protocols for addition, multiplication, and computing modular inverses. The protocols for computing the sum and product of shared fixed-point numbers remain unchanged from the analogous SSS primitives, except that rational operands are encoded to field elements before any protocols are executed. Division is an amalgam of existing protocols and relies on the fact that the various embodiments' mapping for encoding rational numbers to integers is induced by a ring homomorphism, and therefore preserves inverses; likewise is true for the decode mapping. Further elaboration is given below.

[0029] 3.1 Encoding Rationals

[0030] The various embodiments use the encoding map introduced in U.S. patent application Serial No. 18/535,875, filed December 11, 2023, entitled "Methods and Systems for P -Adic Encoding and Decoding of Rational Data for FHE Systems," which maps a subset of rationals, whose denominators are co-prime with a prime p , into $\mathbb{F}_p$ using p -adic encoding and decoding supported by p -adic number theory.. This map is defined by $\text{encode}(x/y) = xy^{-1} \bmod p$, and has as its domain the *Farey rationals*:

$$\mathcal{F}_N := \{x/y : |x| \leq N, 0 < y \leq N, \gcd(x, y) = 1, \gcd(p, y) = 1\},$$

where $N = N(p) := \lfloor \sqrt{(p-1)/2} \rfloor$. encode is induced by a ring isomorphism, so both it and its inverse decode are additively and multiplicatively homomorphic as long as the composition of operands in $\mathcal{F}_N$ remains in $\mathcal{F}_N$ (e.g., $\text{encode}\left(\frac{x_0}{y_0} + \frac{x_1}{y_1}\right) = \text{encode}\left(\frac{x_0}{y_0}\right) + \text{encode}\left(\frac{x_1}{y_1}\right)$ iff $\frac{x_0}{y_0}, \frac{x_1}{y_1}, \frac{x_0}{y_0} + \frac{x_1}{y_1} \in \mathcal{F}_N$). decode can be computed efficiently using a slight

modification of the Extended Euclidean Algorithm. We summarize important properties of `encode`, `decode`, and $\mathcal{F}_N$ with the following lemma.

[0031] Lemma 1. *Let p be a prime, $N = N(p)$, and `encode`, `decode` be the encode and decode maps, respectively.*

- (i) *If $x/y \in \mathcal{F}_N$, then $-x/y \in \mathcal{F}_N$.*
- (ii) *If $x/y \in \mathcal{F}_N$ is nonzero, then $y/x \in \mathcal{F}_N$.*
- (iii) *$\mathcal{F}_N$ is not closed under addition and multiplication.*
- (iv) *$[-N, N] \cap \mathbb{Z} \subseteq \mathcal{F}_N$. Moreover, if $z \in [0, N] \cap \mathbb{Z}$, then `encode`(z) = z .*
- (v) *`encode` and `decode` are homomorphic with regard to addition and multiplication as long as the composition of operands in $\mathcal{F}_N$ remains in $\mathcal{F}_N$.*

[0032] *Proof.* (i)-(iv) are obvious.

[0033] For (v), the mapping $H_{p^r} : \mathcal{F}_N \rightarrow \mathbb{Z}_{p^r}$ and its inverse are defined as:

$$H_{p^r}\left(\frac{x}{y}\right) = xy^{-1} \bmod p^r,$$

$$H_{p^r}^{-1}(h) = \text{MEEA}(p^r, h).$$

[0034] The H -mapping is injective and, therefore, gives a unique representation of each element of $\mathcal{F}_N$ in $\mathbb{Z}_{p^r}$. The inverse of H_{p^r} is well-defined.

[0035] For all $x/y \in \mathcal{F}_N$ and $h \in H_{p^r}(\mathcal{F}_N) \subseteq \mathbb{Z}_{p^r}$,

- (i) $H_{p^r}^{-1}(H_{p^r}(x/y)) = x/y$,
- (ii) $H_{p^r}(H_{p^r}^{-1}(h)) = h$,
- (iii) *If $a, a' \in \mathbb{Z}$ and $a' = a \pmod{p^r}$, then $H_{p^r}^{-1}(a) = H_{p^r}^{-1}(a')$.*

[0036] The mappings H_{p^r} and $H_{p^r}^{-1}$ are homomorphic with respect to addition and multiplication in the following sense.

- (i) *For all $u, u' \in \mathcal{F}_N$, $H_{p^r}(u) \cdot H_{p^r}(u') = H_{p^r}(u \cdot u') \bmod p^r$ and $H_{p^r}(u) + H_{p^r}(u') = H_{p^r}(u + u') \bmod p^r$.*
- (ii) *If $h, h' \in \mathbb{Z}_{p^r}$ and $H_{p^r}^{-1}(h) \cdot H_{p^r}^{-1}(h'), H_{p^r}^{-1}(h) + H_{p^r}^{-1}(h') \in \mathcal{F}_N$, then $H_{p^r}^{-1}(h \cdot h') = H_{p^r}^{-1}(h) \cdot H_{p^r}^{-1}(h')$ and $H_{p^r}^{-1}(h + h') = H_{p^r}^{-1}(h) + H_{p^r}^{-1}(h')$.*

[0037] *Proof.* (i) Let $a/b, c/d \in \mathcal{F}_N$. By definition of the Farey rationals, a, b, c, d are co-prime with p . That H_{p^r} is homomorphic with respect to addition and multiplication follows from the properties of congruences:

$$ac(bd)^{-1} = (ab^{-1})(cd^{-1}) \bmod p^r$$

$$\text{and } (ad + bc)(bd)^{-1} = (ab^{-1} + cd^{-1}) \bmod p^r$$

[0038] (ii) Invoking the homomorphic property of H_{p^r} , from $H_{p^r}^{-1}(h) \cdot H_{p^r}^{-1}(h'), H_{p^r}^{-1}(h) + H_{p^r}^{-1}(h') \in \mathcal{F}_N$ we obtain $h \cdot h', h + h' \in H_{p^r}(\mathcal{F}_N)$. By proposition 1(ii), $H_{p^r}(H_{p^r}^{-1}(h \cdot h')) = h \cdot h'$ and $H_{p^r}(H_{p^r}^{-1}(h + h')) = h + h'$. The result follows from the injectivity of H_{p^r} .

[0039] **PIE: A Rational Encoder/Decoder**

[0040] The encoding/decoding map introduced in U.S. patent application Serial No. 18/535,875, is given as a PIE (p -adic encoding) rational encoder and decoder as described in more detail below.

[0041] Let g be a positive integer, $N = \lfloor \sqrt{(g-1)/2} \rfloor$, and make $\mathcal{F}_N$ the input space. We define encoding and decoding as follows:

$$\text{PIE.Encode}(x/y). \text{ For } x/y \in \mathcal{F}_N \text{ output } H_g(x/y)$$

$$\text{PIE.Decode}(z). \text{ For } z \in \mathcal{Z}_g, \text{ output } H_g^{-1}(z)$$

[0042] For all $m, m' \in \mathcal{F}_N$ such that $m \cdot m' \in \mathcal{F}_N$,

$$\text{PIE.Decode}([\text{PIE.Encode}(m) \cdot \text{PIE.Encode}(m') \bmod g]) = m \cdot m'$$

and $\forall m, m' \in \mathcal{F}_N$ such that $m + m' \in \mathcal{F}_N$

$$\text{PIE.Decode}([\text{PIE.Encode}(m) + \text{PIE.Encode}(m') \bmod g]) = m + m'$$

[0043] Further, let p be a multivariate polynomial with coefficients in $\mathbb{Q}$. For all $m_0, \dots, m_k \in \mathcal{F}_N$ such that $p(m_0, \dots, m_k) \in \mathcal{F}_N$,

$$\text{PIE.Decode}(g, p(\text{PIE.Encode}(g, m_0), \dots, \text{PIE.Encode}(g, m_k)) \bmod g)$$

$$= p(m_0, \dots, m_k).$$

[0044] For the encoding (and decoding) to yield the correct result when used in an MPC scheme, one must ensure that if two or more elements from $\mathcal{F}_N$ are combined using additions and/or multiplications then any intermediates and the final output must not lie outside the set $\mathcal{F}_N$. For this reason, we will define the (rational) message space to be the following subset of $\mathcal{F}_N$:

$$\mathcal{G}_M = \{x/y \in \mathcal{F}_N : 0 \leq |x| \leq M, 1 \leq y \leq M\}$$

[0045] The main idea behind choosing a subset of $\mathcal{F}_N$ as the set of messages is that when elements from $\mathcal{G}_M$ are combined, the resulting element can be in $\mathcal{F}_N$. Ensuring the output lands in $\mathcal{F}_N$ induces a bound on the number of computations that can be performed, and determines the choice of parameters involved therein. At this point, one might wonder whether we need to do something similar with the range $\mathcal{Z}_g$ of the encoder to make sure that overflow modulo g does not occur during computations. The answer is “no”. This is because proposition 3(iii) along with the above message space restriction imply that overflow modulo g does not affect decoding.

[0046] The choice of M depends jointly on the rational data one must encode, and the circuits one must evaluate over those data. We elaborate this in the following section.

[0047] Choosing the Message Space $\mathcal{G}_M$.

[0048] We will describe an arithmetic circuit in terms of the multivariate polynomial it computes. To this end, recall that the ℓ_1 -norm of a polynomial is simply the sum of the absolute values of its coefficients.

[0049] *Polynomials with which PIE is compatible.* Let $\mathcal{P}_{d,t}$ denote the set of polynomials in $\mathbb{Q}[x_1, x_2, \dots]$ with total degree at most d and ℓ_1 -norm at most t , whose coefficients have absolute value at least 1. For example, $\mathcal{P}_{d,t}$ contains polynomials of the form:

$$p(x_1, \dots, x_k) = \sum_{d_1 + \dots + d_k \leq d} \sum_{\alpha=1}^I c_\alpha x_1^{d_1} x_2^{d_2} \dots x_k^{d_k},$$

where each $|c_\alpha| \geq 1$, and $\sum_\alpha |c_\alpha| \leq t$.

[0050] The following establishes an upper bound on the output of a polynomial in $\mathcal{P}_{d,t}$ when all inputs are from $\mathcal{G}_M$.

[0051] If $x_1/y_1, \dots, x_k/y_k \in \mathcal{G}_M$, $p \in \mathcal{P}_{d,t}$ is k -variate, and $p(x_1/y_1, \dots, x_k/y_k) = x/y$, then:

$$|x| \leq t \cdot M^{dt} \text{ and } |y| \leq M^{dt}$$

[0052] *Proof.* Note that $\mathcal{P}_{d,t}$ can be written as, $p = \sum_i c_i p_i$, where $\sum_i |c_i| \leq t$, each $|c_i| \geq 1$, and each p_i is a monomial of degree at most d .

[0053] Let $p = \sum_{i=1}^I c_i p_i$.

[0054] Since $\deg(p_i) \leq d$, the evaluation $p_i(x_1/y_1, \dots, x_k/y_k)$ is a fraction of the form:

$$\frac{a_i}{b_i} = \frac{x_{i_1} x_{i_2} \dots x_{i_\ell}}{y_{i_1} y_{i_2} \dots y_{i_\ell}}, \text{ for some } \ell \leq d \text{ and } \{i_1, \dots, i_\ell\} \subseteq \{1, \dots, k\}.$$

[0055] As each $x_i/y_i \in \mathcal{G}_M$, we have $|a_i|, |b_i| \leq M^\ell \leq M^d$.

[0056] Since $x/y = \sum_{i=1}^I c_i \cdot a_i/b_i$, there are nonzero integers α and β such that:

$$\begin{aligned} \alpha x &= (c_1 a_1) b_2 b_3 \dots b_I + b_1 (c_2 a_2) b_3 \dots b_I + b_1 b_2 \dots b_{I-1} (c_I a_I) \\ &\quad \text{and } \beta y = b_1 b_2 \dots b_I. \end{aligned}$$

[0057] It follows from $\sum |c_i| \leq t$ and the above bound on $|a_i|, |b_i|$ that:

$$|x| \leq \sum_{i=1}^I |c_i| (M^d)^I \leq t \cdot M^{dI} \text{ and } |y| \leq M^{dI}.$$

[0058] The proof is completed by observing that $|c_i| \geq 1$, for all i , implies $I \leq t$.

[0059] A sufficient condition for compatibility of PIE with polynomials in $\mathcal{P}_{d,t}$ as described above:

$$\begin{aligned} &\text{PIE.Decode}(g, p(\text{PIE.Encode}(g, m_0), \dots, \text{PIE.Encode}(g, m_k)) \bmod g) \\ &\quad = p m_0, \dots, m_k \end{aligned}$$

is

$$M \leq \left(\frac{N}{t}\right)^{\frac{1}{dt}}, \text{ equivalently } d \leq \frac{\log(N) - \log(t)}{t \log(M)}.$$

[0060] *Proof.* Suppose M is chosen according to the above equation, and let $p \in \mathcal{P}_{d,t}$ be k -variate. According to the above descriptions, if $\mathbf{m} \notin \mathcal{G}_M^k$ and $p(\mathbf{m}) = x/y$, then:

$$\begin{aligned} |x| &\leq t \cdot M^{dt} \leq t \cdot \left((N/t)^{\frac{1}{dt}}\right)^{dt} = N, \text{ and} \\ |y| &\leq M^{dt} \leq \left((N/t)^{\frac{1}{dt}}\right) = N/t \leq N. \end{aligned}$$

[0061] Clearly $\gcd(g, y) = 1$, since y is a factor of the product of the denominators in $\mathbf{m}$. Thus $p(\mathbf{m}) \in \mathcal{F}_N$, and the proof is completed.

[0062] 3.2 Building Blocks

[0063] SSS primitives will be the algorithms/protocols **Share**, **Recon**, **Add**, **ScalarMult**, and **Mult**. Respectively, these create shares of a secret value $s \in \mathbb{F}_p$, reconstruct a secret value given enough shares $[s]$, compute the shares of the sum of two secrets without revealing either, compute shares of the product of a secret value and a known value without revealing the secret value, and compute shares of the product of two secrets without revealing either. Note that **Recon**, **Add**, and **ScalarMult** are all performed locally. By “performed locally” it is meant that the parties use their shares to execute the protocol without communicating with each other. For example, for **Recon** a party (or the dealer, if one exists) in possession of at least t shares reconstructs the secret using polynomial interpolation. $s \leftarrow \text{Output}([s])$ is used to mean that each of a set of $t - 1$ parties sends their share of s to another party, which subsequently computes $s = \text{Recon}([s])$ and sends s to the other $n - 1$ parties. The Gennaro, Rabin and Rabin multiplication protocol is described below.

$$[xy] \leftarrow \text{Mult}([x], [y])$$

1. For $i = 1, \dots, n$, the i^{th} party P_i computes $z_i = [x]_i \cdot [y]_i$;
2. for $i = 1, \dots, 2t - 1$, P_i computes $[z_i] \leftarrow \text{Share}(z_i)$ (One could use any set of $2t - 1$ parties for this step);
3. once each party $P_1, \dots, P_n$ has received $2t - 1$ shares, each party P_i locally computes $[xy]_i$ using Lagrange interpolation on the received shares;
4. output $[xy]$.

[0064] Two additional protocols, **RandInt** and **Inv**, are required for the rational division protocol of the various embodiments. These protocols allow all parties to obtain shares of a random ring element and compute shares of the multiplicative inverse of a ring element, respectively.

$$[r] \leftarrow \text{RandInt}()$$

1. Each of a set of t parties select a uniformly random $r_i \in \mathbb{F}_p, i = 1, \dots, t$;
2. each of the t parties do $[r_i] \leftarrow \text{Share}(r_i)$, and send the j^{th} share to P_j ;
3. $[r] \leftarrow \text{Add}([r_1], \dots, [r_t])$;
4. return $[r]$.

$$[x^{-1}] \leftarrow \text{Inv}([x])$$

1. $[r] \leftarrow \text{RandInt}()$;
2. $[rx] \leftarrow \text{Mult}([r], [x])$;
3. $rx = \text{Output}([rx])$;

4. abort and restart if $rx = 0$, otherwise continue;
5. each party locally computes $(rx)^{-1} = x^{-1}r^{-1} \bmod p$;
6. each party does $[x^{-1}] = \text{ScalarMult}(x^{-1}r^{-1}, [r])$;
7. return $[x^{-1}]$.

Protocol	Rounds	Comm. Complexity
Share	1	$n - 1$
Recon	0	0
Output	2	$(n - 1) + (t - 1)$
Add	0	0
Mult	1	$(2t - 1)(n - 1)$
ScalarMult	0	0
RandInt	1	$t(n - 1)$
Inv	4	$t(3n - 2) - 1$

[0065] Table 1: *Total communication complexity (measured in field elements) of SSS building block protocols.*

[0066] Note that **Share** has communication complexity n (initial sharing of a secret), unless one party is sharing with the remaining parties, in which case it has communication complexity $n - 1$. The latter is the only one relevant to our protocols.

[0067] **3.3 Rational Addition, Multiplication, Subtraction, and Division**

[0068] Addition and multiplication are obtained by simply pairing the encoder **encode** with **Add** and **Mult**. To represent shares of the encoding of $x/y \in \mathcal{F}_N$, we write $[\text{encode}(x/y)]$. We first present the four protocols, and then list their complexities in table 2. For all protocols, we use the field $\mathbb{F}_p$, and $\frac{x_0}{y_0}, \frac{x_1}{y_1} \in \mathcal{F}_N$ such that:

$$\frac{x_0}{y_0} \pm \frac{x_1}{y_1}, \frac{x_0}{y_0} \cdot \frac{x_1}{y_1}, \frac{x_0}{y_0} \div \frac{x_1}{y_1} \in \mathcal{F}_N. \text{ Let } t_0 = \text{encode}(x_0/y_0) \text{ and } t_1 = \text{encode}(x_1/y_1).$$

[0069] *Remark 1.* We use the prefix “Hg” for our protocols because it is the chemical symbol for Mercury in the Periodic Table of Elements.

$$[\text{encode}(x_0/y_0 + x_1/y_1)] = \text{HgAdd}([\text{encode}(x_0/y_0)], [\text{encode}(x_1/y_1)])$$

1. $[t_0 + t_1] = \text{Add}([t_0], [t_1])$;
2. return $[t_0 + t_1] = [\text{encode}(x_0/y_0 + x_1/y_1)]$.

$$[\text{encode}(x_0 x_1 / y_0 y_1)] = \text{HgMult}([\text{encode}(x_0 / y_0)], [\text{encode}(x_1 / y_1)])$$

1. $[t_0 t_1] \leftarrow \text{Mult}([t_0], [t_1]);$
2. return $[t_0 t_1] = [\text{encode}(x_0 x_1 / y_0 y_1)].$

$$[\text{encode}(x_0 / y_0 - x_1 / y_1)] = \text{HgSubtr}([\text{encode}(x_0 / y_0)], [\text{encode}(x_1 / y_1)])$$

1. All parties compute $\text{encode}(-1) = -1_{\text{field}} \in \mathbb{F}_p;$
2. all parties compute $[-t_1] = \text{ScalarMult}(-1_{\text{field}}, [t_1]);$
3. $[t_0 - t_1] = \text{HgAdd}([t_0], [-t_1]);$
4. return $[t_0 - t_1] = [\text{encode}(x_0 / y_0 - x_1 / y_1)].$

$$[\text{encode}(x_0 y_1 / y_0 x_1)] = \text{HgDiv}([\text{encode}(x_0 / y_0)], [\text{encode}(x_1 / y_1)])$$

1. $[t_1^{-1}] \leftarrow \text{Inv}([t_1]);$
2. $[t_0 t_1^{-1}] \leftarrow \text{HgMult}([t_0], [t_1^{-1}]);$
3. return $[t_0 t_1^{-1}] = [\text{encode}(x_0 y_1 / y_0 x_1)].$

Mercury		
Protocol	Rounds	Comm. Complexity
HgAdd	0	0
HgMult	1	$(2t - 1)(n - 1)$
HgSubtr	0	0
HgDiv	5	$t(5n - 4) - n$

[0070] Table 2: Total communication complexity (measured in field elements) of rational addition, multiplication, subtraction, and division protocols.

[0071] Remark 2. **ScalarMult** can be turned into **HgScalarMult** by simply encoding a public element $\alpha \in \mathcal{F}_N$, and then computing $[\text{encode}(\alpha)s] = \text{ScalarMult}(\text{encode}(\alpha), [s])$. Note that **HgScalarMult** also serves as a *division by public divisor* protocol – simply replace $\alpha \neq 0$ by $1/\alpha$.

[0072] 4 Which rational numbers can we use?

[0073] The various embodiments' protocols use the aforementioned Farey rationals. As mentioned in lemma 1 above, $\mathcal{F}_N$ is closed under additive inverses and under multiplicative inverses, but it is not closed under addition and multiplication. This means that a suitable subset of $\mathcal{F}_N$ must be chosen as the set of rational inputs. In particular, we must include fractions with “small” numerators and denominators so that adding/multiplying these fractions yields fractions

that remain in $\mathcal{F}_N$. Following closely the analysis of previously mentioned U.S. patent application Serial No. 18/535,875, filed December 11, 2023, entitled “Methods and Systems for P -Adic Encoding and Decoding of Rational Data for FHE Systems,” this set will be chosen as:

$$\mathcal{G}_{X,Y} := \{x/y \in \mathcal{F}_N | X, Y \in [0, N], |x| \leq X, 0 < y \leq Y\}.$$

[0074] 4.1 Compatibility with fixed-point numbers

[0075] We now highlight the various embodiments’ (partial) compatibility with fixed-point arithmetic. Further, many previous works designed their protocols with fixed-point arithmetic in mind. So, to facilitate comparison with prior art, we briefly discuss conditions under which $\mathcal{F}_N$ contains a set of fixed-point numbers.

[0076] Fixed-point numbers. These are rational numbers represented as a list of digits split by a radix point, and are defined by an integer (represented in a particular base b) in a given range along with a fixed scaling factor f . For example, we can represent decimal numbers with integral part in the range $(-10^{\ell+1}, 10^{\ell+1})$ and up to f decimal places after the radix point as $a \cdot 10^{-f} = a/10^f, a \in (-10^{\ell+f+1}, 10^{\ell+f+1})$. We will represent a set of fixed point numbers with a tuple of the form (b, ℓ, f) , where b is the base, $(-b^{\ell+1}, b^{\ell+1})$ is range of the integer part, and up to f base- b digits are allowed after the radix point. The set of Farey rationals $\mathcal{F}_N$ contains the fixed-point numbers given by (b, ℓ, f) as long as:

$$N \geq \max\{b^{\ell+f+1} - 1, b^f - 1\}. \quad \text{Eq. 1}$$

[0077] Of course, N should be sufficiently large to ensure that adding/multiplying the fixed-point numbers does not cause overflow. While $\mathcal{F}_N$ can be made to contain a set of fixed-point numbers with precision f , addition and multiplication of Farey rationals does not quite coincide with addition and multiplication of fixed-point numbers. This is because the fixed-point representation requires the precision to remain f after each operation (this necessitates a truncation protocol), while $\mathcal{F}_N$ allows the precision to increase until overflow occurs and the output of a computation is no longer correct. However, this disparity mostly vanishes if the system running the protocols only allows precision f , because the “extra” bits that would be removed via a truncation protocol are removed by the system itself.

[0078] We will use the fact that $\mathcal{F}_N$ contains certain fixed-point numbers in Section 6 where we compare emboiments’ protocols with prior work.

[0079] 4.2 Compatible Circuits

[0080] Again borrowing from previously mentioned U.S. patent application Serial No. 18/535,875, filed December 11, 2023, entitled “Methods and Systems for P -Adic Encoding and Decoding of Rational Data for FHE Systems,” for positive integers d, t we define (d, t) -permitted (arithmetic) circuits over $\mathbb{Q}$ to be those that compute a polynomial $p \in \mathbb{Q}[x_1, x_2, \dots]$ such that: (i) p has ℓ_1 norm at most t , (ii) total degree at most d , and (iii) coefficients with absolute value greater than or equal to 1. Note that nonzero polynomials $p \in \mathbb{Z}[x_1, x_2, \dots]$ with $\|p\|_1$ and $\deg(p) \leq d$ satisfy (iii). Let $\mathcal{C}_{d,t}$ be the set of (d, t) -permitted circuits, and $\mathcal{P}_{d,t}$ be the set of polynomials the circuits compute. The following lemma 2 is further described in U.S. patent application Serial No. 18/535,875 at Proposition 7.

[0081] Lemma 2. Let $C \in \mathcal{C}_{d,t}$ with inputs from $\mathcal{G}_{X,Y} \subseteq \mathcal{F}_N$. If x/y is the output of C , then $|x| \leq tX^dY^{d(t-1)}$ and $|y| \leq Y^{dt}$.

[0082] Proof. This is a slight modification of the proof of U.S. patent application Serial No. 18/535,875 at Proposition 7. The bounds given in lemma 2 allow us to determine the (d, t) -permitted circuits with output fractions in $\mathcal{F}_N$ given inputs from $\mathcal{G}_{X,Y}$. Intuitively, the bound on x is larger than the bound on y because the numerator grows faster than the denominator when fractions are summed versus when they are multiplied. For example, $(a/b)(c/d) = (ad/bc)/bd$ versus $(a/b)(c/d) = ac/bd$. Some “space” is wasted if, for example, one chooses $X = Y$, because then the numerator bound becomes the limiting factor for ensuring the output of a permitted circuit remains in $\mathcal{F}_N$. For fixed d and t , this can be avoided by judiciously choosing X to be smaller than Y . In particular, X and Y should satisfy $\log_2(t)/d + \log_2(X) = \log_2(Y)$. Of course this is not possible for many applications, so typically the numerator bound will remain the limiting factor for depth of compatible circuits.

[0083] For example, suppose p is a prime such that $N = \lfloor \sqrt{(p-1)/2} \rfloor$ is 1024 bits, and let $X = 2^{32}, Y = 2^{14}$. According to Lemma 2, we can only evaluate circuits in $\mathcal{C}_{d,t}$ with inputs in $\mathcal{G}_{X,Y}$ if $t(2^{32})^d(2^{14})^{d(t-1)} \leq 2^{1024}$ or, equivalently, if $\log_2(t) + 18d + 14dt \leq 1024$. Notice how the numerator bound from Lemma 2 is the only one needed to determine d and t . The following table shows some choices for d and t .

$ \text{num} \leq 2^{32}, \text{denom} = 2^{14}$						
d	1	2	3	4	5	10
t	71	35	22	16	13	6

[0084] Table 3: Possible values of d (total degree of polynomial computed by permitted circuit) and t (ℓ_1 norm of polynomial computed by permitted circuit) for fractions with numerators bounded in absolute value by 2^{32} and denominators bounded by 2^{14} .

[0085] This is not particularly useful, as many applications require thousands or even millions of additions to be performed on shared values. However, for many applications one is likely to work with decimal numbers with a small number of significant digits; e.g., fractions with denominators a power of 10 between 7 bits and 17 bits. In such cases, we can significantly improve the bounds on d and t . In general, if the fractional data all have the same denominator, then Lemma 2 yields the following corollary.

[0086] **Corollary 1.** Let $C \in \mathcal{C}_{d,t}$ with inputs from $\mathcal{F}_N$ whose denominators are all some power of integer base $b > 0$, say $B = b^e$, and whose numerators are bounded in absolute value by X . If x/y is the output of C , then $|x| \leq t(XB)^d$ and $y \leq B^d$.

[0087] Rehashing the above example ($X = 2^{32}$ and N 1024 bits) with $B = 2^{14}$ we get $t(2^{32}2^{14})^d \leq 2^{1024} \Rightarrow \log_2(t) \leq 1024 - 46d$ and $(2^{14})^d \leq 2^{1024} \Rightarrow d \leq 73$. The bound on d is in fact even smaller: since the ℓ_1 norm of a polynomial in $\mathcal{P}_{d,t}$ is at least 1, $\log_2(t) \geq 0 \Rightarrow 1024 - 46d \geq 0 \Rightarrow 22 \geq d$. This yields the following table.

$ \text{num} \leq 2^{32}, \text{denom} = 2^{14}$						
d	1	2	10	15	20	22
t	2^{978}	2^{932}	2^{564}	2^{334}	2^{104}	2^{12}

[0088] Table 4: Possible values of d (total degree of polynomial computed by a permitted circuit) and t (ℓ_1 norm of polynomial computed by permitted circuit) for fractions with numerators bounded in absolute value by 2^{32} and denominators all equal to 2^{14} .

[0089] So if we restrict inputs to have the same denominators, we can perform an enormous number of additions and a reasonable number of multiplications before the output lands outside of $\mathcal{F}_N$. We can do even better though.

[0090] *Degree-constant circuits.* Each gate of an arithmetic circuit computes a polynomial over (some of) the inputs. We define a *degree-constant (arithmetic) circuit* to be one in which every gate computes a polynomial whose monomial summands all have the same degree; e.g., a dot product. The goal of introducing these circuits is to ensure that whenever two fractions are summed, they already have a common denominator.

[0091] **Corollary 2.** Let $C \in \mathcal{C}_{d,t}$ be degree-constant with inputs from $\mathcal{F}_N$ whose denominators are all $B = b^e$ and whose numerators are bounded in absolute value by $X > 0$. If x/y is the output of C , then $|x| \leq tX^d$ and $y \leq B^d$.

[0092] *Proof.* This follows easily from the fact that whenever two terms are added during the evaluation of a degree-constant circuit, they already have a common denominator which is a power of B .

[0093] Again, using a 1024 bit N , $X = 2^{32}$, and $B = 2^{14}$, we get the inequalities $\log_2(t) \leq 1024 - 32d$ and $d \leq 32$, yielding the following table.

$ \text{num} \leq 2^{32}, \text{denom} = 2^{14}$						
d	1	2	10	15	25	31
t	2^{992}	2^{960}	2^{704}	2^{544}	2^{384}	2^{32}

[0094] Table 5: Possible values of d (total degree of polynomial computed by a permitted circuit) and t (ℓ_1 norm of polynomial computed by permitted circuit) for degree-constant $C \in \mathcal{C}_{d,t}$ taking inputs from $\mathcal{F}_N$ with numerators bounded in absolute value by 2^{32} and denominators all equal to 2^{14} .

[0095] *Incorporating division.* Once we allow divisions, the bounds given in Corollary 1 and Corollary 2 no longer apply, since the numerator of the divisor becomes a factor of denominator of the quotient. This means we should perform any necessary divisions as late as possible relative to other operations.

[0096] 5 Optimizations

[0097] The complexity of HgMult and HgDiv can be reduced by executing parts of the protocols asynchronously in an *offline phase*. This allows certain parts of the protocols to be executed before the desired computation, thereby reducing the *online* complexity. The complexity of the offline phase depends on chosen primitives, existence of a trusted dealer, etc. Henceforth, we emphasize the online round complexity and the online communication complexity.

[0098] We utilize two ubiquitous tools for optimization: Beaver triples (introduced in [3]) for more efficient multiplication, and Pseudo-Random Secret Sharing (PRSS, [11]) to generate random field elements *sans* interaction.

[0099] In PRSS, the parties agree on a pseudo-random function (PRF) $\psi(\cdot)$ and a common input a . They then use pre-distributed keys r_A (one for each set A of $n - t$ parties) to locally compute shares of a random field element s using Replicated Secret Sharing (see [11] for details). The use of PRSS reduces the online round and communication complexity of RandInt

from 1 and $t(n - 1)$ to 0 and 0, respectively. Further, we assume that the PRF, common input, and keys are agreed upon and distributed during a *set-up phase*, whence using PRSS makes the offline round and communication complexity of **RandInt** both 0.

[00100] Beaver triples are 3-tuples of shares $([a], [b], [c])$ satisfying $ab = c$, and can be generated asynchronously in the offline phase using PRSS and **Mult**. These triples can be used to multiply secrets without interaction. In particular, shares $[xy]$ can be obtained from $[x]$ and $[y]$ using only **Add**, **ScalarMult**, **Recon**, **Output**, and one Beaver triple $([a], [b], [c])$:

$$[xy] = (x + a)[y] - (y + b)[a] + [c].$$

[00101] Used triples must be discarded, else information is leaked. This means that a sufficiently-large reservoir of Beaver triples should be maintained to allow the desired functions to be computed.

[00102] These optimizations reduce the online complexities of **HgMult** and **HgDiv**, and leave the complexities of **HgSubtr** and **HgAdd** the same. Table 6 below summarizes the improvements.

Protocol	Online Rounds	Offline Rounds	Online Comm.	Offline Comm.
HgAdd	0	0	0	0
HgMult	0	0	0	$(2t - 1)(n - 1)$
HgSubtr	0	0	0	0
HgDiv	2	2	$(t - 1) + (n - 1)$	$2(2t - 1)(n - 1)$

[00103] Table 6: *Optimized round and communication complexities for our protocols. Note that the nonzero offline communication complexities come from **Mult** during Beaver triple generation.*

[00104] We use the complexities listed in table 6 for the comparisons in section 6. Henceforth, “rounds” will mean “online rounds + offline rounds”, and “total communication” will mean “online communication + offline communication”.

[00105] 6 Comparison with Prior Work

[00106] Catrina and Saxena introduced semi-honest secure protocols for fixed-point multiplication and division – their division is based on (the iterative) Goldschmidt’s method. A variant of their protocol is used by MP-SPDZ for fixed-point division. Catrina subsequently improved the round and communication complexities. To measure the complexities of their protocols, they use the set of fixed-point numbers given by $(2, 2f, f)$; i.e., the set of rationals $a \cdot 2^{-f}$ with $a \in [-2^{-2f}, 2^{2f}) \cap \mathbb{Z}$. Their fixed-point encoding technique requires a field $\mathbb{F}_q$ with

$q > 2^{2f+\kappa}$, κ a statistical security parameter. For our protocols, we use the same field $\mathbb{F}_q$, whence our set of rationals is $\mathcal{F}_N$ with $N = N(q)$; specifically $\log_2(N) \geq f + \kappa/2$. Table 7 shows that for reasonable values of n and t (e.g., $n = 3$, $t = 2$), our protocols far outperform those of Catrina's subsequent improvements.

	Protocol	Rounds	Online Comm.	Offline Comm.
Mercury	Multiplication	0	0	$(2t - 1)(n - 1)$
	Division	3	$(t - 1) + (n - 1)$	$2(2t - 1)(n - 1)$
[7]	Multiplication	1	1	f
	Division	$9 + \theta$	$10f + 2\theta$	$16f + 4\theta f$

[00107] Table 7: *Complexity comparison between our work and that of [7]. Both the online and offline communication complexity are measured in elements of $\mathbb{F}_q$ sent among **all** parties throughout a protocol. n and t are the number of parties and the threshold, resp., θ is the number of iterations of Goldschmidt's method, and f is the fixed-point precision.*

[00108] For example, the authors suggest $f = 32$ and $\theta = 5$, which results in a 14 round division with online communication complexity 320 field elements. Taking $n = 3$ and $t = 2$, our division requires 2 rounds, and has online communication complexity 3 field elements. There is, however, a bit more subtlety to this comparison. As mentioned in Section 4.1, operations on fixed-point numbers require a truncation, and the protocols of Catrina et al. use truncation. Consequently, there is no limit to how many times they can multiply/divide two fixed-point numbers. However, there is a number of multiplications, say, that will render their outputs of little use because so many bits have been truncated. Our limitation, on the other hand, is overflow: computations over $\mathcal{F}_N$ are only meaningful if all intermediate outputs and the final output are in $\mathcal{F}_N$. We can address this in two ways: (i) only take inputs from the subset $\mathcal{G}_{X,Y} \subseteq \mathcal{F}_N$ defined in section 4, for X, Y sufficiently smaller than N , or (ii) use a larger field than Catrina. As long as we don't choose too large a field, (ii) will preserve our complexity advantage.

[00109] Another interesting solution, albeit only for integer division, was proposed by Veugen and Abspoel. They propose three division variations: public divisor, private divisor (only one party knows the divisor), and secret divisor (hidden from all parties). Their protocols are implemented using MP-SPDZ with three parties, and runtimes along with communication complexities (in MB) for dividing a k -bit integer by a $k/2$ -bit integer are provided. Even though our division protocol uses rationals in general, comparison makes sense because $\mathcal{F}_N$ contains the

integers $[-N, N] \cap \mathbb{Z}$ (see lemma 1). For comparison, we use $n = 3$ and $t = 2$, and use the smallest prime field $\mathbb{F}_p$ allowed by Veugen and Abspoel:

$$\log_2(p) \approx 4\max\{\log_2(\text{dividend}), \log_2(\text{divisor})\} + 40$$

[00110] For example, this means that for a 64 bit dividend and 32 bit divisor, we have $\log_2(p) = 296$ and $N = N(p) \approx 148$ bits.

[00111] The reader should take the comparison in table 8 with a grain of salt since our numbers were estimated and not obtained from implementation.

dividend bits	8	16	32	64
divisor bits	4	8	16	32
Mercury	0.001MB	0.0015MB	0.0024MB	0.0042MB
[20] (semi-honest security)	8.6MB	32.6MB	121.0MB	492.7MB

[00112] Table 8: *Total communication complexity in megabytes (MB) of our division protocol (applied to integers) vs. the (secret divisor) integer division protocol of Veugen and Abspoel. The communication complexities for our work were estimated using table 7.*

[00113] The last comparison we shall show is against Pika. Pika uses Function Secret Sharing to construct a three-party protocol (one party is used only to generate correlated randomness) for computing functions such as reciprocal, sigmoid, and square root. The protocol Pika takes as inputs (binary) fixed-point numbers x with precision f , such that $x \cdot 2^f \in (-2^{k-1}, 2^{k-1}]$, and creates shares in the ring $\mathbb{Z}_{2^\ell}$, where $\ell \geq 2k$. For comparison, we choose $N = N(p) = 2^{k-1}$ (meaning we share secrets over $\mathbb{F}_p$ with $p \approx 2^{2k}$). This guarantees that $\mathcal{F}_N$ contains the fixed-point numbers used by Pika regardless of the chosen precision f . As with the preceding comparisons, we again take $n = 3$ and $t = 2$.

[00114] Using the same parameter values for (semi-honest secure) Pika as the author, we found that that total communication complexity for securely computing the reciprocal with $k = 16$ and $\ell = 32$ was 8524 bits over three rounds (one offline). In contrast, we can compute the reciprocal of an element of $\mathcal{F}_{2^{15}}$ in three rounds (one offline) with communication complexity $9\log_2(p) \approx 18k = 288$ bits.

[00115] 7 Conclusions

[00116] *Conclusion.* This work uses Shamir Secret Sharing with a minority of semi-honest adversaries, but **Mercury** is flexible in the sense that it can be easily realized over other primitives with better security assumptions; e.g. additive secret sharing *à la* MP-SPDZ along with a majority of malicious adversaries. **Mercury** provides an efficient low round and communication complexity solution to exact computation over rational numbers using MPC. A cost of exactness, though, is that our protocols are not intrinsically compatible with fixed-point arithmetic. Instead of truncating after every operation to not exceed the chosen fixed-point precision, we allow the precision to grow until overflow occurs. This means that we may need to work over a larger field $\mathbb{F}_p$ than prior art, but our communication and round complexity are sufficiently low as to make using a slightly larger field not problematic.

Hardware implementation for an embodiment (Fig. 1)

[00117] Fig. 1 is a block diagram 100 of the hardware implementation for an embodiment. A 1st data server computing device 102 is connected over an electronic network/bus connection 110 to 3rd data server computing device 106. A 2nd data server computing device 104 is also connected over an electronic network/bus connection 112 to the 3rd data server computing device 106. The third data server computing device 106 is further connected over an electronic network/bus connection 114 to the 4th data server computing device 108.

[00118] In the embodiment shown in Fig. 1, the 1st data server computing device 102 acts as the source of the 1st encoded integer 110 and the 1st data server computing device 102 sends the 1st encoded integer 110 over the network/bus connection 110 to the 3rd data server computing device 106. The 2nd data server computing device 104 acts as the source of the 2nd encoded integer 112 and the 2nd data server computing device 104 sends the 2nd encoded integer 112 over the network/bus connection 112 to the 3rd data server computing device 106.

[00119] The 3rd data server computing device 106, in compliance with the MPC system 116, performs an arithmetic operation (addition, subtraction, multiplication, and/or division) with the 1st encoded integer 110 and the 2nd encoded integer 112. The 3rd data server computing device 106 then sends the result encoded integer 114 over the network/bus connection 114 to the 4th data server computing device 108. The 4th data server computing device 108 then decodes the result encoded integer 114.

[00120] The 1st encoded integer 110 and the 2nd encoded integer 112 starts at the source computing device as one or more rational numbers (e.g., of the form x/y). An embodiment at the 1st data server computing device 102 and the 2nd data server computing device 104 encodes the

rational numbers into corresponding 1st encoded integer 110 and the 2nd encoded integer 112, respectively as a function of p -adic arithmetic performed on the rational numbers.

[00121] Generally, communications, including concealed/encrypted communications, are bi-directional and may be connected between any of the data server computing devices 102-108, such that data server computing devices 102-108 may change roles as is necessary to accommodate the transfer of data back and forth between the data server computing devices 102-108. Additionally, while the data server computing devices 102-108 are depicted as separate devices in Fig. 1, the functionality of the data server computing devices 102-108 may be shared on a single computing system/device or among two or more computing devices.

[00122] Further, as shown in Fig. 1, the data server computing devices 102-108 appear to be laptop computers, but any computing device or system may be satisfactory. Generally, any computing device capable of communication over any form of electronic network or bus communication platform may be one or more of the data server computing devices 102-108. Additionally, data server computing devices 102-108 may actually be the same physical computing device communicating over an internal bus connection with itself.

[00123] Various embodiments may implement the network/bus communications channel 110-114 using any communications channel 110-114 capable of transferring electronic data between the data server computing devices 102-108. For instance, the network/bus communication connection 110-114 may be an Internet connection routed over one or more different communications channels. Likewise, the network/bus communication connection 110-114 may be an internal communications bus of a computing device, or even the internal bus of a processing or memory storage Integrated Circuit (IC) chip, such as a memory chip or a Central Processing Unit (CPU) chip. The network/bus communication channel 110-114 may utilize any medium capable of transmitting electronic data communications, including, but not limited to: wired communications, wireless electro-magnetic communications, fiber-optic cable communications, light/laser communications, sonic/sound communications, etc., and any combination thereof of the various communication channels.

[00124] The various embodiments may provide the control and management functions detailed herein via an application operating on the data server computing devices 102-108. The data server computing devices 102-108 may each be a computer or computer system, or any other electronic devices capable of performing the communications and computations of an embodiment. The data server computing devices 102-108 may include, but are not limited to: a general-purpose computer, a laptop/portable computer, a tablet device, a smart phone, an industrial control computer, a data storage system controller, a CPU, a Graphical Processing Unit (GPU), an Application Specific Integrated Circuit (ASIC), and/or a Field Programmable Gate

Array (FPGA). Notably, the data server computing devices 102-108 may be the storage controller of a data storage media (e.g., the controller for a hard disk drive). Embodiments may be provided as a computer program product which may include a computer-readable, or machine-readable, medium having stored thereon instructions which may be used to program/operate a computer (or other electronic devices) or computer system to perform a process or processes in accordance with the various embodiments. The computer-readable medium may include, but is not limited to, hard disk drives, floppy diskettes, optical disks, Compact Disc Read-Only Memories (CD-ROMs), Digital Versatile Disc ROMs (DVD-ROMs), Universal Serial Bus (USB) memory sticks, magneto-optical disks, ROMs, random access memories (RAMs), Erasable Programmable ROMs (EPROMs), Electrically Erasable Programmable ROMs (EEPROMs), magnetic optical cards, flash memory, or other types of media/machine-readable medium suitable for storing electronic instructions. The computer program instructions may reside and operate on a single computer/electronic device or various portions may be spread over multiple computers/devices that comprise a computer system. Moreover, embodiments may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer to a requesting computer by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection, including both wired/cabled and wireless connections).

Operational Flow Chart for an Embodiment (Fig. 2)

[00125] Fig. 2 is a flow chart 200 of operations for an embodiment. At process 210, the 1st data server computing device 202 encodes a 1st rational number (e.g., x_0/y_0) into a 1st corresponding encoded integer using p -adic arithmetic that provides the inverse and homomorphic preserving aspects of the p -adic generated integers. At process 212, the 1st data server computing device 202 sends 1st encoded integer to the 3rd data server computing device 206. At process 214, the 2nd data server computing device 204 encodes a 2nd rational number (e.g., x_1/y_1) into a 2nd corresponding encoded integer using p -adic arithmetic that provides the inverse and homomorphic preserving aspects of the p -adic generated integers. At process 216, the 1st data server computing device 202 sends 1st encoded integer to the 3rd data server computing device 206. At process 218, the 3rd data server computing device 205, in accord with the MPC system, computes an arithmetic function with the 1st encoded integer and 2nd encoded integer in to obtain at least one result encoded integer. The potential arithmetic functions are one or more of addition, subtraction, multiplication, and division. At process 220, the 3rd data server computing device 206 sends the result encoded integer to the 4th data server computing device 208. At process 222, the 4th data server computing device decodes the result encoded integer

into a corresponding result rational number (e.g., x_r/y_r) using inverse p -adic arithmetic. The MPC system running on the data server computing devices 202-208 should be of the same type.

[00126] Additionally, while the flow charts and flow chart details described above with respect to Fig. 2 describe a methodology that may be embodied as a method or process, another embodiment may be recognized as a computer system, and/or as an intermediary computing device that stores and/or performs homomorphic operations of encrypted data by implementing the processes described above with respect to the flow chart and flow chart details of Fig. 2. Further, in describing the computing system, and/or the intermediary computing system, one, or more, individual processes described above for the methodology may be broken down and represented as a subsystem of the overall encryption computer system. A subsystem of the computer system, in whole or in part, may be assigned to a particular hardware implemented system, such as a dedicated Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA). One or more subsystems, in whole or in part, may alternatively be implemented as software or firmware instructions defining the operation of a computer system with specific regard to the one or more subsystems implemented as software or firmware instructions. The software or firmware instructions may cause the Central Processing Unit, memory, and/or other systems of a computer system to operate in particular accordance with the particular one or more subsystems designated features.

[00127] The foregoing description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed, and other modifications and variations may be possible in light of the above teachings. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to best utilize the invention in various embodiments and various modifications as are suited to the particular use contemplated.

CLAIMS

What is claimed is:

1. A method for Multi-Party Computation (MPC) on a Multi-Party Computation (MPC) system of rational data encoded in a domain of Farey rationals, the method comprising:
 - encoding by a first data server computing device a first rational number x_0/y_0 into a first encoded integer corresponding to said first rational number x_0/y_0 as an encode function of p -adic arithmetic in said domain of Farey rationals that is induced by a ring isomorphism performed on said first rational number x_0/y_0 , such that said first encoded integer preserves inverses, and additive and multiplicative homomorphic properties;
 - sending by said first data server computing device said first encoded integer to a third data server computing device;
 - encoding by a second data server computing device a second rational number x_1/y_1 into a second encoded integer corresponding to said second rational number x_1/y_1 as said encode function of p -adic arithmetic in said domain of Farey rationals that is induced by a ring isomorphism performed on said second rational number x_1/y_1 , such that said second encoded integer preserves inverses, and additive and multiplicative homomorphic properties;
 - sending by said second first data server computing device said second encoded integer to said third data server computing device;
 - computing by said third data server computing device an arithmetic function as part of said MPC system with said first encoded integer and said second encoded integer to obtain a result encoded integer;
 - sending by said third data server computing device said result encoded integer to a fourth data server computing device; and
 - decoding by said fourth data server computing device said result encoded integer into a result rational number x_r/y_r corresponding to said result encoded integer as a decode inverse function of p -adic arithmetic in said domain of Farey rationals performed on said result encoded integer.
2. The method of claim 1 wherein said arithmetic function is addition performed according to the MPC system functions.

3. The method of claim 1 wherein said arithmetic function is subtraction performed according to the MPC system functions.
4. The method of claim 1 wherein said arithmetic function is division performed by inverting said second encoded integer and multiplying according to the MPC system functions by said first encoded integer to obtain said result encoded integer of said 1st encoded integer divided by said second encoded integer.
5. An Farey rational encoding system for Multi-Party Computation (MPC) on a Multi-Party Computation (MPC) system of rational data encoded in a domain of Farey rationals, the Farey rational encoding system comprising:
 - a first data server computing system that encodes a first rational number x_0/y_0 into a first encoded integer corresponding to said first rational number x_0/y_0 as an encode function of p -adic arithmetic in said domain of Farey rationals that is induced by a ring isomorphism performed on said first rational number x_0/y_0 , such that said first encoded integer preserves inverses, and additive and multiplicative homomorphic properties and sends said first encoded integer to a third data server computing device;
 - a second data server computing device that encodes a second rational number x_1/y_1 into a second encoded integer corresponding to said second rational number x_1/y_1 as said encode function of p -adic arithmetic in said domain of Farey rationals that is induced by a ring isomorphism performed on said second rational number x_1/y_1 , such that said second encoded integer preserves inverses, and additive and multiplicative homomorphic properties and sends said second encoded integer to said third data server computing device;
 - said third data server computing device that computes an arithmetic function as part of said MPC system with said first encoded integer and said second encoded integer to obtain a result encoded integer and sends said result encoded integer to a fourth data server computing device; and
 - said fourth data server computing device that decodes said result encoded integer into a result rational number x_r/y_r corresponding to said result encoded integer as a decode inverse function of p -adic arithmetic in said domain of Farey rationals performed on said result encoded integer.

6. The Farey rational encoding system of claim 5 wherein said arithmetic function is addition performed according to the MPC system functions.
7. The Farey rational encoding system of claim 5 wherein said arithmetic function is subtraction performed according to the MPC system functions.
8. The Farey rational encoding system of claim 5 wherein said arithmetic function is division performed by inverting said second encoded integer and multiplying according to the MPC system functions by said first encoded integer to obtain said result encoded integer of said 1st encoded integer divided by said second encoded integer.