Secure Computation With Fixed-Point Numbers

Octavian Catrina and Amitabh Saxena

Dept. of Computer Science, University of Mannheim, Germany
{catrina, saxena}@uni-mannheim.de

Abstract. Secure computation is a promising approach to business problems in which several parties want to run a joint application and cannot reveal their inputs. Secure computation preserves the privacy of input data using cryptographic protocols, allowing the parties to obtain the benefits of data sharing and at the same time avoid the associated risks. These business applications need protocols that support all the primitive data types and allow secure protocol composition and efficient application development. Secure computation with rational numbers has been a challenging problem. We present in this paper a family of protocols for multiparty computation with rational numbers using fixed-point representation. This approach offers more efficient solutions for secure computation than other usual representations.

Keywords: Secure multiparty computation, secure fixed-point arithmetic, secret sharing.

1 Introduction

Secure computation provides cryptographic protocols that enable a group of parties to run joint applications and preserve the privacy of their inputs. For instance, parties $P_1, P_2, \ldots, P_n$ can use these protocols to evaluate a function $f(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$, where P_i has private input x_i and output y_i. Roughly speaking, the protocols ensure that the output is correct and the computation does not reveal anything else besides the agreed upon output.

Secure computation can solve business problems where input data belongs to different parties and cannot be revealed or shared with other parties. For example, information sharing and collaborative decision making can substantially improve supply chain performance. However, the supply chain partners are not willing to share the necessary sensitive data (e.g., production costs and capacity), since the risks associated with revealing it exceed the benefits gained. Secure computation can offer the benefits of data sharing and at the same time avoid the risks of disclosing private data. Solutions based on secure computation have been studied for various business problems, including privacy-preserving supply chain planning [2], different types of auctions [9,4], benchmarking [3], and collaborative linear programming [20].

A basic requirement of these applications is a protocol family that provides operations with all primitive data types and allows secure protocol composition and efficient application development. The protocols proposed so far offer subsets of operations with boolean and integer data or/and specialized solutions for particular problems. Our goal is to provide practical protocols for secure computation with rational numbers.

Our contribution. We present a family of protocols for multiparty computation with rational numbers using fixed-point representation. The protocols are constructed using secure computation based on secret sharing and provide perfect or statistical privacy in the semi-honest model. The protocol family offers arithmetic and comparison with signed fixed-point numbers and evaluation of boolean functions. Secure addition, subtraction, and comparison of fixed-point numbers are trivial extensions of the integer operations. We present new protocols for scaling, multiplication, and division of fixed-point numbers. We also discuss the methods used to optimize the building blocks of these protocols, including a more efficient solution for bit decomposition.

Related Work. We use standard techniques for constructing multiparty computation protocols based on secret sharing, similar to [7,8,19,6]. However, the solutions presented in [8,19] aim at providing perfect privacy and constant round complexity, while our goal is to obtain efficient protocols for secure computation with fixed-point numbers of typical size (≤ 128 bits). For many building blocks we obtain important performance gains using a combination of techniques that includes additive hiding with statistical privacy (instead of perfect privacy), protocols with logarithmic round complexity (instead of constant round complexity), optimized data encoding (especially for binary values), and non-interactive generation of shared random values.

Related protocols focus on secure computation with field (or ring) elements, binary values, and integers. Protocols for secure division (the most complex task) were developed for particular applications and offer only partial solutions. The division protocol in [15] was designed for two-party computation of statistics and relies on a particular structure of the inputs. The multiparty reciprocal protocol in [1] is restricted to positive integers with known range, $2^{k-1} \leq x < 2^k$. This approach based on the Newton-Raphson method (and its extension to division in [14]) is closer to ours. However, our goal is a general division protocol for signed fixed-point numbers. We present a protocol constructed with more accurate and efficient components and using an algorithm that can better take advantage of their properties. In particular, the absolute error of our integer truncation protocol (division by 2^m) is $|\delta| \leq 0.5$ with high probability (rounding to the nearest integer) and $|\delta| < 1$ in the worst case. The approximate truncation protocol in [1] has absolute error $|\delta| \leq n + 1$, where n is the number of parties.

Secure computation with rational numbers has been a challenging problem. An interesting method was proposed in [13] for addition and multiplication of rational numbers using Paillier homomorphic encryption. This method works only for a limited number of consecutive operations (without decryption), depending on the size of the operands and the modulus of the encryption scheme (e.g., 15 operations for 1024-bit modulus and 32-bit numerator and denominator). Our approach based on fixed-point representation does not have such limitations and offers a complete protocol family for arithmetic and comparison.

Protocols for multiplication and reciprocal of fixed-point numbers were first presented in [5], together with two more general building blocks, reduction modulo 2^m and division by 2^m with rounding down. The fixed-point arithmetic solutions in this paper are more efficient and accurate.

2 Preliminaries

2.1 Secure Computation Framework

Basic framework. Consider a group of $n > 2$ parties, $P_1, \ldots, P_n$, that communicate on secure channels. For $1 \leq i \leq n$, party P_i has private input x_i and output y_i, function of all inputs. Multiparty computation based on secret sharing proceeds as follows. The parties use a linear secret sharing scheme to distribute their private inputs to the group, creating a distributed state of the computation where each party has a share of each secret variable. Certain subsets of parties can reconstruct a secret by pooling together their shares (when needed), while any other subset cannot learn anything about it. Moreover, the properties of the secret sharing scheme allow the parties to compute with shared variables. The protocols used for this purpose take on shared inputs and return shared outputs. This provides the basis for secure protocol composition.

Let X and Y be random variables with finite sample spaces V and W and $\Delta(X,Y) = \frac{1}{2}\sum_{v \in V \bigcup W} |Pr(X = v) - Pr(Y = v)|$ the statistical distance between them. We say that the distributions are perfectly indistinguishable if $\Delta(X,Y) = 0$ and statistically indistinguishable if $\Delta(X,Y)$ is negligible in some security parameter κ. Our protocols offer perfect or statistical privacy, in the sense that the views of protocol execution (consisting of all values learned by an adversary) can be simulated such that the distributions of real and simulated views are perfectly or statistically indistinguishable, respectively.

We assume a basic framework that uses Shamir secret sharing over a finite field $\mathbb{F}$. This framework allows secure arithmetic in $\mathbb{F}$ with perfect privacy against a passive threshold adversary able to corrupt t out of n parties. Essentially, in this model, the parties do not deviate from the specified protocol and any $t+1$ parties can reconstruct a secret, while t or less parties cannot distinguish it from random uniform values in $\mathbb{F}$. We assume $|\mathbb{F}| > n$, to enable Shamir sharing, and $n > 2t$, for multiplication of secret-shared values. We refer the reader to [7] for a more formal and general presentation of this approach to secure computation.

Complexity metrics. In this framework, the running time of the protocols is (usually) dominated by the communication between parties. We evaluate protocol complexity using two metrics that reflect different aspects of the interaction between parties. Communication complexity measures the amount of data sent by each party. For our protocols, a suitable abstract metric of communication complexity is the number of invocations of a primitive during which every party sends a share (field element) to the others, e,g., the multiplication protocol.

Table 1. Secure arithmetic in a finite field $\mathbb{F}$.

Operation	**Purpose**	**Rounds**	**Invocations**
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} + [b]^{\mathbb{F}}$	Add secrets	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} + b$	Add secret and public	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} b$	Multiply secret and public	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} [b]^{\mathbb{F}}$	Multiply secrets	1	1
$a \leftarrow \mathsf{Output}([a]^{\mathbb{F}})$	Reveal a secret	1	1

Round complexity measures the number of sequential invocations. This metric is relevant for the inherent network delay, independent of the amount of data sent. Invocations that can be executed in parallel count as a single round.

We denote $[x]$ a Shamir sharing of x and $[x]^{\mathbb{F}}$ a sharing in a particular field $\mathbb{F}$. Table 1 summarizes the secure arithmetic operations in the basic framework.

2.2 Data Representation

The next step toward secure computation using this approach is to map the application data to field elements. The reverse mapping is performed to extract the application data after the computation. We consider the following data types: boolean values, signed integers, and signed fixed-point numbers.

Fixed-point representation. Fixed-point numbers are rational numbers represented as a sequence of digits split into integer and fractional parts by a virtual radix point. For binary digits, a fixed-point number can be written $\tilde{x} = s \cdot (d_{e-2} \dots d_0.d_{-1} \dots d_{-f})$ and its value is $\tilde{x} = s \cdot \sum_{i=-f}^{e-2} d_i 2^i$, where $s \in \{-1, 1\}$, e is the length of the integer part (including the sign bit), and f is the length of the fractional part. Denote $\bar{x} = s \cdot \sum_{i=0}^{e+f-2} d_i 2^i$ and observe that $\tilde{x} = \bar{x} \cdot 2^{-f}$, hence $\tilde{x}$ is encoded as an integer $\bar{x}$ scaled by the factor 2^{-f}.

We define a fixed-point data type as follows. Let k, e, and f be integers such that $k > 0$, $f \geq 0$, and $e = k - f \geq 0$. Denote $\mathbb{Z}_{\langle k \rangle} = \{x \in \mathbb{Z} \mid -2^{k-1} + 1 \leq x \leq 2^{k-1} - 1\}$. The fixed-point data type with resolution 2^{-f} and range 2^e is the set $\mathbb{Q}_{\langle k,f \rangle} = \{\tilde{x} \in \mathbb{Q} \mid \tilde{x} = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle k \rangle}\}$. Intuitively, $\mathbb{Q}_{\langle k,f \rangle}$ is obtained by sampling the range of real values $[-2^{e-1} + 2^{-f}, 2^{e-1} - 2^{-f}]$ at 2^{-f} intervals.

We use the following compact notation for a range of integers: $[A..B] = \{x \in \mathbb{Z} \mid A \leq x \leq B\}$ and $[A..B) = \{x \in \mathbb{Z} \mid A \leq x < B\}$.

Data encoding in a field. Any secret value in a secure computation has a data type which is public information. Data types are encoded in a field $\mathbb{F}$ as follows.

Denote 0_F and 1_F the additive and multiplicative identities of $\mathbb{F}$. Logical values $false, true$ and bit values $0, 1$ are encoded as 0_F and 1_F, respectively. $\mathbb{F}$ can be a small binary field $\mathbb{F}_{2^m}$ or prime field $\mathbb{Z}_q$. This encoding allows secure evaluation of boolean functions using secure arithmetic in $\mathbb{F}$. Encoding in $\mathbb{F}_{2^m}$ is more efficient, because XOR is a local operation; we can take $m = 8$, which is sufficient for Shamir sharing with $n < 256$ parties.

Table 2. Complexity of the building blocks (the default field is $\mathbb{Z}_q$).

Protocol	**Rounds**	**Invocations**	**Field**
$[r] \leftarrow \mathsf{PRandBit}()$	1	1	$\mathbb{Z}_q$
$[r] \leftarrow \mathsf{PRandBitL}()$	2	2	$\mathbb{Z}_{q_1}$
$[r]^{\mathbb{F}_{2^8}}, [r] \leftarrow \mathsf{PRandBitD}()$	2	2	$\mathbb{Z}_{q_1}$
$[r] \leftarrow \mathsf{PRandInt}(k)$	0	0	0
$[a] \leftarrow \mathsf{BitF2MtoZQ}([a]^{\mathbb{F}_{2^8}})$	2	2	$\mathbb{Z}_{q_1}$
	1	1	$\mathbb{F}_{2^8}$
$[c_{k-1}]^{\mathbb{F}}, \ldots, [c_0]^{\mathbb{F}} \leftarrow \mathsf{PreOR}([a_{k-1}]^{\mathbb{F}}, \ldots, [a_0]^{\mathbb{F}})$	$\log(k)$	$\frac{k}{2}\log(k)$	$\mathbb{F}$
$[c_{m-1}]^{\mathbb{F}}, \ldots, [c_0]^{\mathbb{F}} \leftarrow \mathsf{BitAdd}(a, [b_{m-1}]^{\mathbb{F}}, \ldots, [b_0]^{\mathbb{F}})$	$\log(m)$	$m\log(m)$	$\mathbb{F}$
$[a_{m-1}]^{\mathbb{F}_{2^8}}, \ldots, [a_0]^{\mathbb{F}_{2^8}} \leftarrow \mathsf{BitDec}([a], k, m)$	1	1	$\mathbb{Z}_q$
	2	$2m$	$\mathbb{Z}_{q_1}$
	$\log(m)$	$m\log(m)$	$\mathbb{F}_{2^8}$
$[s] \leftarrow \mathsf{LTZ}([a], k)$	1	1	$\mathbb{Z}_q$
	2	$2k$	$\mathbb{Z}_{q_1}$
	$\log(k)+1$	$2k-3$	$\mathbb{F}_{2^8}$

Signed integers are encoded in $\mathbb{Z}_q$ using the function $\mathsf{fld} : \mathbb{Z}_{\langle k\rangle} \mapsto \mathbb{Z}_q$, $\mathsf{fld}(\bar{x}) = \bar{x} \bmod q$, for $q > 2^k$. For any integers $\bar{a}, \bar{b} \in \mathbb{Z}_{\langle k\rangle}$ and operation $\odot \in \{+, -, \cdot\}$ we have $\bar{a} \odot \bar{b} = \mathsf{fld}^{-1}(\mathsf{fld}(\bar{a}) \odot \mathsf{fld}(\bar{b}))$. Moreover, if $\bar{b}|\bar{a}$ then $\bar{a}/\bar{b} = \mathsf{fld}^{-1}(\mathsf{fld}(\bar{a}) \cdot \mathsf{fld}(\bar{b})^{-1})$. Secure arithmetic with signed integers can thus be computed using secure arithmetic in $\mathbb{Z}_q$.

A secret fixed-point number $\tilde{x}$ of type $\mathbb{Q}_{\langle k,f\rangle}$ is represented as a secret integer $\bar{x} = \tilde{x}2^f$ encoded in $\mathbb{Z}_q$ and public parameters that specify the resolution and the range, f and e (or $k = e + f$). We define the map $\mathsf{int}_f : \mathbb{Q}_{\langle k,f\rangle} \mapsto \mathbb{Z}_{\langle k\rangle}$, $\mathsf{int}_f(\tilde{x}) = \tilde{x}2^f$. Note that the fixed-point representation allows very efficient encoding of a secret rational number, as a single field element.

We also use (when required) a bitwise encoding of integers, where each bit of the binary representation is encoded and shared in a field $\mathbb{F}$ as described above.

We distinguish different representations of a number using the following simplified notation: we denote $\tilde{x}$ a rational number of some fixed-point type $\mathbb{Q}_{\langle k,f\rangle}$ and $\bar{x} = \tilde{x}2^f \in \mathbb{Z}_{\langle k\rangle}$ the integer value of its fixed-point representation; for secure computation using secret-sharing we denote $x = \bar{x} \bmod q \in \mathbb{Z}_q$ the field element that encodes $\bar{x}$ (and hence $\tilde{x}$) and $[x]$ a sharing of x.

2.3 Building Blocks

We provide an overview of several building blocks and techniques used in fixed-point arithmetic protocols. Their complexity is summarized in Table 2.

Shared random values. The protocols often use secret sharing together with additive or multiplicative hiding, taking advantage of their combined capabilities for computing with secret data and efficient conversion methods. For example, given a shared variable $[x]$ the parties can jointly generate a shared random value

$[r]$, compute $[y] = [x] + [r]$, and reveal $y = x + r$. This is similar to one-time pad encryption of x with key r.

For a secret $x \in \mathbb{Z}_q$ and random uniform $r \in \mathbb{Z}_q$ we obtain $\Delta(x+r \bmod q, r) = 0$, hence perfect privacy. Alternatively, for $x \in [0..2^k)$, random uniform $r \in [0..2^{k+\kappa})$, and $q > 2^{k+\kappa+1}$ we obtain $\Delta(x + r \bmod q, r) < 2^{-\kappa}$, hence statistical privacy with security parameter κ. The variant with statistical privacy can substantially simplify the protocols by avoiding wraparound modulo q, although it requires larger q (hence larger shares) for a given data range. Statistical privacy also holds for other distributions of r that can be generated more efficiently or/and meet particular requirements: (1) $r = \sum_i r_i$, where $r_i \in [0..2^{k+\kappa})$ are random uniform integers; (2) $r = 2^k r'' + r'$, where $r'' = \sum_i r''_i$ and $r''_i \in [0..2^\kappa)$ and $r' \in [0..2^k)$ are random uniform integers.

We use Pseudo-random Replicated Secret Sharing (PRSS) [6] to generate without interaction shared random values in $\mathbb{F}$ with uniform distribution and random sharings of zero. Also, we use the integer variant of PRSS (RISS) [10] to generate shared random integers in a given interval, and the ideas in [11] for bit-share conversions (e.g., BitF2MtoZQ converts bit shares from $\mathbb{F}_{2^8}$ to $\mathbb{Z}_q$).

To enable these techniques, we assume in the remainder of the paper that numbers are encoded in $\mathbb{Z}_q$ as specified in Section 2.2 and $q > 2^{k+\kappa+\nu+1}$, where k is the required integer bit-length, κ is the security parameter, $\nu = \lceil \log(\binom{n}{t}) \rceil$, n is the number of parties, and t is the corruption threshold.

Protocol PRandBit generates a random bit shared in $\mathbb{Z}_q$ by combining the protocol RandBit in [8] and protocols in [6]. A random uniform integer $r \in [0..2^k)$ is constructed from shared random bits as $[r] = \sum_{i=0}^{k-1} 2^i [r_i]$. Note that RandBit includes an exponentiation that significantly increases the running time when generating many random bits for large q. PRandBitL generates a shared random bit in a small field $\mathbb{Z}_{q_1}$ to reduce complexity, then converts its shares to the target field $\mathbb{Z}_q$ (e.g., $\lceil \log(q_1) \rceil = 64$). PRandBitD uses a similar technique to generate a random bit shared in both $\mathbb{Z}_q$ and $\mathbb{F}_{2^8}$. Bits shared in $\mathbb{Z}_q$ are used to construct a random uniform integer, while bits shared in $\mathbb{F}_{2^8}$ are used for binary computation. PRandInt(k) generates without interaction a shared random integer $r \in [0..2^{k+\nu})$ distributed as sum of $\binom{n}{t}$ random uniform integers in $[0..2^k)$.

Bit decomposition. Protocol 2.1, BitDec, is a general tool that provides a bridge between secure computation with integers shared in $\mathbb{Z}_q$ and with integers bitwise-shared in $\mathbb{Z}_q$ or $\mathbb{F}_{2^8}$. The inputs are $[a] = [\mathsf{fld}(\bar{a})]$ and the public integers k and m, where $\bar{a} \in \mathbb{Z}_{\langle k \rangle}$ and $0 < m \leq k$. The output is an array of shared bits equal to the m least significant bits of the 2's complement binary representation of $\bar{a}$. The protocol follows the idea in [8,18,19] for bit decomposition of $\mathbb{Z}_q$ elements, but offers a more efficient solution for bounded integers and statistical privacy. Protocol 2.1 extracts m bits in $\log(m) + 3$ rounds with $m \log(m) + 2m + 1$ invocations, while the variant with perfect privacy and constant round complexity [19] extracts $k = \lceil \log(q) \rceil$ bits in 51 rounds with $56k \log(k) + 30k$ invocations.

Correctness. Let $\ell = k + \kappa + \nu$. The protocol generates a random integer $0 \leq r < 2^\ell$ and computes $c = (2^\ell + 2^k + a - r) \bmod q$. For $q > 2^{\ell+1}$ we have $(2^k + a) \bmod q = 2^k + \bar{a}$ and $c = 2^\ell + 2^k + \bar{a} - r$. If $\bar{a} \geq 0$ then $(r + c) \bmod 2^k = \bar{a}$

and if $\bar{a} < 0$ then $(r + c) \bmod 2^k = 2^k - |\bar{a}|$, hence $(r + c) \bmod 2^k$ is equal to the 2's complement representation of $\bar{a}$. The protocol computes the $m \leq k$ least significant bits of $\bar{a}$ using the binary addition protocol BitAdd.

```
Protocol 2.1: $([a_{m-1}]^{\mathbb{F}_{2^8}}, \ldots, [a_0]^{\mathbb{F}_{2^8}}) \leftarrow \mathsf{BitDec}([a], k, m)$
1 foreach $i \in [0..m-1]$ do parallel
2     $[r_i]^{\mathbb{F}_{2^8}}, [r_i] \leftarrow \mathsf{PRandBitD}()$;
3 $[r'] \leftarrow \sum_{i=0}^{m-1} 2^i \cdot [r_i]$;
4 $[r''] \leftarrow \mathsf{PRandInt}(\kappa + k - m)$;
5 $[r] \leftarrow 2^m \cdot [r''] + [r']$;
6 $c \leftarrow \mathsf{Output}(2^{k+\kappa+\nu} + 2^k + [a] - [r])$;
7 $([a_{m-1}]^{\mathbb{F}_{2^8}}, \ldots, [a_0]^{\mathbb{F}_{2^8}}) \leftarrow \mathsf{BitAdd}((c_{m-1}, \ldots, c_0), ([r_{m-1}]^{\mathbb{F}_{2^8}}, \ldots, [r_0]^{\mathbb{F}_{2^8}}))$;
8 return $([a_{m-1}]^{\mathbb{F}_{2^8}}, \ldots, [a_0]^{\mathbb{F}_{2^8}})$;
```

Security. Protocol BitDec can leak information in step 6 when it reveals c. The other building blocks provide perfect privacy or statistical privacy with security parameter κ. Since $\Delta(c, r) < 2^{-\kappa}$ we conclude that BitDec provides statistical privacy with security parameter κ.

Complexity. Table 3 shows the complexity of BitDec and its building blocks. Observe that most of the invocations are in small fields, $\mathbb{Z}_{q_1}$ or $\mathbb{F}_{2^8}$. The double-shared random bits can be generated in parallel in 2 rounds and precomputed. The random integer r is constructed such that to minimize the number of shared random bits. BitAdd uses standard algorithms and is designed to offer a good trade-off between round and communication complexity (a variant with minimal communication needs $2\log(m)$ rounds).

3 Secure Fixed-Point Arithmetic

The protocols for arithmetic with fixed-point numbers are constructed using secure integer arithmetic and scaling. Let $\tilde{a}$, $\tilde{b}$ be fixed-point numbers. We denote $\tilde{a}+\tilde{b}$, $\tilde{a}-\tilde{b}$, $\tilde{a}\cdot\tilde{b}$, $\tilde{a}/\tilde{b}$ the exact arithmetic operations (the result is a real number). The output of a protocol may differ from the exact result, either because the value is truncated to obtain a given fixed-point representation, or because the algorithm computes an approximation of the result.

We present a secure arithmetic operation in three steps: we first give an algorithm for exact arithmetic; then, we derive an algorithm for inputs and output of given fixed-point types and limited precision arithmetic, and evaluate its error; finally, we use this algorithm to obtain a protocol with secret inputs and output. The second algorithm takes as inputs $\bar{a} = \tilde{a}2^f$, $\bar{b} = \tilde{b}2^f$ and computes the result $\bar{c} = \tilde{c}2^f$ using integer arithmetic. For secure computation, fixed-point numbers are encoded in $\mathbb{Z}_q$ and secret-shared. Let $a = \bar{a} \bmod q$, $b = \bar{b} \bmod q$, $c = \bar{c} \bmod q$ the encoded numbers. On input the secret-shared values $[a]$ and $[b]$ the protocol computes the secret-shared output $[c]$ using secure arithmetic in $\mathbb{Z}_q$. Table 3 summarizes the complexity of the protocols presented in this section.

Table 3. Complexity of the fixed-point arithmetic protocols.

Protocol	**Rounds**	**Invocations**	**Field**
$[d] \leftarrow \mathsf{TruncPr}([a], k, m)$	1	1	$\mathbb{Z}_q$
	2	$2m$	$\mathbb{Z}_{q_1}$
$\mathsf{TruncPr}$ after precomputation	1	1	$\mathbb{Z}_q$
$[c] \leftarrow \mathsf{FPMul}([a], [b], k, f)$	2	2	$\mathbb{Z}_q$
	2	$2f$	$\mathbb{Z}_{q_1}$
FPMul after precomputation	2	2	$\mathbb{Z}_q$
$[y] \leftarrow \mathsf{FPDiv}([a], [b], k, f)$	$2\theta + 8$	$4\theta + 8$	$\mathbb{Z}_q$
($e = f$, $k = 2f$, θ iterations)	2	$2k\theta + 6.5k$	$\mathbb{Z}_{q_1}$
	$3\log(k) + 2$	$1.5k\log(k) + 4k - 2$	$\mathbb{F}_{2^8}$
FPDiv after precomputation	$2\theta + 8$	$4\theta + 8$	$\mathbb{Z}_q$
	$3\log(k) + 2$	$1.5k\log(k) + 4k - 2$	$\mathbb{F}_{2^8}$
$[w] \leftarrow \mathsf{AppRcr}([b], k, f)$	7	7	$\mathbb{Z}_q$
after precomputation	$3\log(k) + 2$	$1.5k\log(k) + 4k - 2$	$\mathbb{F}_{2^8}$

3.1 Scaling

The purpose of scaling is to convert a given number to a fixed-point type with different resolution. Let $\tilde{a}_1 = \bar{a}_1 2^{-f_1}$ and suppose we want to convert this value to $\tilde{a}_2 = \bar{a}_2 2^{-f_2}$. Let $m = f_2 - f_1$. We distinguish two cases. If $m \geq 0$ we have to scale up $\tilde{a}_1$ by computing $\bar{a}_2 = \bar{a}_1 2^m$. We obtain $\tilde{a}_2 = \tilde{a}_1$ (same value with higher resolution). If $m < 0$ we have to scale down (truncate) $\tilde{a}_1$. Let $\mathsf{trunc}(\bar{x}, m) = \bar{x}/2^m - \delta_t$, where δ_t is the absolute error of the truncation operation. We scale down $\tilde{a}_1$ by computing $\bar{a}_2 = \mathsf{trunc}(\bar{a}_1, m)$ and obtain $\tilde{a}_2 \approx \tilde{a}_1$ with absolute error $\delta = \tilde{a}_1 - \tilde{a}_2 = \delta_t 2^{-f_2}$. For example, if $\mathsf{trunc}(\bar{x}, m)$ rounds down (discards m bits) then $0 \leq \delta_t < 1$. If it rounds to the nearest integer then $-0.5 < \delta_t \leq 0.5$.

A secret number $[a_1]$ is scaled up without interaction by computing $[a_2] = [a_1]2^m$. Truncation is more complicated. We present an accurate and efficient solution. Let $\bar{a} \in \mathbb{Z}_{\langle k \rangle}$ and $0 < m < k$. Protocol 3.1, $\mathsf{TruncPr}$, takes as inputs $[a]$ and the public integers k and m and returns $[d]$ such that $\bar{d} = \lfloor \bar{a}/2^m \rfloor + u$, where u is a random bit and $Pr(u = 1) = (\bar{a} \bmod 2^m)/2^m$. Therefore, the protocol rounds $\bar{a}/2^m$ to the nearest integer with probability $1 - \alpha$, where α is the distance between $\bar{a}/2^m$ and the nearest integer.

Correctness. A signed integer $\bar{a}$ is encoded in $\mathbb{Z}_q$ as $a = \mathsf{fld}(\bar{a}) = \bar{a} \bmod q$. Step 1 maps $\bar{a} \in [-2^{k-1}..2^{k-1})$ to $b \in [0..2^k)$ by computing $b = (2^{k-1} + a) \bmod q = 2^{k-1} + \bar{a}$. Observe that $b' = b \bmod 2^m = \bar{a} \bmod 2^m$ for any $0 < m \leq k$. Denote $\ell = k + \kappa + \nu$. Steps 2-6 generate a random secret $r \in [0..2^\ell)$ and reveal $c = (b + r) \bmod q$. For $q > 2^{\ell+1}$ we have $q > b + r$ and hence $c = b + r$. Let $c' = c \bmod 2^m$ and $r' = r \bmod 2^m$. We see that $c' = (b' + r') \bmod 2^m = b' + r' - u \cdot 2^m$, where $u \in \{0, 1\}$. Therefore, steps 1-9 compute $a' = (\bar{a} \bmod 2^m) - u \cdot 2^m$.

Let $d' = (a - a') \bmod q$ and observe that $d' = (\bar{a} - (\bar{a} \bmod 2^m) + u \cdot 2^m) \bmod q = (\lfloor \bar{a}/2^m \rfloor \cdot 2^m + u \cdot 2^m) \bmod q$. The protocol returns $d = d'(2^{-m} \bmod q) \bmod q$. We have $d = (\lfloor \bar{a}/2^m \rfloor + u) \bmod q = \mathsf{fld}(\lfloor \bar{a}/2^m \rfloor + u)$, hence the output is correct.

Protocol 3.1: $[d] \leftarrow \mathsf{TruncPr}([a], k, m)$

```
1  $[b] \leftarrow 2^{k-1} + [a]$;
2  **foreach** $i \in [0..m-1]$ **do parallel**
3      $[r_i] \leftarrow \mathsf{PRandBitL}()$;
4  $[r'] \leftarrow \sum_{i=0}^{m-1} 2^i \cdot [r_i]$;
5  $[r''] \leftarrow \mathsf{PRandInt}(q, \kappa + k - m)$;
6  $[r] \leftarrow 2^m \cdot [r''] + [r']$;
7  $c \leftarrow \mathsf{Output}([b] + [r])$;
8  $c' \leftarrow c \bmod 2^m$;
9  $[a'] \leftarrow c' - [r']$;
10 $[d] \leftarrow ([a] - [a'])2^{-m}$;
11 **return** $[d]$;
```

Probabilistic rounding. Observe that $b' + r' \in [0..2^{m+1} - 2]$ and that $u = 1$ if $b' + r' \geq 2^m$ and $u = 0$ if $b' + r' < 2^m - 1$. It follows that $Pr(u = 1) = Pr(r' \geq 2^m - b')$. We see that $p(b') = Pr(u = 1)$ grows with b' from $p(0) = 0$ to $p(2^m - 1) \approx 1$. For example, if r' is random uniform in $[0..2^m - 1]$, we obtain $p(b') = b'/2^m$, hence $p(0) = 0$, $p(2^m/2) = 1/2$, and $p(2^m - 1) = 1 - 2^{-m}$. Note that deterministic rounding is too expensive because it requires comparisons.

Security. Protocol $\mathsf{TruncPr}$ can leak information in step 7 when it reveals $c = b+r$. The other building blocks provide perfect privacy or statistical privacy with security parameter κ. Since $\Delta(c, r) < 2^{-\kappa}$ we conclude that $\mathsf{TruncPr}$ provides statistical privacy with security parameter κ.

Complexity. All random bits are generated in parallel in 1 or 2 rounds depending on the protocol used, $\mathsf{PRandBit}$ or $\mathsf{PRandBitL}$, and can be precomputed. The construction of r minimizes the number of shared random bits generated by the protocol. Table 3 shows the complexity of the variant using $\mathsf{PRandBitL}$.

Extensions. Observe that $b' = c' - r' + u \cdot 2^m$ and that $u = 1$ if $c' < r'$ and $u = 0$ if $c' \geq r'$. We can compute $[u]$ using a comparison protocol for bitwise-shared integers and obtain a protocol $\mathsf{Trunc}([a], k, m)$ that computes $\bar{d} = \lfloor \bar{a}/2^m \rfloor$, i.e., truncates m bits and rounds down. This is the truncation protocol used in [5]. Note that $\mathsf{TruncPr}$ is substantially more efficient than Trunc, since it avoids an expensive bitwise comparison, and at the same time reduces the rounding error with high probability to $|\delta_t| < 0.5$.

Furthermore, if $\bar{a} < 0$ then $\lfloor \bar{a}/2^{k-1} \rfloor = -1$ and if $\bar{a} \geq 0$ then $\lfloor \bar{a}/2^{k-1} \rfloor = 0$. Therefore, we can determine the sign of a secret integer by computing $[s] = -\mathsf{Trunc}([a], k, k-1)$. This is the comparison protocol $\mathsf{LTZ}([a], k)$ in Table 2.

3.2 Addition, Subtraction, and Comparison

We specify addition and subtraction for values of the same fixed-point type. Values of different types have to be converted to the same type. Let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ and $\tilde{c} = \tilde{a} + \tilde{b}$. Since $\tilde{c} = (\bar{a} + \bar{b})2^{-f}$, we obtain the representation of $\tilde{c}$ with resolution 2^{-f} by computing $\bar{c} = \bar{a} + \bar{b}$. For secret-shared values we compute

$[c] = [a] + [b]$ and $[c] = a + [b]$ without interaction. The output is the exact result and has the same type as the inputs. Subtraction is similar.

Integer comparison operators with secret inputs and output can be constructed using the following two protocols: $\mathsf{EQZ}([a])$, that computes $\bar{a} \overset{?}{=} 0$, and $\mathsf{LTZ}([a])$, that computes $\bar{a} \overset{?}{<} 0$. For example, $\mathsf{EQ}([a], [b]) = \mathsf{EQZ}([a] - [b])$ computes $\bar{a} \overset{?}{=} \bar{b}$ and $\mathsf{GE}([a], [b]) = 1 - \mathsf{LTZ}([a] - [b])$ computes $\bar{a} \overset{?}{\geq} \bar{b}$. We can also use these protocols for fixed-point inputs of the same type and obtain exact results.

3.3 Multiplication

We first consider multiplication of two numbers of the same fixed-point type, $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$. Let $\tilde{c} = \tilde{a}\tilde{b}$. Since $\tilde{c} = \bar{a}\bar{b}2^{-2f}$, we obtain the representation of the exact result $\tilde{c}$ with resolution 2^{-2f} by computing $\bar{c} = \bar{a}\bar{b}$ (if overflow does not occur). For secret-shared values we compute $[c] = [a][b]$ with complexity 1 round and 1 invocation. If an input is public then $[c] = a[b]$, without interaction.

The output of a multiplication is usually scaled down to resolution 2^{-f} in order to obtain a value with the same type as the inputs and to limit the size of the integers that encode the fixed-point numbers. Thus, for a typical multiplication we compute $\bar{d} = \mathsf{trunc}(\bar{a}\bar{b}, f)$ and obtain $\tilde{d} \approx \tilde{a}\tilde{b}$ with absolute error $\delta = \tilde{a}\tilde{b} - \tilde{d} = \delta_t 2^{-f}$. The secure computation is shown in Protocol 3.2. Observe that the output overflows when the intermediate value $\bar{a}\bar{b}$ reaches $k + f$ bits. Therefore, $\mathbb{Z}_q$ must support integers of at least $k + f$ bits in order to avoid overflow of $\bar{a}\bar{b}$ for all valid outputs.

Protocol 3.2: $[d] \leftarrow \mathsf{FPMul}([a], [b], k, f)$

```
1 $[c] \leftarrow [a][b]$;
2 $[d] \leftarrow \mathsf{TruncPr}([c], 2k, f)$;
3 **return** $[d]$;
```

Fixed-point multiplication with inputs and outputs of different types can be computed using similar protocols, with the same complexity and accuracy. For example, if the inputs are $\tilde{a} = \bar{a}2^{-f_a}$, $\tilde{b} = \bar{b}2^{-f_b}$ and the output is $\tilde{d} = \bar{d}2^{-f}$, where $f \leq f_a + f_b$, the computation is $\bar{d} = \mathsf{trunc}(\bar{a}\bar{b}, f_a + f_b - f)$. We obtain $\tilde{d} \approx \tilde{a}\tilde{b}$ with absolute error $\delta = \delta_t 2^{-f}$. Truncation is not necessary if one input is an integer and the other one has the same resolution as the output.

We point out two optimizations that improve the efficiency and accuracy of the protocols by reducing the number of truncations. We assume inputs and outputs of the same type $\mathbb{Q}_{\langle k,f \rangle}$ and $|\delta_t| < 1$.

We can evaluate the inner product $\sum_{i=1}^{m} \tilde{a}_i\tilde{b}_i$ with error $\delta = \delta_t 2^{-f}$ by computing $\bar{d} = \mathsf{trunc}(\sum_{i=1}^{m} \bar{a}_i\bar{b}_i, f)$. Computing $\bar{d}' = \sum_{i=1}^{m} \mathsf{trunc}(\bar{a}_i\bar{b}_i, f)$ is both inefficient and less accurate, since the cumulated errors can reach $|\delta'| < m2^{-f}$. A double multiplication $\tilde{a}\tilde{b}\tilde{c}$ can be evaluated with absolute error $\delta = \delta_t 2^{-f}$ by computing $\bar{d} = \mathsf{trunc}(\bar{a}\bar{b}\bar{c}, 2f)$, if the data representation supports integers of $k + 2f$ bits. On the other hand, if we compute $\bar{d}' = \mathsf{trunc}(\mathsf{trunc}(\bar{a}\bar{b}, f)\bar{c}, f)$, the error becomes $\delta' \approx \delta_t 2^{\ell - 2f}$, assuming $\bar{c} \in [2^{\ell-1}..2^{\ell})$.

3.4 Division

Secure division with secret dividend and public divisor follows immediately from fixed-point multiplication. Let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$, and assume $\tilde{a}$ is secret and $\tilde{b}$ is public. We can obtain the secret quotient $\tilde{y} \approx \tilde{a}/\tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ with error $\delta = \delta_t 2^{-f}$ by computing the reciprocal $\tilde{x} \approx 1/\tilde{b}$, $\tilde{x} \in \mathbb{Q}_{\langle k+f,k \rangle}$, and then $[y] = \mathsf{TruncPr}([a] \cdot \mathsf{fld}(\mathsf{int}_k(\tilde{x})), k)$. Therefore, if the divisor is public the division protocol has the same complexity as the truncation. For example, this protocol may be sufficient for secure evaluation of statistics like sample mean $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ and variance $\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$ since N is usually public.

The problem becomes difficult when the divisor is secret. The algorithms for dividing fixed-point numbers follow two main approaches: digit recurrence (subtractive division) and functional iteration (multiplicative division) [12]. Functional iteration is more suitable for secure computation, because the algorithms converge faster and are simpler to implement with the available building blocks. These algorithms fall into two main classes: algorithms that use the Newton-Raphson method for computing the reciprocal and algorithms that use series expansion, in particular Goldschmidt's method. Both methods require a suitable initial approximation, which is the main hurdle for secure computation. Moreover, both offer quadratic convergence and the iterations have similar complexity. The Newton-Raphson iterations are self correcting (truncation errors in an iteration decrease quadratically during next iterations), but the multiplications are dependent and have to be computed sequentially. For Goldschmidt's method, the multiplications can be computed concurrently, but truncation errors cumulate during the iterations. We developed and evaluated protocols for both methods. We present in this paper a protocol based on Goldschmidt's method that offers better efficiency (for similar accuracy).

Goldschmidt's method for computing a/b can be described as follows [16]. Let w_0 be an initial approximation of $1/b$ with relative error $\epsilon_0 < 1$, and let $a_0 = a$, $b_0 = b$. For $i \geq 1$ the algorithm computes: $a_i = a_{i-1}w_{i-1}$, $b_i = b_{i-1}w_{i-1}$, $w_i = 2 - b_i$. Denote $r_i = \prod_{j=0}^{i} w_j$ and observe that:

$$\frac{a}{b} = \frac{aw_0 \ldots w_{i-1}}{bw_0 \ldots w_{i-1}} = \frac{a_i}{b_i} = \frac{a_i w_i}{b_i w_i} = \frac{a_{i+1}}{b_{i+1}} = \frac{ar_i}{br_i}.$$

The relative error of the initial approximation is $\epsilon_0 = 1 - bw_0$. It can be shown (by induction) that $b_i = 1 - \epsilon_0^{2^{i-1}}$ and $w_i = 1 + \epsilon_0^{2^{i-1}}$. Observe that if $|\epsilon_0| < 1$ then b_i converges to 1 and hence a_i converges to the quotient a/b and r_i to the reciprocal $1/b$. Denoting $e_i = \epsilon_0^{2^i}$, the recurrence relations that approximate the quotient can be written as follows: $a_1 = aw_0$; $a_{i+1} = a_i(1 + e_{i-1})$, $e_i = e_{i-1}^2$. After i iterations we obtain $a_{i+1} = (a/b)(1 - \epsilon_0^{2^i})$, hence $a_{i+1} \approx a/b$ with relative error $\epsilon_0^{2^i}$. We can obtain similar recurrence relations for $1/b$.

Initial approximation. A critical issue is to determine an initial approximation that ensures fast convergence. The usual method is to compute a normalized input $c \in [0.5, 1)$ and then find an approximation of $1/c$. We use the linear approximation $w_0 = 2.9142 - 2c$ with relative error $\epsilon_0 < 0.08578$ (3.5 initial bits) [12]. This approximation offers sufficient accuracy for our purposes and can be computed without interaction for secret c.

More accurate initial approximations can be obtained by table lookup [17]. For example, a piece-wise linear approximation using a table with 2^k entries offers initial approximations with accuracy of $2k+2$ bits. A reciprocal with 64-bit accuracy can thus be computed in 2 iterations, with an initial approximation based on a table with only 128 entries. However, the efficiency gain is reduced by the additional cost of the table lookup with secret index.

Division algorithm. The division protocol performs the computation described above using the building blocks in the previous sections. We give an algorithm for positive inputs and then show how to extend it to signed inputs.

Let $\tilde{a}, \tilde{b} \in \mathbb{Q}^+_{\langle k,f\rangle}$ and assume that $2^{\ell-f-1} \le \tilde{a} < 2^{\ell-f}$ and $2^{m-f-1} \le \tilde{b} < 2^{m-f}$, for some $\ell \le k$ and $m \le k$. Our goal is to compute $\tilde{y} \in \mathbb{Q}^+_{\langle k,f\rangle}$ such that $\tilde{y} \approx \tilde{a}/\tilde{b}$ and the maximum absolute error is close to the resolution 2^{-f} of the output. We describe the exact computation (without truncations) followed by the computation with limited precision carried out by the protocol:

1. Computation of the initial approximation $\tilde{w} \approx 1/\tilde{b}$:
   Exact arithmetic: Normalize $\tilde{b}$ to obtain $\tilde{c} \in (0.5, 1)$. The normalized divisor is $\tilde{c} = \tilde{b}2^{f-m} = \tilde{b}2^{u-e}$, where $u = k-m = e+f-m$. Let $\tilde{d} = 2.9142 - 2\tilde{c}$ be the initial approximation of $1/\tilde{c}$. The initial approximation of $1/\tilde{b}$ is $\tilde{w} = \tilde{d}2^{u-e}$.
   Approximate arithmetic: Assume $\tilde{b}$ with resolution 2^{-f}, $\tilde{c}$ with resolution 2^{-k} and $\tilde{w}$ with resolution 2^{-f}. Let $\bar{b} = \tilde{b}2^f$, $\bar{c} = \tilde{c}2^k$, $\bar{w} = \tilde{w}2^f$. Compute: $\bar{c} = \bar{b}2^u$; $\bar{d} = \mathsf{int}_k(2.9142) - 2\bar{c}$; $\bar{w} = \mathsf{trunc}(\bar{d}2^u, 2e)$.
2. Computation of $\tilde{y} \approx \tilde{a}/\tilde{b}$:
   Exact arithmetic: Let $\tilde{y}_0 = \tilde{a}\tilde{w}$ and $\tilde{x}_0 = 1 - \tilde{b}\tilde{w}$ (note that $\tilde{x}_0$ is the relative error of $\tilde{w}$ and $|\tilde{x}_0| < 1$). For $1 \le i < \theta$ do: $\tilde{y}_i = \tilde{y}_{i-1} + \tilde{y}_{i-1}\tilde{x}_{i-1}$; $\tilde{x}_i = \tilde{x}_{i-1}\tilde{x}_{i-1}$. Let $\tilde{y} = \tilde{y}_\theta = \tilde{y}_{\theta-1} + \tilde{y}_{\theta-1}\tilde{x}_{\theta-1}$ (last iteration). We obtain $\tilde{y} \approx \tilde{a}/\tilde{c}$ with relative error $\epsilon_\theta < \epsilon_0^{2^\theta}$.
   Approximate arithmetic: Assume $\tilde{a}$, $\tilde{b}$, $\tilde{w}$, $\tilde{y}_i$ with resolution 2^{-f}, and $\tilde{x}_i$ with resolution 2^{-2f}. Denote $\bar{a} = \tilde{a}2^f$, $\bar{b} = \tilde{b}2^f$, $\bar{y}_i = \tilde{y}_i2^f$, $\bar{w} = \tilde{w}2^f$, and $\bar{x}_i = \tilde{x}_i2^{2f}$. Let $\bar{x}_0 = \mathsf{int}_{2f}(1.0) - \bar{b}\bar{w}$ and $\bar{y}_0 = \mathsf{trunc}(\bar{a}\bar{w}, f)$. For $1 \le i < \theta$ do: $\bar{y}_i = \bar{y}_{i-1} + \mathsf{trunc}(\bar{y}_{i-1}\bar{x}_{i-1}, 2f)$; $\bar{x}_i = \mathsf{trunc}(\bar{x}_{i-1}\bar{x}_{i-1}, 2f)$. Let $\bar{y} = \bar{y}_{\theta-1} + \mathsf{trunc}(\bar{y}_{\theta-1}\bar{x}_{\theta-1}, 2f)$ (last iteration).

Correctness. Since $2^{m-1} \le \tilde{b}2^f < 2^m$, we have $2^{k-1} \le \tilde{b}2^f2^{k-m} < 2^k$ and $2^{-1} \le \tilde{b}2^{f-m} < 1$, so the normalized divisor is $\tilde{c} = \tilde{b}2^{f-m} = \tilde{b}2^{u-e}$.

The initial approximation of $1/\tilde{c}$ is $\tilde{d} = 2.9142 - 2\tilde{c}$. From $\tilde{d} \approx 1/(\tilde{b}2^{u-e})$ it follows that $\tilde{w} = \tilde{d}2^{u-e} \approx 1/\tilde{b}$. The relative error of $\tilde{w}$ is $\tilde{x} = (1/\tilde{b} - \tilde{w})/(1/\tilde{b}) = 1 - \tilde{b}\tilde{w}$. For the fixed-point types in the algorithm we obtain: $\tilde{c}2^k = \bar{b}2^{-f}2^{u-e}2^k = \bar{b}2^u$ hence $\bar{c} = \bar{b}2^u$; $\tilde{d}2^k = (\mathsf{int}_k(2.9142)2^{-k} - 2\bar{c}2^{-k})2^k$ hence $\bar{d} = \mathsf{int}_k(2.9142) - 2\bar{c}$; $\tilde{w}2^f = \bar{d}2^{-k}2^{u-e}2^f = \bar{d}2^u2^{-2e}$ hence $\bar{w} = \mathsf{trunc}(\bar{d}2^u, 2e)$; and $\tilde{x}_0 2^{2f} = (\mathsf{int}_{2f}(1.0)2^{-2f} - \bar{b}2^{-f}\bar{w}2^{-f})2^{2f}$ hence $\bar{x}_0 = \mathsf{int}_{2f}(1.0) - \bar{b}\bar{w}$.

The iterations follow the simple recurrence relations presented earlier. Correctness of the computation with limited precision is easy to verify. Observe that the two fixed-point multiplications in an iteration can be computed in parallel, and in the last iteration it is sufficient to compute $\tilde{y}_\theta$.

Signed inputs. Since $\tilde{x}_i + 1 \geq 0$ the division algorithm works for $\tilde{a} \leq 0$ without modification. The extension to $\tilde{b} < 0$ affects only the initial approximation algorithm, which is modified to return $\tilde{w} \approx 1/\tilde{b}$ with the correct sign. Thus, $\tilde{y}_0 = \tilde{a}\tilde{w}$ is initialized with the correct sign, and the iterations preserve it.

Accuracy. The quotient error has two main components: the approximation error of the method, which depends on the initial approximation and the number of iterations, and the truncation error due to computation of the iterations with limited precision. The accuracy is limited by the resolution 2^{-f} of the output.

For exact computation of the iterations, the relative error after iteration θ is $\epsilon_\theta < \epsilon_0^{2^\theta}$, where ϵ_0 is the relative error of the initial approximation of $1/\tilde{b}$. For example, we use a linear approximation with $\epsilon_0 < 0.08578$, so the approximation error of $\tilde{y}_5$ is $\epsilon_5 < 7.4\ 10^{-35}$. This implies 113 exact quotient bits, hence an absolute error less than 2^{-f} for $k = 2f \leq 112$ bits.

For θ iterations, the cumulated absolute error δ_T due to truncations is upper bounded by $\theta 2^{-f}$. This error is essentially caused by truncation of $\tilde{y}_i$, which adds an error $|\delta_{T_y}| < 2^{-f}$ per iteration. Truncation of $\tilde{x}_i$ introduces a negligible error $|\delta_{T_x}| < 2^{-2f} \ll |\delta_{T_y}|$. Assuming sufficient iterations for an approximation error 2^{-f}, the overall error of the algorithm is bound by $(\theta + 1)2^{-f}$. The error bound can be reduced to 2^{-f} by slightly increasing the resolution of $\tilde{y}_i$.

We note that the average accuracy of the truncations is better than the worst case considered above. The error bound observed in experiments with an implementation of the division protocol is actually close to 2^{-f}.

Protocols. Let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ and $\tilde{b} \neq 0$. On input $[a]$, $[b]$ the Protocol 3.3, FPDiv, computes $[y]$ such that $\tilde{y} \in \mathbb{Q}_{\langle k,f \rangle}$ and $\tilde{y} \approx \tilde{a}/\tilde{b}$, using the algorithm described above. Protocol 3.4, AppRcr, provides the initial approximation of $1/\tilde{b}$. It takes as input the divisor $[b]$ and returns $[w]$ such that $\tilde{w} \in \mathbb{Q}_{\langle k,f \rangle}$ and $\tilde{w} \approx 1/\tilde{b}$. The linear approximation is computed using the normalized value of the divisor obtained by Protocol 3.5, Norm.

Correctness. The correctness of FPDiv and AppRcr is easy to verify based on the algorithm description. Protocol Norm takes as input $[b]$, for $\tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ and computes the secret integer values $[c]$ and $[v']$ such that $2^{k-1} \leq \bar{c} < 2^k$ and $\bar{c} = \bar{b}\bar{v}'$. Suppose that $2^{m-1} \leq |\bar{b}| < 2^m$, $m \leq k$. Observe that $|\bar{v}'| = 2^{k-m}$ and $|\bar{c}| = (|\tilde{b}|2^{f-m})2^k$. Therefore, $\bar{c}$ is the representation of the normalized input $\tilde{b}2^{f-m} \in [0.5, 1)$ with resolution 2^{-k} and $\bar{v}'$ is the signed scale factor. Steps 1-2 compute the sign of $\bar{b}$ as a secret integer $\bar{s} \in \{-1, 1\}$ using the protocol LTZ, and then the absolute value of the input $\bar{x} = \bar{s}\bar{b} = |\bar{b}|$. Steps 3-10 determine the scale factor 2^{k-m} using bit decomposition and the protocol PreOR, which returns all prefixes $[y_i] = \bigvee_{j=i}^{k-1}[x_i]$, for $0 \leq i < k$. Finally, steps 11-12 compute (in parallel) the normalized input $\bar{c} = \bar{x}2^{k-m}$ and the signed scale factor $\bar{v}' = \bar{s}2^{k-m}$.

Protocol 3.3: $[y] \leftarrow \mathsf{FPDiv}([a], [b], k, f)$

```
$(\theta, \alpha) \leftarrow (\lceil \log \frac{k}{3.5} \rceil, \mathsf{fld}(\mathsf{int}_{2f}(1.0)))$;
$[w] \leftarrow \mathsf{AppRcr}([b], k, f)$;
$[x] \leftarrow \alpha - [b][w]$;
$[y] \leftarrow [a][w]$;
$[y] \leftarrow \mathsf{TruncPr}([y], 2k, f)$;
for $i \in [1..\theta - 1]$ do
    $[y] \leftarrow [y](\alpha + [x])$;
    $[x] \leftarrow [x][x]$;
    $[y] \leftarrow \mathsf{TruncPr}([y], 2k, 2f)$;
    $[x] \leftarrow \mathsf{TruncPr}([x], 2k, 2f)$;
$[y] \leftarrow [y](\alpha + [x])$;
$[y] \leftarrow \mathsf{TruncPr}([y], 2k, 2f)$;
return $[y]$;
```

Protocol 3.4: $[w] \leftarrow \mathsf{AppRcr}([b], k, f)$

```
$\alpha \leftarrow \mathsf{fld}(\mathsf{int}_k(2.9142))$;
$([c], [v]) \leftarrow \mathsf{Norm}([b], k, f)$;
$[d] \leftarrow \alpha - 2[c]$;
$[w] \leftarrow [d][v]$;
$[w] \leftarrow \mathsf{TruncPr}([w], 2k, 2(k - f))$;
return $[w]$;
```

Protocol 3.5: $([c], [v']) \leftarrow \mathsf{Norm}([b], k, f)$

```
$[s] \leftarrow 1 - 2 \cdot \mathsf{LTZ}([b], k)$;
$[x] \leftarrow [s][b]$;
$([x_{k-1}]^{\mathbb{F}_{2^8}}, \ldots, [x_0]^{\mathbb{F}_{2^8}}) \leftarrow \mathsf{BitDec}([x], k, k)$;
$([y_{k-1}]^{\mathbb{F}_{2^8}}, \ldots, [y_0]^{\mathbb{F}_{2^8}}) \leftarrow \mathsf{PreOR}([x_{k-1}]^{\mathbb{F}_{2^8}}, \ldots, [x_0]^{\mathbb{F}_{2^8}})$;
foreach $i \in [0..k - 1]$ do parallel
    $[y_i] \leftarrow \mathsf{BitF2MtoZQ}([y_i]^{\mathbb{F}_{2^8}})$;
foreach $i \in [0..k - 2]$ do
    $[z_i] \leftarrow [y_i] - [y_{i+1}]$;
$[z_{k-1}] \leftarrow [y_{k-1}]$;
$[v] \leftarrow \sum_{i=0}^{k-1} 2^{k-i-1} [z_i]$;
$[c] \leftarrow [x][v]$;
$[v'] \leftarrow [s][v]$;
return $([c], [v'])$;
```

Security. The division algorithm performs the same sequence of operations regardless of the secret values. The loop counters depend on the desired accuracy of the division operation and the fixed-point representation, which are public parameters. The three protocols do not reveal any secret-shared variable and all their sub-protocols provide either perfect or statistical privacy. We conclude that FPDiv provides statistical privacy.

Complexity. The round and communication complexity of the protocols FPDiv and AppRcr are shown in Table 3 for $k = 2f$. Observe that most of the invocations are in a small field, $\mathbb{Z}_{q_1}$ or $\mathbb{F}_{2^8}$, so their communication and computation overhead is low. All shared random bits used in FPDiv and its subprotocols can be generated in parallel in 2 rounds. An iteration is computed in 2 rounds (two fixed-point multiplications in parallel).

The complexity of FPDiv is clearly dominated by the initial approximation, especially the normalization step. For example, if $k = 112$ and $\theta = 5$ (≈ 112 bits accuracy), steps 3-12 of FPDiv are computed in 12 rounds, and AppRcr adds 29 rounds (27 rounds for Norm), giving a total of 43 rounds. A variant of FPDiv with positive divisor is sufficient in many applications and can be computed in 33 rounds by skipping the steps 1, 2, and 12 of Norm.

Note that the building blocks in Table 2 are optimized for low communication complexity. The round complexity of FPDiv can be reduced using building blocks that trade off higher communication complexity for a lower number of rounds.

4 Conclusions

Business applications of secure computation need a protocol family that provides operations with all primitive data types and allows secure protocol composition and efficient application development. We presented a protocol family that fills an important gap by enabling secure computation with rational numbers.

Fixed-point representation offers the most efficient encoding of rational numbers as well as efficient protocols for the most frequent operations: addition, subtraction, multiplication, and comparison. Division is simple for public divisor, but becomes quite complex when the divisor is secret. On the other hand, secure arithmetic with floating-point numbers is clearly not practical.

The protocols have been implemented in Java and tested in complex applications like secure linear programming using Simplex (with a variant of the division protocol that was optimized for multiple divisions with the same divisor).

On-going work focuses on improving the efficiency of division and adding protocols for secure evaluation of other mathematical functions.

Acknowledgements. Part of the work presented in this paper was funded by the European Commission through the grant FP7-213531 to the SecureSCM project. The authors thank the anonymous reviewers for their helpful comments.

References

1. J. Algesheimer, J. Camenish, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 417–432. Springer-Verlag, 2002.

2. M. Atallah, M. Blanton, V. Deshpande, K. Frikken, J. Li, and L. Schwarz. Secure Collaborative Planning, Forecasting, and Replenishment (SCPFR). In *Multi-Echelon/Public Applications of Supply Chain Management Conference*, 2006.
3. M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. In *Proc. WPES 2004*, Washington, 2004.
4. F. Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5(4):201–216, 2006.
5. O. Catrina and C. Dragulin. Multiparty Computation of Fixed-Point Multiplication and Reciprocal. In *Proc. 20th International Workshop on Database and Expert Systems Application (DEXA 2009)*, pages 107–111. IEEE Computer Society, 2009.
6. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proc. of 2nd Theory of Cryptography Conference (TCC'05)*, pages 342–362, 2005.
7. R. Cramer, I. Damgård, and U. Maurer. General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer-Verlag, 2000.
8. I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proc. of 3rd Theory of Cryptography Conference (TCC'06)*, volume 3876 of *LNCS*, pages 285–304. Springer-Verlag, 2006.
9. I. Damgård, J. Nielsen, T. Toft, J. I. Pagter, T. Jakobsen, P. Bogetoft, and K. Nielsen. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. In *Proc. of Financial Cryptography 2006*, volume 4107 of *LNCS*, pages 142–147. Springer-Verlag, 2006.
10. I. Damgård and R. Thorbek. Non-interactive Proofs for Integer Multiplication. In *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 412–429. Springer-Verlag, 2007.
11. I. Damgard and R. Thorbek. Efficient Conversion of Secret-shared Values Between Different Fields. Cryptology ePrint Archive, Report 2008/221, 2008.
12. M. D. Ercegovac and T. Lang. *Digital Arithmetic.* Morgan Kaufmann, 2003.
13. P. Fouque, J. Stern, and G. Wackers. CryptoComputing with Rationals. In *FC 2002*, volume 2357 of *LNCS*, pages 136–146. Springer-Verlag, 2003.
14. S. L. From and T. Jakobsen. Secure Multi-Party Computation on Integers. Master's thesis, Univ. of Aarhus, Denmark, BRICS, Dep. of Computer Science, 2006.
15. E. Kiltz, G. Leander, and J. Malone-Lee. Secure Computation of the Mean and Related Statistics. In *Proc. Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *LNCS*. Springer-Verlag, 2005.
16. P. Markstein. Software Division and Square Root Using Goldschmidt's Algorithms. In *Proc. 6th Conference on Real Numbers and Computers*, pages 146–157, 2004.
17. N. T. Masayuki Ito and S. Yajima. Efficient Initial Approximation for Multiplicative Division and Square Root by a Multiplication with Operand Modification. *IEEE Transactions on Computers*, 46(4), 1997.
18. T. Nishide and K. Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In *PKC 2007*, volume 4450 of *LNCS*, pages 343–360. Springer-Verlag, 2007.
19. T. Toft. *Primitives and Applications for Multi-party Computation.* PhD dissertation, Univ. of Aarhus, Denmark, BRICS, Dep. of Computer Science, 2007.
20. T. Toft. Solving Linear Programs Using Multiparty Computation. In *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.