PAPER Special Section on Cryptography and Information Security

Efficiency and Accuracy Improvements of Secure Floating-Point Addition over Secret Sharing*

Kota SASAKI[†], *Nonmember and* **Koji NUIDA**[††,†††a)], *Member*

SUMMARY In secure multiparty computation (MPC), floating-point numbers should be handled in many potential applications, but these are basically expensive. In particular, for MPC based on secret sharing (SS), the floating-point addition takes many communication rounds though the addition is the most fundamental operation. In this paper, we propose an SS-based two-party protocol for floating-point addition with 13 rounds (for single/double precision numbers), which is much fewer than the milestone work of Aliasgari et al. in NDSS 2013 (34 and 36 rounds, respectively) and also fewer than the state of the art in the literature. Moreover, in contrast to the existing SS-based protocols which are all based on "roundTowardZero" rounding mode in the IEEE 754 standard, we propose another protocol with 15 rounds which is the first result realizing more accurate "roundTiesToEven" rounding mode. We also discuss possible applications of the latter protocol to secure Validated Numerics (a.k.a. Rigorous Computation) by implementing a simple example.

key words: *secure multiparty computation, floating-point numbers, secret sharing*

1. Introduction

Secure multiparty computation (*MPC*) is a cryptographic technology enabling two or more parties to compute functions on their individual inputs in a way that each party's input is kept secret during the computation. Besides the interest from cryptology, MPC is also important in other areas as a core technology in privacy-preserving data mining, which is expected by the increasing social demands for both big-data utilization and sensitive data protection.

Major underlying cryptographic primitives in recent studies for MPC are *garbled circuit* (*GC*), *homomorphic encryption* (*HE*), and *secret sharing* (*SS*). Each of them has its own advantages and disadvantages, and there are studies for developing general and efficient MPC protocols by combining these different primitives (e.g., [10], [11]). On the other hand, studying MPC based on a single primitive is still important by at least the following two reasons. First, such protocols can fully benefit from the advantages specific to the primitive. For example, SS (with ideally generated randomness) is the only primitive among those listed above that can achieve information-theoretic security. Secondly, improvements of building-block protocols based on each primitive can also potentially contribute to better performances of the mixed-primitive type MPC mentioned above. From such viewpoints, in this paper we focus on SS-based MPC.

In SS-based MPC, efficient constructions of three-party computation (3PC), i.e., with three computing parties, have been studied well (e.g., [4], [5], [26]). On the other hand, SS-based two-party computation (2PC) also has an advantage that it requires less hardware resources (e.g., fewer computing servers) than 3PC. This paper focuses on the latter, SS-based 2PC. In both 2PC and 3PC, a recent trend is to divide the whole computation into the input-independent offline (pre-computation) phase and the input-dependent online phase and to make the online phase more efficient. We also follow this strategy. Moreover, a major disadvantage of SS-based MPC is that it requires many communication rounds, which is frequently dominant among the whole running time due to the unavoidable network latency for each communication round, especially over Wide Area Network (WAN) with larger latency. Hence, reducing the number of communication rounds is particularly important in SS-based MPC.

1.1 Secure Computation for Floating-Point Numbers

There are studies of MPC for various kinds of applications where floating-point numbers are required (e.g., [2], [12], [16], [23]). A milestone in SS-based MPC for floating-point numbers is the work by Aliasgari et al. [3], where 3PC protocols for basic arithmetic addition and multiplication (note that subtraction is immediately obtained from addition by flipping the sign of the second input) as well as for some advanced operations such as square root and logarithm are proposed. Their high-level protocol constructions are also applied to the 2PC case in [1] (but the underlying primitive in [1] is HE instead of SS.)

However, there is a typical difficulty in SS-based MPC for floating-point numbers. For the cases of integers or fixed-point numbers, usually the addition is almost for free, i.e., executable locally at each party without communication. In contrast, for floating-point numbers, the addition requires complicated operations (e.g., alignment of the significands according to the difference of exponents) and therefore is much expensive (with more than 27 rounds in [3]), even more than the multiplication (11 rounds in [3]). As the addition is the most fundamental operation, it is undoubtedly important

Manuscript received March 15, 2021.
Manuscript revised June 9, 2021.
Manuscript publicized September 9, 2021.

[†]The author is with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113-8656 Japan.
[††]The author is with Institute of Mathematics for Industry (IMI), Kyushu University, Fukuoka-shi, 819-0395 Japan.
[†††]The author is with National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, 135-0064 Japan.

*This is the full version of a paper presented at IWSEC 2020.

a) E-mail: nuida@imi.kyushu-u.ac.jp

DOI: 10.1587/transfun.2021CIP0013

Table 1 Comparison of the SS-based floating-point addition protocols. (*) The encoding format is not compatible with [3]. (**) Changing probabilistically. (***) Depending on the signs of inputs (see the main text). (†) Assuming the trusted initializer model for pre-computation (see the main text).

Protocol	# of parties	Rounding direction	# of rounds		Bit length of encoding	Security
			binary32	binary64		
[3]	3	Truncation	34	36	$2l+1$	statistical
[7](*)	3	Truncation	15	15	$l+1$	statistical
[8]	3	unstable(**)	19	19	$l+2$	statistical
[9]	3	unstable(**)	16	16	$l+2$	statistical
[17]	2(†)	unstable(***)	N/A	N/A	$l+2$	perfect
Ours (trunc)	**2**(†)	**Truncation**	**13**	**13**	$\mathbf{l+2}$	**perfect**
Ours (even)	**2**(†)	**Round-to-Even**	**15**	**15**	$\mathbf{l+4}$	**perfect**

to improve the efficiency of the addition protocol. There are studies of improving MPC for floating-point arithmetic (including the addition) by combining SS with GC (e.g., [14], [22]). In this paper, we focus on SS-based (without GC) 2PC for the floating-point addition; only few studies for this topic exist in the literature (see Sect. 1.2).

1.2 Our Contributions

In this paper, we propose an improvement of SS-based secure floating-point addition (and subtraction) with two computing parties in the semi-honest model. For the binary64 format in the IEEE 754 standard [15] (with 53-bit significand), our proposed 2PC protocol requires only 13 rounds in the online phase, which is approximately 36% of the number of communication rounds (36 rounds) in [3] (the latter takes $\log_2 l + \log_2 \log_2 l + 27$ rounds as shown in Table 1 of [3] and we have $\lceil \log_2 53 + \log_2 \log_2 53 + 27 \rceil = 36$). Some comparisons with the existing results are summarized in Table 1; see the text below for the details.

As mentioned in Sect. 1.1, the 3PC protocol in [3] can be converted into 2PC by replacing the subprotocols with those executable by 2PC. Instead of the aforementioned HE-based ones in [1], here we use the round-efficient SS-based 2PC protocols in a recent paper [20]. (The combination of our construction with other kinds of round-efficient protocols such as in [6] will be a future research topic.) However, we emphasize that our protocol is not just a straightforward combination of [20] with [3], as explained below.

In [3], each l-bit significand is encoded as a $(2l+1)$-bit (or larger) integer and the alignment of significands is done by left-shift (toy example: treating $1.101 \times 2^3 + 1.010 \times 2^1$ as $(110.100 + 1.010) \times 2^1$). However, by this method, even a binary32 (resp. binary64) number with $l = 24$ (resp. 53) cannot be implemented by 32-bit (resp. 64-bit) integers. This is not only inconvenient in implementation, but also a source of inefficiency in a way that the efficiency of the subprotocols in [20] depends largely on the bit length of integers, therefore it is better to avoid unnecessarily large integers. Instead of the left-shift alignment, here we adopt the opposite, right-shift alignment for this purpose. However, we should also take care of the fact that a naive use of right-shift may make the direction of rounding unstable depending on the signs of inputs. We explain it by toy examples. When $1.110 \times 2^4 + 1.100 \times 2^1$ is treated as $(1.110 + 0.001\cancel{100}) \times 2^4$, the exact result 1.111100×2^4 is rounded *down* to 1.111×2^4. On the other hand, when $1.110 \times 2^4 - 1.100 \times 2^1$ is treated as $(1.110 - 0.001\cancel{100}) \times 2^4$, the exact result 1.100100×2^4 is rounded *up* to 1.101×2^4. (This kind of error in fact happens in the protocol of [17].) We avoid this issue by carefully combining the left-shift and the right-shift, obtaining a correct protocol with the significands encoded by just $(l+2)$-bit integers. See Sect. 4.2 for details.

We note also that among the various rounding-direction attributes in the IEEE 754 standard, the existing SS-based floating-point addition protocols support *roundTowardZero* only (i.e., the absolute value of the precise result is rounded down). For simplicity, we call it *Truncation mode* in this paper. On the other hand, our protocol (with two additional communication rounds) also supports *roundTiesToEven*, which rounds the precise result to the nearest floating-point number; we call it *Round-to-Even mode* in this paper. In other words, *our protocol newly supports the most accurate mode for floating-point addition*. See Sect. 4.3 for details. This would be useful in potential applications of MPC where the accuracy is very important, e.g., secure numerical simulations using expensive experimental data held by private companies. As a toy example, based on our protocol with Round-to-Even mode, we implemented a 2PC protocol for error-free transformation for the sum of two floating-point numbers (see e.g., [24]), which is a simple kind of Validated Numerics (a.k.a. Rigorous Computation) in the area of numerical analysis. See Sect. 6 for details. Our future research topics include extension of our proposed protocol to other arithmetic operations for floating-point numbers (i.e., multiplication and division).

We summarize comparison results of our proposed protocols with the existing SS-based MPC for floating-point addition in Table 1 (we also note about other protocols in [16], [21] where the number of communication rounds is not clear and the improvement relative to [3] seems not very large). We show the numbers of communication rounds in the online phase for binary32 and binary64 formats. The "Bit length of encoding" column shows the bit length of integers to which the l-bit significands are encoded during each protocol. The "Security" column shows whether each protocol (assuming ideally generated randomness) is perfectly secure or only statistically secure. In [7], the encoding format of floating-point numbers is slightly different from [3]; the *signed* significand is encoded as a single *signed* integer,

while the sign bit and the unsigned significand are separated in [3], therefore the protocol in [7] does not have compatibility with those in [3]. In [8], [9], some subprotocol inside the whole protocol performs rounding *probabilistically* towards one of the two directions. The protocols in [3], [7]–[9] use imperfect random masking in some subprotocol and therefore have only statistical security. In [17], the number of communication rounds is not clear, and its rounding direction is not stable as mentioned above. By Table 1, our 2PC protocol (in Truncation mode) has round complexity even smaller than the best known (accurate) 3PC protocol [7]. We note that our protocols (as well as the protocols in [20]) require somewhat complicated correlated randomness and therefore the use of the so-called trusted initializer model, such as the client-aided model (e.g., [18], [19]), in the offline phase suits well.

1.3 Publication Information

A conference version of this work has been published in Proceedings of IWSEC 2020 [25]. Compared to the conference version, we added more detailed explanations of correctness of the main proposed protocols (such as Theorem 1) and of some subprotocol (as Proposition 1), added more explanation about client-aided secure multiplication (as Sect. 2.1.2), and performed editorial improvements such as changing the notation for shares for the sake of readability.

2. Preliminaries

2.1 Secure Multiparty Computation over Secret Sharing

2.1.1 Additive Secret Sharing and Secure Addition

This paper deals with two-party computation (2PC) based on the following 2-out-of-2 additive secret sharing (SS) scheme $(\mathsf{Share}, \mathsf{Reconst})$ over a ring $\mathcal{M} = \mathbb{Z}/2^n\mathbb{Z}$. For a plaintext $m \in \mathcal{M}$, the algorithm $\mathsf{Share}(m)$ chooses $s_0 \leftarrow \mathcal{M}$ uniformly at random, sets $s_1 \leftarrow m - s_0$, and outputs $\langle m \rangle = (\langle m \rangle_0, \langle m \rangle_1) \leftarrow (s_0, s_1)$. Each $\langle m \rangle_i$ is called a *share* of m for i-th party, and by abusing the terminology, the pair $\langle m \rangle$ is also called a share of m. On the other hand, for a share $\langle m \rangle = (\langle m \rangle_0, \langle m \rangle_1)$, the algorithm $\mathsf{Reconst}(\langle m \rangle)$ outputs a plaintext $\langle m \rangle_0 + \langle m \rangle_1$. We note that the reconstruction of the plaintext requires one communication round between the two parties to send each party's share to the other party. When $n = 1$ and a plaintext is regarded as a bit, each share is called a Boolean share and denoted by putting a superscript 'B' like $\langle m \rangle^{\mathsf{B}}$. Otherwise, each share is called an Arithmetic share and denoted by putting a superscript 'A' like $\langle m \rangle^{\mathsf{A}}$.

Given two Arithmetic shares $\langle m_1 \rangle^{\mathsf{A}}$ and $\langle m_2 \rangle^{\mathsf{A}}$ (with the same bit length n), addition of plaintexts is executable locally (without communication) by adding each party's shares; $\langle m_1 + m_2 \rangle_i^{\mathsf{A}} \leftarrow \langle m_1 \rangle_i^{\mathsf{A}} + \langle m_2 \rangle_i^{\mathsf{A}}$ $(i = 0, 1)$. We write this protocol simply as $\langle m_1 \rangle^{\mathsf{A}} + \langle m_2 \rangle^{\mathsf{A}}$. Subtraction $\langle m_1 \rangle^{\mathsf{A}} - \langle m_2 \rangle^{\mathsf{A}}$ and scalar multiplication $c \cdot \langle m \rangle^{\mathsf{A}}$ can be locally executed in a similar way. Addition by constant $\langle m \rangle^{\mathsf{A}} + c$ can be also done by using $\langle c \rangle^{\mathsf{A}} = (c, 0)$. By replacing + with $\oplus$, exclusive OR (XOR) for Boolean shares $\langle b_1 \rangle^{\mathsf{B}} \oplus \langle b_2 \rangle^{\mathsf{B}}$ are obtained. Then NOT operation is given by $\overline{\langle b \rangle}^{\mathsf{B}} = \neg \langle b \rangle^{\mathsf{B}} = \langle b \rangle^{\mathsf{B}} \oplus 1$.

2.1.2 Client-Aided Secure Multiplication

Multiplication of plaintexts over the additive SS, denoted by $\langle m_1 \cdot m_2 \rangle^{\mathsf{A}} = \langle m_1 \rangle^{\mathsf{A}} \cdot \langle m_2 \rangle^{\mathsf{A}}$, can be executed with one communication round sending one share to each other, by using auxiliary inputs $(\langle a \rangle^{\mathsf{A}}, \langle b \rangle^{\mathsf{A}}, \langle c \rangle^{\mathsf{A}})$ called *Beaver Triple* (*BT*) where $a, b \in \mathcal{M}$ are uniformly random and not known by any party and we have $c = ab$. See e.g., [20] for the concrete protocol construction. Boolean AND $\langle b_1 \wedge b_2 \rangle^{\mathsf{B}} = \langle b_1 \rangle^{\mathsf{B}} \wedge \langle b_2 \rangle^{\mathsf{B}}$ can be computed similarly (with one round sending one bit each), and Boolean OR $\langle b_1 \vee b_2 \rangle^{\mathsf{B}} = \langle b_1 \rangle^{\mathsf{B}} \vee \langle b_2 \rangle^{\mathsf{B}}$ can be computed as well with the same communication complexity (by combining Boolean AND with locally executable Boolean NOT).

In this paper, we assume that any auxiliary input such as BT is ideally generated in the offline phase. For example, this may be realized by adopting *client-aided model* [18], [19] for SS-based MPC (which is a kind of so-called trusted initializer model) where some trusted party other than the computing parties is supposed to be active only at the offline phase and to generate and send the necessary auxiliary inputs to the computing parties.

2.1.3 Security Notion

Here we explain the security notion for MPC specialized to the two-party case, as we only deal with 2PC in this paper. Suppose that two parties P_0 and P_1 compute functionality $f(x_0, x_1) = (f_i(x_0, x_1))_{i=0}^1$ with input x_i for P_i. Let the *view* $\mathsf{View}_i(x_0, x_1)$ for Party P_i during a protocol be the chronological list of all messages sent from the other party to P_i together with the local input x_i and the internal randomness for P_i used in the protocol. Let $\mathsf{Out}(x_0, x_1)$ denote the pair of the two parties' outputs at the end of the protocol. Then we say that a protocol is *secure* (*in the semi-honest model*) *against* P_i if there is a probabilistic polynomial-time algorithm $\mathcal{S}_i$ satisfying that the probability distribution of the pair $(\mathcal{S}_i(x_i, f_i(x_0, x_1)), f(x_0, x_1))$ is indistinguishable from the probability distribution of $(\mathsf{View}_i(x_0, x_1), \mathsf{Out}(x_0, x_1))$; see e.g., Definition 7.5.1 of [13]. In particular, when the two distributions above are identical (resp. statistically indistinguishable), the protocol is said to be perfectly (resp. statistically) secure.

The Composition Theorem (Theorem 7.5.7 of [13]) states, roughly speaking, that if a protocol $\Pi^{(g_1, \ldots, g_L)}$ using oracles computing functionalities $g_1, \ldots, g_L$ is secure and each g_j can be computed securely, then the protocol obtained by replacing each oracle call for g_j in $\Pi^{(g_1, \ldots, g_L)}$ with the secure protocol computing g_j is also secure. Now we note that, in any protocol proposed in this paper, the subprotocols used in the protocol are known to be secure, while the protocol assuming ideal subprotocols is trivially secure as

the parties send data to each other only during some subprotocol; as a result, the whole protocol is proven secure due to the Composition Theorem. According to this fact, in the rest of this paper we safely omit discussions about the security of our proposed protocols.

2.2 Floating-Point Numbers

2.2.1 Floating-Point Representation

The floating-point numbers are the most common way of representing in computers an approximation to real numbers. Here, we explain the details of binary floating-point formats.

The set of all floating-point numbers consists of finite numbers (including normalized numbers, unnormalized numbers, and zero), infinities, and NaN (not-a-number). For simplicity we only treat normalized numbers and zero.

Following the format in [3], a floating-point number x is defined as a tuple (f, e, s, z), where $f \in [2^{l-1}, 2^l - 1] \cup \{0\}$ is the unsigned, normalized significand, $e \in \mathbb{Z}/2^k\mathbb{Z}$ is the biased exponent, s is the sign bit which is set to 1 when the value is negative, and z is the zero-test bit which is set to 1 if and only if $x = 0$. The value of the number is

$$x = (1 - 2s) \cdot (1 - z) \cdot 2^{-l+1} f \cdot 2^{e - \mathsf{bias}} \ ,$$

where $\mathsf{bias} = 2^{k-1} - 1$. We note that both f and e are set to 0 when x is zero.

The parameters l, k are not secret and determine the precision and range of the numbers, and these parameters affect the numbers of communication rounds and the computational complexity of the protocols. In this paper, we focus on $(l, k) = (24, 8)$ which corresponds to IEEE 754 single precision, and $(l, k) = (53, 11)$ which corresponds to IEEE 754 double precision [15]. For example, if $(l, k) = (24, 8)$, then the value $x = 1.0$ is represented as

$$f = \underbrace{10 \cdots 0}_{24}, \ e = 0 + (2^{8-1} - 1) = 127, \ s = 0, \ z = 0 \ .$$

2.2.2 Floating-Point Addition/Subtraction and Rounding Mode

The floating-point addition/subtraction is executed as follows:

1. Compare the absolute values of the two numbers.
2. Shift the smaller absolute value to the right until its exponent would match the larger exponent.
3. Add or subtract the significands.
4. Normalize the result, either shifting right and incrementing the exponent or shifting left and decrementing the exponent.
5. Round the significand to the appropriate number of bits.
6. Check if the result is still normalized or not. If the result is not normalized, normalize the result again.

In this paper, we assume for simplicity that no overflow or underflow occurs in addition/subtraction. We note that there are no SS-based protocols for floating-point arithmetic in the literature that handle overflow or underflow. An extension of our result to handling overflow/underflow detection as well is left as a future research topic.

IEEE 754 standard defines five rounding rules; roundTiesToEven, roundTies-ToAway, roundTowardPositive, roundTowardNegative, and roundTowardZero. The first two rules round to a nearest value, the others are called directed roudings. In this paper, we use roundTiesToEven (we call it *Round-to-Even* mode) and roundTowardZero (we call it *Truncation* mode).

Round-to-Even mode rounds to the nearest value. With this mode, if two candidate values with equal distance exist, the number is rounded to the one with the least significant bit being 0. This mode is the most commonly used. Truncation mode is a direct rounding towards zero. Note that we have to check once that the intermediate result is normalized after rounding with Round-to-Even mode, but we do not need this check in Truncation mode.

2.2.3 Extra Bits to Obtain Accurate Result

To get the same result as if the intermediate result were calculated with infinite precision and then rounded, the following bits are defined for determining the magnitude lower than the units in the last place (ulp); *guard bit* has weight $1/2$ ulp, *round bit* has weight $1/4$ ulp, and *sticky bit* is the OR of all bits with weight $1/8$ ulp or lower. With these three extra bits, we can obtain accurate result with Round-to-Even mode. By almost the same idea, we can obtain accurate result with Truncation mode using one extra bit, which is set whenever there is some nonzero bit to the right of ulp.

3. The Previous Works

3.1 Secure Floating-Point Operations

In [3], SS-based protocols to compute fundamental arithmetic operations (including the addition) and some advanced functions for floating-point numbers are proposed. In their protocols, the two components f, e in the format described in Sect. 2.2.1 are treated as Arithmetic shares and the other two components s, z are treated as Boolean shares.

In the addition protocol of [3], first the two input numbers are sorted with respect to their significands (which requires secure comparison protocol) for the ease of the remaining steps. Then, in contrast to the addition algorithm for plain (non-secret) inputs explained in Sect. 2.2.2, the protocol in [3] aligns the significands by using left-shift instead of right-shift (see also an explanation in Sect. 1.2); this is convenient to ensure the accuracy of the resulting value, while this requires (almost) twice the bit length of integers to hold the intermediate significand. After that, the most

Table 2 Numbers of communication rounds for the subprotocols in [20] (upper part) and our subprotocols in Sect. 4.1 (lower part); here Arithmetic shares are n-bit integers. (*) $\beta_j = 1$ if and only if $b_j = 1$ and $b_k = 0$ for any $k > j$. (**) $b = 1$ if and only if $(\langle x\rangle_0^{\mathsf{A}} \bmod 2^k) + (\langle x\rangle_1^{\mathsf{A}} \bmod 2^k) \geq 2^k$. (***) Here $x = (x_{n-1} \cdots x_1 x_0)_2$ in binary.

Protocol	Functionality	# of communication rounds	
		$(n = 8)$	$(n \in \{16, 32, 64\})$
$\langle z\rangle \leftarrow N\text{-Mult}((\langle x_j\rangle)_{j=1}^N)$	$z = x_1 \cdots x_N$	1	1
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}})$	$b = (x \stackrel{?}{=} y)$	1	2
$(\langle \beta_j\rangle^{\mathsf{B}})_{j=0}^{N-1} \leftarrow \mathsf{MSNZB}((\langle b_j\rangle^{\mathsf{B}})_{j=0}^{N-1})$	(*)	1	2
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Overflow}(\langle x\rangle^{\mathsf{A}}, k)$	(**)	1	2
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Comparison}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}})$	$b = (x \stackrel{?}{<} y)$	2	3
$\langle z\rangle^{\mathsf{A}} \leftarrow \mathsf{B2A}(\langle b\rangle^{\mathsf{B}})$	$z = b$	1	1
$\langle z\rangle^{\mathsf{A}} \leftarrow \langle b\rangle^{\mathsf{B}} \times \langle x\rangle^{\mathsf{A}}$	$z = b \cdot x$	1	1
$\langle z\rangle^{\mathsf{A}} \leftarrow \langle b\rangle^{\mathsf{B}} \times \langle c\rangle^{\mathsf{B}}$	$z = b \cdot c$	1	1
$\langle z\rangle^{\mathsf{A}} \leftarrow \langle b\rangle^{\mathsf{B}} \times \langle c\rangle^{\mathsf{B}} \times \langle x\rangle^{\mathsf{A}}$	$z = b \cdot c \cdot x$	1	1
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Modeq}(\langle x\rangle^{\mathsf{A}}, k)$	$b = (x \bmod 2^k \stackrel{?}{=} 0)$	1	2
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Extractbit}(\langle x\rangle^{\mathsf{A}}, k)$	$b = x_k$ (***)	1	2
$(\langle b[j]\rangle^{\mathsf{B}})_{j=0}^{n-1} \leftarrow \mathsf{Bitdec}(\langle x\rangle^{\mathsf{A}})$	$b[j] = x_j$ (***)	1	2

significant non-zero bit of the intermediate significand is searched (which requires secure bit extraction protocol) and the intermediate significand is normalized by an appropriate shift. The protocol also includes procedures to handle some exceptional case (e.g., one of the two inputs is zero) and to maintain the bits s and z correctly. Here we omit the details of their protocol (see the original paper [3]), but we note that it is a three-party computation (3PC) protocol and it (with l-bit significands for inputs) takes $\log_2 l + \log_2 \log_2 l + 27$ rounds. This becomes 34 and 36 rounds for binary32 and binary 64 inputs with $l = 24$ and 53, respectively.

3.2 Round-Efficient Protocols over Secret Sharing

In our proposed protocols, we use the 2PC protocols in [20] as subprotocols. We omit their concrete constructions here and only refer to [20]. Their functionalities and numbers of communication rounds are listed in the upper part of Table 2.

We note that in [20], the N-fan-in multiplication (N-Mult) for $N > 2$ (and similarly, N-fan-in AND/OR) is realized with one round by introducing an extension of Beaver Triple (BT) called *Beaver Triple Extension* (*BTE*). The fan-in number N can in principle be arbitrary, but the computational and the communication costs grow exponentially in N, therefore the choice of N is limited as $N \leq 9$ in [20]. For the Equality protocol, roughly speaking, the computation is reduced to the computation of OR for at most n significant bits of the input; by using N-OR's for $N \leq 9$, it takes two rounds when $n \in \{16, 32, 64\}$ as described in [20], while it can be done simply with one round for $n = 8$. The situation is similar for the most significant non-zero bit search protocol MSNZB, the overflow detection protocol Overflow, and the less-than comparison protocol Comparison, where the number of rounds for $n = 8$ is fewer than those for $n \in \{16, 32, 64\}$ by one. On the other hand, the Boolean-to-Arithmetic conversion protocol B2A and the multiplication protocols for Boolean and Arithmetic inputs (regarded as Arithmetic values) are executable with one round for any $n \in \{8, 16, 32, 64\}$.

We note that, in our proposed protocols in this paper, we only need N-Mult for $N \leq 5$ (with at most $(2^5 - 1)n = 31n$ bits of BTE) and N-AND/OR for $N \leq 9$ (with at most $2^9 - 1 = 511$ bits of BTE), therefore the communication complexity of our protocol in the offline phase is not too large.

4. Our Proposed Protocols

4.1 Some More Subprotocols

Here we present some more subprotocols to be used in our proposed protocols. Their functionalities and numbers of communication rounds are listed in the lower part of Table 2.

4.1.1 Modeq

A protocol $\mathsf{Modeq}(\langle x\rangle^{\mathsf{A}}, k)$ outputs $\langle b\rangle^{\mathsf{B}}$, where $b = 1$ if and only if $(x \bmod 2^k) = 0$. By almost the same idea as Equality, we can construct this protocol with the same number of communication rounds as Equality owing to the relation $x \bmod 2^k = (\langle x\rangle_0^{\mathsf{A}} \bmod 2^k) + (\langle x\rangle_1^{\mathsf{A}} \bmod 2^k)$ in $\mathbb{Z}/2^k\mathbb{Z}$.

4.1.2 Extractbit

A protocol $\mathsf{Extractbit}(\langle x\rangle^{\mathsf{A}}, k)$ outputs $\langle b\rangle^{\mathsf{B}}$, where $b = x_k$ is the $(k+1)$-th least significant bit of (binary expanded) $x = (x_{n-1} \cdots x_1 x_0)_2$. Using Overflow, we can construct Extractbit with the same number of communication rounds as follows, where $\mathsf{Overflow}(\langle x\rangle^{\mathsf{A}}, k)$ is a protocol to output a share $\langle b\rangle^{\mathsf{B}}$ with the property that $b = 1$ if and only if $(\langle x\rangle_0^{\mathsf{A}} \bmod 2^k) + (\langle x\rangle_1^{\mathsf{A}} \bmod 2^k) \geq 2^k$:

1. P_i ($i \in \{0, 1\}$) locally extend $\langle x\rangle_i^{\mathsf{A}}$ to binary and obtain a bit string $(\langle t[n-1]\rangle_i^{\mathsf{B}}, \cdots, \langle t[0]\rangle_i^{\mathsf{B}})$, then set $\langle v\rangle_i^{\mathsf{B}} = \langle t[k]\rangle_i^{\mathsf{B}}$. P_i also compute $\langle w\rangle^{\mathsf{B}} \leftarrow \mathsf{Overflow}(\langle x\rangle^{\mathsf{A}}, k)$.

2. P_i compute $\langle z\rangle^{\mathsf{B}} = \langle v\rangle^{\mathsf{B}} \oplus \langle w\rangle^{\mathsf{B}}$.

Proposition 1. *The protocol above correctly computes the functionality of* Extractbit.

Proof. By noting that the $(k+1)$-th least significant bit of $\langle x\rangle_i^{\mathsf{A}}$ is now denoted by $\langle t[k]\rangle_i^{\mathsf{B}}$, if an overflow from the lower bit in the binary addition of $\langle x\rangle_0^{\mathsf{A}} + \langle x\rangle_1^{\mathsf{A}}$ does not occur, then the $(k+1)$-th least significant bit of x is equal to $\langle t[k]\rangle_0^{\mathsf{B}} + \langle t[k]\rangle_1^{\mathsf{B}} \bmod 2 = \langle v\rangle_0^{\mathsf{B}} \oplus \langle v\rangle_1^{\mathsf{B}} = v$. On the other hand, if an overflow from the lower bit occurs, then the $(k+1)$-th least significant bit of x is the negation of the v. As the occurrence of the overflow is detected by the bit w, the claim holds. □

4.1.3 Bitdec

A bit decomposition protocol $\mathsf{Bitdec}(\langle x\rangle^{\mathsf{A}})$ outputs a Boolean share vector $\langle b\rangle^{\mathsf{B}} = (\langle b[n-1]\rangle^{\mathsf{B}}, \cdots, \langle b[0]\rangle^{\mathsf{B}})$, where $b[k] = x_k$ is the $(k+1)$-th least significant bit of (binary expanded) x. We can construct Bitdec by parallelly executing Extractbit n times.

4.2 Addition/Subtraction in Truncation Mode

Here we show our proposed floating-point addition protocol in Truncation mode (Algorithm 1). We abuse some notations; for example, we write $\langle b\rangle^{\mathsf{B}} \times \langle x\rangle^{\mathsf{A}}$ protocol simply as $\langle b\rangle^{\mathsf{B}}\langle x\rangle^{\mathsf{A}}$. In the algorithm, the numbers at the right side with symbols '▷' denote the numbers of communication rounds required for each step; for example, Step 1 takes three rounds where Comparison to compute $\langle c\rangle^{\mathsf{B}}$ is dominant among the parallel processes. We note that, for example, the formula to compute $\langle f'_{\min}\rangle^{\mathsf{A}}$ in Step 4 seems to require two consecutive multiplications, but this description is just for clarifying the structure of the formula, and the actual computation is performed with its expanded form

$$\sum_{j=0}^{l} \langle t[j]\rangle^{\mathsf{B}}\langle t'[j]\rangle^{\mathsf{A}} - 2\sum_{j=0}^{l} \langle s_2\rangle^{\mathsf{B}}\langle t[j]\rangle^{\mathsf{B}}\langle t'[j]\rangle^{\mathsf{A}}$$

which requires only one round. Similar conventions for descriptions of functions are also adopted in several places. Summarizing, the total number of communication rounds is 13 for both binary32 and binary64 floating-point numbers.

Theorem 1. *Algorithm 1 correctly computes the functionality of floating-point addition in Truncation mode.*

Proof. In our encoding rule, the zero value $x_i = 0$ is represented by $(f_i, e_i, s_i, z_i) = (0,0,0,1)$. First we discuss exceptional cases where at least one of the input values is zero.

- We suppose that $x_0 = 0$, i.e., $(f_0, e_0, s_0, z_0) = (0,0,0,1)$. As $\overline{z_0} = 0$, in Step 5 we have $z_2 = z_3 = 0$. Therefore, in Step 6 we have $f = f_1$ and $e = e_1$. On the other hand, in Step 1 we have $s_2 = s_1$, and in Step 2 we have $s = a \wedge s_1 \oplus b \wedge c \wedge s_1$ and $z = b \wedge b' \wedge s_1 \oplus z_1$. Now if $x_1 = 0$, then we have $(f_1, e_1, s_1, z_1) = (0,0,0,1)$, therefore $s = 0 \oplus 0 = 0$ and $z = 0 \oplus 1 = 1$. Hence $(f, e, s, z) = (0,0,0,1)$, which corresponds to the property $x_0 + x_1 = 0$. On the other hand, if $x_1 \neq 0$, then we have $e_1 > 0$ and $z_1 = 0$. This implies that $a = 1$ and $b = 0$, therefore $s = s_1 \oplus 0 = s_1$ and $z = 0 \oplus z_1 = z_1$. Hence $(f, e, s, z) = (f_1, e_1, s_1, z_1)$, which corresponds to the property $x_0 + x_1 = x_1$.
- We suppose that $x_1 = 0$, i.e., $(f_1, e_1, s_1, z_1) = (0,0,0,1)$, and $x_0 \neq 0$, i.e., $e_0 > 0$ and $z_0 = 0$ (note that the case $x_0 = x_1 = 0$ has been considered above). By a similar argument, in Step 5 we have $z_2 = z_3 = 0$, and in Step 6 we have $f = f_0$ and $e = e_0$. Moreover, in Steps 1 and 2, we have $a = b = 0$, $s = 0 \oplus 0 \oplus s_0 \oplus 0 \oplus 0 = s_0$, and $z = 0 \oplus z_0 = z_0$. Hence $(f, e, s, z) = (f_0, e_0, s_0, z_0)$, which corresponds to the property $x_0 + x_1 = x_0$.

From now, we consider the other case where $x_0 \neq 0$ and $x_1 \neq 0$, i.e., $z_0 = z_1 = 0$. Then by Step 2, we have $z = 1$ if and only if $b = b' = s_2 = 1$, i.e., $(e_0, f_0) = (e_1, f_1)$ and $s_0 \neq s_1$. The latter means that x_0 and x_1 have the same absolute value and different signs, which is equivalent to $x_0 + x_1 = 0$. Hence the value of z is correct.

During Steps 1 and 2, the algorithm swaps the inputs (x_0, x_1) to $(x_{\max}, x_{\min})$ so that they are ordered by magnitude, and computes the corresponding significands and exponents, denoted by $f_{\max}, f_{\min}, e_{\max}, e_{\min}$, respectively. These steps also compute the sign bit s for the output of this protocol. The algorithm also computes $e_\Delta = e_{\max} - e_{\min}$ and $s_2 = s_0 \oplus s_1$. Here s_2 determines whether to perform addition or subtraction of the significands; if $s_2 = 0$ (the signs of inputs are the same), the values are being added; otherwise they are being subtracted. We explain the correctness of the behaviors of Steps 1 and 2 by case-by-case analysis:

- If $e_0 < e_1$ (i.e., $a = 1$ and $b = 0$), then (regardless of the values of c and b') we have $e_{\max} = e_1$, $e_{\min} = e_0$, $f_{\max} = f_1$, $f_{\min} = f_0$, and $s = s_1$. This corresponds to the property that the absolute value of x_1 is dominant among the two inputs.
- If $e_0 > e_1$ (i.e., $a = b = 0$), then (regardless of the values of c and b') we have $e_{\max} = e_0$, $e_{\min} = e_1$, $f_{\max} = f_0$, $f_{\min} = f_1$, and $s = s_0$. This corresponds to the property that the absolute value of x_0 is dominant among the two inputs.
- If $e_0 = e_1$ (i.e., $a = 0$ and $b = 1$), then we have $e_{\max} = e_0 = e_1 = e_{\min}$,

  $$f_{\max} = c \cdot f_1 + \overline{c} \cdot f_0\,,\ f_{\min} = c \cdot f_0 + \overline{c} \cdot f_1\ .$$

  Hence we have $(e_{\max}, f_{\max}, e_{\min}, f_{\min}) = (e_1, f_1, e_0, f_0)$ if $f_0 < f_1$ (i.e., $c = 1$), and $(e_{\max}, f_{\max}, e_{\min}, f_{\min}) = (e_0, f_0, e_1, f_1)$ if $f_1 \geq f_0$ (i.e., $c = 0$). This is the desired behavior. Moreover:
  - If $f_0 < f_1$, i.e., the absolute value of x_1 is dominant, then we have $c = 1$ and $b' = 0$, therefore

Algorithm 1 FLADDtrunc

Functionality: $\langle x\rangle \leftarrow$ FLADDtrunc($\langle x_0\rangle, \langle x_1\rangle$)
Ensure: $\langle x\rangle$, where $x = x_0 + x_1$.
1: P_i ($i \in \{0,1\}$) parallelly compute ▷ 3
$\langle a\rangle^{\mathsf{B}} \leftarrow$ Comparison($\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}}$),
$\langle b\rangle^{\mathsf{B}} \leftarrow$ Equality($\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}}$),
$\langle c\rangle^{\mathsf{B}} \leftarrow$ Comparison($\langle f_0\rangle^{\mathsf{A}}, \langle f_1\rangle^{\mathsf{A}}$),
$\langle b'\rangle^{\mathsf{B}} \leftarrow$ Equality($\langle f_0\rangle^{\mathsf{A}}, \langle f_1\rangle^{\mathsf{A}}$),
and then locally compute
$\langle s_2\rangle^{\mathsf{B}} \leftarrow \langle s_0\rangle^{\mathsf{B}} \oplus \langle s_1\rangle^{\mathsf{B}}$.
2: P_i parallelly compute ▷ 1
$\langle e_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}}$,
$\langle e_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}}$,
$\langle f_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}}$,
$\langle f_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}}$,
$\langle s\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\,\overline{\langle b'\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle b'\rangle^{\mathsf{B}}\overline{\langle s_2\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}}$,
$\langle z\rangle^{\mathsf{B}} \leftarrow \langle b\rangle^{\mathsf{B}}\langle b'\rangle^{\mathsf{B}}\langle s_2\rangle^{\mathsf{B}} \oplus \langle z_0\rangle^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}}$,
and then locally compute
$\langle e_\Delta\rangle^{\mathsf{A}} \leftarrow \langle e_{\max}\rangle^{\mathsf{A}} - \langle e_{\min}\rangle^{\mathsf{A}}$.
3: For $j = 0, 1, \cdots, l$, P_i parallelly compute ▷ 3
$\langle t[j]\rangle^{\mathsf{B}} \leftarrow$ Equality($\langle e_\Delta\rangle^{\mathsf{A}}, j$),
$\langle t'[j]\rangle^{\mathsf{A}} \leftarrow$ Rightshift($2\langle f_{\min}\rangle^{\mathsf{A}}, j$),
$\langle t''[j]\rangle^{\mathsf{A}} \leftarrow$ B2A($\neg$Modeq($2\langle f_{\min}\rangle^{\mathsf{A}}, j$)),
and then locally compute
$\langle c_1\rangle^{\mathsf{B}} \leftarrow \neg\bigoplus_{j=0}^{l}\langle t[j]\rangle^{\mathsf{B}}$.
4: P_i parallelly compute ▷ 1
$\langle f'_{\min}\rangle^{\mathsf{A}} \leftarrow (1 - 2\langle s_2\rangle^{\mathsf{B}})\sum_{j=0}^{l}\langle t[j]\rangle^{\mathsf{B}}\langle t'[j]\rangle^{\mathsf{A}}$,
$\langle \delta\rangle^{\mathsf{A}} \leftarrow -\langle s_2\rangle^{\mathsf{B}}(\langle c_1\rangle^{\mathsf{B}} + \sum_{j=0}^{l}\langle t[j]\rangle^{\mathsf{B}}\langle t''[j]\rangle^{\mathsf{A}})$,
and then locally compute
$\langle f_2\rangle^{\mathsf{A}} \leftarrow 2\langle f_{\max}\rangle^{\mathsf{A}} + \langle f'_{\min}\rangle^{\mathsf{A}} + \langle \delta\rangle^{\mathsf{A}}$.
5: P_i compute ▷ 4
$\langle d\rangle^{\mathsf{B}} \leftarrow$ Bitdec($\langle f_2\rangle^{\mathsf{A}}$),
and then compute
$\langle d'\rangle^{\mathsf{B}} \leftarrow$ MSNZB($\langle d\rangle^{\mathsf{B}}$).
Parallelly, P_i compute
$\langle f[j]\rangle^{\mathsf{A}} \leftarrow 2^{l-j-1}\langle f_2\rangle^{\mathsf{A}}$ for $j \in \{0, \cdots, l-1\}$,
$\langle f[j]\rangle^{\mathsf{A}} \leftarrow$ Rightshift($\langle f_2\rangle^{\mathsf{A}}, j - l + 1$) for $j \in \{l, l+1\}$,
$\langle z_2\rangle^{\mathsf{B}} \leftarrow \overline{\langle z_0\rangle}^{\mathsf{B}}\overline{\langle z_1\rangle}^{\mathsf{B}}$.
$\langle z_3\rangle^{\mathsf{B}} \leftarrow \overline{\langle z\rangle}^{\mathsf{B}}\overline{\langle z_0\rangle}^{\mathsf{B}}\overline{\langle z_1\rangle}^{\mathsf{B}}$.
6: P_i parallelly compute ▷ 1
$\langle f\rangle^{\mathsf{A}} \leftarrow \langle z_2\rangle^{\mathsf{B}}\sum_{j=0}^{l+1}\langle d'[j]\rangle^{\mathsf{B}}\langle f[j]\rangle^{\mathsf{A}} + \langle z_1\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle z_0\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}}$,
$\langle e\rangle^{\mathsf{A}} \leftarrow \langle z_3\rangle^{\mathsf{B}}(\langle e_{\max}\rangle^{\mathsf{A}} + \sum_{j=0}^{l+1}(j-l)\langle d'[j]\rangle^{\mathsf{B}}) + \langle z_1\rangle^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}} + \langle z_0\rangle^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}}$.
7: **return** ($\langle f\rangle^{\mathsf{A}}, \langle e\rangle^{\mathsf{A}}, \langle s\rangle^{\mathsf{B}}, \langle z\rangle^{\mathsf{B}}$).

$s = 0 \oplus s_1 \oplus 0 \oplus 0 \oplus 0 = s_1$, as desired.

- If $f_0 > f_1$, i.e., the absolute value of x_0 is dominant, then we have $c = b' = 0$, therefore $s = 0 \oplus 0 \oplus 0 \oplus s_0 \oplus 0 = s_0$, as desired.
- If $f_0 = f_1$, i.e., $|x_0| = |x_1|$, then we have $c = 0$ and $b' = 1$, therefore

$$s = 0 \oplus 0 \oplus 0 \oplus 0 \oplus \overline{s_2} \cdot s_0 = \overline{s_2} \cdot s_0 \ .$$

If $s_0 = s_1$ (i.e., $x_0 = x_1$), then we have $s_2 = 0$ and $s = s_0$, which corresponds to the property $x_0 + x_1 = 2x_0$. On the other hand, if $s_0 \neq s_1$ (i.e., $x_0 = -x_1$), then we have $s_2 = 1$ and $s = 0$, which corresponds to the property $x_0 + x_1 = 0$.

Therefore Steps 1 and 2 behave as described above.

In Steps 3 and 4, we implement the addition/subtraction of the significands (depending on either $s_2 = 0$ or $s_2 = 1$); we compute $f_2 = 2f_{\max} \pm (2f_{\min} \gg e_\Delta) + \delta$. In the addition/subtraction of the significands, we execute 1-bit left-shift for both of the significands (see below for the reason of the left-shift) and then execute right-shift alignment using e_Δ. δ is a correction term; $\delta = -1$ if subtraction is performed and there is any bit 1 in the least e_Δ bits of $2f_{\min}$ (in this case, $2f_{\max} - (2f_{\min} \gg e_\Delta)$ is the rounding up of the exact value and hence it should be further subtracted by one to be rounded down), and $\delta = 0$ otherwise. By this method, we can get the intermediate result f_2 needed for accurate rounding in Truncation mode. Note that f_2 is at most $(l+2)$-bit integer.

Note for the left-shift: For addition, or subtraction with $e_\Delta = 0$, no accuracy loss occurs here. We consider subtraction with $e_\Delta > 0$. If $e_\Delta \geq 2$, then at most 1-bit loss of accuracy may happen, and the 1-bit left-shift is sufficient for preventing the accuracy loss. On the other hand, if $e_\Delta = 1$,

Algorithm 2 FLADDeven

Functionality: $\langle x\rangle \leftarrow \mathsf{FLADDeven}(\langle x_0\rangle, \langle x_1\rangle)$

Ensure: $\langle x\rangle$, where $x = x_0 + x_1$.

1: P_i $(i \in \{0,1\})$ parallelly compute ▷ 3

$\langle a\rangle^{\mathsf{B}} \leftarrow \mathsf{Comparison}(\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}})$,
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}})$,
$\langle c\rangle^{\mathsf{B}} \leftarrow \mathsf{Comparison}(\langle f_0\rangle^{\mathsf{A}}, \langle f_1\rangle^{\mathsf{A}})$,
$\langle b'\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle f_0\rangle^{\mathsf{A}}, \langle f_1\rangle^{\mathsf{A}})$,

and locally compute

$\langle s_2\rangle^{\mathsf{B}} \leftarrow \langle s_0\rangle^{\mathsf{B}} \oplus \langle s_1\rangle^{\mathsf{B}}$.

2: P_i parallelly compute ▷ 1

$\langle e_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}}$,
$\langle e_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}}$,
$\langle f_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}}$,
$\langle f_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}}$,
$\langle s\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\,\overline{\langle b'\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle b'\rangle^{\mathsf{B}}\overline{\langle s_2\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}}$,
$\langle z\rangle^{\mathsf{B}} \leftarrow \langle b\rangle^{\mathsf{B}}\langle b'\rangle^{\mathsf{B}}\langle s_2\rangle^{\mathsf{B}} \oplus \langle z_0\rangle^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}}$,

and then locally compute

$\langle e_\Delta\rangle^{\mathsf{A}} \leftarrow \langle e_{\max}\rangle^{\mathsf{A}} - \langle e_{\min}\rangle^{\mathsf{A}}$.

3: For $j = 0, 1, \cdots, l+1$, P_i parallelly compute ▷ 3

$\langle t[j]\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle e_\Delta\rangle^{\mathsf{A}}, j)$,
$\langle t'[j]\rangle^{\mathsf{A}} \leftarrow \mathsf{Rightshift}(4\langle f_{\min}\rangle^{\mathsf{A}}, j)$,
$\langle t''[j]\rangle^{\mathsf{A}} \leftarrow \mathsf{B2A}(\neg\mathsf{Modeq}(4\langle f_{\min}\rangle^{\mathsf{A}}, j))$,

and then locally compute

$\langle c_1\rangle^{\mathsf{B}} \leftarrow \neg\bigoplus_{j=0}^{l+1}\langle t[j]\rangle^{\mathsf{B}}$.

4: P_i parallelly compute ▷ 1

$\langle f'_{\min}\rangle^{\mathsf{A}} \leftarrow (1 - 2\langle s_2\rangle^{\mathsf{B}})\sum_{j=0}^{l+1}\langle t[j]\rangle^{\mathsf{B}}\langle t'[j]\rangle^{\mathsf{A}}$,
$\langle \delta\rangle^{\mathsf{A}} \leftarrow (1 - 2\langle s_2\rangle^{\mathsf{B}})(\langle c_1\rangle^{\mathsf{B}} + \sum_{j=0}^{l+1}\langle t[j]\rangle^{\mathsf{B}}\langle t''[j]\rangle^{\mathsf{A}})$,

and then locally compute

$\langle f_2\rangle^{\mathsf{A}} \leftarrow 8\langle f_{\max}\rangle^{\mathsf{A}} + 2\langle f'_{\min}\rangle^{\mathsf{A}} + \langle \delta\rangle^{\mathsf{A}}$.

5: P_i parallelly compute ▷ 6

$\langle d\rangle^{\mathsf{B}} \leftarrow \mathsf{Bitdec}(\langle f_2\rangle^{\mathsf{A}})$, $\quad\langle S\rangle^{\mathsf{B}} \leftarrow \neg\mathsf{Modeq}(\langle f_2\rangle^{\mathsf{A}}, 2)$,

and then compute

$\langle d'\rangle^{\mathsf{B}} \leftarrow \mathsf{MSNZB}(\langle d\rangle^{\mathsf{B}})$,
$\langle r[l+3]\rangle^{\mathsf{A}} \leftarrow \langle d[3]\rangle^{\mathsf{B}}\langle d[2]\rangle^{\mathsf{B}} \oplus \langle d[3]\rangle^{\mathsf{B}}\overline{\langle d[2]\rangle}^{\mathsf{B}}\langle S\rangle^{\mathsf{B}} \oplus \langle d[4]\rangle^{\mathsf{B}}\langle d[3]\rangle^{\mathsf{B}}\overline{\langle d[2]\rangle}^{\mathsf{B}}\,\overline{\langle S\rangle}^{\mathsf{B}}$,
$\langle r[l+2]\rangle^{\mathsf{A}} \leftarrow \langle d[2]\rangle^{\mathsf{B}}\langle d[1]\rangle^{\mathsf{B}} \oplus \langle d[2]\rangle^{\mathsf{B}}\overline{\langle d[1]\rangle}^{\mathsf{B}}\langle d[0]\rangle^{\mathsf{B}} \oplus \langle d[3]\rangle^{\mathsf{B}}\langle d[2]\rangle^{\mathsf{B}}\overline{\langle d[1]\rangle}^{\mathsf{B}}\,\overline{\langle d[0]\rangle}^{\mathsf{B}}$,
$\langle r[l+1]\rangle^{\mathsf{A}} \leftarrow \langle d[1]\rangle^{\mathsf{B}}\langle d[0]\rangle^{\mathsf{B}} \oplus \langle d[2]\rangle^{\mathsf{B}}\langle d[1]\rangle^{\mathsf{B}}\overline{\langle d[0]\rangle}^{\mathsf{B}}$.

Parallelly, P_i compute

$\langle f[j]\rangle^{\mathsf{A}} \leftarrow 2^{l-j-1}\langle f_2\rangle^{\mathsf{A}}$ for $j \in \{2, \cdots, l-1\}$,
$\langle f[j]\rangle^{\mathsf{A}} \leftarrow \mathsf{Rightshift}(\langle f_2\rangle^{\mathsf{A}}, j-l+1)$ for $j \in \{l, \cdots, l+3\}$,

then locally compute

$\langle f[j]\rangle^{\mathsf{A}} \leftarrow \langle f[j]\rangle^{\mathsf{A}} + \langle r[j]\rangle^{\mathsf{A}}$ for $j \in \{l+1, l+2, l+3\}$,

then compute

$\langle \mathrm{of}[j]\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle f[j]\rangle^{\mathsf{A}}, 2^l)$ for $j \in \{l+1, l+2, l+3\}$.

6: P_i parallelly compute ▷ 1

$\langle f\rangle^{\mathsf{A}} \leftarrow \sum_{j=2}^{l}\langle d'[j]\rangle^{\mathsf{B}}\langle f[j]\rangle^{\mathsf{A}} + \sum_{j=l+1}^{l+3}\langle d'[j]\rangle^{\mathsf{B}}(\overline{\langle \mathrm{of}[j]\rangle}^{\mathsf{B}}\langle f[j]\rangle^{\mathsf{A}} + \langle \mathrm{of}[j]\rangle^{\mathsf{B}}2^{l-1})$,
$\langle e\rangle^{\mathsf{A}} \leftarrow \overline{\langle z\rangle}^{\mathsf{B}}\langle e_{\max}\rangle^{\mathsf{A}} + \sum_{j=2}^{l}(j-l-2)\langle d'[j]\rangle^{\mathsf{B}} + \sum_{j=l+1}^{l+3}(j-l-2+\langle \mathrm{of}[j]\rangle^{\mathsf{B}})\langle d'[j]\rangle^{\mathsf{B}}$.

7: **return** $(\langle f\rangle^{\mathsf{A}}, \langle e\rangle^{\mathsf{A}}, \langle s\rangle^{\mathsf{B}}, \langle z\rangle^{\mathsf{B}})$.

then the 1-bit left-shift can keep all the bits of $f_{\min}$ even after the e_Δ (= 1)-bit right-shift, therefore no accuracy loss happens as well. Hence the accuracy is kept in any case.

In Steps 5 and 6, we normalize the significand. We compute the most significant non-zero bit (MSNZB) of f_2 and rounds f_2 to l-bit integer by either right-shift (when MSNZB is at the l-th or $(l+1)$-th least bit) or left-shift (otherwise). We also adjust the exponent of $x_{\max}$ to obtain the resulting exponent according to the position of MSNZB. This gives the correct output. □

4.3 Addition/Subtraction in Round-to-Even Mode

Here we show our proposed floating-point addition protocol in Round-to-Even mode (Algorithm 2). The notations and conventions are the same as the case of Truncation mode, and the total number of communication rounds is 15 for both binary32 and binary64 floating-point numbers.

Theorem 2. *Algorithm 2 correctly computes the functionality of floating-point addition in Round-to-Even mode.*

Proof. We note that Steps 1 and 2 of Algorithm 2 are ex-

actly the same as Steps 1 and 2 of Algorithm 1 (Truncation mode), therefore the argument to Algorithm 1 is also applicable. The processes during Steps 3 and 4 are also almost the same as Truncation mode, except that now we perform 3-bit (instead of 1-bit) left-shift to keep the information on the guard bit, the round bit, and the sticky bit (see Sect. 2.2.3 for the terminology). More precisely, $f_{\min}$ is 2-bit left-shifted first to keep the guard bit and the round bit, and then 1-bit left-shifted later to reserve the space for the sticky bit. On the other hand, the correction bit δ now plays the role of the sticky bit for $f_{\min}$. Note that now the intermediate significand f_2 is at most $(l+4)$-bit integer.

In Step 5, which is the main modification from the case of Truncation mode, we decide whether f_2 has to be rounded up or rounded down. We divide the argument into the following cases depending on the place of the MSNZB of f_2.

- When MSNZB is the $(l+3)$-th least bit: now the guard bit G is $d[3]$, the round bit R is $d[2]$, and the sticky bit is $S = d[1] \vee d[0]$. If $G = 0$, then it should be rounded down, which is represented by setting $r[l+3] = 0$. From now, we consider the other case $G = 1$. If $R = 1$ or $S = 1$, then it should be rounded up, which is represented by setting $r[l+3] = 1$. In the other case where $R = S = 0$, it should be rounded up if and only if $d[4] = 1$ due to the rounding rule in Round-to-Even mode. This is correctly expressed by the formula of $r[l+3]$ in the algorithm.
- When MSNZB is the $(l+2)$-th or $(l+1)$-th least bit: now the argument is similar to the previous case; in fact, the case of $(l+1)$-th least bit is even simpler because now the sticky bit is always zero.
- Otherwise: by the same argument as the case of Truncation mode, such a large move of the place of MSNZB (which was originally at the $(l+3)$-th least bit) can occur only when $e_\Delta \le 1$. In such a case, the least two bits $d[1]$ and $d[0]$ are always zero due to the 3-bit left-shift, therefore all the necessary information is involved in the bits $d[l], d[l-1], \ldots, d[2]$ and only the normalization (without any rounding) is sufficient.

These arguments show that the direction of rounding is correctly chosen in the algorithm. However, we have another possibility to concern, which was not necessary in Truncation mode. That is, in the new case of rounding up, the bit length of the significand may be increased from l to $l+1$. This happens only in the case $11\cdots1 \mapsto 100\cdots0$; in the algorithm, this possibility is checked by using the bits "of$[j]$", and the resulting output is adjusted according to the values of of$[j]$'s. This gives the correct output. □

5. Experimental Results

We implemented our proposed protocols and performed experiments for the addition of two floating-point numbers. The results are summarized in Table 3.

In our experiments, the computation (Comp.) times (sec) assuming the client-aided model and the communication (Comm.) sizes (bit) for each party were measured for the offline and the online phases separately; the offline communication means communication sending the pre-generated BTs/BTEs from the client to each computing party. The computation times shown in the table are averages of 10 trials for each parameter setting. To calculate the data transfer (trans.) time (msec) and the communication latency in total for the online phase, we did not use a real network and, instead, adopted the following simulated WAN environment setting which is the same as [20]: 10 MB/sec (= 80000 bit/msec) bandwidth and 40 ,msec round-trip time (RTT) latency. Our protocols were implemented on a single laptop computer

Table 3 Experimental results on our proposed 2PC protocols for floating-point addition over simulated WAN with 10 MB/s bandwidth and 40 ms RTT latency.

	Offline		Online				
Batch size	Comp. time (s)	Comm. size (bit)	Comp. time (s)	Comm. size (bit)	Data trans. time (ms)	Comm. latency (s)	Total time (s)
Truncation mode (binary32)							
1	0.163	352, 565	0.065	74, 373	0.930	0.520	0.586
10	1.538	3, 525, 650	0.079	743, 730	9.297		0.608
100	13.697	35, 256, 500	0.175	7, 437, 300	92.966		0.788
1000	145.553	352, 565, 000	1.758	74, 373, 000	929.663		3.208
Truncation mode (binary64)							
1	1.075	2, 506, 416	0.131	324, 617	4.058	0.520	0.655
10	15.739	25, 064, 160	0.223	3, 246, 170	40.577		0.784
100	160.971	250, 641, 600	1.405	32, 461, 700	405.771		2.331
Round-to-Even mode (binary32)							
1	0.185	298, 923	0.077	64, 095	0.801	0.600	0.678
10	1.639	2, 989, 230	0.089	640, 950	8.012		0.697
100	14.408	29, 892, 300	0.177	6, 409, 500	80.119		0.857
1000	141.476	298, 923, 000	1.526	64, 095, 000	801.188		2.927
Round-to-Even mode (binary64)							
1	1.718	2, 265, 310	0.135	270, 835	3.385	0.600	0.738
10	16.038	22, 653, 100	0.227	2, 708, 350	33.854		0.861
100	162.959	226, 531, 000	1.329	27, 083, 500	338.544		2.268

with Intel Core i7-820HQ 2.9 GHz and 16.0 GB RAM, and with Python3.7 and Numpy v1.16.3.

For inputs with binary32 formats, we performed the experiments for the cases of 1/10/100/1000 batches; the computation and the data transfer times grow as the batch numbers increase, while the communication latency depends only on the number of communication rounds (13 rounds) and is independent of the batch numbers. For the case of binary64 inputs (with 15 rounds), we performed the experiments similarly for the cases of 1/10/100 batches; the larger batches could not be performed due to memory error.

Our experimental results show that in the online phase, the communication latency is dominant among the total execution time especially for small batches (e.g., ≤ 100 batches for binary32 and ≤ 10 batches for binary64). In such cases, our improvement in reducing the communication rounds in this paper has given significant effects to reduce the total online execution time. On the other hand, for the offline phase, the computation time might be reducible by just improving the implementation, while improvements of protocol designs are needed to reduce the communication size further, which is an important future research topic.

6. Application: Error-Free Transformation

Here we show a privacy-preserving error-free transformation protocol for the sum of two floating-point numbers; that is, we transform any pair of floating-point numbers (a, b) into a new pair (x, y) with $x = \mathrm{fl}(a + b)$ and $x + y = a + b$. Here $\mathrm{fl}(\cdot)$ denotes that the expression inside the parenthesis is calculated in floating-point. According to Dekker's algorithm in [24], we can construct privacy-preserving error-free transformation with 45 communication rounds as in Algorithm 3 using our protocol FLADDeven. In this algorithm, we use swap (Algorithm 4) as a subprotocol so that the input is ordered by magnitude. We performed experiments for inputs with binary32 and binary64 where the computer and the (simulated) network environments are the same as

Algorithm 3 FastTwoSum

Functionality: $(\langle x\rangle, \langle y\rangle) \leftarrow \mathsf{FastTwoSum}(\langle a\rangle, \langle b\rangle)$
Ensure: $x = \mathrm{fl}(a + b)$ and $x + y = a + b$.
1: P_i $(i \in \{0, 1\})$ parallelly compute ▷ 15
$\langle x\rangle \leftarrow \mathsf{FLADDeven}(\langle a\rangle, \langle b\rangle)$,
$(\langle a'\rangle, \langle b'\rangle) \leftarrow \mathsf{swap}(\langle a\rangle, \langle b\rangle)$.
2: P_i compute $\langle q\rangle \leftarrow \mathsf{FLADDeven}(\langle x\rangle, -\langle a'\rangle)$. ▷ 15
3: P_i compute $\langle y\rangle \leftarrow \mathsf{FLADDeven}(\langle b'\rangle, -\langle q\rangle)$. ▷ 15
4: **return** $(\langle x\rangle, \langle y\rangle)$.

Algorithm 4 swap

Functionality: $(\langle y_0\rangle, \langle y_1\rangle) \leftarrow \mathsf{swap}(\langle x_0\rangle, \langle x_1\rangle)$
Ensure: $\{y_0, y_1\} = \{x_0, x_1\}$ and $|y_0| \geq |y_1|$.
1: P_i $(i \in \{0, 1\})$ parallelly compute ▷ 3
$\langle a\rangle^{\mathsf{B}} \leftarrow \mathsf{Comparison}(\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}})$,
$\langle b\rangle^{\mathsf{B}} \leftarrow \mathsf{Equality}(\langle e_0\rangle^{\mathsf{A}}, \langle e_1\rangle^{\mathsf{A}})$,
$\langle c\rangle^{\mathsf{B}} \leftarrow \mathsf{Comparison}(\langle f_0\rangle^{\mathsf{A}}, \langle f_1\rangle^{\mathsf{A}})$.
2: P_i parallelly compute ▷ 1
$\langle e_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}}$,
$\langle e_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle e_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\langle e_1\rangle^{\mathsf{A}}$,
$\langle f_{\max}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}}$,
$\langle f_{\min}\rangle^{\mathsf{A}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle f_0\rangle^{\mathsf{A}} + \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}} + \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle f_1\rangle^{\mathsf{A}}$,
$\langle s_{\max}\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}}$,
$\langle s_{\min}\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle s_0\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle s_1\rangle^{\mathsf{B}}$,
$\langle z_{\max}\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle z_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle z_0\rangle^{\mathsf{B}}$,
$\langle z_{\min}\rangle^{\mathsf{B}} \leftarrow \langle a\rangle^{\mathsf{B}}\langle z_0\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\langle c\rangle^{\mathsf{B}}\langle z_0\rangle^{\mathsf{B}} \oplus \overline{\langle a\rangle}^{\mathsf{B}}\,\overline{\langle b\rangle}^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}} \oplus \langle b\rangle^{\mathsf{B}}\overline{\langle c\rangle}^{\mathsf{B}}\langle z_1\rangle^{\mathsf{B}}$.
3: **return** $(\langle f_{\max}\rangle^{\mathsf{A}}, \langle e_{\max}\rangle^{\mathsf{A}}, \langle s_{\max}\rangle^{\mathsf{B}}, \langle z_{\max}\rangle^{\mathsf{B}}), (\langle f_{\min}\rangle^{\mathsf{A}}, \langle e_{\min}\rangle^{\mathsf{A}}, \langle s_{\min}\rangle^{\mathsf{B}}, \langle z_{\min}\rangle^{\mathsf{B}})$.

Table 4 Experimental results on our proposed 2PC protocols for error-free transformation over simulated WAN with 10 MB/s bandwidth and 40 ms RTT latency.

Batch size	Offline		Online				
	Comp. time (s)	Comm. size (bit)	Comp. time (s)	Comm. size (bit)	Data trans. time (ms)	Comm. latency (s)	Total time (s)
binary32							
1	0.536	922, 170	0.233	199, 743	2.497	1.800	2.035
10	4.259	9, 221, 700	0.242	1, 997, 430	24.968		2.067
100	40.848	92, 217, 000	0.538	19, 974, 300	249.679		2.588
1000	427.950	922, 170, 000	4.498	199, 743, 000	2396.788		8.795
binary64							
1	4.916	6, 867, 189	0.426	823, 005	10.288	1.800	2.236
10	46.952	68, 671, 890	0.715	8, 230, 050	102.876		2.618
100	475.751	686, 718, 900	3.984	82, 300, 500	1028.756		6.813

Sect. 5. The experimental results are as in Table 4. Our protocol FLADDeven rounds the precise result to the nearest floating-point number, so we can obtain accurate result via Algorithm 3.

Acknowledgments

This work was partly supported by the Ministry of Internal Affairs and Communications SCOPE Grant Number 182103105 and by JST CREST JPMJCR19F6. The authors thank Satsuya Ohata for his implemented library of basic protocols proposed in [20]. This work was done when the first author was an undergraduate student at Department of Mathematical Engineering and Information Physics, School of Engineering, The University of Tokyo.

References

[1] M. Aliasgari and M. Blanton, "Secure computation of hidden Markov models," Proc. SECRYPT 2013, pp.242–253, 2013
[2] M. Aliasgari, M. Blanton, and F. Bayatbabolghani, "Secure computation of hidden Markov models and secure floating-point arithmetic in the malicious model," Int. J. Inf. Secur., vol.16, no.6, pp.577–601, 2017.
[3] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, "Secure computation on floating point numbers," NDSS 2013, San Diego, California, USA, Feb. 2013.
[4] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida, "Generalizing the SPDZ compiler for other protocols," Proc. ACM CCS 2018, pp.880–895, 2018.
[5] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," Proc. ACM CCS 2016, pp.805–817, 2016.
[6] E. Boyle, N. Gilboa, and Y. Ishai, "Secure computation with preprocessing via function secret sharing," Proc. TCC 2019 (Part I), LNCS vol.11891, pp.341–371, 2019.
[7] O. Catrina, "Towards practical secure computation with floating-point numbers," Proc. BalkanCryptSec 2018, 2018.
[8] O. Catrina, "Efficient secure floating-point arithmetic using shamir secret sharing," Proc. SECRYPT 2019, pp.49–60, 2019.
[9] O. Catrina, "Optimization and tradeoffs in secure floating point computation: Products, powers, and polynomials," Proc. 6th Conference on the Engineering of Computer Based Systems (ECBS 2019), pp.7:1–7:10, 2019.
[10] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Automated synthesis of optimized circuits for secure computation," Proc. ACM CCS 2015, pp.1504–1517, 2015.
[11] D. Demmler, T. Schneider, and M. Zohner, "ABY — A framework for efficient mixed-protocol secure two-party computation," NDSS 2015, San Diego, California, USA, Feb. 2015.
[12] F. Eigner, M. Maffei, I. Pryvalov, F. Pampaloni, and A. Kate, "Differentially private data aggregation with optimal utility," Proc. ACSAC 2014, ACM, pp.316–325, 2014.
[13] O. Goldreich, Foundations of Cryptography, Volume II, Cambridge Univ. Press, 2004.
[14] B. Hemenway, S. Lu, R. Ostrovsky, and W. Welser, IV, "High-precision secure computation of satellite collision probabilities," Proc. SCN 2016, LNCS vol.9841, pp.169–187, 2016.
[15] 754-2019 - IEEE Standard for Floating-Point Arithmetic, 2019.
[16] L. Kamm and J. Willemson, "Secure floating point arithmetic and private satellite collision analysis," Int. J. Inf. Secur., vol.14, no.6, pp.531–548, 2015.
[17] Y.-C. Liu, Y.-T. Chiang, T.-S. Hsu, C.-J. Liau, and D.-W. Wang, "Floating point arithmetic protocols for constructing secure data analysis application," Proc. KES 2013, Procedia Computer Science, vol.22, pp.152–161, 2013.
[18] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," Proc. IEEE S&P 2017, pp.19–38, 2017.
[19] H. Morita, N. Attrapadung, T. Teruya, S. Ohata, K. Nuida, and G. Hanaoka, "Constant-round client-aided secure comparison protocol," Proc. ESORICS 2018 (Part II), LNCS vol.11099, pp.395–415, 2018.
[20] S. Ohata and K. Nuida, "Communication-efficient (client-aided) secure two-party protocols and its application," Proc. Financial Cryptography and Data Security (FC) 2020, LNCS vol.12059, pp.369–385, 2020 (also available at https://arxiv.org/abs/1907.03415v2)
[21] W. Omori and A. Kanaoka, "Efficient secure arithmetic on floating point numbers," Proc. The 20th International Conference on Network-Based Information Systems (NBiS 2017), pp.924–934, 2018.
[22] P. Pullonen and S. Siim, "Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations," Proc. Financial Cryptography and Data Security (FC) 2015, LNCS vol.8976, pp.172–183, 2015.
[23] M.G. Raeini and M. Nojoumian, "Secure trust evaluation using multi-path and referral chain methods," Proc. 15th International Workshop on Security and Trust Management (STM 2019), LNCS vol.11738, pp.124–139, 2019.
[24] S.M. Rump, T. Ogita, and S. Oishi, "Accurate floating-point summation," Technical Report 05.12, Faculty for Information- and Communication Sciences, Hamburg University of Technology, Nov. 2005.
[25] K. Sasaki and K. Nuida, "Efficiency and accuracy improvements of secure floating-point addition over secret sharing," Proc. IWSEC 2020, LNCS vol.12231, pp.77–94, 2020.
[26] Y. Zhang, A. Steele, and M. Blanton, "PICCO: A general-purpose compiler for private distributed computation," Proc. ACM CCS 2013, pp.813–826, 2013.

Kota Sasaki received the Bachelor's degree at The University of Tokyo in 2020. He is currently a Master's student in Graduate School of Information Science and Technology, The University of Tokyo. His research interest is mainly in machine learning.

Koji Nuida received the Ph.D. degree in Mathematical Science from The University of Tokyo, Japan, in 2006. Currently, he is mainly working as a professor at Institute of Mathematics for Industry (IMI), Kyushu University, Japan. His research interest is mainly in mathematics and mathematical cryptography.