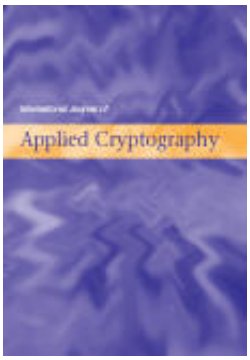
INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research



International Journal of Applied Cryptography

ISSN online: 1753-0571 - ISSN print: 1753-0563
https://www.inderscience.com/ijact

Secure addition of floating points

Thijs Veugen, Robert Wezeman, Alessandro Amadori, Sven Bootsma, Bart Kamphorst

DOI: 10.1504/IJACT.2025.10070533

Article History:

Received:	07 September 2024
Last revised:	10 January 2025
Accepted:	21 February 2025
Published online:	15 April 2025

Secure addition of floating points

Thijs Veugen*

Unit ICT, Strategy and Policy,
TNO,
The Hague, The Netherlands
and
Department of Semantics, Cybersecurity and Services,
University of Twente,
Enschede, The Netherlands
Email: thijs.veugen@tno.nl
*Corresponding author

Robert Wezeman

Unit ICT, Strategy and Policy,
TNO,
Groningen, The Netherlands
Email: robert.wezeman@tno.nl

Alessandro Amadori

Unit ICT, Strategy and Policy,
TNO,
Eindhoven, The Netherlands
Email: alessandro.amadori@tno.nl

Sven Bootsma

Unit ICT, Strategy and Policy,
TNO,
Groningen, The Netherlands
Email: sven.bootsma@tno.nl

Bart Kamphorst

Unit ICT, Strategy and Policy,
TNO,
The Hague, The Netherlands
Email: bart.kamphorst@tno.nl

Abstract: Secure multi-party computation (MPC) and homomorphic encryption are very powerful tools to compute with secret numbers without revealing inputs or any intermediate values. To securely achieve high accuracy with varying number sizes, one needs to work with floating points in the secret (secret-shared or encrypted) domain. The main bottleneck of secure floating points is addition. We improve its efficiency by designing a protocol for multiple additions, using standard building blocks available in most MPC platforms. The more additions n were combined, the larger the relative gain, up to a factor 13 with n = 1,024. Additionally, we introduce a new protocol for securely computing the bitlength (given upper bound M), the first one with linear time complexity and constant round complexity. It reduces secure multiplications with a factor 4 (for the constant-round solution), or the number of communication rounds with a factor $M/2$ (for the logarithmic-round solution). We evaluate accuracy, execution time and communication complexity of our protocols, and release them open source, such that they can be used to improve the efficiency of secure floating-point arithmetic.

Keywords: secure multi-party computation; floating-point arithmetic; bit length protocol; cryptography; homomorphic encryption.

Reference to this paper should be made as follows: Veugen, T., Wezeman, R., Amadori, A., Bootsma, S. and Kamphorst, B. (2025) 'Secure addition of floating points', *Int. J. Applied Cryptography*, Vol. 5, No. 5, pp.1–11.

Biographical notes: Thijs Veugen received his two MSc, one in Mathematics and one in Computer Science, both Cum Laude, and his PhD in the field of Information Theory, all from Eindhoven University of Technology. Since 1999, he has been researching information security, in particular applied cryptography, at TNO, The Netherlands, where he is Principal Scientist. Since 2022, he is also affiliated as a Full Professor Applied Cryptography with University of Twente, The Netherlands.

Robert Wezeman received his two MSc in Physics and Mathematics (Cum Laude) from the University of Groningen (RUG). Since October 2018, he has been working as a Scientist in the Applied Cryptography and Quantum Algorithms Department at TNO, The Netherlands. His main research focus is on quantum algorithms and their applications in the field of optimisation and machine learning tasks.

Alessandro Amadori received his MSc in Mathematics (Cum Laude) from the University of Trento in 2016. He then obtained his PhD in the field of White-box Cryptography from Eindhoven University of Technology. Since 2021, he has been a Scientist in the Applied Cryptography and Quantum Algorithms group at TNO, The Netherlands.

Sven Bootsma received the BSc and MSc in Mathematics from the University of Groningen, in 2023. From 2023 to now, he is a researcher at TNO working in Applied Cryptography and Quantum Algorithms. His research interests include (applied) cryptography, quantum algorithms and elliptic curves.

Bart Kamphorst obtained his MSc in Applied Mathematics at Leiden University (2013). He received his PhD from Eindhoven University of Technology while exploring the boundaries of Applied Stochastics and Scheduling Theory (2018). Since then, Bart has worked at TNO as a scientist to explore and expand the application of privacy-enhancing technologies.

1 Introduction

Secure multi-party computation (MPC) and homomorphic encryption are very powerful tools to compute with secret numbers without revealing inputs or any intermediate values. However, like all cryptographic systems, they can only deal with integers, whereas applications, e.g., based on machine learning, often require more accurate floating-point arithmetic.

This problem is often tackled by computing with fixed-point arithmetic, i.e., scaled integers. In case both very small and very large numbers need to be handled, very large (scaled) integers are needed to preserve accuracy, which introduces a large overhead during computation, and while communicating between parties.

To achieve high accuracy with regular integer sizes, one needs to work with floating points in the secret (secret-shared or encrypted) domain. This requires a secret significand and a secret exponent (see first paragraph of Section 2) for each secret number, and a set of cryptographic protocols for different floating-point operations that work on these secret pairs.

Since the first secure floating-point protocols by Aliasgari et al. (2012), many similar frameworks have been developed, sometimes as part of more generic MPC frameworks (see Subsection 1.1). The general conclusion is that rounding, comparison and division with secure floating points is cheaper than with secure fixed points, multiplication is slightly more involved, but the main bottleneck is additions [Aliasgari et al., (2012), Table 2]. Therefore, the focus of this paper is to improve the efficiency of secure floating-point additions.

1.1 Related work

Since the initial work by Aliasgari et al. (2012), many similar secure floating-point frameworks have been developed. Some are part of larger generic MPC platforms like Sharemind (Krips and Willemson, 2013; Kamm and Willemson, 2015; Kerik et al., 2014), ABY from Demnler et al. (2015), MPyC by Schoenmakers (2018), SCALE-MAMBA by Abdelraham and Smart (2019) and MP-SPDZ in Keller (2020).

Although the software platform and implementation differ, they all use a similar approach for securely computing $y = x_1 + x_2$, given two floating point numbers $x_i = s_i \cdot 2^{e_i}$, $i = 1, 2$, each with a significand s_i of ℓ bits:

1 Find the input $x_{\max}$ with the largest exponent.
2 Is the difference between the two significant?
  Compute $d = \min\{e_{\max} - e_{\min}, \ell\}$.
3 Compute sum: $s \leftarrow s_{\max} + s_{\min} \cdot 2^{-d}$.
4 Normalise s to an integer s_y of exactly ℓ bits:
  a Compute the bits of s.
  b Use the bits to compute bit size m of s.
  c Set $s_y \leftarrow \lfloor s \cdot 2^{\ell - m} \rfloor$.
5 Set exponent of sum y: $e_y \leftarrow e_{\max} + m - \ell$.

Furthermore, Liu et al. (2013) made basic secure floating-point operations that follow the IEEE-754 standard, and more recently Rathee et al. (2022) optimised for

a two-party setting, and for machine learning (Rathee et al., 2023). Sasaki and Nuida (2020) developed two-party protocols for securely adding floating points within a small number of communication rounds. Belorgey et al. (2023) constructed a platform with an offline precomputation phase and a fast online phase that mitigates overflow while securely computing with floating points.

Of particular interest is Catrina (2020a) who introduced a secure protocol for adding multiple floating points, similar to ours. Later, Blanton et al. (2023) developed a more accurate (but less efficient) version of this.

Krips and Willemson (2013) use a scaled integer as significand, just as Schoenmakers (2018), and optimise specific functions as inverse, square-root, exponential and Gaussian error. Abdelraham and Smart (2019) compare the efficiency of functions like square root, sine, cosine, exponentiation and logarithm with secure fixed-point implementations.

1.2 Our contribution

As addition is shown the bottleneck of secure floating-point operations by Aliasgari et al. (2012), we focus on improving its efficiency. During a floating-point addition, one needs to compute the bit length of the new significand (variable m in step 4 in Subsection 1.1) to properly scale it, which introduces a large overhead. We present a new secure bit length protocol that, unlike previous solutions, does not require bit decomposition (as in step 4a of Subsection 1.1), and reduces complexity from $\mathcal{O}(M \log_2 M)$ to $\mathcal{O}(M)$ secure multiplications with constant round complexity, where M is the maximal number of input bits. We show how to simultaneously compute functions on the bit length without extra secure multiplications.

The new bit length protocol additionally speeds up the computation of many secure floating-point functions that use Padé polynomials, because their inputs need to be scaled to fixed intervals.

Furthermore, we present a protocol that combines several secure floating-point additions in one [we found a similar approach in Catrina (2020a, Protocol 3)], and analyse its accuracy. Because the bit length protocol needs to be invoked only once, this significantly reduces complexity. We show a couple of optimisations that further accelerate the multiple addition protocol, which can be used more generic in secure floating-point arithmetic. In particular, speeding up the exponentiation in step 3 of Subsection 1.1, and the secure division in step 4c.

Summarising the main contributions of this paper:

1 The accuracy analysis of a protocol that combines the addition of multiple floating points (see Subsection 2.3).

2 A new bit length protocol with linear time complexity and constant round complexity (see Subsection 3.1).

3 Speeding up the secure division while adding floating points (see Subsection 3.2).

4 Speeding up the secure exponentiation while adding floating points (see Subsection 3.3).

5 Open source software from TNO (2024), based on MPyC, implementing our secure floating point arithmetic, including multiplication, division, square root and logarithm functions.

Although implemented in MPyC, our improvements are applicable to all secure floating point platforms (see Subsection 4.4).

2 Design

We implement a secure floating point with a secret signed integer for the significand and a secret signed integer for the exponent. More specific, a floating point x is represented as $s_x \cdot 2^{e_x}$, where $2^{\ell-1} \le |s_x| < 2^\ell$ and $|e_x| < 2^k$, such that significand and exponent have bit sizes $\ell+1$ and $k+1$ respectively, and the significand contains a maximal number of significant bits.

Unlike other implementations, we do not use a separate sign bit. We also avoid a zero bit and allow the exception $s_x = 0$, in case $x = 0$.

Although similar protocols could be designed for homomorphic encryption, we restrict ourselves to MPC, and use the MPyC framework by Schoenmakers (2018) for implementations. The significand has type SecInt within MPyC, which creates a Shamir secret sharing modulus with a headroom of κ bits, κ being the statistical security parameter. This is convenient for secure operations that require additive blinding. Similar for the exponent.

We use $\langle . \rangle$ to denote secret-shared values, and the notation $(x < y)$ to denote the binary outcome of the comparison $x < y$. The symbol $\div$ is used for integer division, yielding the integer quotient of the division. We use $\oplus$ for exclusive-or of bits.

Before presenting our multiple addition protocol, we introduce the accuracy requirements of (secure) floating points.

2.1 Accuracy requirements

How to decide whether a floating point with integer significand s_x and integer exponent e_x is an accurate representation of real number x? First, the exponent should be correct, which means that there is a real number s, such that $2^{\ell-1} \le |s| < 2^\ell$ and $s \cdot 2^{e_x} = x$. This means that

$$e_x = \lfloor \log_2 |x| \rfloor - \ell + 1,$$

provided $x \neq 0$. Because we are not able to represent zero consistently, this case needs to be handled separately. In our protocol from Subsection 2.2 we then obtain a zero significand, but it is also possible to introduce a separate zero flag for this case.

Because the exponent is limited to k bits, we might run into overflow ($e_x \ge 2^k$) or underflow ($e_x \le -2^k$) problems, which we conveniently neglect in this paper.

Given the exponent, the significand should be sufficiently accurate, which means that s_x should be the integer closest to $x \cdot 2^{-e_x}$. As we are computing in the encrypted (or secret-shared) domain, where rounding off is expensive compared to truncation, we relax this a little to

$$|s_x - x \cdot 2^{-e_x}| < 1.$$

To show that it is not always easy to meet these accuracy requirements with secure floating points, consider the computation of $y = \log_2 x$, given x. As $y = e_x + \log_2 s_x = e_x + \ell - 1 + \epsilon$, for some $0 \le \epsilon < 1$, it might seem sufficient to find an approximation of ϵ with an absolute accuracy of $2^{-\ell}$ to determine s_y. However, if $x \approx 1$, then $y \approx 0$, so either $\epsilon \approx 0$, or $\epsilon \approx 1$. One can show that $|y| \ge 2^{-\ell}$ (if $x \ne 1$) because s_x is an integer, but that means we need to approximate ϵ with an absolute accuracy of $2^{-2\ell}$ to make sure that s_y will be accurate. This and similar observed errors were also described in Rathee et al. (2022).

2.2 Multiple additions

We describe a protocol for securely adding multiple floating points, with the following specification, where all significands and exponents are secret:

Input: $x_i = s_i \cdot 2^{e_i}$, $1 \le i \le n$, with
$2^{\ell-1} \le |s_i| < 2^\ell$ and $|e_i| < 2^k$

Output: $y = s_y \cdot 2^{e_y}$, with
$2^{\ell-1} \le |s_y| < 2^\ell$ and $y \approx \sum_{i=1}^n x_i$

The output should fulfil the accuracy requirements of the previous subsection. However, the analysis in the next subsection shows that in some exceptional cases, our generic approach will not lead to a perfectly accurate outcome.

Because the largest inputs will determine e_y, it makes sense to compute $e_{\max} = \max\{e_i \mid 1 \le i \le n\}$, and write $y = s \cdot 2^{e_{\max}}$, with

$$s = \sum_{i=1}^n s_i \cdot 2^{e_i - e_{\max}}.$$

We know that $0 \le |s| < n \cdot 2^\ell \le 2^\lambda$, where $\lambda = \ell + \lceil \log_2 n \rceil$, but s is not necessarily integer. Performing n secure divisions $t_i \leftarrow s_i \div 2^{e_{\max} - e_i}$ will be costly, but if we neglect the inputs with small exponents e_i, i.e., $e_i < e_{\max} - \lambda$, we can use the integer $t_i = s_i \cdot 2^{e_i - e_{\max} + \lambda}$ with scaling factor $2^{-\lambda}$.

The final step is to compute $T = \sum_i t_i$ and scale the integer T to an s_y with ℓ significant bits. This leads to the following approach:

1 Find the largest exponent $\langle e_{\max} \rangle$, where
$e_{\max} = \max\{e_i \mid 1 \le i \le n\}$.

2 For $i = 1$ to n do

a Compute $\langle \delta_i \rangle = \langle (e_i \ge e_{\max} - \lambda) \rangle$
{Neglect inputs with $\delta_i = 0$}.

b Compute $\langle t_i \rangle \leftarrow \langle s_i \rangle \cdot \langle 2^{e_i - e_{\max} + \lambda} \rangle \cdot \langle \delta_i \rangle$.

3 Compute $\langle T \rangle \leftarrow \sum_{i=1}^n \langle t_i \rangle$
{Approximation of s with scaling factor $2^{-\lambda}$}.

4 Normalise T to an integer s_y of exactly ℓ bits:

a Compute bit length $\langle m \rangle$ of $\langle |T| \rangle$, $0 \le m \le 2\lambda$.

b Compute integer division $\langle s_y \rangle \leftarrow \langle T \div 2^{m-\ell} \rangle$.

5 Set $\langle e_y \rangle$ to $\langle e_{\max} \rangle - \lambda + (\langle m \rangle - \ell)$.

The multiple additions protocol is designed to achieve both high accuracy and good efficiency. Its accuracy is analysed in the next subsection. In Section 3, we describe efficient implementations of the various steps.

2.3 Accuracy analysis

We will show that $s_y \cdot 2^{e_y}$ is an accurate approximation of $y = \sum_{i=1}^n \delta_i \cdot x_i$. We have

$$\begin{aligned} y &= 2^{e_{\max}} \cdot \sum_{i=1}^n \delta_i \cdot s_i \cdot 2^{e_i - e_{\max}} \\ &= 2^{e_{\max}} \cdot \sum_{i=1}^n t_i \cdot 2^{-\lambda} \\ &= 2^{e_{\max} - \lambda} \cdot T \\ &= 2^{e_y + \ell - m} \cdot T. \end{aligned}$$

Therefore, given that s_y is an accurate computation of the integer division $T \div 2^{m-\ell}$, i.e., $|s_y - \frac{T}{2^{m-\ell}}| < 1$, we conclude that $|s_y - y \cdot 2^{-e_y}| < 1$, because $y \cdot 2^{-e_y} = T \cdot 2^{\ell - m}$, and indeed $s_y \cdot 2^{e_y}$ turns out to be an accurate approximation of $y = \sum_{i=1}^n \delta_i \cdot x_i$.

The δ_i are chosen to exclude small exponents with $e_i < e_{\max} - \lambda$, because then

$$\left| \sum_{i, \delta_i = 0} s_i \cdot 2^{e_i - e_{\max}} \right| < (n-1) \cdot 2^\ell \cdot 2^{-1-\lambda} < \frac{1}{2}.$$

This does not imply that $s_y \cdot 2^{e_y}$ always is an accurate approximation of $y = \sum_{i=1}^n x_i$, although in many cases it will be. For example when $|\sum_{i,\delta_i=1} s_i \cdot 2^{e_i - e_{\max}}| \ge 2^{\ell-1}$, which is the case when all large inputs have the same sign. Another example is the case where the input exponents are close to each other, i.e., all δ_i equal one.

The computation is accurate in the standard case $n = 2$ as well, because either both δ_i are one, or exactly one significand remains (with at least ℓ significant bits).

A typical negative exception is the case $n = 3$ with $|x_2| \ll |x_1|$ and $x_3 = -x_1$, because then $y = x_2$ is much smaller than expected. On the other hand, two standard floating-point additions would also yield the same erroneous result $y = (x_1 + x_2) + x_3 = x_1 + x_3 = 0$. This

case is easily generalised to larger values of n: one group of large numbers that sum to zero, and another group of smaller numbers that will be neglected.

A way to mitigate this inaccuracy (at the obvious cost of efficiency) is to not only compute $y = \sum_{i=1}^{n} \delta_i \cdot x_i$, but also $y' = \sum_{i=1}^{n} (1 - \delta_i) \cdot x_i$ in a similar way, and add those together. Another way would be to increase λ, or use the superaccumulator of Blanton et al. (2023). A general measure to increase accuracy is to add up all positive, and all negative numbers, separately.

2.4 Security

We implemented our protocol with MPyC, which uses Shamir secret sharing, and used standard building blocks from de Hoogh (2012) that are available in the platform. MPyC tolerates a dishonest minority of up to t, $0 \leq t < \eta/2$, parties out of the total number η of parties (Schoenmakers, 2018). Therefore, it automatically follows that our protocols (see Subsection 2.2 and Section 3) are secure in the same security model.

The only exception might be step 2 of the bit length protocol (see Subsection 3.1), where an intermediate value c is revealed. In this step, standard additive blinding is used, which is secure because the random number r has κ (the statistical security parameter) more bits than the secret value x. All other steps consist of standard computations with secret sharings without additionally revealing intermediate results.

Our protocol is described by standard building blocks that are available in most MPC platforms (see Subsection 4.4). When implemented in a different platform than MPyC, however, our protocol would inherit the security model of the other platform.

2.5 Multiple multiplications

We designed an efficient protocol for adding multiple floating points. This is the main bottle neck of secure floating points, because the sum needs to be normalised by means of an expensive bit length protocol.

There are two reasons why combining n multiplications is less attractive:

1 It requires less effort to normalise the product of n significands, because the range of the product is smaller: $2^{n(\ell-1)} \leq |\prod_{i=1}^{n} s_i| < 2^{n\ell}$. The bit length protocol can be adjusted to work with lower bounds.

2 With growing n, the size of the product of the significands increases more than its sum, and computing with larger (secret-shared) numbers is less efficient.

Nevertheless, we designed a similar protocol for multiple multiplications, which is available in our open source library. For $n = 128$ the execution time was reduced by a factor 4.7, compared to factor 10 with the same number of additions (TNO, 2024).

3 Increasing efficiency

We assume standard solutions for the less advanced steps, like secure comparison (step 2a) and secure maximum (step 1). In this section we describe efficient implementations for the complicated steps of the multiple addition protocol. In particular, Subsection 3.1 shows a new way to securely compute the bit length of an integer (step 4a), the most intensive step of the protocol as confirmed by Catrina (2020b). In Subsection 3.2, the integer division from step 4b is worked out to a less intensive computation. And finally, the exponentiations from step 2b are computed in an alternative, more efficient way in Subsection 3.3.

By avoiding secure bit decomposition in all steps, we reduced the multiple addition protocol to linear complexity (with small constant) and a constant number of communication rounds. See Section 4 for performance graphs.

3.1 Bit length

Computing the bit length is the problem of determining $\langle m \rangle$, such that $2^{m-1} \leq x < 2^m$, given secret-sharing $\langle x \rangle$ of integer x, $0 \leq x < 2^M$. Here M is a known upper bound, e.g., $M = 2\lambda$ in the multiple addition protocol. This operation is frequently used in secure floating points and often forms the bottle neck of performance. It is used for rescaling the significand after addition, or assuring the proper range for Padé polynomial approximations.

Previous solutions tackled this problem (known, e.g., as 'most significant non-zero bit (MSNZB)' in Rathee et al., 2022) by computing the bits of x, leading to a complexity of $\mathcal{O}(M \log_2 M)$ secure multiplications and $\mathcal{O}(\log_2 M)$ communication rounds (de Hoogh, 2012). Bit decomposition is called Z_n-to-Z_2 in Liu et al. (2013), and can be done in constant rounds, but for practical values of M a logarithmic round solution is preferred (de Hoogh, 2012). We found a new solution that, as far as we are aware, is the first to take only $\mathcal{O}(M)$ secure multiplications within constant rounds, thereby significantly improving the efficiency of secure floating points.

The main idea is to additively blind $\langle x \rangle$ with a large random number $\langle r \rangle$, reveal $c = x + r$, and study the right most (least significant) $M + 1$ (secret) bits $(c \oplus r)_M \dots (c \oplus r)_0$ of exclusive-or sequence $\langle c \oplus r \rangle$. If there were no carry-overs in the bitwise addition of x and the right most $M + 1$ bits of r, then these bits would equal the bits of x, and the left most 1 would be at position $m - 1$. The latter still holds, as long as there was no carry-over at specifically position $m - 1$. If there was a carry-over at position $m - 1$, then the left most 1 of $(c \oplus r)_M \dots (c \oplus r)_0$ would end up beyond position $m - 1$.

By analysing the bitwise addition, we are able to trace back the propagation of a potential carry-over at position $m - 1$ and obtain a sequence t, such that the position of the left most 1 of $t_M \dots t_0$ is (almost) equal to the position of the left most 1 of x, i.e., $m - 1$. In Figure 1 an example

is depicted to support the reader while reading the next paragraphs and understanding the protocol and its variables.

Let α_i be the carry-over at position i, $0 \le i \le M$, then

$$c_i = x_i \oplus r_i \oplus \alpha_{i-1}, \text{ and}$$
$$\alpha_i = (x_i + r_i + \alpha_{i-1} > 1).$$

Suppose the carry-over at index $m-1$ propagated as far as index μ, then $\alpha_i = 0$, $i \ge \mu$, and $\alpha_i = 1$, $m-1 \le i < \mu$. We consider two cases:

- For $m \le i < \mu$, we have $x_i = 0$, such that $c_i \oplus r_i = \alpha_{i-1} = 1$, and r_i should be 1 to set $\alpha_i = 1$.
- For $i = \mu$, we also have $c_i \oplus r_i = \alpha_{i-1} = 1$, but now $r_i = 0$ to avoid setting α_μ.

This means we can distinguish between $m \le i < \mu$ and $i = \mu$ with the pair (c_i, r_i), which equals $(0,1)$ in the first case, and $(1,0)$ in the second case. These $(0,1)$ cases for $0 \le i \le M$ are circled in Table 1.

Furthermore, we can trace back the carry-over propagation to position m by setting $t_i = 0$ (these are the framed zeros $\boxed{0}$ of t in Table 1) whenever $(c_{i-1}, r_{i-1}) = (0,1)$, and setting $t_i = (c \oplus r)_i$, otherwise, for $1 \le i \le M$. This works as long as $(c_{i-1}, r_{i-1}) \ne (0,1)$ for $i = m-1$, because we don't want to incorrectly set t_{m-1} to zero. For $i = m-1$ we have $x_{m-1} = 1$ and $c_{m-1} \oplus r_{m-1} = 1 \oplus \alpha_{m-2}$. We distinguish two cases:

- If $\alpha_{m-2} = 1$, then $c_{m-1} \oplus r_{m-1} = 0$, such that t_m will not be set to zero.
- If $\alpha_{m-2} = 0$, then $c_{m-1} \oplus r_{m-1} = 1$. If $c_{m-2} \oplus r_{m-2} = 1$ (and t_{m-1} might be set to zero), then $x_{m-2} \oplus \alpha_{m-3} = 1$. Given that α_{m-2} is not set, we derive $r_{m-2} = 0$ and safely conclude that t_{m-1} will not be set to zero.

Therefore, the back propagation of the carry-over will stop at either index m or $m-1$. When we have computed the index m' of the left most 1 of t (the index is one less than the bit length), we know that either $m' = m$, or $m' = m-1$. We can distinguish both cases by the secure comparison $\langle \delta \rangle \leftarrow \langle (x < 2^{m'}) \rangle$ and obtain $\langle m \rangle \leftarrow \langle m' \rangle - \langle \delta \rangle + 1$. In the example of Table 1, we have $m' = 4$ (because $t_4 = 1$ and $t_5 = t_6 = 0$) and $x = 26$, such that $\delta = (26 < 2^4) = 0$.

Given sequence t, we can compute the index m' of the left most 1 by adding the bits of the prefix-or p of t, which are defined as $p_i = t_0 \vee t_1 \vee \ldots \vee t_i$, $0 \le i \le M$. This entire analysis leads to the following protocol:

1 Compute $\langle r \rangle$ of random number r, consisting of at least κ more bits than x, κ being the statistical security parameter:
   a Securely generate secret sharings of $M+1$ random bits $\langle r_i \rangle$, $0 \le i \le M$, and a secret-sharing of a random number r' of at least κ bits.
   b Compute the random number $\langle r \rangle \leftarrow 2^{M+1} \cdot \langle r' \rangle + \sum_{i=0}^{M} \langle r_i \rangle \cdot 2^i$.
2 Compute and reveal $\langle c \rangle \leftarrow \langle x \rangle + \langle r \rangle$.
3 Locally compute the secret sequence $\langle c \oplus r \rangle$ for the first $M+1$ bits:
   For $i \leftarrow 0$ to M do:
   If $c_i = 0$ then $\langle (c \oplus r)_i \rangle \leftarrow \langle r_i \rangle$,
   else $\langle (c \oplus r)_i \rangle \leftarrow 1 - \langle r_i \rangle$.
4 Trace back the carry-over in one round, obtaining sequence $\langle t \rangle$:
   a $\langle t_0 \rangle \leftarrow \langle (c \oplus r)_0 \rangle$.
   b For $i \leftarrow 1$ to M do:
   if $c_{i-1} = 0$ then $\langle t_i \rangle \leftarrow (1 - \langle r_{i-1} \rangle) \cdot \langle (c \oplus r)_i \rangle$
   else $\langle t_i \rangle \leftarrow \langle (c \oplus r)_i \rangle$.
5 Compute the prefix-or $\langle p \rangle$ of $\langle t \rangle$:
   a Compute $\langle y_i \rangle \leftarrow 1 + \langle t_i \rangle$, $0 \le i \le M$.
   b Compute $\langle z_i \rangle \leftarrow \prod_{j=i}^{M} \langle y_j \rangle$, $0 \le i \le M$ with one fan-in multiplication.
   c Compute $\langle p_i \rangle \leftarrow 1 - \langle (z_i \bmod 2) \rangle$, $0 \le i \le M$.
6 Given p, compute first guess $\langle m' \rangle \leftarrow -1 + \sum_{i=0}^{M} \langle p_i \rangle$ and simultaneously
   $\langle 2^{m'} \rangle \leftarrow \langle p_0 \rangle + \sum_{i=1}^{M} 2^{i-1} \cdot \langle p_i \rangle$.
7 Run the secure comparison $\langle \delta \rangle \leftarrow \langle (x < 2^{m'}) \rangle$.
8 Compute $\langle m \rangle \leftarrow \langle m' \rangle - \langle \delta \rangle + 1$.

The protocol is also correct for $x = 0$, because then $p_0 = 0$, $m' = -1$, and $2^{m'} = 0$, such that $\delta = 0$ and $m = 0$. Table 1 shows a small example to illustrate the role of all variables.

Table 1 Example for bit length protocol

$x = 26 = 11010$							*input*
6	5	4	3	2	1	0	index $M \ldots 0$
0	0	1	1	0	1	0	x
1	0	①	0	0	①	1	r
1	1	⓪	1	1	⓪	1	$c = x + r$
0	0	1	0	0	1	0	carry-over α
0	1	1	1	1	1	0	$c \oplus r$
0	$\boxed{0}$	1	1	$\boxed{0}$	1	0	t (traced back)
0	0	1	1	1	1	1	prefix-or p
$m = m' - (x < 2^{m'}) + 1 = 4 - 0 + 1 = 5$							*output*

We presented a constant-round solution from (de Hoogh, 2012; PreOrC) for the prefix-or in step 5, but other approaches are known. It uses a fan-in multiplication to simultaneously compute the products z_i, $0 \le i \le M$, in two communication rounds (de Hoogh, 2012; KMulC). In Schoenmakers and Tuyls (2012), an efficient solution can be found to compute the parity bits $z_i \bmod 2$, known as least significant bit gate.

Similarly as $\langle 2^{m'} \rangle$ in step 6, we can use the prefix-or $\langle p \rangle$ to compute $\langle 2^{-m} \rangle$ without additional secure multiplications, which value is needed elsewhere (see next subsection) in the (multiple) addition protocol: $\langle 2^{-m'} \rangle \leftarrow 2 - \langle p_0 \rangle - \sum_{i=1}^{M} 2^{-i} \langle p_i \rangle$ and $\langle 2^{-m} \rangle \leftarrow (1 + \langle \delta \rangle) \cdot \langle 2^{-m'} \rangle \cdot 2^{-1}$.

3.2 Normalisation

We show an efficient implementation for the secure integer division $s_y \leftarrow T \div 2^{m-\ell}$ of step 4b. The first idea is to transform the secret divisor to a public divisor, which leads to a more efficient operation:

$$T \div 2^{m-\ell} = (T \cdot (2^{2\lambda} \cdot 2^{-m})) \div 2^{2\lambda-\ell}.$$

Assuming that 2^{-m} has been computed simultaneously in step 4a, we can compute the integer $2^{2\lambda-m}$ and multiply it with T, resulting in an integer of at most 2λ bits (because $|T| < 2^m$). Subsequently, a division with public divisor $2^{2\lambda-\ell}$ will yield s_y.

Catrina (2020a) uses a different, less efficient approach: he precomputes $T \div 2^{m-\ell}$ for all possible values of m (PreDiv2m) to ensure a fast online phase.

The second idea is to further relax the division with public divisor, also called truncation, to not only improve efficiency, but also increase accuracy. To explain that, we elaborate on the truncation operation, which is often needed to remove the least significant bits of an integer:

$$\text{Trunc}(\langle x \rangle, d) = \langle \lfloor x \cdot 2^{-d} \rfloor \rangle,$$

where d is a public positive integer. The protocol for truncation typically starts by blinding x with a large secret number r: $\langle c \rangle = \langle x \rangle + \langle r \rangle$, revealing c, and then computing

$$\text{Trunc}(\langle x \rangle, d) = (c \div 2^d) - \langle r \div 2^d \rangle - \langle \delta \rangle,$$

where δ is the binary outcome of the secure comparison $(c \bmod 2^d < \langle r \bmod 2^d \rangle)$ (Veugen, 2014). As the computation of $\langle \delta \rangle$ forms the computational bottleneck of the truncation protocol, one could easily accelerate it by skipping the secure comparison at the cost of a small inaccuracy (Veugen, 2014). This new truncation is also known as probabilistic rounding (TruncPr) (Schoenmakers, 2018) and Div2mP (Catrina, 2020a).

However, in our goal of removing the d least significant bits, TruncPr will not decrease, but improve accuracy, because $\text{TruncPr}(x, d)$ will on average produce a better integer approximation of $x \cdot 2^{-d}$ than $\text{Trunc}(x, d)$. The reason is that when $x \bmod 2^d$ is large, then the addition of r is likely to cause a carry-over modulo 2^d, in which case $\delta = 1$. Removing $\delta = 1$ means that $x \cdot 2^{-d}$ is rounded upwards instead of downwards, which is correct in case $x \bmod 2^d$ is large. It is less likely that the rounding is incorrectly performed upwards in case $x \bmod 2^d$ is small.

The only problem with TruncPr is that there is a slight probability of overflow, which will lead to $s_y = 2^\ell$ instead of $s_y = 2^\ell - 1$. If $m \leq \ell$, then the division by $2^{m-\ell}$ is actually a multiplication and the outcome will be exact. Only if $m > \ell$, overflow might happen. This effect will probably fade out during multiple floating-point computations, and could be easily eliminated by using a secure equality ($\langle s_y \rangle \leftarrow \langle s_y \rangle - \langle (s_y = 2^\ell) \rangle$), which takes less effort than a secure comparison.

As a side note, probabilistic rounding can also be used during secure multiplication of floating points, without the risk of overflow, because then $|s_1 \cdot s_2| \leq (2^\ell - 1)^2 = 2^{2\ell} - 2^{\ell+1} + 1$, so $|s_1 \cdot s_2| \div 2^\ell < 2^\ell - 1$.

3.3 Avoiding secure exponentiation

In step 2b of the multiple addition protocol (see Subsection 2.2) we need to compute $\langle 2^{e_i - e_{\max} + \lambda} \rangle$ for each term i. The straightforward approach is to use a secure exponentiation protocol. Computing $\langle 2^x \rangle$ securely for secret integer $\langle x \rangle$ is usually done through bit decomposition, because $2^x = \prod_i (2^{2^i} \cdot x_i + 1 - x_i)$, given the bits $x = \sum_i 2^i \cdot x_i$, which has a $\mathcal{O}(\ell \log \ell)$ complexity (de Hoogh, 2012; Abdelraham and Smart, 2019).

However, in our multiple addition protocol, we only need to compute $\langle 2^x \rangle$ for $x \in \{0, \ldots, \lambda\}$, i.e., the inputs with $\delta_i = 1$. An alternative way of computing these is to generate a Lagrange polynomial $q(x)$ of degree $\lambda + 1$ such that $q(x) = 2^x$ for $x \in \{0, \ldots, \lambda\}$. Computing $\langle 2^{e_i - e_{\max} + \lambda} \rangle = \langle q(e_i - e_{\max} + \lambda) \rangle$ will take λ multiplications and λ communication rounds using Horner's rule, or one fan-in multiplication (to compute $\langle (e_i - e_{\max} + \lambda)^j \rangle$ for all j) in one round.

By avoiding the bit decomposition we reduced the computational complexity from $\mathcal{O}(\ell \log \ell)$ down to $\mathcal{O}(\ell)$. Catrina (2020a) uses the Lagrange polynomials $q_i(x) = (x = i)$, $i \in \{0, \ldots, \lambda\}$ together with the precomputed divisions (see Subsection 3.2), which also avoids secure exponentiations, but in a slightly less efficient way.

4 Performance

We implemented secure floating points, including the multiple addition protocol, with the MPyC framework, and released it open source (TNO, 2024). We also implemented a variant (see the Appendix) that additionally stores $2^e \bmod 2^{2\lambda}$ to accelerate secure floating-point operations.

We used secure integer type SecInt(2λ) from MPyC for the significand (and SecInt($k + 1$) for the exponent), which automatically assigns a $(2\lambda + \kappa)$-bit Shamir secret-sharing modulus to the significand.

The number of secure multiplications determines both computational (execution time) and communication complexity (amount of communication), because secure additions can be performed without communication. The network latency is mostly determined by the number of communication rounds.

4.1 Multiple additions

We ran the multiple addition protocol, as presented here, and measured execution times on a single computer (with

three parties) for $n = 2$ up to $n = 128$ addends (with increasing secret-sharing modulus because λ increases with n), and compared it with n single additions (our multiple addition protocol with two addends and a $(2(\ell + 1) + \kappa)$-bit modulus).

Figure 1 Multiple additions (see online version for colours)

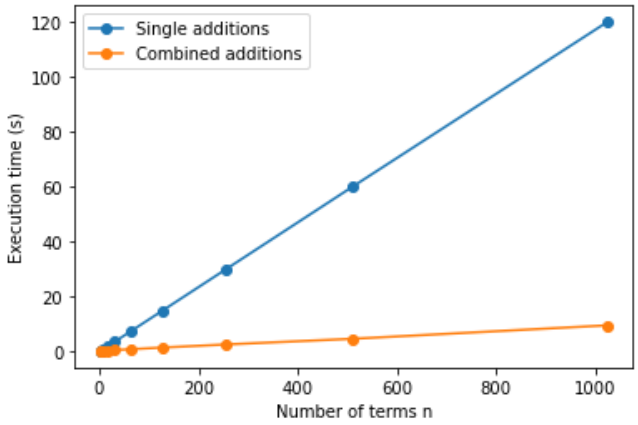


Figure 1 shows the clear gain of combining additions, caused by eliminating $n - 1$ bit length computations. The execution time was reduced by a factor that increases in n up to factor 13 with $n = 1{,}024$ combined additions.

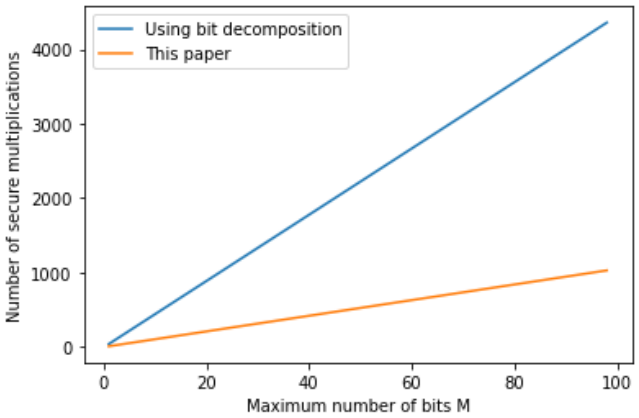
Because we used a single computer to conduct our experiments, communication has been neglected. We expect the advantage of combined addition to be actually larger, because the single addition approach has substantially more communication rounds.

4.2 *Bit length protocol*

Previous solutions (see Subsection 3.1) use a bit decomposition protocol to determine the bit length. The bit decomposition protocol replaces the first four steps of our bit length protocol and produces a similar sequence t, in their case exactly representing the input bits. The fifth step is the same for everyone, and computes the prefix-or p of sequence t.

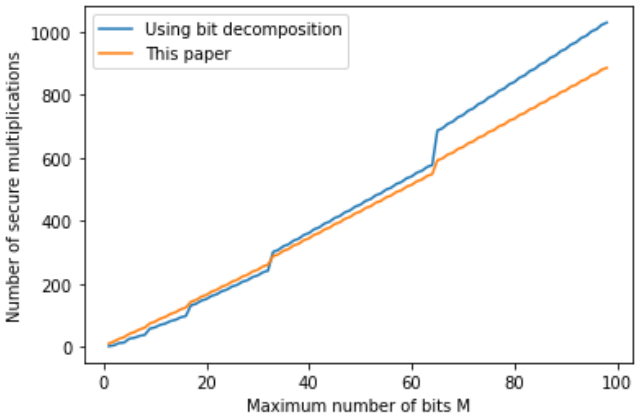
Logarithmic and constant-round solutions exist for both bit decomposition and prefix-or, the constant-round ones requiring more secure multiplications. Therefore, for smaller input sizes, it might be interesting to use a solution with logarithmic instead of constant communication rounds. To clearly distinguish the constant communication round solution from the logarithmic one, we counted the number of secure multiplications and the number of communication rounds. We used the PhD thesis of de Hoogh (2012) for this, which describes the bit decomposition (BitDec) and prefix-or (PreOr) protocols that are used in MPyC.

Figure 2 Constant round bit length protocol (see online version for colours)



The gain of the new bit length protocol is shown in Figure 2 (constant rounds) and Figure 3 (logarithmic rounds), where we used constant-round and logarithmic-round solutions respectively for both bit decomposition and prefix-or. The number of secure multiplications (invocations) of our protocol is clearly smaller in the case of constant rounds, going from roughly $44.5 \cdot M$ to $10.5 \cdot M$ invocations, and is more or less equal for the logarithmic round protocols.

Figure 3 Logarithmic round bit length protocol (see online version for colours)



The advantage of the new bit length protocol in the case of logarithmic rounds becomes clear when counting the total number of communication rounds, as depicted in Table 2. The number of communication rounds is reduced with a factor $M/2$, M being the maximal number of input bits.

Table 2 Number of communication rounds

Bit length protocol	*Constant rounds*	*Logarithmic rounds*
Using bit dec.	15	$2 + (1 + M/2) \log_2 M$
This paper	10	$7 + \log_2 M$

4.3 Normalisation and exponentiation

To show the gain of our other two improvements, namely the use of TruncPr instead of Trunc, and replacing of exponentiation by polynomial evaluation, we counted the number of secure multiplications and the number of communication rounds, as before, using de Hoogh (2012).

As depicted in Table 3, the alternative truncation protocol saves a factor 4 in multiplications, and halves the number of communication rounds.

Table 3 Complexity of truncating d bits

Protocol	*Secure multiplications*	*rounds*
Trunc	$4d+2$	4
TruncPr	$d+1$	2

The main difference between a secure exponentiation and a secure polynomial evaluation is the additional bit decomposition protocol for the exponentiation. Both solutions need a fan-in multiplication (KMul) to either multiply the ℓ bitwise exponents, or compute the exponents of the $\lambda = \ell + \lceil \log_2 n \rceil$ terms.

Table 4 Complexity of computing exponents

Protocol	*Secure multiplications*	*Rounds*
Exponentiation	$3\ell + \ell \log_2 \ell$	$\log_2 \ell + 14$
Polynomial evaluation	$3(\ell + \lceil \log_2 n \rceil) - 1$	2

The results have been summarised in Table 4, and show the clear advantage of polynomial evaluation, even for large number n of addends.

4.4 Other platforms

We have shown several improvements for secure floating points in MPC. Although implemented in MPyC, they apply to other platforms as well. This holds for the multiple addition approach, the improved bit length protocol, the normalisation with public divisor and probabilistic rounding, and the use of Lagrange interpolation to avoid secure exponentiations.

Although the multiple addition (and multiplication) approach is suitable for all MPC platforms, the specific improvements work with secret sharing systems over a field that have a head room to work with a statistical security parameter, like SPDZ (Keller, 2020) and SCALE-MAMBA (Abdelraham and Smart, 2019). The conversion with public divisor and the Lagrange interpolation do not specifically require a head room. Although garbled circuits work on the bit level (Demnler et al., 2015), they can still profit from combining multiple additions (and multiplications), but the specific steps would be worked out differently.

4.5 Application

In general, floating points are preferred over fixed points, in case computations have to be performed with both very large and very small numbers, because of their accuracy. In other situations where there is less variation in size, fixed points will be the most efficient choice.

A typical application for floating points is the Cox proportion hazard model that is used for, e.g., survival analysis in the medical domain (Kamphorst et al., 2022). During the iterative optimisation of the model parameters β, variables G_r^n need to be computed, given by

$$G_r^n = \frac{Z_r^1 e^{\beta^T \cdot Z^1} + \ldots + Z_r^n e^{\beta^T \cdot Z^n}}{e^{\beta^T \cdot Z^1} + \ldots + e^{\beta^T \cdot Z^n}}, \tag{1}$$

where Z_r^i is the value of covariate r for subject i [Kamphorst et al., 2022, equation (7)]. They rewrote equation (1) as a weighted covariate $G_r^n = \sum_{i=1}^n \eta_i \cdot Z_r^i$, where

$$\eta_i = (e^{\beta^T \cdot (Z^1 - Z^i)} + \ldots + e^{\beta^T \cdot (Z^n - Z^i)})^{-1}, \tag{2}$$

such that the computations could be performed with secure fixed points with modest length, leading to a reasonably accurate model, as illustrated in Figure 4.

Figure 4 Accuracy in Cox proportion hazard model (see online version for colours)

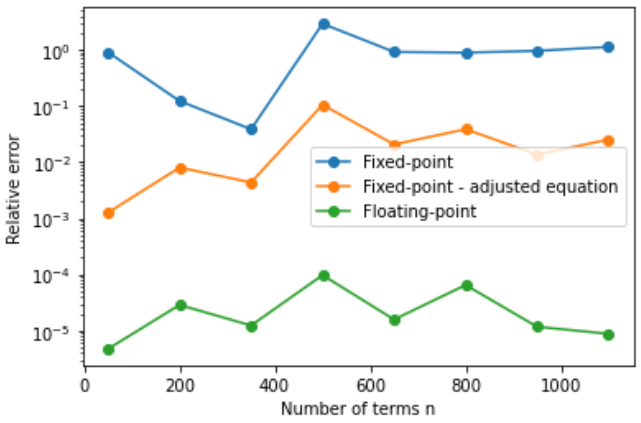
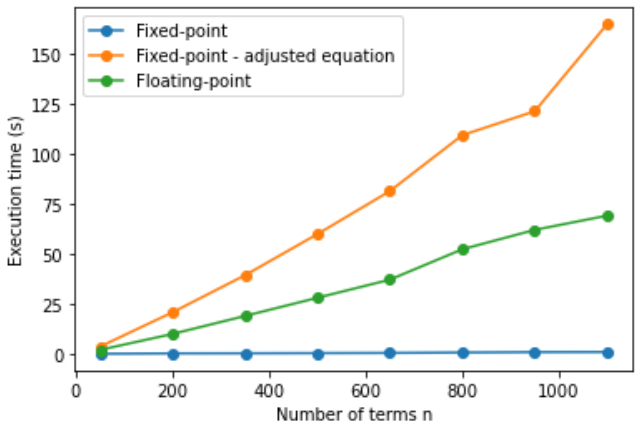


Figure 5 Execution time in Cox proportion hazard model (see online version for colours)



However, when using our secure floating point approach, the computation of equation (1) would have been straightforward, leading to more accurate solutions and

requiring less computation time. The different execution times have been depicted in Figure 5.

For these small experiments, we randomly sampled realistic values for the model parameters and covariates, and used fixed-points with 32 bits and scaling factor 2^{-16}, and similar floating-points with a significand and exponent of 16 bits. Although not all terms are positive, our multiple addition protocol achieved much better accuracy, and was faster than using equation (2) with fixed-points.

An important application of our new bit length protocol is the use of Padé polynomials in secure floating point arithmetic (Hart, 1978). These polynomials provide numerical approximations of various mathematical functions like logarithm, sine, exponentiation, etc., which are efficiently computable in the encrypted domain (Thissen, 2019). Since the approximations are valid within small intervals, the inputs need to be scaled beforehand, which requires computing the bit length of the inputs.

5 Conclusions and future research

We were able to significantly speed up the bottleneck of secure floating points by presenting a new protocol that combines multiple additions, which can be used in many MPC platforms. A typical application for our multiple addition protocol was given. We further accelerated it by eliminating secure division, and reducing exponentiations to a two-round evaluation of a Lagrange polynomial.

Furthermore, we reduced the complexity of the bit length protocol, and by that also of the entire addition protocol, to linear time with constant rounds. The new bit length protocol reduces the number of secure multiplications with a factor 4 (for the constant-round solution), or the number of communication rounds with a factor $M/2$ (for the logarithmic-round solution).

Accuracy requirements of (secure) floating points were clearly defined, and the accuracy of the multiple addition protocol was analysed. A different truncation protocol was suggested to improve both accuracy and efficiency.

The new secure floating-point framework, available as open source software, was tested and improvements were quantified. The more additions n were combined, the larger the relative gain, up to a factor 13 with $n = 1024$, which would have been larger if we had accounted for communication time. Similar to combining multiple additions, we also combined multiple floating-point multiplications, although a larger secret-sharing modulus was needed.

5.1 Future research

Instead of the common attempt to reduce the number of multiplications in a computation, one might develop new algorithms that minimise the number of additions, because these form the bottleneck in floating point arithmetic with MPC.

Furthermore, there is a trade-off between significand size and the number of terms, which can be explored.

References

Abdelrahaman, A. and Smart, N.P. (2019) 'Benchmarking privacy preserving scientific operations', *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019*, 5–7 June, Bogota, Colombia.

Aliasgari, M., Blanton, M., Zhang, Y. and Steele, A. (2012) 'Secure computation on floating point number', *Network and Distributed System Security (NDSS) Symposium 2013*, San Diego, USA, 24–27 February [online] https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/secure-computation-floating-point-numbers/.

Aliasgari, M., Blanton, M., Zhang, Y. and Steele, A. (2013) 'Secure computation on floating point numbers', *20th Annual Network and Distributed System Security Symposium, NDSS 2013*, 24–27 February, The Internet Society, San Diego, California, USA.

Belorgey, M.G., Carpov, S., Deforth, K. et al. (2023) 'Manticore: a framework for efficient multiparty computation supporting real number and Boolean arithmetic', *J. Cryptol.*, Vol. 36, p.31, https://doi.org/10.1007/s00145-023-09464-4

Blanton, M., Goodrich, M.T. and Yuan, C. (2023) 'Secure and accurate summation of many floating-point numbers', *Proceedings on Privacy Enhancing Technologies*, Vol. 2023, No. 3, pp.432–445.

Catrina, O. (2020a) 'Optimising secure floating-point arithmetic: sums, dot products, and polynomials', *Proceedings of the Romanian Academy, Series A*, Vol. 21, No. 1/2020, pp.21–28.

Catrina, O. (2020b) 'Performance analysis of secure floating-point sums and dot products', *2020 13th International Conference on Communications (COMM)*, Bucharest, Romania, pp.465–470, DOI: 10.1109/COMM48946.2020.9141961.

de Hoogh, S. (2012) *Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming*, PhD thesis 1, Research TU/e/Graduation TU/e), Mathematics and Computer Science, Eindhoven University of Technology.

Demmler, D., Dessouky, G., Koushanfar, F., Sadeghi, A-R., Schneider, T. and Zeitouni, S. (2015) 'Automated synthesis of optimized circuits for secure computation', *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*, Association for Computing Machinery, New York, NY, USA, pp.1504–1517, https://doi.org/10.1145/2810103.2813678.

Hart, J.F. (1978) *Computer Approximations*, Krieger Publishing Co., Inc., Melbourne, FL, USA, ISBN: 0882756427.

Kamm, L. and Willemson, J. (2015) 'Secure floating-point arithmetic and private satellite collision analysis', *Int. J. Inf. Sec.*, Vol. 14, No. 6, pp.531–548.

Kamphorst, B., Rooijakkers, T., Veugen, T., Cellamare, M. and Knoors, D. (2022) 'Accurate training of the Cox proportional hazards model on vertically-partitioned data while preserving privacy', *BMC Medical Informatics and Decision Making*, Vol. 22, p.49, https://doi.org/10.1186/s12911-022-01771-3.

Keller, M. (2020) 'MP-SPDZ: a versatile framework for multi-party computation', in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*, Association for Computing Machinery, New York, NY, USA, pp.1575–1590, https://doi.org/10.1145/3372297.3417872.

Kerik, L., Laud, P. and Randmets, J. (2016) 'Optimizing MPC for robust and scalable integer and floating-point arithmetic', *International Conference on Financial Cryptography and Data Security*, February.

Krips, T. and Willemson, J. (2014) 'Hybrid model of fixed and floating-point numbers in secure multiparty computations', *Information Security – 17th International Conference, ISC 2014, Proceedings. Lecture Notes in Computer Science*, 12–14 October, Springer, Hong Kong, China, Vol. 8783, pp.179–197.

Liu, Y-C., Chiang, Y-T., Hsu, T.S., Liau, C-J. and Wang, D-W. (2013) 'Floating-point arithmetic protocols for constructing secure data analysis application', *Procedia Computer Science*, Vol. 22, pp.152–161, ISSN: 1877-0509.

Rathee, D., Bhattacharya, A., Gupta, D., Sharma, R. and Song, D. (2023) 'Secure floating-point training', *32nd USENIX Security Symposium (USENIX Security 23)*, pp.6329–6346.

Rathee, D., Bhattacharya, A., Sharma, R., Gupta, D., Chandran, N. and Rastogi, A. (2022) 'SecFloat: accurate floating-point meets secure 2-party computation', *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, pp.576–595.

Sasaki, K. and Nuida, K. (2020) 'Efficiency and accuracy improvements of secure floating-point addition over secret sharing', in Aoki, K. and Kanaoka, A. (Eds.): *Advances in Information and Computer Security. IWSEC 2020. Lecture Notes in Computer Science*, Springer, Cham, Vol. 12231, https://doi.org/10.1007/978-3-030-58208-1_5.

Schoenmakers, B. and Tuyls, P. (2006) 'Efficient binary conversion for Paillier encryptions', *EUROCRYPT 2006*, Springer Verlag, Vol. 4004 of *LNCS*, pp.522–537.

Schoenmakers, B. (2018) 'MPyC: multiparty computation in Python', *Theory and Practice of Multiparty Computation (TPMPC) 2018 Workshop*, Aarhus, Denmark, 30 May [online] https://www.win.tue.nl/~berry/mpyc/.

TNO (2024) *Secure Floating Points*, December, Open Source Software [online] https://github.com/TNO-MPC/mpyc.floating_point (accessed 11 March 2025).

Thissen, K.K.A. (2019) *Achieving Differential Privacy in Secure Multiparty Computation*, Master thesis, Eindhoven University of Technology [online] https://research.tue.nl/files/130176372/Report.Kimberly_Thissen_pdf.pdf (accessed 11 March 2025).

Veugen, T. (2014) 'Encrypted integer division and secure comparison', *International Journal of Applied Cryptography*, Vol. 3, No. 2, pp.166–180.

Appendix

Storing exponents

In our source code (TNO, 2024), implemented within MPyC, we added the option of storing exponents to accelerate secure floating-point operations. The idea is that for each floating point x, not only significand s_x (a 2λ-bit SecInt) and exponent e_x (a $k+1$-bit SecInt) should be stored, but also the exponentiation $2^{e_x} \bmod 2^{2\lambda}$.

During the addition of secure floating points, this avoids computing the $2^{e_i - e_{\max}}$ from scratch (see step 2b in Subsection 2.2). Given that e_y is eventually set to $e_{\max} - \lambda + m - \ell$, the new exponentiation 2^{e_y} is easily computed with two secure multiplications.

When multiplying two secure floating points, the precomputed exponentiations are not really beneficial, but can be easily updated for the product output. However, when at some point secure floating points need to be converted to secure integers, they do reduce computational time.