USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Improved Secure Two-party Computation from a Geometric Perspective

Hao Guo, *School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen;* Liqiang Peng, *Alibaba Group;* Haiyang Xue, *Singapore Management University;* Li Peng and Weiran Liu, *Alibaba Group;* Zhe Liu, *Zhejiang Lab;* Lei Hu, *Institute of Information Engineering, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity25/presentation/guo-hao-improved

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Improved Secure Two-party Computation from a Geometric Perspective

Hao Guo[1], Liqiang Peng[2], Haiyang Xue[3], Li Peng[2], Weiran Liu[2], Zhe Liu[4,*] and Lei Hu[5]

[1]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen
[2]Alibaba Group
[3]Singapore Management University
[4]Zhejiang Lab
[5]Institute of Information Engineering, Chinese Academy of Sciences

Abstract

Multiplication and other non-linear operations are widely recognized as the most costly components of secure two-party computation (2PC) based on linear secret sharing. Moreover, the comparison protocol (or Wrap protocol) is essential for various operations such as truncation, signed extension, and signed non-uniform multiplication. This paper aims to optimize these protocols by avoiding invoking the costly comparison protocol, thereby improving their efficiency.

We propose a novel approach to study 2PC from a geometric perspective. Specifically, we interpret the two shares of a secret as the horizontal and vertical coordinates of a point in a Cartesian coordinate system, with the secret itself represented as the corresponding point. This reformulation allows us to address the comparison problem by determining the region where the point lies. Furthermore, we identify scenarios where the costly comparison protocol can be replaced by more efficient evaluating AND gate protocols within a constrained range. Using this method, we improve protocols for truncation, signed extension and signed non-uniform multiplication, all of which are fundamental to 2PC. In particular, for the one-bit error truncation protocol and signed extension protocols, we reduce the state-of-the-art communication complexities of Cheetah (USENIX'22) and SirNN (S&P'21) from $\approx \lambda(l+1)$ to $\approx \lambda$ in two rounds, where l is the input length and λ is the security parameter. For signed multiplication with non-uniform bit-width, we reduce the communication cost of SirNN's by 40% to 60%.

1 Introduction

In today's digital age, data plays a crucial role in society and individual lives. However, certain data is sensitive and can not be exposed to others, which limits its potential utility. To address this issue, the concept of secure computation has emerged. Secure multi-party computation (MPC) [7, 26, 27], a fundamental cryptographic primitive, enables multiple participants to jointly evaluate a function without exposing their inputs. Participants can not obtain additional information except for the function's output.

Machine learning has become a highly influential field in recent years, demonstrating significant potential for application and innovation across various domains. However, addressing data security concerns in machine learning has gradually become a research hot-spot. Notable contributions in two-party secure inference have been proposed recently, such as CrypTFlow2 [24], Delphi [20], SirNN [23], CipherGPT [11], moreover the works of SecureML [21], MiniONN [17], Cheetah [13], SecretFlow [18] and ABY2 [22] also focus on the training phase. Due to the complex operations involved in machine learning frameworks, general methods in secure two-party computation are not suitable for directly designing privacy-preserving machine learning (PPML) schemes. This necessitates the development of customized and efficient protocols, as demonstrated in works such as [6, 9, 16, 19, 28].

It is well established that multiplication and non-linear operations, such as truncation and comparison, are the primary performance bottlenecks in secure two-party computation (2PC). For example, truncation contributes more than 50% of communication overhead in CrypTFlow2 [13, 24], and the comparison protocol accounts for more than 68% of the total runtime in CryptGPU [25, 28]. In PPML schemes, real numbers must be encoded into fixed-point representation, and secure two-party computation protocols are performed over a ring $\mathbb{Z}_{2^l}$. As for non-linear operations, OT-based protocols on ring $\mathbb{Z}_{2^l}$ can perform 40% $\sim$ 60% better than on the prime field $\mathbb{Z}_p$ in terms of bandwidth consumption [13]. For linear layers, computing the product of two fixed-point numbers is required to perform matrix multiplication and convolution operations. As a result, the decimal places in the product with fixed-point representation form will increase, then truncation is required to maintain the precision of decimal and prevent overflow. A detailed analysis of this protocol shows that the cost of truncation is nearly half that of multiplication, accounting for over one-third of the total cost in the

*Corresponding author: sduliuzhe@gmail.com

multiplication-then-truncation protocol.

The expensive truncation protocol primarily depends on a comparison or Wrap protocol, which can be regarded as a variant of the Millionaires' problem [27]. Specifically, in secure two-party computation, a secret value x is shared as $x = x_0 + x_1 \mod L$, where participant P_0 holds x_0 and participant P_1 holds x_1. Alternatively, this can be expressed as $x = x_0 + x_1 - w \cdot L$, where $w = 1$ if $x_0 + x_1 \geq L$, and $w = 0$ otherwise. To implement the truncation protocol, we first need to compute w, which involves determining whether $x_0 + x_1 \geq L$ (i.e., whether $x_0 \geq L - x_1$, where P_0 holds x_0 and P_1 holds $L - x_1$). The communication complexity of this comparison protocol is $O(\lambda l)$ in $O(\log l)$ rounds [24].

We also observed that the comparison protocol significantly contributes not only to truncation but also to signed extension and signed non-uniform multiplication in SirNN [23], making these operations computationally and communicationally expensive when dealing with signed numbers. We focus on secure two-party computation (2PC) in the semi-honest setting and are motivated by the need to reduce the costs associated with truncation, signed extension, and signed non-uniform multiplication protocols, thereby enhancing the overall performance of 2PC.

1.1 Our Contributions

In this work, we address the challenges associated with designing protocols for functions with signed inputs by proposing a novel geometric method. This method allows us to replace the costly comparison protocol with more efficient evaluating AND gate protocols when the input is constrained, thereby reducing communication costs and improving efficiency.

Formal contributions. We achieve the following contributions:

- We introduce an innovative geometric method for designing and optimizing secure two-party computation protocols for signed functions, which can also serve as a valuable research tool in MPC.
- We apply our geometric method to develop new protocols for truncation, signed extension and signed multiplication with non-uniform bitwidths. Compared to state-of-the-art protocols, our approach significantly improves both runtime and communication efficiency.

To be more specific, the results of our proposed applications can be listed as follows.

New truncation protocols. This work proposes both one-bit error truncation protocol and faithful truncation protocol, designed to truncate k bits from an l-bit shared value x. Compared to the state-of-the-art truncation protocols in Cheetah [13] and SirNN [23], our new one-bit error and faithful truncation protocols achieve a communication improvement by factors approximately $l+1$ and $\frac{l+3}{k+1}$, respectively. The details of communication comparison are shown in Table 1(a). For typical parameters $l = 37$ and $k = 12$, experimental results show that our one-bit error and faithful truncation protocols have an approximate $35\times$ and $> 3\times$ improvement compared to the works in Cheetah and SirNN, respectively. Additionally, the performance of one-bit error truncation protocol with known MSB in Cheetah [13] is also improved by $> 1.6\times$.

New signed extension protocols. We also propose new signed extension protocols with constraint, achieving an $(m+1)\times$ improvement in communication compared to SirNN. Moreover, experimental results demonstrate a $> 18.47\times$ improvement in runtime. The details are provided in Table 1(b).

New signed multiplication with non-uniform bitwidths protocol. We propose a new signed non-uniform multiplication protocol for inputs of m-bit x and n-bit y. A detailed comparison of the communication between SirNN's multiplication protocol and ours for input length (m,n) and $(m+1,n+1)$ is shown in Table 2. Moreover, our implementation demonstrates that our multiplication protocol outperforms SirNN's, achieving approximately $1.7\times$ improvement in runtime and $1.5\times$ reduction in communication costs for parameters $m = 20$ and $n = 30$.

1.2 Related works

1.2.1 Truncation Protocol

The truncation protocol truncates k bits from $x \in \mathbb{Z}_L$, where $L = 2^l$, to compute $x \gg k$. SecureML [21] proposed a local truncation protocol with no communication but introducing two types of errors: e_{small} and e_{big}. e_{small} is a one-bit error that occurs with a probability of $\frac{1}{2}$, while the e_{big} error is bounded by 2^l and occurs with a probability of $2^{l_x - l + 1}$, where l_x represents the significant number of bits of input. CrypTFlow2 [24] proposed a faithful truncation protocol that eliminates errors entirely by invoking two comparison protocols over l bits and k bits to address e_{big} and e_{small}, respectively. Subsequently, SirNN [23] proposed an optimized faithful truncation protocol that replaces the two comparison protocols with the Wrap protocol over $l-k$ and k bits, followed by an evaluating AND gate protocol.

Recently, Cheetah [13] proposed a truncation protocol that results in only e_{small} errors, also known as the one-bit error truncation protocol, asserting that the impact of a one-bit error on some practical application scenarios such as PPML is negligible. This approach requires only one call to the comparison protocol and one call to B2A protocol.

1.2.2 Protocols for functions with signed input

SirNN [23] proposed several protocols for computing functions with signed inputs, including truncation, signed exten-

Table 1: Comparison of overhead with prior work for truncation and signed extension protocols. We suppose the input is shared on $\mathbb{Z}_{2^l}$ (for truncation protocol) or $\mathbb{Z}_{2^m}$ (for signed extension protocol) and $|x| \le 2^{l_x}$ where $l_x \le l-1$ (or $l_x \le m-1$). Then for general $x \in \mathbb{Z}_{2^l}$, we have $l_x \le l-1$, and the constraint $|x| \le \frac{2^l}{3}$ and $|x| \le \frac{2^l}{4}$ can be denoted as $l_x \le l-1.58$ and $l_x \le l-2$, respectively. λ is the security parameter and is usually set as 128.

(a) Comparison with the state-of-the-art of secure 2PC truncation protocols, where k is the number of bits to be truncated.

Benchmark	Method	Comm. (bits)	Round	Constraint
Trun. (1-bit error)	Cheetah [13]	$< \lambda(l+1)+14l+k$	$\log l+2$	$l_x \le l-1$
	Π^k_{trun1}, Sec.5.1.1	$\mathbf{2\lambda+2k}$	**2**	$l_x \le l-1.58$
	Π^k_{trun2}, Sec.5.1.1	$\boldsymbol{\lambda+k}$	**2**	$l_x \le l-2$
Trun. known MSB (1-bit error)	Cheetah [13]	$2\lambda+k+2$	4	known $\mathsf{MSB}(x)$
	Ours, Sec.5.1.2	$\boldsymbol{\lambda+k}$	**2**	known $\mathsf{MSB}(x)$
Trun. (Faithful)	CrypTFlow2 [24]	$< \lambda(l+2+k)+19l+14k$	$\log l+2$	$l_x \le l-1$
	SirNN [23]	$< \lambda(l+3)+15l+k+20$	$\log l+3$	$l_x \le l-1$
	$\Pi^k_{\mathsf{trun_f}}$, Sec.5.1.3	$\boldsymbol{< \lambda(k+2)+l+15k}$	$\log \boldsymbol{k+2}$	$l_x \le l-2$

(b) Comparison with the state-of-the-art of secure 2PC signed extension protocol that extend $x \in \mathbb{Z}_{2^m}$ to $x \in \mathbb{Z}_{2^n}$, where $M < N$.

Benchmark	Method	Comm. (bits)	Round	Constraint
SExt.	SirNN [23]	$< \lambda(m+1)+13m+n$	$\log m+2$	$l_x \le m-1$
	$\Pi^{m,n}_{\mathsf{SExt1}}$, Sec.B.1	$\mathbf{2(\lambda+n-m)}$	**2**	$l_x \le m-1.58$
	$\Pi^{m,n}_{\mathsf{SExt2}}$, Sec.B.1	$\boldsymbol{\lambda+n-m}$	**2**	$l_x \le m-2$

Table 2: Comparison of the communication with SirNN's signed multiplication with our $\Pi^{m,n}_{\mathsf{SMul}}$, where $\mu = \min\{m,n\}$ and $\nu = \max\{m,n\}$. For given parameter (m,n), we list the communication of SirNN's multiplication protocol with inputs shared on m-bit and n-bit ring. While for our multiplication protocol, we list the communication of $\Pi^{m,n}_{\mathsf{SMul}}$ and $\Pi^{m+1,n+1}_{\mathsf{SMul}}$, where the second protocol ensures the constraints $|x| < \frac{2^{m+1}}{4}$ and $|y| < \frac{2^{n+1}}{4}$ are satisfied.

Input length	SirNN [23]	$\Pi^{m,n}_{\mathsf{SMul}}$ and $\Pi^{m+1,n+1}_{\mathsf{SMul}}$, Sec.5.3	Improvement
(m,n)	$< \lambda(3\mu+\nu+4)+\mu(\mu+2\nu+1)$ $+16(m+n)$	$\lambda(2\mu+12)+\mu(\mu+1)+2mn+4(m+n)$	$\approx \frac{2\mu+(m+n)+4}{2\mu+12}\times$
		$\lambda(2\mu+14)+\mu(\mu+3)+2mn+6(m+n)$	$\approx \frac{2\mu+(m+n)+4}{2\mu+14}\times$
$(10,50)$	< 12300	5446	2.25 ×
		5854	2.21 ×
$(20,30)$	< 13920	8476	1.64 ×
		8884	1.64 ×
$(30,30)$	< 19020	12186	1.56 ×
		12634	1.56 ×

sion and signed multiplication with non-uniform bitwidths protocols. They first design the protocols for unsigned inputs and then convert them to signed versions. However, their method requires invoking costly comparison protocols, leading to significant overhead.

1.3 Organisation

The remainder of this paper is organized as follows. In Section 2, we introduce the overview of our techniques. In Section 3, we provide the necessary preliminaries. Section 4 introduces our geometric method for computing signed value. We then apply this method to design new truncation, signed extension and signed multiplication protocols in Section 5, We implement experiments for these protocols in Section 6 and conclude this paper in Section 7.

2 Overview of Our Techniques

Notations. We use $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ to represent the floor and ceiling functions, respectively. We consider an l-bit ring $\mathbb{Z}_L$ where $L = 2^l$. Let $\mathbf{1}\{\mathsf{state}\}$ denote the indicator function, which

equals 1 if state is true and 0 otherwise. For $x = x_0 + x_1 \mod L \in \mathbb{Z}_L$, the Most Significant Bit (MSB) for $x \in \mathbb{Z}_L$ is defined as $\mathsf{MSB}(x) = \mathbf{1}\{x \geq 2^{l-1}\}$. Additionally, we define $\mathsf{Wrap}(x) = \mathsf{Wrap}(x_0, x_1, L) = \mathbf{1}\{x_0 + x_1 \geq L\}$. The functions $\mathsf{int}(x)$ and $\mathsf{uint}(x)$ represent the signed and unsigned values in $\mathbb{Z}$, respectively, where $\mathsf{int}(x) = \mathsf{uint}(x) - \mathsf{MSB}(x) \cdot L$. For a signed integer x, we write $x \gg k$ to denote the arithmetic right-shift of x by k-bit. Additionally, for convenience, we define $|x| < B$ as $x \in [0, B) \cup [L - B, L)$, where we allow x to take on the value of $L - B$. To denote shares on the rings $\mathbb{Z}_2$ and $\mathbb{Z}_L$, we use the notation $[\![\cdot]\!]^B$ and $[\![\cdot]\!]^L$, respectively. The expression $x =_L y$ indicates $x \equiv y \mod L$. Suppose $\mathcal{D}_1$ and $\mathcal{D}_2$ are two families of distributions, the symbol $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ denotes they are computationally indistinguishable. In our following protocols, λ is the security parameter and is typically set to 128.

2.1 Protocols for functions with signed inputs

In two-party secret sharing, a secret value is encoded as an unsigned integer on ring $\mathbb{Z}_L$, and shared as $\mathsf{uint}(x) = x = x_0 + x_1 \mod L$. However, most functions take signed input $\mathsf{int}(x)$. For example, the truncation protocol can be computed as $x \gg k = \lfloor \frac{\mathsf{int}(x)}{2^k} \rfloor$. SirNN [23] proposed a method that first designs a protocol for functions with unsigned input and then converts it to a signed version. In this work, we directly take the signed value as input and design protocols for functions with signed input. We begin by determining how to compute $\mathsf{int}(x)$ from x_0 and x_1 without $\mod L$ operation.

For $\mathsf{uint}(x) = x = x_0 + x_1 \mod L$, we have that $\mathsf{uint}(x) = x_0 + x_1 - \mathsf{Wrap}(x_0, x_1, L) \cdot L$ and $\mathsf{int}(x) = \mathsf{uint}(x) - \mathsf{MSB}(x) \cdot L$, where $\mathsf{MSB}(x) = 1$ if $x \geq \frac{L}{2}$, else $\mathsf{MSB}(x) = 0$, and $\mathsf{Wrap}(x_0, x_1, L)$ outputs 1 if $x_0 + x_1 \geq L$, else outputs 0. Thus, the signed x can be written as:

$$\mathsf{int}(x) = x_0 + x_1 - (\mathsf{Wrap}(x_0, x_1, L) + \mathsf{MSB}(x)) \cdot L. \quad (1)$$

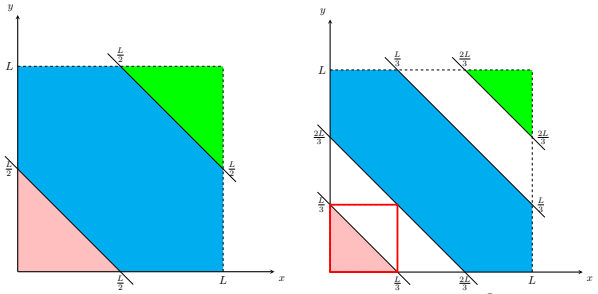
We define the signed coefficient as $\mathsf{MW}(x_0, x_1, L) = \mathsf{Wrap}(x_0, x_1, L) + \mathsf{MSB}(x)$, and abbreviate it as $\mathsf{MW}(x, L)$ or simply $\mathsf{MW}(x)$ when there is no ambiguity. From the definitions of Wrap and MSB function we have:

$$\mathsf{MW}(x) = \mathsf{MW}(x_0, x_1, L) = \begin{cases} 0, & \text{if } x_0 + x_1 \in [0, \frac{L}{2}) \\ 1, & \text{if } x_0 + x_1 \in [\frac{L}{2}, \frac{3L}{2}) \\ 2, & \text{if } x_0 + x_1 \in [\frac{3L}{2}, 2L) \end{cases}. \quad (2)$$

Once we obtain $\mathsf{MW}(x)$, we can compute $\mathsf{int}(x)$ and proceed to design protocols for public functions. Thus, the key to performing protocols with signed input is computing $\mathsf{MW}(x)$. Traditionally, this computation involves invoking a comparison protocol, which introduces significant overhead. In this work, we propose a new geometric method to compute $\mathsf{MW}(x)$, requiring one or two cheap AND operations and avoiding the costly comparison protocol.

2.2 The idea for computing $\mathsf{MW}(x)$

For $x = x_0 + x_1 \mod L$, we interpret x_0 as the horizontal coordinate and x_1 as the vertical coordinate, which together represent the point $P(x_0, x_1)$ in a two-dimensional plane. Given x, considering x_0 as the independent variable and x_1 as the dependent variable, the expression $d = x_0 + x_1$ (without $\mod L$) or $x_1 = -x_0 + d$ represents a straight line with a slope of -1 and an intercept of d. Thus, the expression $x_0 + x_1 > a$ represents the area above the straight line $x_1 = -x_0 + a$. Further, $x_0 + x_1 \in [a, b)$ corresponds to the area between two lines: $x_1 = -x_0 + a$ and $x_1 = -x_0 + b$. From this geometric perspective, the problem of computing $\mathsf{MW}(x)$ in Equation 2 reduces to determining whether the point P lies below the line $x_1 = -x_0 + \frac{L}{2}$ or above the line $x_1 = -x_0 + \frac{3L}{2}$, as depicted in Figure 1(a). In other words, $\mathsf{MW}(x) = 0, 1$ and 2 if P falls into the pink, blue and green area, respectively. Therefore, computing $\mathsf{MW}(x)$ is equivalent to identifying which area P falls into.



(a) $x \in \mathbb{Z}_L$. (b) $|x| < \frac{L}{3}$.

Figure 1: Feasible region of $P(x_0, x_1)$.

We can efficiently determine whether a point $P(x_0, x_1)$ is located within a square area by performing a simple AND operation. Specifically, $\mathbf{1}\{a < x_0 < b\} \wedge \mathbf{1}\{c < x_1 < d\} = 1$ if and only if $P(x_0, x_1)$ located in the square area with vertices $(a, c), (b, c), (b, d)$ and (a, d). However, to compute $\mathsf{MW}(x)$, we need to identify a triangle area (the pink and green area in Figure 1(a)). To address this, we separate the three feasible regions of point P so that they are sufficiently far apart, ensuring that a square covers only one triangular area. Specifically, we limit the input as $|x| < \frac{L}{3}$, where $x \in [0, \frac{L}{3}) \bigcup [\frac{2L}{3}, L)$ and $x_0 + x_1 \in [0, \frac{L}{3}) \bigcup [\frac{2L}{3}, \frac{4L}{3}) \bigcup [\frac{5L}{3}, 2L)$, with the feasible region of P shown in Figure 1(b).

In this figure, P falls within the pink area if and only if it falls into the square area with vertices $(0, 0), (0, \frac{L}{3}), (\frac{L}{3}, \frac{L}{3})$ and $(\frac{L}{3}, 0)$. Therefore, $\mathsf{MW}(x) = 0$ if and only if $\mathbf{1}\{x_0 < \frac{L}{3}\} \wedge \mathbf{1}\{x_1 < \frac{L}{3}\} = 1$. Similarly, $\mathsf{MW}(x) = 2$ if and only if $\mathbf{1}\{x_0 \geq \frac{2L}{3}\} \wedge \mathbf{1}\{x_1 \geq \frac{2L}{3}\} = 1$, and $\mathsf{MW}(x) = 1$ in other cases. Overall, only two AND operations are required to compute $\mathsf{MW}(x)$ when $|x| < \frac{L}{3}$, avoiding the need for the costly comparison protocol. Further, only one AND gate is needed to compute $\mathsf{MW}(x)$ when $|x| < \frac{L}{4}$. The details are shown in Section 4.2

and Section 4.3.

2.3 Application

We apply our method to design protocols for functions with signed inputs, including the truncation, signed extension and signed non-uniform multiplication protocols, where these functions all take the form $f(\mathsf{int}(x))$, as referred in Equation 3, Equation 9 and Equation 10. By leveraging our geometric method to compute $\mathsf{MW}(x)$ and $\mathsf{int}(x)$, we can efficiently implement these protocols when the input is constrained by the boundaries $\frac{L}{3}$ or $\frac{L}{4}$. Additionally, we discuss how to meet the constraints in Section 5.4.

3 Preliminaries

3.1 Cryptographic Primitives

3.1.1 Fixed-Point representation

To perform cryptographic operations, real numbers are encoded as elements on the ring $\mathbb{Z}_L$ using their fixed-point representation. In this representation, a real number $\hat{x}$ is represented by an l-bit integer $x \in \mathbb{Z}_L$, where the first $l-k$ bits of x denote the integer part of $\hat{x}$ and the last k bits represent the fractional part. This representation can be expressed as $x = \mathsf{Fix}(\hat{x}, l, k) = \lfloor \hat{x} \cdot 2^k \rfloor \mod L$. To decode x back into the real number $\hat{x}$, we simply calculate $\frac{x}{2^k}$.

3.1.2 Secret Sharing Scheme

In this paper, we utilize a 2-out-of-2 additive secret sharing scheme. A secret value x is shared between two parties P_0 and P_1 as $[\![x]\!]^L = ([\![x]\!]^L_0, [\![x]\!]^L_1)$ such that $x = [\![x]\!]^L_0 + [\![x]\!]^L_1 \mod L$. For $L = 1$, we use $[\![x]\!]^B$ to denote the boolean shares. Moreover, the state "P_0 and P_1 hold (or output) $[\![x]\!]^L$" means that each party P_i holds (or outputs) $[\![x]\!]^L_i$ for $i \in \{0,1\}$. For simplicity, we abbreviate $[\![x]\!]^L_i$ as x_i when the context is clear.

Additive secret sharing provides perfect secrecy of x since each party P_i only knows x_i, revealing no information about x even if he has infinite computational power. In this scheme, to securely compute a public function $y = f(x)$, we design a protocol Π_f that takes $[\![x]\!]^L$ as inputs and outputs y_0 and y_1 such that $y = y_0 + y_1 \mod L = f(x)$.

3.1.3 Oblivious Transfer

Oblivious transfer (OT) is a fundamental protocol in secure multi-party computation [4]. In a general $\binom{2}{1}$-OT$_l$, the sender inputs two l-bit messages m_0, m_1 and the receiver inputs a choice bit b. The protocol ensures that the receiver learns m_b without gaining any knowledge of $m_{b\oplus 1}$. Simultaneously, the sender remains unaware of the value of b and learns nothing.

The communication cost for $\binom{2}{1}$-OT$_l$ is $\lambda + 2l$ in 2 rounds. In the scenarios where the sender's messages are correlated, a more efficient correlated OT (COT) [2] $\binom{2}{1}$-COT$_l$ is used, with communication cost of $\lambda + l$ in 2 rounds. Additionally, we can implement $\binom{n}{1}$-OT$_l$ using the IKNP-style OT extension [14], with communication $2\lambda + nl$.

3.1.4 2PC Functionalities

We use the symbol Π^l_f to denote the protocol that securely evaluates function $f(\cdot)$ with input shared over the ring $\mathbb{Z}_{2^l}$. In this work, we use the following two-party protocols.

Comparison. The comparison protocol is also known as Millionaires' or Wrap protocol, which is used to compare two values without revealing them. In Millionaires' protocol, P_0 inputs a $x \in \mathbb{Z}_L$ and P_1 inputs a $y \in \mathbb{Z}_L$, then output $[\![\mathbf{1}\{x < y\}]\!]^B$. The Wrap protocol also can be realized using Π^l_{Mill} with input $L-1-x$ and y, since $\mathsf{Wrap}(x,y,L) = \mathbf{1}\{L-1-x < y\}$. Recently, CrypTFlow2 [24] proposed an efficient Π^l_{Mill} with communication less than $\lambda l + 14l$ in $\log l$ rounds.

Evaluating AND gate. The Π_{AND} takes inputs $([\![x]\!]^B, [\![y]\!]^B)$ and outputs $[\![x \wedge y]\!]^B$. CrypTFlow2 [24] implement this protocol using Beaver bit-triples [3,24], with a total communication $\lambda + 20$ in 2 rounds.

Boolean to Arithmetic (B2A). The protocol Π^l_{B2A} takes boolean shares $[\![x]\!]^B$ as input and outputs arithmetic shares $[\![x]\!]^L$ of the same value. This conversion can be achieved using COT with a communication $\lambda + l$ bits in 2 rounds [24].

Bit multiplication. In this work, we propose a new bit multiplication protocol Π^l_{BitMul}, where P_0 inputs a bit u and P_1 inputs a bit v, then returns $[\![u \wedge v]\!]^L$, with communication $\lambda + l$ in 2 rounds. There are two key differences between Π^l_{BitMul} and Π_{AND}: (1) Π^l_{BitMul} takes inputs u and v held by P_0 and P_1, respectively, while Π_{AND} operates on two shared bits; (2) Π^l_{BitMul} directly outputs $u \wedge v$ in algebraic sharing form, whereas Π_{AND} outputs Boolean shares that require an additional Π^l_{B2A} conversion to algebraic shares, increasing the total communication to $2\lambda + l + 20$. The details of Π^l_{BitMul} are provided in Appendix A.

Multiplexer (MUX). Π^l_{MUX} has two inputs, an arithmetic shares $[\![x]\!]^L$ and a boolean shares $[\![y]\!]^B$. The output is an arithmetic shared $[\![z]\!]^L$, where $z = x$ if $y = 1$ and $z = 0$ if $y = 0$. Recently, SirNN [23] proposed an implementation of this protocol by using COTs, with a total communication $2\lambda + 2l$ in 2 rounds.

3.2 Threat Model and Security

We consider a static semi-honest probabilistic polynomial time (PPT) adversary $\mathcal{A}$, where each participant follows the specification of the protocol and the adversary can corrupt at most one participant. Formally, the standard simulation-based notion of security in the presence of semi-honest adversaries [5, 8] is that

Definition 1. *(Semi-Honest Security). Let $f : X_1 \times X_2 \to Y$ be a randomized functionality and let Π be a protocol. We say that Π securely computes f in the presence of a single semi-honest corruption if there exists an efficient simulator $\mathcal{S}$ such that for every corrupted party $i \in \{0,1\}$ and every input $x \in X_1 \times X_2$ we have:*

$$\{output^{\Pi}(x), view_i^{\Pi}(x)\} \overset{c}{\approx} \{f(x), \mathcal{S}(i, x_i, f_i(x))\}$$

where $view_i^{\Pi}(x)$ is the view of party i in an execution of Π on input x, $output^{\Pi}(x)$ is the output of all parties in an execution of Π on input x, and $f_i(x)$ denotes the i-th output of $f(x)$.

3.3 Truncation Protocol

Truncation is a crucial non-linear operation in fixed-point calculation, often combined with multiplication protocol to maintain the precision of decimal values. Given two real number $\hat{x}, \hat{y} \in \mathbb{R}$, where $\hat{x} \cdot 2^k, \hat{y} \cdot 2^k \in \mathbb{Z}$, the product of $\hat{x}$ and $\hat{y}$ in fixed-point representation is expressed as $\mathsf{Fix}(\hat{x}\hat{y}, l, k) = xy \cdot 2^k = \lfloor \frac{\mathsf{Fix}(\hat{x},l,k) \cdot \mathsf{Fix}(\hat{y},l,k)}{2^k} \rfloor$. Consequently, a truncation operation that shifts k bits is required after computing the product of two fixed-point numbers.

We denote the truncate or arithmetic right-shift k-bit operation for $x \in \mathbb{Z}_L$ as $x \gg k$. For $\mathsf{int}(x) \in \mathbb{Z}$, the truncation operation can be expressed as $x \gg k = \lfloor \frac{\mathsf{int}(x)}{2^k} \rfloor$. From Equation 1, the truncation operation can be computed as:

$$\begin{aligned} x \gg k &=_L \lfloor \frac{x_0 + x_1 - \mathsf{MW}(x) \cdot L}{2^k} \rfloor \\ &= \lfloor \frac{x_0}{2^k} \rfloor + \lfloor \frac{x_1}{2^k} \rfloor - \mathsf{MW}(x) \cdot 2^{l-k} + \delta \end{aligned}, \tag{3}$$

where $\delta = \{x_0 \mod 2^k + x_1 \mod 2^k > 2^k\} = \mathsf{Wrap}(x_0 \mod 2^k, x_1 \mod 2^k, 2^k) \in \{0,1\}$, and $\mathsf{MW}(x)$ is defined as Equation 2. The proof of Equation 3 is provided in Appendix C.1. Therefore, the essential step in computing $x \gg k$ involves computing $\mathsf{MW}(x)$ and δ. Additionally, in some scenarios where 1-bit errors are tolerated, δ can be disregarded, resulting in the computation of $x \gg k - \delta$, known as one-bit error truncation.

3.4 Signed Extension protocol

The extension protocol extends an m-bit number $x \in \mathbb{Z}_M$ to an n-bit number $y \in \mathbb{Z}_N$, where $M = 2^m$, $N = 2^n$, and $m < n$. SirNN [23] proposed zero and signed extension protocols for extending the bitwidths of unsigned and signed numbers, respectively.

For $x \in \mathbb{Z}_M$, the zero extension is defined as $y = \mathsf{ZExt}(x,m,n) \in \mathbb{Z}_N$, where $\mathsf{uint}(x) = \mathsf{uint}(y)$. While the signed extension is $y = \mathsf{SExt}(x,m,n) \in \mathbb{Z}_N$, where $\mathsf{int}(x) = \mathsf{int}(y)$. For $x = x_0 + x_1 \mod M$, the zero extension protocol outputs $\mathsf{ZExt}(x,m,n) = x_0 + x_1 - \mathsf{Wrap}(x_0, x_1, M) \cdot M$. The signed extension protocol can be computed as $\mathsf{SExt}(x,m,n) = \mathsf{ZExt}(x',m,n) - 2^{m-1}$, where $x' = x + 2^{m-1} \mod M$, with the same overhead as the zero extension protocol.

3.5 Signed Multiplication with non-uniform bitwidths

For $x \in \mathbb{Z}_M$ and $y \in \mathbb{Z}_N$, where $M = 2^m$ and $N = 2^n$, the non-uniform multiplication protocol computes $z = xy \in \mathbb{Z}_{MN}$. SirNN [23] initially proposed an unsigned non-uniform multiplication protocol to compute $z = \mathsf{uint}(x) \cdot \mathsf{uint}(y)$ as:

$$\begin{aligned} \mathsf{uint}(x) \cdot \mathsf{uint}(y) &= (x_0 + x_1 - w_x \cdot 2^m) \cdot (y_0 + y_1 - w_y \cdot 2^n) \\ &= x_0 y_0 + x_1 y_1 + x_0 y_1 + x_1 y_0 - 2^m \cdot w_x y \\ &\quad - 2^n \cdot w_y x - 2^{m+n} \cdot w_x \cdot w_y, \end{aligned} \tag{4}$$

where $w_x = \mathsf{Wrap}(x_0, x_1, 2^m)$ and $w_y = \mathsf{Wrap}(y_0, y_1, 2^n)$. The main overhead lies in computing the cross terms and the Wrap function. Then they propose the signed non-uniform multiplication protocol based on unsigned non-uniform multiplication protocol, with the same overhead.

4 Computing MW Using Geometric Method

4.1 Two-party Secret sharing from Geometric Perspective

For $x = x_0 + x_1 \mod L$, we regard $P(x_0, x_1)$ as a point in the Cartesian coordinate system, where x_0 is the abscissa and x_1 the ordinate. According to the definition $\mathsf{MSB}(x) = \mathbf{1}\{x \geq \frac{L}{2}\}$, we have that $\mathsf{MSB}(x) = 0$ if and only if (note that there is no mod L operation):

$$0 \leq x_0 + x_1 < \tfrac{L}{2} \quad \text{or} \quad L \leq x_0 + x_1 < L + \tfrac{L}{2}$$

and $\mathsf{MSB}(x) = 1$ if and only if:

$$\tfrac{L}{2} \leq x_0 + x_1 < L \quad \text{or} \quad L + \tfrac{L}{2} \leq x_0 + x_1 < 2L.$$

The feasible region for the point $P(x_0, x_1)$ is depicted in Figure 2(a), where $\mathsf{MSB}(x) = 0$ if and only if point $P(x_0, x_1)$ falls into the pink area, and $\mathsf{MSB}(x) = 1$ if and only if P falls into the green area.

In many practical application scenarios, although x is shared on $x \in \mathbb{Z}_L$, the actual data range might be smaller and bounded by L_x. Specifically, the signed input $\mathsf{int}(x) \in \mathbb{Z}$ is within the interval $[-L_x, L_x)$, or denoted as $|\mathsf{int}(x)| < L_x$ or $|x| < L_x$, where $L_x \leq \frac{L}{2}$. Therefore, $x \in [0, L_x) \bigcup [L - L_x, L)$, where the first interval represents $\mathsf{int}(x)$ being positive and less than L_x, while the second interval represents $\mathsf{int}(x)$ being negative but not less than $-L_x$. In this case, $\mathsf{MSB}(x) = 0$ if and only if

$$0 \leq x_0 + x_1 < L_x \quad \text{or} \quad L \leq x_0 + x_1 < L + L_x$$

and $\mathsf{MSB}(x) = 1$ if and only if

$$L - L_x \le x_0 + x_1 < L \quad \text{or} \quad 2L - L_x \le x_0 + x_1 < 2L.$$

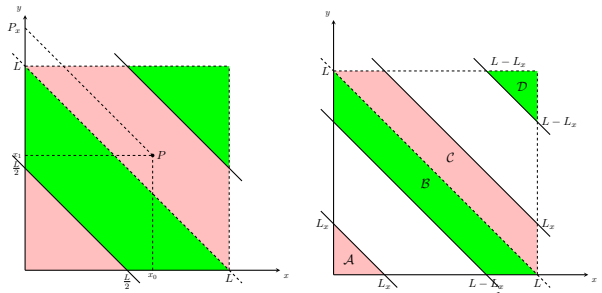
The feasible regions for P when $|x| < L_x$ and $L_x < \frac{L}{2}$ are shown in Figure 2(b), where the four possible regions for P are defined as:

$$\mathcal{A} : \{(x_0, x_1) | x_0, x_1 \in \mathbb{Z}_L \ , \ 0 \le x_0 + x_1 < L_x\},$$
$$\mathcal{B} : \{(x_0, x_1) | x_0, x_1 \in \mathbb{Z}_L \ , \ L - L_x \le x_0 + x_1 < L\},$$
$$\mathcal{C} : \{(x_0, x_1) | x_0, x_1 \in \mathbb{Z}_L \ , \ L \le x_0 + x_1 < L + L_x\},$$
$$\mathcal{D} : \{(x_0, x_1) | x_0, x_1 \in \mathbb{Z}_L \ , \ 2L - L_x \le x_0 + x_1 < 2L\}.$$

From Figure 2(b) we can deduce that $\mathsf{MSB}(x) = 0$ if and only if $P \in \mathcal{A} \bigcup \mathcal{C}$, and $\mathsf{MSB}(x) = 1$ if and only if $P \in \mathcal{B} \bigcup \mathcal{D}$. Further, since $\mathsf{Wrap}(x_0, x_1, L) = \mathbf{1}\{x_0 + x_1 \ge L\}$, the Wrap function determines whether the point P lies above or below the line $y = -x + L$, where $\mathsf{Wrap}(x) = 1$ if $P \in \mathcal{C} \bigcup \mathcal{D}$ and $\mathsf{Wrap}(x) = 0$ if $P \in \mathcal{A} \bigcup \mathcal{B}$. Thus, the $\mathsf{MW}(x)$ can be computed by determining the area P falls into, where

$$\mathsf{MW}(x) = \begin{cases} 0, & \text{if } P \in \mathcal{A} \\ 1, & \text{if } P \in \mathcal{B} \bigcup \mathcal{C} \\ 2, & \text{if } P \in \mathcal{D} \end{cases} . \tag{5}$$

Then the problem of computing $\mathsf{MW}(x)$ is reduced to determining the area in which P lies.



(a) General $x \in \mathbb{Z}_L$ (b) $|x| < 2^{l_x}$

Figure 2: Feasible region of $P(x_0, x_1)$ from a geometric perspective.

4.2 Computing MW When $|x| < \frac{L}{3}$

From Figure 2(b), it can be observed that if $L_x \ll L$, then $P \in \mathcal{A}$ if and only if $x_0 < L_x$ and $x_1 < L_x$. Therefore, in this case, only one AND operation is required to determine whether $P \in \mathcal{A}$. Based on this observation, we set $L_x = \frac{L}{3}$, then the feasible region of (x_0, x_1) for $|x| < L_x$ is shown in Figure 3. We define

$$\begin{cases} a = \mathbf{1}\{x_0 < \frac{L}{3}\} \wedge \mathbf{1}\{x_1 < \frac{L}{3}\} \\ d = \mathbf{1}\{x_0 \ge \frac{2L}{3}\} \wedge \mathbf{1}\{x_1 \ge \frac{2L}{3}\} \end{cases} , \tag{6}$$

then from Figure 3 we can intuitively deduce that $a = 1$ if and only if $x_0 < \frac{L}{3}$ and $x_1 < \frac{L}{3}$, indicating $P \in \mathcal{A}$. Similarly, $d = 1$ if and only if $x_0 \ge \frac{2L}{3}$ and $x_1 \ge \frac{2L}{3}$, indicating $P \in \mathcal{D}$. If $a = d = 0$, then $P \in \mathcal{B} \bigcup \mathcal{C}$. Theorem 1 formalizes this conclusion, with the proof provided in Appendix C.2.

Theorem 1. *For $x = x_0 + x_1 \mod L$ and $|x| < \frac{L}{3}$, we have $x_0 + x_1 \in [0, \frac{L}{3}) \bigcup [\frac{2L}{3}, \frac{4L}{3}) \bigcup [\frac{5L}{3}, 2L)$. Furthermore, $x_0 + x_1 \in [0, \frac{L}{3})$ if and only if $x_0 < \frac{L}{3}$ and $x_1 < \frac{L}{3}$; $x_0 + x_1 \in [\frac{5L}{3}, 2L)$ if and only if $x_0 \ge \frac{2L}{3}$ and $x_1 \ge \frac{2L}{3}$; otherwise, $x_0 + x_1 \in [\frac{2L}{3}, \frac{4L}{3})$.*

Based on Theorem 1, we can determine the range of $x_0 + x_1$ or the area P falls into by computing the values of a and d in Equation 6. Then the $\mathsf{MW}(x)$ can be computed as $\mathsf{MW}(x) = 1 - a + d$, which requires only two AND operations but not the costly comparison protocol. We propose new protocol $\Pi^{l,l'}_{\mathsf{MW1}}$ to compute $[\![\mathsf{MW}(x, L)]\!]^{L'}$ with input $[\![x]\!]^L$, where $L' = 2^{l'}$. The details are provided in Algorithm 1. Since P_0 holds $\mathbf{1}\{x_0 < \frac{L}{3}\}$ and $\mathbf{1}\{x_0 \ge \frac{2L}{3}\}$, and P_1 holds $\mathbf{1}\{x_1 < \frac{L}{3}\}$ and $\mathbf{1}\{x_1 \ge \frac{2L}{3}\}$, we can invoke the $\Pi^{l'}_{\mathsf{BitMul}}$ to compute $[\![a]\!]^{L'}$ and $[\![d]\!]^{L'}$. Finally, we can compute $[\![\mathsf{MW}(x, L)]\!]^{L'} = 1 - [\![a]\!]^{L'} + [\![d]\!]^{L'}$.

Algorithm 1: Computing $\mathsf{MW}(x, L)$ with $|x| < \frac{L}{3}$, $\Pi^{l,l'}_{\mathsf{MW1}}$:

Input: P_0 and P_1 hold $[\![x]\!]^L$ where $|x| < \frac{L}{3}$ and $L = 2^l$.

Output: P_0 and P_1 output $[\![\mathsf{MW}(x, L)]\!]^{L'}$ where $\mathsf{MW}(x, L) = \mathsf{Wrap}(x_0, x_1, L) + \mathsf{MSB}(x)$.

1 P_0 and P_1 invoke $\Pi^{l'}_{\mathsf{BitMul}}$ with input $x_0 < \frac{L}{3}$ and $x_1 < \frac{L}{3}$ to learn $[\![a]\!]^{L'}$.

2 P_0 and P_1 invoke $\Pi^{l'}_{\mathsf{BitMul}}$ with input $x_0 \ge \frac{2L}{3}$ and $x_1 \ge \frac{2L}{3}$ to learn $[\![d]\!]^{L'}$.

3 P_0 and P_1 output $1 - [\![a]\!]^{L'} + [\![d]\!]^{L'}$.

Correctness and security. The correctness of $\Pi^{l,l'}_{\mathsf{MW1}}$ is ensured by Equation 2 and Theorem 1. The only interaction between P_0 and P_1 occurs in performing $\Pi^{l'}_{\mathsf{BitMul}}$. Therefore, the security of $\Pi^{l,l'}_{\mathsf{MW1}}$ relies on the security of $\Pi^{l'}_{\mathsf{BitMul}}$.

Complexity. $\Pi^{l,l'}_{\mathsf{MW1}}$ calls $\Pi^{l'}_{\mathsf{BitMul}}$ twice in parallel to obtain the shared value of $\mathsf{MW}(x)$ over the ring $\mathbb{Z}_{L'}$, with a total communication $2\lambda + 2l'$ in 2 rounds.

4.2.1 Why limit $|x| < \frac{L}{3}$

In Theorem 1 and Algorithm 1, we constrain the input range to $|x| < \frac{L}{3}$. We now explain why this constraint can not be further relaxed. When $a = 1$, the point P actually falls in the square region $S_{\mathcal{A}}$ with vertices at $(0,0), (0, \frac{L}{3}), (\frac{L}{3}, 0)$ and $(\frac{L}{3}, \frac{L}{3})$, as indicated by the red square in the bottom left corner of Figure 3. However, since we limit $|x| < \frac{L}{3}$, it follows that $x_0 + x_1 < \frac{L}{3}$, restricting P to the triangular area with vertices at $(0,0), (0, \frac{L}{3})$ and $(\frac{L}{3}, 0)$, ensuring that P falls within area $\mathcal{A}$.

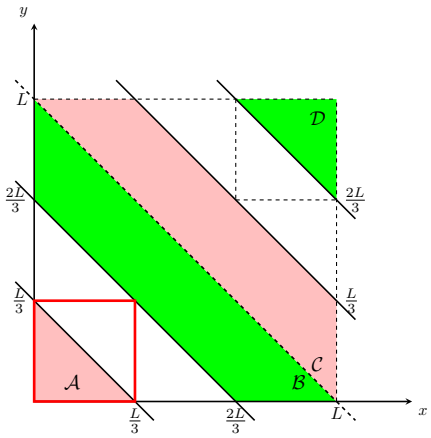


Figure 3: Feasible region of $P(x_0, x_1)$ for $|x| < \frac{L}{3}$.

To ensure $P \in \mathcal{A}$ when $a = 1$, the square region $S_{\mathcal{A}}$ must not overlap with any other area, specifically area $\mathcal{B}$. Therefore, we impose the constraint $|x| < \frac{L}{3}$. Otherwise, if this constraint is relaxed to $|x| < L'_x$ where $L'_x > \frac{L}{3}$, then $a = \mathbf{1}\{x_0 < L'_x\} \wedge \mathbf{1}\{x_1 < L'_x\} = 1$ when $x_0 = x_1 = \frac{L}{3}$. While point $(\frac{L}{3}, \frac{L}{3})$ falls in area $\mathcal{B}$, leading to misclassification.

4.3 Computing MW When $|x| < \frac{L}{4}$

The feasible region for $P(x_0, x_1)$ under the constraint $|x| < L_x$ consists of three distinct areas: $\mathcal{A}$, $\mathcal{B} \bigcup \mathcal{C}$, and $\mathcal{D}$. As a result, $\mathsf{MW}(x)$ is a two-bit value, requiring two AND operations. However, by translating P to the left by $L_x \pmod L$, we get a new point $P^*(x_0^*, x_1)$ where the feasible region of P^* is reduced to two distinct areas. In this scenario, $\mathsf{MW}(x^*)$ becomes a one-bit value and may be computed with only one AND operation. In the remainder of this subsection, we demonstrate how to compute $\mathsf{MW}(x^*)$ and subsequently derive $\mathsf{MW}(x)$ from $\mathsf{MW}(x^*)$.

By limiting $|x| < \frac{L}{4}$, the feasible regions for $P(x_0, x_1)$ and $P^*(x_0^*, x_1)$ where $x_0^* = x_0 - \frac{L}{4} \mod L$ are shown in Figure 4(a) and Figure 4(b), respectively. According to the definition of MW in Equation 2 and Figure 4(b), we have that $\mathsf{MW}(x^*) = 1$ if and only if $P^* \in \mathcal{B}' \bigcup \mathcal{C}'$, and $\mathsf{MW}(x^*) = 2$ if and only if $P^* \in \mathcal{D}' \bigcup \mathcal{E}'$. Additionally, Figure 4(b) indicates that $P^* \in \mathcal{D}' \bigcup \mathcal{E}'$ if and only if $x_0^* \geq \frac{L}{2}$ and $x_1 \geq \frac{L}{2}$ since $|x| < \frac{L}{4}$. Therefore, for $|x| < \frac{L}{4}$, $\mathsf{MW}(x^*)$ can be computed as $\mathsf{MW}(x^*) = \mathbf{1}\{x_0^* \geq \frac{L}{2}\} \wedge \mathbf{1}\{x_1 \geq \frac{L}{2}\} + 1$, requiring only one AND operation. Finally, we give the following Theorem 2 to show how to compute $\mathsf{MW}(x)$ from $\mathsf{MW}(x^*)$. And the proof of this theorem is provided in Appendix C.3.

Theorem 2. *For $x = x_0 + x_1 \mod L$ and $|x| < \frac{L}{4}$, if $x_0^* = x_0 - \frac{L}{4} \mod L$ and $x^* = x_0^* + x_1 \mod L$, then $\mathsf{MW}(x) = \mathsf{MW}(x^*) - \mathbf{1}\{x_0 < \frac{L}{4}\}$.*

Based on Theorem 2, we propose Algorithm 2 to compute $\mathsf{MW}(x)$, where we limit $|x| < \frac{L}{4}$ and requiring only one call to Π_{BitMul}.

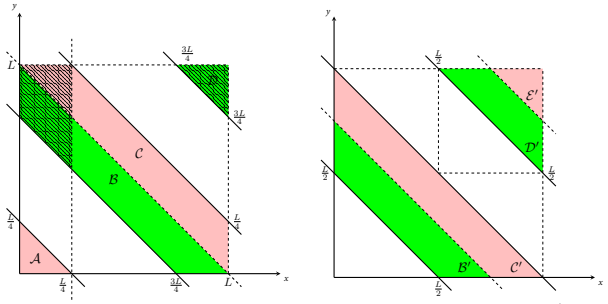
Algorithm 2: Computing $\mathsf{MW}(x, L)$ with $|x| < \frac{L}{4}$, $\Pi^{l,l'}_{\mathsf{MW2}}$:

Input: P_0 and P_1 hold $[\![x]\!]^L$ where $|x| < \frac{L}{4}$ and $L = 2^l$.
Output: P_0 and P_1 output $[\![\mathsf{MW}(x, L)]\!]^{L'}$.

1 P_0 sets $x_0^* = (x_0 - \frac{L}{4}) \mod L$.
2 P_0 and P_1 invoke $\Pi^{l'}_{\mathsf{BitMul}}$ with input $\mathbf{1}\{x_0^* \geq \frac{L}{2}\}$ and $\mathbf{1}\{x_1 \geq \frac{L}{2}\}$ to learn $[\![d^*]\!]^{L'}$.
3 P_0 output $[\![d^*]\!]^{L'}_0 + 1 - \mathbf{1}\{x_0 < \frac{L}{4}\} \mod L'$ and P_1 output $[\![d^*]\!]^{L'}_1$.

Correctness and security. The correctness of $\Pi^{l,l'}_{\mathsf{MW2}}$ is ensured by Theorem 2, and the security comes from the security of Π_{BitMul}.

Complexity. $\Pi^{l,l'}_{\mathsf{MW2}}$ requires only a single call to $\Pi^{l'}_{\mathsf{BitMul}}$, resulting in a total communication cost of $\lambda + l'$ in 2 rounds.



(a) Feasible region of P. (b) Feasible region of P^*.

Figure 4: Feasible region of $P(x_0, x_1)$ and $P^*(x_0^*, x_1)$ for $|x| < \frac{L}{4}$, where $x_0^* = x_0 - \frac{L}{4} \mod L$.

4.3.1 Why set $|x| < \frac{L}{4}$

Now we interpret why we set the bound B as $B = \frac{L}{4}$ and it can not be larger. Intuitively, from Figure 4(b) we can deduce that if $B > \frac{L}{4}$, then there will intersection between the dashed box area in the upper right corner and area $\mathcal{C}$. For example, for point $(x_0^*, x_1) = (\frac{L}{2} - 1, \frac{L}{2})$, $\mathsf{MW}(x^*)$ should be 1 as it falls into area $\mathcal{C}$, while $\mathbf{1}\{x_0^* \geq \frac{L}{2}\} \wedge \mathbf{1}\{x_1 \geq \frac{L}{2}\} + 1 = 2$, therefore a misclassification occurs. Formally, for $x = x_0 + x_1 \mod L$ and $|x| < B$, let $x^* = x_0 - B \mod L$, then $x^* + x_1 \in [L - 2B, L) \bigcup [2L - 2B, 2L)$. We define $d^* = \mathbf{1}\{x_0^* \in [L - 2B, L)\} \wedge \mathbf{1}\{x_1 \in [L - 2B, L)\}$, then to prevent misclassification, the minimum $x_0^* + x_1$ that makes $d^* = 1$ must exceed the maximum $x_0^* + x_1$ that makes $d^* = 0$, which yields $B \leq \frac{L}{4}$ and $|x| < \frac{L}{4}$.

4.4 Computing MW Probabilistically Locally

For $|x| < L_x$, an intuitive observation from Figure 2(b) is that if $L_x \ll L$, then the areas of $\mathcal{A}$ and $\mathcal{D}$ will be significantly small, making it highly probable that P will fall within $\mathcal{B} \cup \mathcal{C}$. Consequently, we can assume that P always falls within $\mathcal{B} \bigcup \mathcal{C}$ and directly set $\mathsf{MW}(x) = 1$ and $\mathsf{int}(x) = x_0 + x_1 - L$. This introduces a small failure probability as the assumption fails when $P \in \mathcal{A} \bigcup \mathcal{D}$. We give an accurate probability for $P \in \mathcal{B} \bigcup \mathcal{C}$ or $\mathsf{MW}(x) = 1$ for a given $|x| < L_x$ in Theorem 3, and the proof is shown in Appendix C.4.

Theorem 3. *For a given input $x = x_0 + x_1 \mod L$ satisfying $|x| < L_x$ where $L_x \leq \frac{L}{2}$, the probability that $x_0 + x_1 \in [L - L_x, L + L_x)$ is $1 - \frac{|\mathsf{int}(x)+1|}{L}$.*

The failure probability in Theorem 3 is $\frac{|\mathsf{int}(x)+1|}{L}$, which is bounded by $\frac{L_x+1}{L}$ for $|x| < L_x$. Therefore, in some scenarios, we can set a sufficiently small L_x and large enough L to make the failure probability negligible, allowing us to directly output $\mathsf{MW}(x) = 1$ without communication.

4.5 Computing MW with Known MSB

In certain scenarios, the MSB of the input is known in clear or secret sharing form. In these cases, computing $\mathsf{MW}(x)$ reduces to computing $\mathsf{Wrap}(x)$ from $\mathsf{MSB}(x)$. SirNN [23] proposed an *MSB-to-Wrap* protocol to compute $\mathsf{Wrap}(x)$ from $\mathsf{MSB}(x)$ as follows:

$$\mathsf{Wrap}(x_0, x_1, L) = ((1 \oplus \mathsf{MSB}(x)) \wedge (m_0 \oplus m_1)) \oplus (m_0 \wedge m_1), \tag{7}$$

where m_i is the most significant bit of x_i for $i \in \{0, 1\}$. Therefore, when $\mathsf{MSB}(x)$ is known in shared form, two Π_{AND} are required to compute $[\![\mathsf{Wrap}]\!]^B$ along with two $\Pi^{l'}_{\mathsf{B2A}}$ to compute $[\![\mathsf{MW}(x)]\!]^{L'} = [\![\mathsf{Wrap}(x)]\!]^{L'} + [\![\mathsf{MSB}(x)]\!]^{L'}$, where $L' = 2^{l'}$. The total communication cost is $4\lambda + 2l' + 40$ in 4 rounds. For the case $\mathsf{MSB}(x)$ is known in clear, only one $\Pi^{l'}_{\mathsf{BitMul}}$ is needed to compute $[\![\mathsf{Wrap}(x)]\!]^{L'}$, with a total communication cost $\lambda + l'$ in 2 rounds.

5 Application to Practical Protocols

In this section, we propose new truncation, signed extension and signed multiplication with non-uniform bitwidth protocols using the geometric method in Section 4.

5.1 New Truncation Protocols

We begin with the truncation protocol, where the truncation operation is computed as Equation 3. Moreover, the one-bit error truncation operation can be written as

$$x \gg k - \delta =_L = \lfloor \frac{x_0}{2^k} \rfloor + \lfloor \frac{x_1}{2^k} \rfloor - \mathsf{MW}(x) \cdot 2^{l-k}. \tag{8}$$

Thus, the one-bit error truncation protocol can be implemented by computing $\mathsf{MW}(x)$, as discussed in Section 4.

5.1.1 One-bit Error Truncation Protocol with Constraint

We propose two new one-bit error truncation protocol Π^k_{trun1} and Π^k_{trun1}, with constraints $|x| < \frac{L}{3}$ and $|x| < \frac{L}{4}$, where we can invoke Π_{MW1} and Π_{MW2} to compute $\mathsf{MW}(x)$, respectively. The details are listed in Algorithm 3 and Algorithm 4. Note that only $[\![\mathsf{MW}]\!]^{2^k}$ but not $[\![\mathsf{MW}]\!]^L$ needs to be computed, as $[\![\mathsf{MW}]\!]^{2^k} \cdot \frac{L}{2^k} =_L [\![\mathsf{MW}]\!]^l \cdot \frac{L}{2^k}$.

The correctness of both Π^k_{trun1} and Π^k_{trun2} come from Equation 8, where $\mathsf{MW}(x)$ is computed based on Π^k_{MW1} and Π^k_{MW2}, respectively. Therefore the security of these two protocols comes from the security of Π^k_{MW1} and Π^k_{MW2}. Moreover, Π^k_{MW1} and Π^k_{MW2} need two and one calls for Π^k_{BitMul}, resulting the communication costs of Π^k_{trun1} and Π^k_{trun2} to be $2\lambda + 2k$ and $\lambda + k$, and both need two rounds of communication. It is worth noting that the communication cost of our one-bit error truncation is independent of the input length l.

Algorithm 3: One-bit error truncation with constraint $|x| < \frac{L}{3}$, Π^k_{trun1}:

Input: P_0 and P_1 hold $[\![x]\!]^L$ where $|x| < \frac{L}{3}$ and $L = 2^l$.
Output: P_0 and P_1 get $[\![x \gg k]\!]^L$ with one-bit error.

1 P_0 and P_1 invoke Π^k_{MW1} with input $[\![x]\!]^L$ to learn $[\![\mathsf{MW}]\!]^{2^k}$.
2 P_0 and P_1 output $\lfloor \frac{x_0}{2^k} \rfloor + \lfloor \frac{x_1}{2^k} \rfloor - [\![\mathsf{MW}]\!]^{2^k} \cdot \frac{L}{2^k}$.

Algorithm 4: One-bit error truncation with constraint $|x| < \frac{L}{4}$, Π^k_{trun2}:

Input: P_0 and P_1 hold $[\![x]\!]^L$, where $|x| < \frac{L}{4}$ and $L = 2^l$.
Output: P_0 and P_1 get $[\![x \gg k]\!]^L$ with one-bit error.

1 P_0 and P_1 invoke Π^k_{MW2} with input $[\![x]\!]^L$ to learn $[\![\mathsf{MW}]\!]^{2^k}$.
2 P_0 and P_1 output $\lfloor \frac{x_0}{2^k} \rfloor + \lfloor \frac{x_1}{2^k} \rfloor - [\![\mathsf{MW}]\!]^{2^k} \cdot \frac{L}{2^k}$.

5.1.2 One-bit Error Truncation Protocol with Known MSB

When MSB is known, we can apply the method in Section 4.5 to compute $\mathsf{MW}(x)$, followed by using Equation 8 to perform the one-bit error truncation protocol. Recently, Cheetah [13] proposed a one-bit error truncation protocol with known MSB in clear, involving invoking a Π_{AND} to compute $\mathsf{MSB}(x_0) \wedge \mathsf{MSB}(x_1)$, followed by a Π^k_{B2A} to convert the result to algebraic sharing. In contrast, our method directly computes the algebraic sharing of $\mathsf{MSB}(x_0) \wedge \mathsf{MSB}(x_1)$ using

Π^k_{BitMul}. This reduces the communication cost from $2\lambda+2+k$ to $\lambda+k$ and eliminates 2 rounds of communication.

For the case $[\![\mathsf{MSB}]\!]^B$ is known, $\mathsf{MW}(x)$ can be computed using Equation 7, and the one-bit truncation protocol can be implemented using Equation 8. The total communication for this method is $3\lambda+k+40$ in 4 rounds.

5.1.3 Faithful Truncation Protocol

We analyzed the one-bit error truncation protocols in the previous subsections, which output $[\![x \gg k]\!]^L - \delta$. To implement the faithful truncation protocol, we need to compute $\delta = \mathsf{Wrap}(x_0 \mod 2^k, x_1 \mod 2^k, 2^k)$, where a comparison protocol with k-bit input is invoked. We construct the faithful truncation protocol based on Π_{trun2} which limits $|x| < \frac{L}{4}$, as detailed in Algorithm 5. Similarly, the faithful truncation protocol can also be implemented based on Π_{trun1} or the one-bit error truncation protocol with known MSB, depending on the range of x.

Algorithm 5: Faithful truncation with constraint $|x| < \frac{L}{4}$, $\Pi^k_{\mathsf{trun_f}}$:

Input: P_0 and P_1 hold $[\![x]\!]^L$ where $|x| < \frac{L}{4}$ and $L = 2^l$.
Output: P_0 and P_1 get $[\![x \gg k]\!]^L$.

1. P_0 and P_1 invoke Π^k_{trun2} to learn $[\![t]\!]^L$.
2. P_0 and P_1 invoke Π^k_{Mill} with input $x_0 \mod 2^k$ and $2^k - (x_1 \mod 2^k)$ to learn output $[\![\varepsilon]\!]^B$.
3. P_0 and P_1 invoke Π^l_{B2A} with input $[\![\varepsilon]\!]^B \oplus 1$ to learn $[\![\delta]\!]^L$.
4. P_0 and P_1 output $[\![t]\!]^L + [\![\delta]\!]^L$.

Correctness and Security. The correctness of $\Pi^k_{\mathsf{trun_f}}$ comes from Equation 3, and the security comes from the security of Π^k_{trun2}, Π^k_{Mill} and Π^l_{B2A}.

Complexity. $\Pi^k_{\mathsf{trun_f}}$ need one call to Π^k_{trun2}, one call to Π^k_{Mill} and one call to Π^l_{B2A}. The total communication is $< \lambda(k+2)+l+15k$ in $\log k+2$ rounds as the Π_{trun2} and Π_{Mill} can be implemented in parallel.

5.1.4 Probabilistic Truncation Protocol

Using the idea in Section 4.4, we can assume $\mathsf{MW}(x) = 1$ and compute $x \gg k - \delta = \lfloor \frac{x_0}{2^k} \rfloor + \lfloor \frac{x_1}{2^k} \rfloor - 2^{l-k}$ locally. The failure probability of this probabilistic truncation protocol is $\frac{|\mathsf{int}(x)+1|}{L}$ according to Theorem 3, which is bounded by 2^{l_x-l} for $x \in \mathbb{Z}_{2^l}$ with constraint $|x| < 2^{l_x}$.

The local truncation protocol proposed in SecureML [21] is the same as ours. However, they estimated the failure probability to be bounded by 2^{l_x+1-l}. In their proof, they introduce a variable r, and set $x_0 = x + r \mod L$ and $x_1 = L - r$, then demonstrate the local truncation protocol holds when $2^{l_x} \le r < L - 2^{l_x}$, yielding a failure probability of 2^{l_x-l+1}. However, their proof overlooks the fact that for a given input x, the sign of x is fixed. Therefore, it is only necessary to ensure either $2^{l_x} \le r$ or $r < L - 2^{l_x}$. We address this oversight and give a more accurate failure probability for $\Pi_{\mathsf{local\_trun}}$, which is half of SecureML's estimate.

Security issue. In USENIX'23, Li et al. [16] identified a security issue in the probabilistic truncation protocol used in SecureML and also ours, demonstrating that the views of an ideal adversary and a corrupted party are not indistinguishable. However, the probabilistic truncation protocols in both SecureML and ours are performed locally. Therefore, this security issue stems from the overly strict definition of the ideal probabilistic truncation functionality (Functionality 2 in [16]) adopted in their work. To address this issue, we redefine the ideal probabilistic truncation functionality $\mathcal{F}_{\mathsf{trunPr}}$ as follows: $\mathcal{F}_{\mathsf{trunPr}}$ receive x_0 and x_i from P_0 and P_1, respectively. Then it computes $t_0 = \lfloor \frac{x_0}{2^d} \rfloor$ and $t_1 = L - \lfloor \frac{L-x_1}{2^d} \rfloor$ and sends t_i to P_i for $i \in \{0,1\}$. Under this revised definition, the probabilistic truncation protocol is clearly secure.

5.2 New Signed Extension Protocol

Similar to the truncation protocol, the signed extension protocol can be implemented by computing $\mathsf{MW}(x)$. We propose a new method to implement the signed extension protocol based on the following Lemma 1.

Lemma 1. *For $x = x_0 + x_1 \mod M$, suppose $y = y_0 + y_1 \mod N$ where $N > M$, and*

$$(y_0, y_1) = \begin{cases} (x_0, x_1), & \text{if } x_0 + x_1 \in [0, \frac{M}{2}) \\ (x_0, x_1 + N - M), & \text{if } x_0 + x_1 \in [\frac{M}{2}, \frac{3M}{2}) \\ (x_0 + N - M, x_1 + N - M), & \text{if } x_0 + x_1 \in [\frac{3M}{2}, 2M) \end{cases}.$$

Then $\mathsf{int}(x) = \mathsf{int}(y)$.

The proof of Lemma 1 is provided in Appendix C.5. Based on Lemma 1, for $x \in \mathbb{Z}_M$ and $y \in \mathbb{Z}_N$, the signed extension protocol can be computed as:

$$y =_N x_0 + x_1 + \mathsf{MW}(x_0, x_1, M) \cdot (N - M). \tag{9}$$

Thus, the signed extension protocol can be performed by computing $\mathsf{MW}(x_0, x_1, M)$ as described Section 4. The details of protocols for signed extension under constraints $|x| < \frac{M}{3}$, $|x| < \frac{M}{4}$, and probabilistic signed extension protocols are provided in Appendix B.

5.3 Application to Signed Multiplication with Non-uniform Bitwidths

In this section, we study the signed non-uniform multiplication protocol, which takes as input $[\![x]\!]^M$ and $[\![y]\!]^N$, where $x = \mathsf{uint}(x) = x_0 + x_1 \mod M$, $y = \mathsf{uint}(y) = y_0 + y_1 \mod N$, $M = 2^m$ and $N = 2^n$, and outputs $[\![z]\!]^{MN}$ satisfying

$\mathsf{int}(z) = \mathsf{int}(x) \cdot \mathsf{int}(y)$. Using Equation 1, we can compute $\mathsf{int}(x) \cdot \mathsf{int}(y)$ as:

$$\begin{aligned}\mathsf{int}(x) \cdot \mathsf{int}(y) &= (x_0 + x_1 - M_x \cdot 2^m) \cdot (y_0 + y_1 - M_y \cdot 2^n) \\ &= x_0y_0 + x_1y_1 + x_0y_1 + x_1y_0 - 2^m \cdot M_x \mathsf{int}(y) \\ &\quad - 2^n \cdot M_y \mathsf{int}(x) - 2^{m+n} \cdot M_x \cdot M_y \\ &= x_0y_0 + x_1y_1 + x_0y_1 + x_1y_0 - 2^m M_x y - 2^n M_y x \\ &\quad + 2^{m+n}(M_x \cdot \mathsf{MSB}(y) + M_y \cdot \mathsf{MSB}(x) - M_x \cdot M_y)\end{aligned} \tag{10}$$

where $M_x = \mathsf{MW}(x_0, x_1, M)$ and $M_y = \mathsf{MW}(y_0, y_1, N)$. In this equation, the term $x_i y_i$ for $i \in \{0,1\}$ can be computed locally, and taking the result modulo 2^{m+n} eliminates the last term. We follow the method from SirNN to compute the cross-terms $x_i y_{1-i}$ for $i \in \{0,1\}$ by invoking two instances of $\Pi^{m,n}_{\mathsf{CrossTerm}}$. The details of $\Pi^{m,n}_{\mathsf{CrossTerm}}$ are provided in Appendix D. The primary difference between our method and SirNN's is that we compute $\mathsf{MW}(x)$ and $\mathsf{MW}(y)$ instead of $\mathsf{Wrap}(x)$ and $\mathsf{Wrap}(y)$, thereby avoiding the need for costly comparison protocols.

Another challenge is that $\mathsf{MW}(x), \mathsf{MW}(y) \in \{0,1,2\}$ are two-bit values, therefore we can not use the Π_{MUX} to compute $\mathsf{MW}(x) \cdot y$ and $\mathsf{MW}(y) \cdot x$. Instead, we propose a new multiplexer protocol Π_{MUX3} with two-bit choice, which is then used in our signed non-uniform multiplication protocol.

5.3.1 Multiplexer Protocol with Two-bit Choice

We introduce a new multiplexer protocol Π^l_{MUX3} which takes as input arithmetic shares of a and c on ring $\mathbb{Z}_L$ and $\mathbb{Z}_4$, respectively, where $c \in \{0,1,2\}$. The protocol returns shares of $y = a \cdot c$ on the same ring $\mathbb{Z}_L$. We parse c as $c[0]||c[1]$, giving $y = 2c[0] \cdot a + c[1] \cdot a$ where $c[0], c[1] \in \{0,1\}$. This requires a digit decomposition protocol and two Π_{MUX} protocols to compute $a \cdot c$.

For a two-bit value c shared on $\mathbb{Z}_4$ as $c = [\![c]\!]^4_0 + [\![c]\!]^4_1 \mod 4$, it can be represented as $c = c[0]||c[1] = \mathsf{MSB}(c)||(c \mod 2)$. The boolean sharing of $c[1]$ can be computed locally as $c[1] = ([\![c]\!]^4_0 \mod 2) \oplus ([\![c]\!]^4_1 \mod 2)$. To compute $c[0] = \mathsf{MSB}(c)$, we use the DReLU or MSB protocol from CrypTFlow2 [24]. CrypTFlow2 computes $\mathsf{MSB}(c)$ as $\mathsf{MSB}(c) = \mathsf{msb}_0 \oplus \mathsf{msb}_1 \oplus \mathsf{carry}$ where $\mathsf{msb}_i = [\![c]\!]^4_i[0]$ for $i \in \{0,1\}$ and $\mathsf{carry} = \mathbf{1}\{[\![c]\!]^4_0[1] + [\![c]\!]^4_1[1] > 1\}$. However, since $[\![c]\!]^4_0[1]$ and $[\![c]\!]^4_1[0]$ are two one-bit values, we have $\mathbf{1}\{[\![c]\!]^4_0[1] + [\![c]\!]^4_1[1] > 1\} = [\![c]\!]^4_0[1] \wedge [\![c]\!]^4_1[1]$. As a result, only one Π_{AND} is needed to convert $[\![c]\!]^4$ to $[\![c]\!]^B$. We then perform our Π^l_{MUX3} to compute $y = c \cdot a$ as described in Algorithm 6.

Note that for the case $c = 2$, $y = 2a$ may overflow L, and the output of Π^l_{MUX3} is actually $2a \mod L$. However, this overflow does not affect the correctness of our multiplication protocol, as $2^m \cdot M_x \cdot y =_{2^{m+n}} 2^m \cdot (M_x \cdot y \mod 2^n)$.

Algorithm 6: Multiplexer protocol with two-bit choice, Π^l_{MUX3}:

Input: P_0 and P_1 hold $[\![a]\!]^L$ and $[\![c]\!]^4$ where $a \in \mathbb{Z}_L$, and $c \in \mathbb{Z}_4$ is a two-bit number.
Output: P_0 and P_1 output $[\![y]\!]^L$, where $y = c \cdot a \mod L$.

1 For $i \in \{0,1\}$, P_i parses $[\![c]\!]^4_i$ as $[\![c]\!]^4_i[0]||[\![c]\!]^4_i[1]$, and set $[\![c[1]]\!]^B_i = [\![c]\!]^4_i[1]$.
2 P_0 and P_1 invoke the Π_{AND} with input $[\![c]\!]^4_0[1]$ and $[\![c]\!]^4_1[1]$ to get $[\![\mathsf{carry}]\!]^B$.
3 P_0 and P_1 set $[\![c[0]]\!]^B$ as $[\![c]\!]^4_0[0] \oplus [\![c]\!]^4_1[0] \oplus [\![\mathsf{carry}]\!]^B$.
4 P_0 and P_1 invoke the Π^l_{MUX} with input $[\![2a]\!]^L$ and $[\![c[0]]\!]^B$ to learn $[\![t_0]\!]^L$.
5 P_0 and P_1 invoke the Π^l_{MUX} with input $[\![a]\!]^L$ and $[\![c[1]]\!]^B$ to learn $[\![t_1]\!]^L$.
6 P_0 and P_1 output $[\![t_0]\!]^L + [\![t_1]\!]^L$.

5.3.2 New Signed Multiplication with Non-uniform Bitwidths Protocol

Now we propose our signed non-uniform multiplication protocol $\Pi^{m,n}_{\mathsf{SMul}}$ based on Equation 10, and the details are shown in Algorithm 7. In this algorithm, we limit the input ranges as $|x| < \frac{M}{4}$ and $|y| < \frac{N}{4}$. Accordingly, we invoke two $\Pi^{L,4}_{\mathsf{Mw2}}$ to compute M_x and M_y in line 4-5. Next, two Π_{MUX3} are invoked to compute $g = M_y \cdot x$ and $h = M_x \cdot y$, followed by the computation of $N \cdot g$ and $M \cdot h$. The output is shared on ring $\mathbb{Z}_L$ where $L = 2^l$ and $l = m + n$.

Algorithm 7: Signed non-uniform multiplication protocol, $\Pi^{m,n}_{\mathsf{SMul}}$:

Input: P_0 and P_1 hold $[\![x]\!]^M$ and $[\![y]\!]^N$ where $M = 2^m$, $N = 2^n$, $|x| < \frac{M}{4}$ and $|y| < \frac{N}{4}$.
Output: P_0 and P_1 output $[\![xy]\!]^L$ where $L = 2^l$ and $l = m + n$.

1 P_0 and P_1 invoke the following protocols:
2 $\quad \Pi^{m,n}_{\mathsf{CrossTerm}}$ with input x_0, y_1 to learn $[\![c]\!]^L$.
3 $\quad \Pi^{m,n}_{\mathsf{CrossTerm}}$ with input x_1, y_0 to learn $[\![d]\!]^L$.
4 $\quad \Pi^{m,2}_{\mathsf{MW2}}$ with input $[\![x]\!]^M$ to learn $[\![M_x]\!]^4$.
5 $\quad \Pi^{n,2}_{\mathsf{MW2}}$ with input $[\![y]\!]^N$ to learn $[\![M_y]\!]^4$.
6 $\quad \Pi^{m}_{\mathsf{MUX3}}$ with input $[\![x]\!]^M$ and $[\![M_y]\!]^4$ to learn $[\![g]\!]^M$.
7 $\quad \Pi^{n}_{\mathsf{MUX3}}$ with input $[\![y]\!]^N$ and $[\![M_x]\!]^4$ to learn $[\![h]\!]^N$.
8 P_0 and P_1 output $x_0y_0 + x_1y_1 + [\![c]\!]^L + [\![d]\!]^L - N \cdot [\![g]\!]^M - M \cdot [\![h]\!]^N \mod 2^l$.

Correctness and Security. The correctness of $\Pi^{m,n}_{\mathsf{SMul}}$ comes from Equation 10, and the security comes from the security of $\Pi_{\mathsf{CrossTerm}}$, Π_{MW} and Π_{MUX3}.

Complexity. $\Pi^{m,n}_{\mathsf{SMul}}$ invokes two $\Pi^{m,n}_{\mathsf{CrossTerm}}$, two $\Pi^{L,4}_{\mathsf{MW2}}$, one $\Pi^{l,2,m}_{\mathsf{MUX3}}$ and one $\Pi^{l,2,n}_{\mathsf{MUX3}}$. The communication of two $\Pi^{m,n}_{\mathsf{CrossTerm}}$ is $\mu(2\lambda+\mu+1)+2mn$ where $\mu=\min(m,n)$, and the communication of the remaining sub-protocols is $12\lambda+4m+4n+44$. Therefore, the total communication is about $(2\mu+12)\lambda+\mu^2+\mu+2mn+4(m+n)$.

5.3.3 Signed Uniform Multiplication Protocol with Known MSB

SirNN [23] implements the multiplication protocol by first computing $\mathsf{uint}(x)\cdot\mathsf{uint}(y)$ as Equation 4, which requires calculating $w_x=\mathsf{Wrap}(x_0,x_1,2^m)$ and $w_y=\mathsf{Wrap}(y_0,y_1,2^n)$. When the MSB is known, the *MSB-to-Wrap* protocol can be invoked to compute the Wrap function, thereby reducing the overall overhead of the unsigned uniform multiplication protocol. Following SirNN's method, this method can then be converted to signed version without introducing extra communication.

5.3.4 Comparison with SirNN's Multiplication Protocol

The signed multiplication protocol in SirNN requires two calls to $\Pi^{m,n}_{\mathsf{CrossTerm}}$ to compute the cross term. Additionally, it involves invoking one instance each of Π^{m}_{Wrap}, Π^{n}_{Wrap}, Π^{m}_{MUX} and Π^{n}_{MUX}, resulting in a communication cost of $(m+n+4)\lambda+16(m+n)$. The total communication is roughly $\lambda(3\mu+\nu+4)+\mu(\mu+2\nu+1)+16(m+n)$. Compared with SirNN's signed multiplication, our $\Pi^{m,n}_{\mathsf{SMul}}$ reduces the communication overhead by approximately $(m+n-8)\lambda+12(m+n)$, leading to a total reduction in communication cost by about 40% to 60%.

The constraint on our $\Pi^{m,n}_{\mathsf{SMul}}$ is that $|x|<\frac{M}{4}$ and $|y|<\frac{N}{4}$. To meet this constraint for an m-bit input x and an n-bit input y without priori knowledge, they must be shared on $\mathbb{Z}_{2^{m+1}}$ and $\mathbb{Z}_{2^{n+1}}$, respectively, which allows us to perform $\Pi^{m+1,n+1}_{\mathsf{SMul}}$ and obtain $[\![z]\!]^{m+n+2}$, where $\mathsf{int}(z)=\mathsf{int}(x)\cdot\mathsf{int}(y)$. Although this extension necessitates using $\Pi^{m+1,n+1}_{\mathsf{SMul}}$ instead of $\Pi^{m,n}_{\mathsf{SMul}}$, which increases the communication required for computing the cross-term, the overall overhead is still reduced since the costly comparison protocols are avoided. The communication comparison in Table 2 confirms this conclusion. Moreover, to obtain $[\![z]\!]^{2^{m+n}}$, P_i just set $[\![z]\!]^{2^{m+n}}_i=[\![z]\!]^{2^{m+n+2}}_i \mod 2^{m+n}$ locally for $i\in\{0,1\}$.

5.4 Satisfying the Constraint

The primary challenge now lies in ensuring that the constraint is satisfied. We first demonstrate that only one bit of redundancy is required to satisfy the constraint $\frac{L}{3}$ or $\frac{L}{4}$. For $x\in\mathbb{Z}_L$, we already have $-\frac{L}{2}\le\mathsf{int}(x)<\frac{L}{2}$ implying $|x|\le\frac{L}{2}$, where the first bit represents the sign of x. Therefore, reducing the range of x by at most half is sufficient to meet the constraint $|x|\le\frac{L}{3}$ or $|x|\le\frac{L}{4}$. We then consider the following three scenarios to show how to meet the constraint.

- **Given or assumed in advance that $|x|$ is relatively small.** In many practical application scenarios, the input x is shared on a large ring $\mathbb{Z}_L$ to prevent overflow, while the actual value of x often relatively small, and bounded by a small range L_x. For example, SecureML [21] sets $L_x=2^{32}$ while $L=2^{64}$, and Bicoptor [28] considers a 13-bit input on 64-bit ring. In these scenarios, the constraint is naturally satisfied, making our protocols applicable.
- **The plaintext x can be shared over larger ring.** In a secure computation task where $F(x)$ is being computed, the data holder first shares his l-bit data x over ring $\mathbb{Z}_L$ as $[\![x]\!]^L$, then performs protocols to securely evaluate the function $f(x)$. To satisfy the constraint in our protocols, we let the data holder shares the l-bit x over a slightly larger ring $\mathbb{Z}_{2L}$ as $[\![x]\!]^{2L}$. This ensures that the constraint $|x|<\frac{L}{4}$ is satisfied, enabling our protocols work.
- **P_0 and P_1 have $[\![x]\!]^L$.** In case where P_0 and P_1 only have shares of x on ring $\mathbb{Z}_L$, a traditional signed extension protocol in SirNN [23] is required to extend $[\![x]\!]^L$ to $[\![x]\!]^{2L}$, thereby satisfying our constraint. Although this extension protocol involves invoking the comparison protocol, which increases communication, we expect that when the function $F(\cdot)$ contains many truncation, extension or multiplication operations, and the total overhead will be reduced by using our protocols.

Although computing $F(x)$ on a larger ring may increase the overhead for some sub-protocols, such as the computing cross-term protocol in multiplication protocol with non-uniform bitwidth, the improvements that come from our optimized protocols significantly reduce the overall overhead (see Section 6.2 for details). Furthermore, it should be noted that this extension does not increase the communication cost of the MSB or DReLU protocol, which is frequently used and has high communication. Since x is an l-bit number, we can reduce the $[\![x]\!]^{2L}$ to $[\![x]\!]^L$ by performing a local $\mod L$ operation, and then compute $[\![\mathsf{MSB}(x)]\!]^B$ with the l-bit input.

6 Experiment

We conduct experiments to demonstrate the improvements achieved by our novel approach, using the state-of-the-art protocols proposed in Cheetah [13] and SirNN [23] as the baselines. The open source code for the implementations of SirNN [23] are available in the Secure and Correct Inference (SCI) library [1]. For a comprehensive and fair comparison, we also implement our protocols on the same unified platform, leveraging the fundamental protocols provided by the SCI library and EMP toolkit. Additionally, we conduct experiments based on the IKNP-style OT.

All experiments were conducted on a single machine running Ubuntu 20.04 with Intel Core i9-9900K 3.6GHz and 128GB of memory. We simulated network conditions using the Linux 'tc' command. The simulated network settings include a LAN environment (1Gbps bandwidth and 0.3ms RTT latency) and a WAN environment (30Mbps bandwidth with 30ms RTT latency). Our protocols are performed entirely online without requiring an offline phase, and we report the online time as total runtime. The code is publicly available at https://zenodo.org/records/14643158 .

6.1 Microbenchmarks

Experiments on truncation and signed extension protocols. We first implement our one-bit error truncation protocol Π^k_{trun2}, faithful truncation $\Pi^k_{\mathsf{trun_f}}$ (based on Π^k_{trun2}) and one-bit error truncation with known MSB protocol and compared them with the state-of-the-art works. The experimental results are presented in Table 3. In this experiment, we consider two settings Our[1] and Our[2], where the second setting allows the one-bit redundancy. Compared to Cheetah [13], the communication cost of the one-bit error truncation protocol can be improved by 34.86×. The running time can be reduced by 35.29× and 26.56× under the LAN setting and WAN setting, respectively. Additionally, by leveraging the bit multiplication protocol, we improve the performance of the one-bit error truncation protocol with known MSB in plaintext by 1.6×. Compared to SirNN [23], our faithful truncation protocol without error showcases a $>3\times$ improvement in both runtime and communication. Moreover, our signed extension protocol achieved improvements ranging from 12× to 18×.

Experiments on signed multiplication with non-uniform bitwidth protocols. We implement our signed non-uniform multiplication protocol and compare it to SirNN's [23], with the experimental results presented in Table 4. We consider an m-bit x and an n-bit y, and for SirNN's multiplication protocol, we let x and y be shared on rings $\mathbb{Z}_M$ and $\mathbb{Z}_N$, where $M=2^m$ and $N=2^n$, respectively. For our protocol, we evaluate two scenarios to satisfy the constraints. In the first scenario, we assume the constraints are already met, meaning that we could estimate or set $|x|<\frac{M}{4}$ and $|y|<\frac{N}{4}$ in advance, allowing us to directly implement our $\Pi^{m,n}_{\mathsf{SMul}}$ with parameters m and n. In the second scenario, we assume x and y are two plaintexts with no prior knowledge. In this case, we share m-bit x and n-bit y on rings $\mathbb{Z}_{2M}$ and $\mathbb{Z}_{2N}$, to satisfy the constraints $|x|<\frac{2M}{4}$ and $|y|<\frac{2N}{4}$. We then implement $\Pi^{m+1,n+1}_{\mathsf{SMul}}$, with parameters $m+1$ and $n+1$. Table 4 illustrates that $\Pi^{m+1,n+1}_{\mathsf{SMul}}$ only increases the overhead by approximately 4% compared to $\Pi^{m,n}_{\mathsf{SMul}}$. Both $\Pi^{m,n}_{\mathsf{SMul}}$ and $\Pi^{m+1,n+1}_{\mathsf{SMul}}$ outperform SirNN's multiplication protocol, achieving a 1.7× improvement under LAN and a 1.4× under WAN.

Table 3: Comparing the running time and communication costs of our protocols with the state-of-the-art. For truncation protocol, we let $l=37$ and the truncated length as $k=12$. For the signed extension protocol, we set $m=20$ and $n=30$. Our $\Pi_{\mathsf{trun_f}}$ is based on Π_{trun2}. For setting Ours[1], we let x shared over ring $\mathbb{Z}_{2^l}$ or $\mathbb{Z}_{2^m}$. For setting Ours[2], we add an additional bit and suppose x shared over ring $\mathbb{Z}_{2^{l+1}}$ or $\mathbb{Z}_{2^{m+1}}$ to meet the constraints. All experiments were conducted using a single thread. The communication and timing are accumulated for 2^{16} runs of the protocols.

Benchmark	Method	Time (ms)		Comm.
		LAN	WAN	
Trun. (1-bit error)	[13]	600	11797	38.00MB
	Ours[1]	17	455	1.09MB
	Ours[2]	17	444	1.09MB
		35.29×	**26.56**×	**34.86**×
Trun. (Faithful)	[23]	672	12821	40.53MB
	Ours[1]	211	3935	11.32MB
	Ours[2]	214	3920	11.33MB
		3.14×	**3.27**×	**3.57**×
Trun. (1-bit error, known MSB)	[13]	29	834	2.11MB
	Ours[1]	16	455	1.09MB
	Ours[2]	18	498	1.09MB
		1.61×	**1.67**×	**1.93**×
SExt.	[23]	314	5866	17.85MB
	Ours[1]	18	450	1.07MB
	Ours[2]	17	459	1.07MB
		18.47×	**12.77**×	**16.68**×

Table 4: Comparing the running time and communication costs of SirNN's signed multiplication with non-uniform bitwidths protocol with our $\Pi^{m,n}_{\mathsf{SMul}}$. We set parameters $m=20$ and $n=30$ and implemented SirNN's multiplication protocol with input $[\![x]\!]^{2^m}$ and $[\![y]\!]^{2^n}$. For our protocol, we implemented $\Pi^{m,n}_{\mathsf{SMul}}$ and $\Pi^{m+1,n+1}_{\mathsf{SMul}}$, where the latter ensures the constraints are satisfied. All experiments were conducted using a single thread. The communication and timing are accumulated for 2^{16} runs of the protocols.

Method	Time (ms)		Comm.(MB)
	LAN	WAN	
SirNN [23]	1456	34825	104.12
Our $\Pi^{m,n}_{\mathsf{SMul}}$	817	23557	66.62
	1.78×	**1.47**×	**1.56**×
Our $\Pi^{m+1,n+1}_{\mathsf{SMul}}$	822	24594	69.81
	1.77×	**1.41**×	**1.49**×

6.2 Biometric Matching (Minimum Euclidean Distance)

Consider the scenario involving k shared biometric samples $(\vec{s_1},...,\vec{s_k})$ and a shared query biometric sample $\vec{c}$. In the privacy-preserving biometric matching task, the goal is to identify the sample $\vec{s_i}$ closest to $\vec{c}$. We use Squared Euclidean Distance (SED) to measure the distance, and then this task involves computing SED and identifying the minimum distance, utilizing signed multiplication and comparison protocols. Additionally, the truncation and reduction protocol from SirNN [23] is applied to reduce input bitwidth and consequently lower the overhead of the comparison protocols.

Let the elements in each $\vec{s_i}$ (for $i = 1,...,k$) and $\vec{c}$ be m-bit values. When using our Π_{SMul}, we represent these elements as shared over $m+1$ bits rings, respectively. To ensure a fair comparison and show the overhead introduced by the extra bit, we also perform experiments using SirNN's multiplication protocols with bitwidths (m,m) and $(m+1,m+1)$. The experimental results are summarized in Table 5, which indicate that while sharing elements over larger rings will slightly increase the overhead of SirNN's protocol, our method improves efficiency by more than 50%.

Table 5: Comparison of the overhead for Minimum Euclidean Distance with $m = 30$ and $k = 10$.

Method	Bitwidth	Time (s)		Comm (MB)
		LAN	WAN	
[23]	m	5.84	193.14	401.91
	$m+1$	5.94	195.13	407.72
Ours	$m+1$	3.79	137.72	261.03
		1.54×	**1.40**×	**1.54**×

6.3 Secure Inference

In this subsection, we showcase the performance of our improved truncation protocol in secure deep neural network (DNN) inference on network ResNet50 [10] and DenseNet121 [12]. The results are summarized in Table 6. In these tasks, we observe that the input values to truncation protocols naturally satisfy our constraint $|x| < \frac{L}{4}$, eliminating the need for bitwidth adjustments. Consequently, we directly replace the original truncation protocol with our optimized version.

Our work follows the settings of prior work in CrypTFlow2 [24], using $l = 37$ and truncating $k = 12$ bits for ResNet50, and $l = 32$ and $k = 11$ for DenseNet121. Experimental results demonstrate a 13× to 24× enhancement in the truncation protocol's performance. Additionally, the total numbers of communication rounds are reduced by over 30%, resulting in an improvement for the overall runtimes by 1.15× to 1.19×. These improvements stem solely from the new truncation protocol, as no modifications were made to other framework components. All secure inference experiments are conducted using 4 threads on the network with 10Gbps bandwidth and 3ms RTT latency.

Table 6: Performance comparison with CrypTFlow2 on large-scale DNNs, where RN50 denotes ResNet50 and DNet denotes DenseNet121. Runtimes are in seconds and communications are in GB.

Network	System	Truncation		Total		
		Comm.	Time	Comm.	Time	Round
RN50	[24]	7.61	67.03	367.55	493.94	4652
	Ours	0.34	2.69	360.28	428.67	3084
DNet	[24]	6.62	58.46	212.63	347.59	10002
	Ours	0.39	4.22	206.40	291.86	6392

7 Conclusion and Future Work

In this paper, we propose a new geometric perspective to study the secure two-party computation. Using this geometric method, we can compute signed values to avoid invoking the costly comparison protocols. We then apply this method on truncation, signed extension and signed non-uniform multiplication protocols, where we can replace the comparison protocols in these protocols with evaluating AND gate, thus achieving an improvement. Compared to previous work, our truncation and signed extension protocol achieve tens of times improvement in run-time, and our signed non-uniform multiplication protocol achieves a 1.4× to 1.7× improvement. Since we optimize the underlying operators, we believe that our results can improve the efficiency of the entire two-party computation framework. Besides, we anticipate that this geometric method will emerge as a new research tool for MPC and open avenues for further exploration and innovation.

Acknowledgments

We thank the anonymous shepherd and reviewers for their valuable feedback. We also thank Chunzao Huang and Zhaoqian Liu for their help in the implementations and suggestions. This work was supported in part by National Key Research and Development Program of China (Grant No. 2022YFB2703003), National Natural Science Foundation of China (Grant No. 62132008, U22B2030, 12141108), Major Programs of the National Social Science Foundation of China (Grant No. 22&ZD147), National Key R&D Program of China (Grant No. 2022YFA1005000), Natural Science Foundation of Jiangsu Province (BK20220075), Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

Ethics Considerations and Compliance

All authors of this paper unanimously agree and declare the following statements.

1. We attest that we read the ethics considerations discussions in the conference call for papers, the detailed submission instructions, and the guidelines for ethics documents.

2. We attest that our research complies with all ethical guidelines and open science policy, and no ethical concerns are relevant to this study. As the research did not involve human participants, data privacy concerns, or other sensitive issues, the need for informed consent does not apply. We believe our team's next-step plans (e.g., after publication) are ethical.

Compliance with the Open Science Policy

We fully support the principles of the Open Science Policy. We have merged our research artifact into an open-source GitHub repository and ready for commit. We will openly share our research artifact in the final version of our paper. We grant the scientific community unrestricted access to review, validate, and expand upon our work.

References

[1] Secure and correct inference (sci) library. https://github.com/mpc-msri/EzPC/tree/master/SCI.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[4] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238. Springer, 1986.

[5] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

[6] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 31–59. Springer, 2020.

[7] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018.

[8] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[9] Jiaxing He, Kang Yang, Guofeng Tang, Zhangjie Huang, Li Lin, Changzheng Wei, Ying Yan, and Wei Wang. Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 2490–2504. ACM, 2024.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[11] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. Cryptology ePrint Archive, Paper 2023/1147, 2023. https://eprint.iacr.org/2023/1147.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society, 2017.

[13] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 809–826. USENIX Association, 2022.

[14] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh,

editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[16] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3pc for binary circuits with application to maliciously-secure DNN inference. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 5377–5394. USENIX Association, 2023.

[17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 619–631. ACM, 2017.

[18] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. Secretflow-spu: A performant and user-friendly framework for privacy-preserving machine learning. In Julia Lawall and Dan Williams, editors, *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, pages 17–33. USENIX Association, 2023.

[19] Kiwan Maeng and G. Edward Suh. Approximating relu on a reduced ring for efficient mpc-based private inference. *CoRR*, abs/2309.04875, 2023.

[20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2505–2522. USENIX Association, 2020.

[21] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

[22] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2165–2182. USENIX Association, 2021.

[23] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure RNN inference. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1003–1020. IEEE, 2021.

[24] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 325–342. ACM, 2020.

[25] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the GPU. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1021–1038. IEEE, 2021.

[26] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[27] Chi Chih Yao. How to generate and exchange secrets. In *Symposium on Foundations of Computer Science*, 2008.

[28] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party nonlinear computation without preprocessing for privacy-preserving machine learning. *CoRR*, abs/2210.01988, 2022.

A Bit Multiplication Protocol

Suppose P_0 holds a bit a, P_1 holds a bit b, and they want to get the share of $a \cdot b$ on $\mathbb{Z}_L$, namely, $[\![a \wedge b]\!]^L$. A straightforward method is first invoking Π_{AND} to get $[\![a \wedge b]\!]^B$, followed by a Π_{B2A} to get $[\![a \wedge b]\!]^L$, with total communication $2\lambda + l + 20$ bits in 4 rounds. Inspired by MASCOT [15] , we propose an optimized method to directly calculate $[\![a \wedge b]\!]^L$ from a and b by invoking a single $\binom{2}{1}$-COT$_l$, with communication $\lambda + l$ bits in 2 rounds. Our observation is that once bits a, b are treated as arithmetic values, then $[\![a \wedge b]\!]^L = [\![a \cdot b]\!]^L$, the calculation of which giving a special case of oblivious product evaluation

proposed in MASCOT [15] with 1 bit input range. Then, a single instance of $\binom{2}{1}$-COT$_l$ satisfied. The details are shown in Algorithm 8. It is easy to see that by correctness of $\binom{2}{1}$-COT$_l$, $[\![a \wedge b]\!]_1^L = x + a \cdot b$. So $[\![a \wedge b]\!]_0^L + [\![a \wedge b]\!]_1^L \mod L = a \cdot b = a \wedge b$ as required. Security directly follows from the security of $\binom{2}{1}$-COT$_l$.

Algorithm 8: Π^l_{BitMul}.

Input: P_0 holds a bit a and P_1 holds a bit b.
Output: P_0 and P_1 output $[\![a \wedge b]\!]^L$, where $L = 2^l$.

1 P_0, P_1 invoke $\binom{2}{1}$-COT$_l$, where P_0 is the sender with correlation function $f(x) = x + a$ and P_1 is the receiver with input b. P_0 learns x and sets $[\![a \wedge b]\!]_0^L = L - x$ and P_1 learns $[\![a \wedge b]\!]_1^L$.

B Our Signed Extension Protocol

B.1 Signed Extension Protocol with Constraint

B.1.1 When $|x| < \frac{M}{3}$

For the case $|x| < \frac{M}{3}$, we can compute $\mathsf{MW}(x)$ as Section 4.2 then perform our signed extension protocol $\Pi^{m,n}_{\mathsf{SExt1}}$. The details are shown in Algorithm 9. The correctness of $\Pi^{m,n}_{\mathsf{SExt1}}$ follows Equation 9 and Section 4.2. The security of $\Pi^{m,n}_{\mathsf{SExt1}}$ comes from the security of Π_{BitMul}. $\Pi^{m,n}_{\mathsf{SExt1}}$ invokes two $\Pi^{n-m}_{\mathsf{BitMul}}$ in parallel, therefore the total communication is $2(\lambda + n - m)$ in 2 rounds.

Algorithm 9: Signed extension protocol, $\Pi^{m,n}_{\mathsf{SExt1}}$:

Input: P_0 and P_1 hold $[\![x]\!]^M$ satisfying $|x| < \frac{M}{3}$ where $M = 2^m$.
Output: P_0 and P_1 output $[\![y]\!]^N$, where $M < N$, $\mathsf{int}(x) = \mathsf{int}(y)$ and $N = 2^n$.

1 P_0 and P_1 invoke Π^k_{MW1} with input $[\![x]\!]^M$ to learn $[\![\mathsf{MW}]\!]^{2^{n-m}}$.
2 P_0 and P_1 output $x_0 + x_1 + N - [\![MW]\!]^{2^{n-m}} \cdot M$.

B.1.2 When $|x| < \frac{M}{4}$

For the case $|x| < \frac{M}{4}$, we can use the method in Section 4.3 to compute MW, as shown in Algorithm 10. The correctness of $\Pi^{m,n}_{\mathsf{SExt}}$ follows Equation 9 and Theorem 2. The security comes from the security of Π_{BitMul}. $\Pi^{m,n}_{\mathsf{SExt}}$ invokes one $\Pi^{n-m}_{\mathsf{BitMul}}$ in parallel, therefore the total communication is $\lambda + n - m$ in 2 rounds.

Algorithm 10: Signed extension protocol, $\Pi^{m,n}_{\mathsf{SExt2}}$:

Input: P_0 and P_1 hold $[\![x]\!]^M$ satisfying $|x| < \frac{M}{4}$ where $M = 2^m$.
Output: P_0 and P_1 output $[\![y]\!]^N$, where $M < N$, $\mathsf{int}(x) = \mathsf{int}(y)$ and $N = 2^n$.

1 P_0 and P_1 invoke Π^k_{MW2} with input $[\![x]\!]^M$ to learn $[\![\mathsf{MW}]\!]^{2^{n-m}}$.
2 P_0 and P_1 output $x_0 + x_1 + [\![\mathsf{MW}]\!]^{2^{n-m}} \cdot (N - M)$.

B.2 Signed Extension Protocol with Known MSB

For the case MSB is known, we can first compute $\mathsf{MW}(x)$ using the idea in Section 4.5, then perform the signed extension protocol by Equation 9, with the same overhead of computing $\mathsf{MW}(x)$.

B.3 Probabilistic Signed Extension Protocol

Similar to the probabilistic one-bit error truncation protocol, for signed extension protocol we can assume $P \in \mathcal{B} \bigcup \mathcal{C}$ or $x_0 + x_1 \in [\frac{M}{2}, \frac{3M}{2})$. Then P_0 and P_1 can locally compute and output $y_0 = x_0$ and $y_1 = x_1 + N - M$ locally. From Theorem 3, the failure probability of the probabilistic signed extension protocol is $\frac{|\mathsf{int}(x)+1|}{M}$.

C Proofs

C.1 Proof of Equation 3

We first propose the following Lemma:

Lemma 2. *For $a, b, d \in \mathbb{Z}$, where $d \neq 0$, we have:*

$$\lfloor \frac{a+b}{d} \rfloor = \lfloor \frac{a}{d} \rfloor + \lfloor \frac{b}{d} \rfloor + \delta,$$

where $\delta = \mathbf{1}\{\frac{a \mod d}{d} + \frac{b \mod d}{d} \geq 1\}$.

Proof. For an integer a and a divisor d, we have $a = d \cdot \lfloor \frac{a}{d} \rfloor + a \mod d$, therefore $\frac{a}{d} = \lfloor \frac{a}{d} \rfloor + \frac{a \mod d}{d}$. Then we have:

$$\lfloor \frac{a+b}{d} \rfloor = \lfloor \frac{a}{d} \rfloor + \lfloor \frac{b}{d} \rfloor + \lfloor \frac{a \mod d + b \mod d}{d} \rfloor.$$

Therefore, the lemma is proven. □

Based on Lemma 2, we can deduce that Equation 3 holds.

C.2 Proof of Theorem 1

Proof. For $x_0 + x_1 \in [0, \frac{L}{3})$, it is obvious that $x_0 < \frac{L}{3}$ and $x_1 < \frac{L}{3}$ as $x_0, x_1 \in \mathbb{Z}_L$. Conversely, if $x_0 < \frac{L}{3}$ and $x_1 < \frac{L}{3}$, then $x_0 + x_1 < \frac{2L}{3}$, implying $x_0 + x_1 \in [0, \frac{L}{3})$ as $|x| < \frac{L}{3}$. For $x_0 + x_1 \in$

$[\frac{5L}{3}, 2L)$, it follows that $x_0 \geq \frac{2L}{3}$ and $x_1 \geq \frac{2L}{3}$. If $x_0 \geq \frac{2L}{3}$ and $x_1 \geq \frac{2L}{3}$, then $x_0 + x_1 \geq \frac{4L}{3}$ which implies $x_0 + x_1 \in [\frac{5L}{3}, 2L)$. For other values of x_0 and x_1, $x_0 + x_1$ will not falls into the first and third intervals, therefore $x_0 + x_1 \in [\frac{2L}{3}, \frac{4L}{3})$. □

C.3 Proof of Theorem 2

Proof. For $|x| < \frac{L}{4}$, we have $x_0 + x_1 \in [0, \frac{L}{4}) \bigcup [\frac{3L}{4}, \frac{5L}{4}) \bigcup [\frac{7L}{4}, 2L)$. Then we prove this theorem under the following cases:

- If $x_0 + x_1 \in [0, \frac{L}{4})$, then $x_0^* + x_1 = x_0 - \frac{L}{4} + L + x_1 \in [\frac{3L}{4}, L)$. In this case, we have $\mathsf{MW}(x) = 0$, $\mathsf{MW}(x^*) = 1$ and $\mathbf{1}\{x_0 < \frac{L}{4}\} = 1$.
- If $x_0 + x_1 \in [\frac{3L}{4}, \frac{5L}{4})$, then:
  - If $x_0 < \frac{L}{4}$, then $x_0^* + x_1 = x_0 - \frac{L}{4} + L + x_1 \in [\frac{6L}{4}, 2L)$. In this case $\mathsf{MW}(x) = 1$, $\mathsf{MW}(x^*) = 2$ and $\mathbf{1}\{x_0 < \frac{L}{4}\} = 1$.
  - If $x_0 \geq \frac{L}{4}$, then $x_0^* + x_1 = x_0 - \frac{L}{4} + x_1 \in [\frac{6L}{4}, 2L)$. In this case $\mathsf{MW}(x) = 1$, $\mathsf{MW}(x^*) = 1$ and $\mathbf{1}\{x_0 < \frac{L}{4}\} = 0$.
- If $x_0 + x_1 \in [\frac{7L}{4}, 2L)$, then $x_0^* + x_1 = x_0 - \frac{L}{4} + x_1 \in [\frac{6L}{4}, \frac{7L}{4})$. In this case, we have $\mathsf{MW}(x) = 2$, $\mathsf{MW}(x^*) = 2$ and $\mathbf{1}\{x_0 < \frac{L}{4}\} = 0$.

□

C.4 Proof of Theorem 3

Proof. For a given $x \in \mathbb{Z}_L$, we have $x = x_0 + x_1 - \mathsf{Wrap}(x) \cdot L$, and x can be either negative or non-negative. We first consider the non-negative case, where $x_0 + x_1 \in [0, L_x) \bigcup [L, L + L_x)$. The probability $\Pr[x_0 + x_1 \in [0, L_x)] = \Pr[x_0 + x_1 = x] = \Pr[x_0 \leq x] = \frac{x+1}{L}$. Therefore, for non-negative x, $\Pr[x_0 + x_1 \in [L, L + L_x)] = 1 - \frac{x+1}{L} = 1 - \frac{|\mathsf{int}(x)+1|}{L}$.

For the case x is negative, we have $x_0 + x_1 \in [L - L_x, L) \bigcup [2L - L_x, 2L)$. The probability $\Pr[x_0 + x_1 \in [L - L_x, L)] = \Pr[x_0 + x_1 = x] = \Pr[x_0 \leq x] = \frac{x+1}{L}$. Since x is negative and $x = x_0 + x_1$, we have $\mathsf{int}(x) = x - L$. Therefore, $\Pr[x_0 + x_1 \in [L - L_x, L)] = \frac{x+1}{L} = 1 - \frac{L-x-1}{L} = 1 - \frac{|\mathsf{int}(x)+1|}{L}$.

In summary, for a given x, the probability that $x_0 + x_1 \in [L - L_x, L + L_x)$ is $1 - \frac{|\mathsf{int}(x)+1|}{L}$. □

C.5 Proof of Lemma 1

Proof. For $x = x_0 + x_1 \mod M$, $\mathsf{uint}(x) = x = x_0 + x_1 - \mathsf{wrap}(x) \cdot M$. Then $\mathsf{int}(x) = \mathsf{uint}(x) - \mathsf{MSB}(x) \cdot M = x_0 + x_1 - (\mathsf{wrap}(x) + \mathsf{MSB}(x)) \cdot M$. Similarly, $\mathsf{int}(y) = y_0 + y_1 - (\mathsf{wrap}(y) + \mathsf{MSB}(y)) \cdot N$.

- For the case $x_0 + x_1 \in [0, \frac{M}{2})$, $\mathsf{wrap}(x) = \mathsf{MSB}(x) = 0$. As $y_0 = x_0$ and $y_1 = x_1$, we have $y_0 + y_1 < \frac{M}{2}$. Therefore $\mathsf{wrap}(y) = \mathsf{MSB}(y) = 0$ and $\mathsf{int}(x) = \mathsf{int}(y) = x_0 + x_1$.
- For the case $x_0 + x_1 \in [\frac{M}{2}, M)$, $\mathsf{wrap}(x) = 0$ and $\mathsf{MSB}(x) = 1$. As $y_0 = x_0$ and $y_1 = x_1 + N - M$, we have $y_0 + y_1 = x_0 + x_1 + N - M \in [N - \frac{M}{2}, N)$. Therefore $\mathsf{wrap}(y) = 0$ and $\mathsf{MSB}(y) = 1$. Then $\mathsf{int}(x) = \mathsf{int}(y) = x_0 + x_1 - M$.
- For the case $x_0 + x_1 \in [M, \frac{3M}{2})$, $\mathsf{wrap}(x) = 1$ and $\mathsf{MSB}(x) = 0$. As $y_0 = x_0$ and $y_1 = x_1 + N - M$, we have $y_0 + y_1 = x_0 + x_1 + N - M \in [N, N + \frac{M}{2})$. Therefore $\mathsf{wrap}(y) = 1$ and $\mathsf{MSB}(y) = 0$. Then $\mathsf{int}(x) = \mathsf{int}(y) = x_0 + x_1 - M$.
- For the case $x_0 + x_1 \in [\frac{3M}{2}, 2M)$, $\mathsf{wrap}(x) = \mathsf{MSB}(x) = 1$. As $y_0 = x_0 + N - M$ and $y_1 = x_1 + N - M$, we have $y_0 + y_1 = x_0 + x_1 + 2N - 2M \in [N - \frac{M}{2}, 2N)$. Therefore $\mathsf{wrap}(y) = \mathsf{MSB}(y) = 1$. Then $\mathsf{int}(x) = \mathsf{int}(y) = x_0 + x_1 - 2M$.

In total, we have $\mathsf{int}(x) = \mathsf{int}(y)$. □

D Cross Term Multiplication Protocol in SirNN [23]

In this work, we use the cross-term multiplication protocol in SirNN [23] to compute the cross term $x_0 y_1$ and $x_1 y_0$, as shown in Algorithm 11.

Algorithm 11: Cross Term Multiplication, $\Pi^{m,n}_{\mathsf{CrossTerm}}$:

Input: P_0 holds $x \in \mathbb{Z}_{2^m}$ and P_1 holds $y \in \mathbb{Z}_{2^n}$, where $m \leq n$.

Output: P_0 and P_1 output $[\![z]\!]^{2^l}$, where $z = x \cdot y \mod 2^l$ and $l = m + n$.

1. P_0 parses x as an m-bit string $x = x_{m-1} || \cdots || x_0$, where $x_i \in \{0, 1\}$.
2. **for** $i = \{0, ..., m-1\}$ **do**
3. $\quad P_0$ and P_1 invoke $\binom{2}{1}$-COT_{l-i}, where P_0 is the sender with input x_i and P_1 is the receiver with input y, and learn $[\![t_i]\!]^{l-i}$.
4. **end**
5. For $b \in \{0, 1\}$, P_b sets $[\![z]\!]^{2^l}_b = \sum_{i=0}^{m-1} 2^i \cdot [\![t_i]\!]^{l-i}_b$.