# Lightning Fast Secure Comparison for 3PC PPML

Tianpei Lu[13], Bingsheng Zhang[13]✉, Lichun Li[2], Yuzhou Zhao[1] and Kui Ren[1]
[1]The State Key Laboratory of Blockchain and Data Security, Zhejiang University,
Email: {lutianpei, bingsheng, zhaoyuzhou, kuiren}@zju.edu.cn
[2]Ant Group Co.,Ltd, Hangzhou, China, lichun.llc@antgroup.com
[3]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

***Abstract*—Privacy-preserving machine learning (PPML) techniques have gained significant popularity in the past years. Those protocols have been widely adopted in many real-world security-sensitive machine-learning scenarios. Secure comparison is one of the most important non-linear operations in PPML. In this work, we focus on maliciously secure comparison in the 3-party MPC over ring $\mathbb{Z}_{2^\ell}$ setting. In particular, we propose a novel constant round *sign-bit extraction* protocol in the preprocessing model. The communication of its semi-honest version is only 12.5% of the state-of-the-art (SOTA) constant-round semi-honest comparison protocol by Zhou *et al.* (Bicoptor, S&P 2023); communication complexity of its malicious version are approximately 25% of the SOTA by Patra and Suresh (BLAZE, NDSS 2020), for $\ell = 64$. Finally, the resulting ReLU protocol outperforms the SOTA secure ReLU evaluation solution (Bicoptor, S&P 2023) by $6\times$ in the semi-honest setting and $20\times$ in the malicious setting, respectively.**

## I. Introduction

In the era of big data, privacy protection, and compliance continues to be a matter of paramount concern among individuals and organizations alike. The need for privacy-preserving mechanisms has intensified with the rise of various privacy regulations, such as GDPR. Privacy-preserving machine learning (PPML) is an emerging privacy-enhancing technique that enables secure data mining and machine learning while maintaining the privacy and confidentiality of the underlying data.

Secure multi-party computation (MPC) [42], [18], [5] allows $n$ parties to jointly evaluate certain functions without revealing their private inputs, and it is a typical cryptographic approach to realize PPML [33], [35], [10], [38], [31], [41] in the multi-server setting. (This work focuses on 3-party MPC, denoted as 3-PC.) A number of works [36], [32], [38], [37], [34], [24], [28], [40], [41], [33], [12], [27], [35], [26] utilizes secure multi-party computation techniques to achieve efficient PPML, including traditional machine learning models such as decision trees and logistic regression, as well as neural network models like ResNet and VGG, and text generation models like GPT-2 and Llama. According to their benchmark reports, the PPML cost of non-linear layers has become the main performance bottleneck.

Secure comparison plays a critical role in evaluating those PPML non-linear functions; for example, the activation functions used in machine learning, such as Rectified Linear Unit (ReLU), and MaxPool. A typical PPML approach is to use piecewise polynomials to approximate arbitrary non-linear functions [24], [25], [36]. The main idea of these methods lies in using comparisons to determine the interval in which the data resides and subsequently selecting the appropriate polynomial evaluation result. Besides, comparisons are also widely used in traditional machine learning tasks, such as decision trees, k-means clustering, and more.

In the literature, the existing comparison protocols can be broadly divided into constant-round [7] and non-constant-round [22], [36], [33], [13]. Empirically, constant-round protocols often incur higher communication costs, whereas non-constant-round protocols can achieve reduced communication by sacrificing the round complexity. The trade-off between communication and round complexity for some representative protocols is depicted in Fig. I, including constant-round protocols like garbled circuits (GC), function secret sharing (FSS), and specialized protocols like CryptFlow, SecureNN, and Bicoptor; as well as non-constant-round protocols such as Falcon and general secret-sharing transformation-based protocols.

For constant-round protocols, GC and FSS only require one-round communication in the online phase (an extra round in the offline is needed). Bicoptor requires two communication rounds without preprocessing. However, the overall communication cost of all these protocols are quit heavy. Let $\mathbb{Z}_{2^\ell}$ be the ring. Garbled circuits and function secret sharing (in particular, its distributed comparison function, DCF) require $O(\ell\kappa)$ communication, where $\kappa$ is the security parameter. Bicoptor [46] realizes $O(\ell^2)$ communication cost comparison protocol based on probabilistic truncation, and it costs $O(\ell^2)$ communication. Nevertheless, when analyzing its specific overhead, it amounts to $\ell(\ell + \lambda)$, where $\lambda$ is the security parameter. Considering $\ell = 64$ and a 64-bit truncation error, its communication cost can even exceed that of DCF. In addition, CryptFlow and SecureNN represent another type of constant-round protocol with communication requirements significantly lower than the two-round protocols. Nevertheless, CryptFlow and SecureNN require 10 rounds to complete a comparison. When considering $\ell = 64$, their round count approaches that of logarithmic-round protocols in most cases.

Turning to non-constant-round protocols, a typical approach is to evaluate comparison circuits using Boolean circuits. For specific types of comparison circuits, employing a parallel prefix adder (PPA) ensures logarithmic rounds and $O(\ell \log \ell)$ communication. If a standard full adder is used, it achieves nearly the minimal communication cost ($3\ell$ in the 3-PC setting) with linear rounds ($\ell$ rounds). Other implementations,

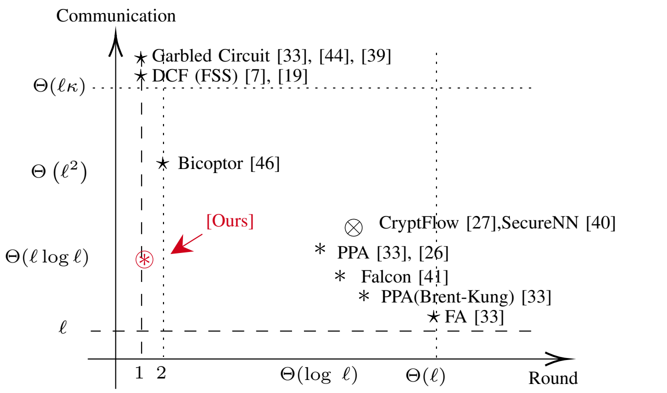



Fig. 1: Comparison of communication and round between prior non-linear protocols and ours. The exact costs of each protocol are depicted in Table I.

such as Falcon or the Brent-Kung algorithm-based PPA, can reduce communication to $O(\ell)$ (with a corresponding coefficient significantly greater than 3), while still maintaining logarithmic rounds. Nevertheless, even logarithmic rounds entail considerable performance overhead in high-latency network environments due to the increased number of communication rounds. To the best of our knowledge, current research lacks a focus on comparison protocols that achieve both low constant rounds and minimal communication overhead.(Cf Appendix. B for related work)

**Our results.** In this work, we aim to reduce the communication overhead of the 3-PC comparison protocol while maintaining a low number of communication rounds. Fig. I depicts the comparison of the theoretical overhead (passively secure version) between our protocol and other protocols. Our protocol achieves communication cost close to that of logarithmic-round protocols while keeping low communication round. At the same time, we apply this protocol to the malicious security while maintaining the constant round complexity. The underlying secret sharing scheme of our 3-PC protocol originates from a variant of the replicated secure sharing (RSS) [12]; that is, to share $x \in \mathbb{Z}_{2^\ell}$, $P_0$ holds $(r_1, r_2)$, $P_1$ holds $(m := x - r, r_1)$, and $P_2$ holds $(m := x - r, r_2)$, where $r := r_1 + r_2$.

One-round semi-honest secure sign-bit extraction. The secure comparison problem in the 3-PC over ring $\mathbb{Z}_{2^\ell}$ setting is equivalent to the sign-bit (i.e. the left-most bit) extraction problem; namely, let $\mathsf{sign}(x)$ denotes the sign-bit of $x \in \mathbb{Z}_{2^\ell}$, and we have $x \geq 0$ iff $\mathsf{sign}(x) = 0$.

Intuitively, our 1-round sign-bit extraction protocol works as follows. Given $x \in \mathbb{Z}_{2^\ell}$, let $\hat{x} := x - 2^{\ell-1} \cdot \mathsf{sign}(x)$ denote the value $x$ after removing its sign-bit. Alternatively, we write $x = \mathsf{sign}(x) \| \hat{x}$. According to our secret sharing scheme, $x = m + r \bmod 2^\ell$, where $m := \mathsf{sign}(m) \| \hat{m}$ and $r := \mathsf{sign}(r) \| \hat{r}$. The sign-bit of $x$ $\mathsf{sign}(x) := \mathsf{sign}(r) \oplus \mathsf{sign}(m) \oplus (\hat{m} \overset{?}{\geq} 2^{\ell-1} - \hat{r})$ where the boolean check $(\hat{m} \overset{?}{\geq} 2^{\ell-1} - \hat{r})$ represents the carry bit from $\hat{m} + \hat{r}$. Since $\mathsf{sign}(r)$ is known to $P_0$ and $\mathsf{sign}(m)$ is known to both $P_1$ and $P_2$, our main task is to obliviously determine $(\hat{m} \overset{?}{\geq} 2^{\ell-1} - \hat{r})$, where $2^{\ell-1} - \hat{r}$ is held by $P_0$ and $\hat{m}$ is held by both $P_1$ and $P_2$.

Let $s := \hat{m} \oplus (2^{\ell-1} - \hat{r})$. It is easy to see that, from left to right, the first non-zero bit of $s$ indicates the left-most position where $2^{\ell-1} - \hat{r}$ and $\hat{m}$ differ, when they are viewed as two binary vectors. Denote the $\zeta$-th bit of $\hat{m}$ as $\hat{m}_\zeta$. We have $\hat{m}_\zeta = (\hat{m} \overset{?}{\geq} 2^{\ell-1} - \hat{r})$.

Without considering security, $\hat{m}_\zeta$ can be determined through the following steps. (i) Compute $s'$ as the prefix-sum of $s$, i.e., $s'_i := \sum_{k=0}^{i} s_k$ for $i \in \mathbb{Z}_\ell$. (ii) Compute $s''_i := s'_i - 2s_i + 1$. We argue that $s''$ will only contain one zero at the position of the first non-zero bit of $s$. Indeed, it converts all the prefix zero bits of $s'$ to 1 (namely, if $s'_i = 0 \wedge s_i = 0$ then $s''_i = 1$); it converts the first non-zero bit of $s'$ to 0 (namely, if $s'_i = 1 \wedge s_i = 1$ then $s''_i = 0$); it converts the suffix bits to non-zero values (namely, in case $s_i = 0$, $s'_i \geq 1$, we have $s''_i = s'_i - 2s_i + 1 \geq 2$; in case $s_i = 1$, $s''_i \geq 2$, we have $s''_i = s'_i - 2s_i + 1 \geq 1$). (iii) $P_1$ and $P_2$ opens $\hat{m}$ and $s''$ to $P_0$; $P_0$ then locates $\zeta$ as the position of the only zero bit in $s''$, and outputs $\hat{m}_\zeta$ as the sign-bit extraction (a.k.a. comparison) result.

Achieving malicious security. We adopt SPDZ style IT-secure MAC [16] and the dual execution technique [23] for malicious security. We overcome the one-bit leakage introduced by dual execution and realize a efficient constant-round actively secure sign-bit extraction protocol. Our main observation is that if we introduce an IT-secure MAC (Cf. TABLE. II, below) to the share of $s''$ on top of the semi-honest version, $P_0$ can verify the correctness of $s''$ through the MAC check, which prevents malicious $P_1$ or $P_2$ from tampering with $s''$. Next, since there is at most one malicious adversary among the 3 parties under static corruption, we can adopt the dual execution paradigm [23] and perform the verification protocol twice, but switch the role of the players, i.e., we nominate a different party to play the role of the $P_0$ and let him generate an IT-secure MAC and check the execution correctness. The comparison result shall be accepted if and only if both verifications pass.

**Performance.** Table I depicts the performance comparison between our protocols and SOTA 3-PC-based ReLU protocols. As we can see, our protocols achieve a significant performance improvement compared to other protocols.

*Semi-honest secure sign-bit extraction.* Our comparison (a.k.a. sign-bit extraction) protocol can be further used as the essential building block of the ReLU and MaxPool evaluations. For the semi-honest (passive) setting, compared with CrypTFlow [27] (8-round with $6\ell \log \ell + 14\ell$ bits of communication) and Bicoptor [46] (2-round with the $(\lambda + \ell)(2 + \ell)$ bits of communication, $\lambda$ is the security parameter such that the error probability is bounded by $2^{1-\lambda}$), our solution demonstrates a significant improvement, i.e., 1-round with $2\ell \log \ell$ communication. Specifically, when $\ell = 64$ and $\lambda = 64$, our protocol reduces the communication cost by 88% from SOTA (Bicoptor), resulting in a $6\times$ speedup in real-world benchmark tests.

*Actively secure sign-bit extraction.* Our actively (mali-

TABLE I: Comparison of 3-PC based ReLU. ($\ell$ is the ring size, $\ell^*$ is the security parameter for truncation error $2^{1-\ell^*}$, $\kappa = 128$ is the computational security parameter of GC, and $\lambda = 7$ is the statistical security parameter for soundness error $2^{-(\lambda \log \ell + \lambda)}$.)

| Protocol | Offline | Online | | Malicious |
|---|---|---|---|---|
| | Communication (bits) | Rounds | Communication (bits) | |
| ABY3[33] | $60\ell$ | $3 + \log \ell$ | $45\ell$ | ✓ |
| BLAZE[35] | $5\kappa\ell + 6\ell + \kappa$ | 4 | $\kappa\ell + 6\ell$ | ✓ |
| Fantastic-3PC[15] | - | $3 + \log \ell$ | $114\ell + 6\kappa + 1$ | ✓ |
| SWIFT[26] | $21\ell$ | $3 + \log \ell$ | $16\ell$ | ✓ |
| Falcon[41] | - | $5 + \log \ell$ | $32\ell$ | ✓ |
| Bicoptor[46] | 0 | 2 | $(\ell^* + \ell)(2 + \ell)$ | × |
| CryptFlow[27] | - | 10 | $(6 \log \ell + 19)\ell$ | × |
| SecureNN[40] | - | 10 | $(8 \log \ell + 24)\ell$ | × |
| Edabits[17] | - | $5 + \log \ell$ | $80\ell$ | ✓ |
| DCF[19], [7] | $(\ell + 2)\kappa$ | 1 | $2\ell$ | × |
| **Ours (Semi-honest)** | $(\ell - 1) \log \ell + 2\ell$ | 2 | $2\ell(\log \ell + 1) + 2\ell$ | × |
| **Ours (Malicious)** | $2((\lambda+1)(\ell-1) \log \ell+(\ell-1) \log \ell + 2\ell$ | 3 | $4\ell(\log \ell + 1) + 8\ell$ | ✓ |

ciously) secure sign-bit extraction protocol requires amortized 3-round with $4\ell \log \ell + 10\ell$ bits communication in online phase, and $10\ell + 6\ell(\lambda + 1) \log \ell$ bits communication in offline phase where $\lambda$ is the statistical security parameter such that the soundness error is $2^{-(\lambda \log \ell+\lambda)}$. To the best of our knowledge, our maliciously secure protocol significantly reduces communication of SOTA constant-round maliciously secure solutions. Compared with BLAZE [35] (5-round with $5\kappa\ell + 6\ell + \kappa$ bits of communication in the offline phase and 4-round and $\kappa\ell + 6\ell$ bits of communication in the online phase), our protocol reduces the communication by 75% in the online phase and 60% in the offline phase, when $\ell = 64, \kappa = 128$ and $\lambda = 7$ (with statistical soundness error $2^{-49}$). Besides, the computational cost of our protocol is significantly lower than that of BLAZE which is based on Garbled Circuit. Our benchmark demonstrates that our protocol achieves a 20× speedup over BLAZE.

## II. Preliminaries

**Notation.** Let $\mathcal{P} := \{P_0, P_1, P_2\}$ be the three MPC parties. We assume the ring size is $\mathbb{Z}_{2^\ell} := \{0, \ldots, 2^\ell - 1\}$ for range $[0, 2^{\ell-1}) \cup [-2^{\ell-1}, -1]$ and represent the negative integers in $[-2^{\ell-1}, -1]$ as $[2^{\ell-1}, 2^\ell - 1]$. This encoding sets the the first bit as the sign bit. In our work, we choose the finite field $\mathbb{Z}_p$, where $p$ is the largest prime in the interval $(\ell, 2\ell]$. For a vector $\mathbf{x} = (x^{(0)}, ..., x^{(n-1)})$, the subscript $x^{(i)}$ denotes its $i$-th element. When processing the bits of $x \in \mathbb{Z}_{2^\ell}$, we abuse the representation of subscripts $x_i$ to denote the $i^{\text{th}}$ bit from big-endian. We denote $\gamma(x) = \alpha \cdot x$ as the MAC of the field element $x$ where $\alpha$ is the MAC key. Considering our field $\mathbb{Z}_p$ is small, we take $\lambda$ numbers of MAC keys for soundness, namely, $\gamma(x) := (\alpha_0 \cdot x, \ldots, \alpha_{\lambda-1} \cdot x)$, and we represent $i^{\text{th}}$ MAC as $\gamma(x)_i := \alpha_i \cdot x$. We denote $\mathsf{sign}(x)$ as the sign-bit of $x$ and $\hat{x}$ as the value dropping the sign-bit, namely, $x = \mathsf{sign}(x)||\hat{x}$. We use $\eta_{j,k}$ to denote the common seed held by both $P_j$ and $P_k$. Our protocol contains five types of secret sharing:

- $[\cdot]^k$-sharing: We define $[\cdot]^k$-sharing over ring $\mathbb{Z}_{2^\ell}$ as $[x]^k := ([x]_{k-1} \in \mathbb{Z}_{2^\ell}, [x]_{k+1} \in \mathbb{Z}_{2^\ell})$ where $x = [x]_{k-1} + [x]_{k+1} \pmod 2)^\ell$. $P_j$ for $j \in \{0, 1, 2\}/k$ holds share $[x]_j$.
- $\langle\cdot\rangle^k$-sharing: We define $\langle\cdot\rangle^k$-sharing over ring $\mathbb{Z}_{2^\ell}$ as $\langle x\rangle^k := (m_x, [r_x]^k)$ where $r_x$ is a fresh random value and $x = m_x + r_x$. $P_j$ for $j \in \{0, 1, 2\}/k$ hold $(m_x \in \mathbb{Z}_{2^\ell}, [r_x]_j \in \mathbb{Z}_{2^\ell})$ and $P_k$ holds $([r_x]_{k-1}, [r_x]_{k+1})$.
- $[\![\cdot]\!]^k$-sharing: It is the finite field version of $[\cdot]^k$. We define $[\![x]\!]^k := ([\![x]\!]_{k+1} \in \mathbb{Z}_p, [\![x]\!]_{k-1} \in \mathbb{Z}_p)$ where $x = [\![x]\!]_{k+1} + [\![x]\!]_{k-1} \pmod p$. $P_j$ for $j \in \{k+1, k-1\}$ hold share $[\![x]\!]_j$.
- $\langle\!\langle\cdot\rangle\!\rangle^k$-sharing: It is the finite field version of $\langle\cdot\rangle^k$. We define $\langle\!\langle x\rangle\!\rangle^k := ([\![r_x]\!]^k, m_x)$ where $x = m_x + r_x \pmod p$. $P_j$ for $j \in \{0, 1, 2\}/k$ hold $(m_x \in \mathbb{Z}_p, [\![r_x]\!]_j \in \mathbb{Z}_p)$ and $P_k$ holds $([\![r_x]\!]_{k-1}, [\![r_x]\!]_{k+1})$.
- $\|\cdot\|^{\lambda,k}$-sharing: We define $\|\cdot\|^{\lambda,k}$-sharing over finite field $\mathbb{Z}_p$ as $\|x\|^{\lambda,k} := ([\![x]\!]^k, \{[\![\alpha_j]\!]^k, [\![\gamma(x)_j]\!]^k\}_{j\in\mathbb{Z}_\lambda})$. In our sign-bit verification protocol, one party $P_k$ holds $\{\alpha_j\}_{j\in\mathbb{Z}_\lambda}$ which are the plaintext of MAC keys, and the other parties $P_i$ for $i \in \{k-1, k+1\}$ hold the share $([\![x]\!]_i, \{[\![\alpha_j]\!]_i, [\![\gamma(x)_j]\!]_i\}_{j\in\mathbb{Z}_\lambda})$.

Table II gives several secret sharing structures for different values of $k$. Note that in the definition above, $k$ is used to indicate which party takes on the asymmetric role. For example, in the replicated secret sharing schemes $\langle x\rangle^0$ and $\langle x\rangle^1$, $\langle x\rangle^0$ denotes that $P_0$ holds the plaintext $r_x$ while $P_1$ and $P_2$ hold the corresponding secret shares, whereas $\langle x\rangle^1$ denotes that $P_1$ holds the complete $r_x$ while $P_0$ and $P_2$ hold the secret shares. We denote the replicated secret sharing fragment as $\langle x\rangle^0 := (m_x, [r_x]_2, [r_x]_1)$, where the element at index $i$ indicates the share fragment unknown to $P_i$. Similarly, we have $\langle x\rangle^1 := ([r_x]_2, m_x, [r_x]_1)$. We use hollow brackets $[\![\cdot]\!]$ and $\langle\!\langle\cdot\rangle\!\rangle$ to denote the field version of $[\cdot]$-sharing and $\langle\cdot\rangle$-sharing. For $\|\cdot\|^{\lambda,k}$, the superscript $\lambda, k$ denotes that $P_k$ holds $\lambda$ MAC keys $\alpha_0, \ldots, \alpha_{\lambda-1}$, and the other parties hold the corresponding secret share over $\mathbb{Z}_p$. Since all MACs are verified at the end of the protocol execution, the MAC keys can be reused. We let any two secret shares $\|x\|^{\lambda,k}$ and $\|y\|^{\lambda,k}$ for the same key holder $P_k$ use the same MAC keys. For simplicity, we ignore the superscript such as $[\cdot]$, $\langle\cdot\rangle$ when semantics are clear. In our description, by default $\langle\cdot\rangle$ refers to $\langle\cdot\rangle^0$.

All the aforementioned secret-sharing forms have the linear homomorphic property. That is, $[x] + [y] = ([x]_1 + [y]_1, [x]_2 +$

TABLE II: Some secret share structure of our protocols.

| | $[\![x]\!]^{p,0}$ | $\|x\|^{\lambda,0}$ | $\langle x\rangle^0$ | $[x]^0$ | $[x]^1$ | $\langle x\rangle^0$ | $\langle x\rangle^1$ |
|---|---|---|---|---|---|---|---|
| $P_0$ | – | $\{\alpha_j\}_{j\in\mathbb{Z}_\lambda}$ | $([\![r_x]\!]_1, [\![r_x]\!]_2 \in \mathbb{Z}_p)$ | – | $[x]_0 \in \mathbb{Z}_{2^\ell}$ | $([r_x]_1, [r_x]_2 \in \mathbb{Z}_{2^\ell})$ | $([r_x]_0, m_x)$ |
| $P_1$ | $[\![x]\!]_1 \in \mathbb{Z}_p$ | $([\![x]\!]_1, \{[\![\alpha_j]\!]_1, [\![\gamma(x)_j]\!]_1\}_{j\in\mathbb{Z}_\lambda})$ | $([\![r_x]\!]_1, m_x = r_x + x)$ | $[x]_1 \in \mathbb{Z}_{2^\ell}$ | – | $([r_x]_1, m_x = r_x + x)$ | $([r_x]_0, [r_x]_2 \in \mathbb{Z}_{2^\ell})$ |
| $P_2$ | $[\![x]\!]_2^p \in \mathbb{Z}_p$ | $([\![x]\!]_2, \{[\![\alpha_j]\!]_1^p, [\![\gamma(x)_j]\!]_2^p\}_{j\in\mathbb{Z}_\lambda})$ | $([\![r_x]\!]_2, m_x = r_x + x)$ | $[x]_2 \in \mathbb{Z}_{2^\ell}$ | $[x]_2 \in \mathbb{Z}_{2^\ell}$ | $([r_x]_2, m_x = r_x + x)$ | $([r_x]_2, m_x)$ |

$[y]_2)$ and $c\cdot[x] = (c\cdot[x]_1, c\cdot[x]_2)$ and $[x]+c = ([x]_1+c, [x]_2)$, where $c$ is a public value. The same linear operations apply to $\langle\cdot\rangle$, $[\![\cdot]\!]$, and other variants. For $\|\cdot\|$, we have $\|x\| + \|y\| = ([\![x]\!] + [\![y]\!], \{[\![\alpha_j]\!], [\![\gamma(x)_j]\!] + [\![\gamma(y)_j]\!]\}_{j\in\mathbb{Z}_\lambda})$, $c\cdot\|x\| = (c\cdot[\![x]\!], \{[\![\alpha_j]\!], c\cdot[\![\gamma(x)_j]\!]\}_{j\in\mathbb{Z}_\lambda})$ and $c+\|x\| = (c+[\![x]\!], \{[\![\alpha_j]\!], c\cdot[\![\alpha_j]\!] + [\![\gamma(x)_j]\!]\}_{j\in\mathbb{Z}_\lambda})$.

**Secret Sharing.** Let $\Pi_{[\cdot]}$, $\Pi_{[\![\cdot]\!]}$, and $\Pi_{\langle\cdot\rangle}$ denote the corresponding secret sharing protocols of $[\cdot]$, $[\![\cdot]\!]$ and $\langle\cdot\rangle$. By $\Pi^k_{[\cdot]}(x)$ with specified input $x$, we mean that $x$ is shared by $P_k$; by $\Pi^k_{[\cdot]}$ without input, we mean the parties jointly generate a shared random value. We utilize pseudorandom generators (PRG) to reduce the communication [43]. In our protocol description, when we let parties $P_j$ and $P_k$ pick random values together, we mean that these parties invoke PRG with seed $\eta_{j,k}$. The brief sketch of secret sharing schemes is as follows.

- $[x]^k \leftarrow \Pi^k_{[\cdot]}(x)$: (Generate shares of $x$.)
  - $P_k$ and $P_{k+1}$ pick random value $[x]_{k+1} \in \mathbb{Z}_{2^\ell}$ with seed $\eta_{k,k+1}$;
  - $P_k$ sends $x_{k-1} = x - [x]_{k+1} \pmod{2^\ell}$ to $P_{k-1}$.
- $[x]^k \leftarrow \Pi^k_{[\cdot]}$: (Generate shares of a random value.)
  - $P_k$ and $P_1$ pick random value $[x]_{k+1} \in \mathbb{Z}_{2^\ell}$ with seed $\eta_{k,k+1}$;
  - $P_k$ and $P_2$ pick random value $[x]_{k-1} \in \mathbb{Z}_{2^\ell}$ with seed $\eta_{k,k-1}$;
  - $P_k$ calculates $x = [x]_{k+1} + [x]_{k-1}$.
- $[\![x]\!]^k \leftarrow \Pi^k_{[\![\cdot]\!]}(x)$: (Generate shares of $x$.)
  - $P_k$ and $P_{k+1}$ pick random value $[\![x]\!]_{k+1} \in \mathbb{Z}_p$ with seed $\eta_{k,k+1}$;
  - $P_k$ sends $[\![x]\!]_{k-1} = x - [\![x]\!]_{k+1} \pmod{p}$ to $P_{k-1}$.
- $[\![x]\!]^k \leftarrow \Pi^k_{[\![\cdot]\!]}$: (Generate shares of a random value.)
  - $P_k$ and $P_{k+1}$ pick random value $[\![x]\!]_{k+1} \in \mathbb{Z}_p$ with seed $\eta_{k,k+1}$;
  - $P_k$ and $P_{k-1}$ pick random value $[\![x]\!]_{k-1} \in \mathbb{Z}_p$ with seed $\eta_{k-1,k}$;
  - $P_k$ calculates $x = [\![x]\!]_{k+1} + [\![x]\!]^p_{k-1}$.
- $\langle x\rangle^k \leftarrow \Pi^k_{\langle\cdot\rangle}(x)$: (Generate shares of $x$.)
  - All parties perform $[r_x]^k \leftarrow \Pi^k_{[\cdot]}$ in the offline phase, and $P_k$ holds both seeds of $[r_x]_{k+1}$ and $[r_x]_{k-1}$ generation;
  - $P_k$ send $m_x = x - [r_x]_{k+1} - [r_x]_{k-1}$ to $P_{k-1}$ and $P_{k+1}$.
- $\langle x\rangle^k \leftarrow \Pi^k_{\langle\cdot\rangle}$: (Generate shares of a random value.)
  - All parties perform $[r_x]^k \leftarrow \Pi^k_{[\cdot]}$ in the offline phase;
  - $P_{k+1}$ and $P_{k-1}$ pick random value $m_x$ with seed $\eta_{k+1,k-1}$.

**Verifiability of share reconstruction.** Note that the shared form $\langle\cdot\rangle$ has the verifiable reconstruction property against a single malicious party. To be precise, for shared value, $\langle x\rangle$, a single active adversary cannot deceive the honest parties into accepting an incorrect reconstruction result $x + e$ with

**Functionality $\mathcal{F}_{\mathsf{Mult}}[R]$**

$\mathcal{F}_{\mathsf{Mult}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, (m_{x,k-1}, m_{x,k+1}, m_{y,k-1}, m_{y,k+1}))$ from $P_k$ for $k \in \mathbb{Z}_3$, $\mathcal{F}_{\mathsf{Mult}}$ does:
  - if any input messages of two parties is inconsistant, abort;
  - compute $z = (m_{x,0} + m_{x,1} + m_{x,2}) \cdot (m_{y,0} + m_{y,1} + m_{y,2}) \pmod{R}$;
- Upon receiving $(\mathsf{Output}, \mathsf{sid}, \mathsf{abort})$ from $\mathcal{S}$, if $\mathsf{abort} = 1$, $\mathcal{F}_{\mathsf{Mult}}$ abort, else picks $m_{z,0} \leftarrow \mathbb{Z}_R$ and $m_{z,1} \leftarrow \mathbb{Z}_R$.
- calculate $m_{z,2} = z - m_{z,0} - m_{z,1} \pmod{R}$, send $(\mathsf{Output}, \mathsf{sid}, (m_{z,k-1}, m_{z,k+1}))$ to $P_k$ for $k \in \mathbb{Z}_3$.

Fig. 2: The ideal functionality $\mathcal{F}_{\mathsf{Mult}}$.

a non-zero error $e$. This is because any two honest parties can collaboratively reconstruct the secret, and invalid shares will be detected by the honest parties. In addition, the shared form $\|\cdot\|^k$ also maintains the verifiability when one of $P_{k-1}$, $P_{k+1}$ is malicious, since $P_k$ can verify the correctness of the share via MAC checks. We apply the hash function $H$ to reduce the communication cost during the reconstruction of $\langle x\rangle$ [14], where the duplicated messages will be aggregated into a single hash message. Formally, the verifiable reconstruct protocol $\Pi_{\mathsf{Rec}}$ is described as follows:

- $x \leftarrow \Pi_{\mathsf{Rec}}(\langle x\rangle)$:
  - $P_0$ sends $[r_x]_1$ to $P_2$ and $[r_x]_2$ to $P_1$;
  - $P_1$ sends $m_x$ to $P_0$ and $H([r_x]_1)$ to $P_2$;
  - $P_2$ sends $H(m_x)$ to $P_0$ and $H([r_x]_2)$ to $P_1$;

  If the received messages from the other parties are inconsistent, $P_i$ output abort. Otherwise $P_i$ output $x = m_x + [r_x]_1 + [r_x]_2$.
- $x \leftarrow \Pi^k_{\mathsf{Rec}}(\langle x\rangle)$: All parties send their shares (or the hash value) to $P_k$. If the received messages from the other parties are inconsistent, $P_k$ output abort. Otherwise $P_k$ output $x = m_x + [r_x]_1 + [r_x]_2$.
- $x \leftarrow \Pi^k_{\mathsf{Rec}}(\|x\|)$:
  - Each party $P_i$ for $i \neq k$ sends its shares $[\![x]\!]_i, \{[\![\gamma(x)_j]\!]_i\}_{j\in\mathbb{Z}_\lambda}$ to $P_k$;
  - $P_k$ reconstructs $x$ and $\{\gamma(x)_j\}_{j\in\mathbb{Z}_\lambda}$, aborts if any $\gamma(x)_j \neq \alpha_j \cdot x$ for $j \in \mathbb{Z}_\lambda$.

**Preprocessing.** We follow the "preprocessing" paradigm [6] which splits the protocol into two phases: the preprocessing/offline phase is data-independent and can be executed without data input, and the online phase is data-dependent and is executed after data input. Specifically, all the items $r_x$ of share $\langle x\rangle$ of our protocols can be generated in the circuit-depend offline phase. What the parties need to do in the online phase is to collaborate in computing $m_x$ for $P_1$ and $P_2$.

**Multiplication Gate.** We adopt the multiplication protocol $\Pi_{\mathsf{Mult}}$ of ASTRA[12], which is secure under the semi-honest setting. For multiplication $z = x \cdot y$ with input $\langle x \rangle$, $\langle y \rangle$ and output $\langle z \rangle$, all parties first generate $[r_z] \leftarrow \Pi_{[\cdot]}$ for the output wire in the offline phase. To calculate $m_z$ for $P_1$ and $P_2$ in the online phase, it can be written as

$$\begin{aligned} m_z = xy - r_z &= (m_x + r_x)(m_y + r_y) - r_z \\ &= m_x m_y + m_x r_y + m_y r_x + r_x r_y - r_z \ . \end{aligned}$$

$[\Gamma'] = m_x m_y + m_x[r_y] + m_y[r_x]$ can be calculated by $P_1$ and $P_2$ locally and $[\Gamma] = [r_x \cdot r_y] - [r_z]$ can be secret shared by $P_0$ to $P_1$ and $P_2$ in the preprocessing phase. In the online phase, $P_1$ and $P_2$ calculate and reconstruct $[m_z] = [\Gamma'] + [\Gamma]$.

**Inner Product Gate.** For the replicated secret share $\langle \cdot \rangle$, the communication cost of any dimension inner product equals to single multiplication. For inner product $z = \sum_{i=1}^{N} x_i \cdot y_i$ with two input vector $\{\langle x_i \rangle\}_{i \in \mathbb{Z}_N}$, $\{\langle y_i \rangle\}_{i \in \mathbb{Z}_N}$ and output inner product result $\langle z \rangle$, all parties first generate $[r_z] \leftarrow \Pi_{[\cdot]}(r_z)$ for the output wire in the offline phase. To calculate $m_z$ for $P_1$ and $P_2$ in the online phase, it can be written as

$$\begin{aligned} m_z &= \sum_{i=1}^{N} x_i \cdot y_i + r_z = \sum_{i=1}^{N} (m_{x_i} + r_{x_i})(m_{y_i} + r_{y_i}) - r_z \\ &= \sum_{i=1}^{N} (m_{x_i} m_{y_i} + m_{x_i} r_{y_i} + m_{y_i} r_{x_i}) + \sum_{i=1}^{N} r_{x_i} \cdot r_{y_i} - r_z \ . \end{aligned}$$

Similarly, $[\Gamma'] = \sum_{i=1}^{N} m_{x_i} m_{y_i} + m_{x_i}[r_{y_i}] + m_{y_i}[r_{x_i}]$ can be calculated by $P_1$ and $P_2$ locally and $[\Gamma] = [\sum_{i=1}^{N} r_x \cdot r_y] - [r_z]$ can be secret shared by $P_0$ to $P_1$ and $P_2$ in the preprocessing phase. In the online phase, $P_1$ and $P_2$ calculate and reconstruct $[m_z] = [\Gamma'] + [\Gamma]$.

**Batch Verification of Multiplication for Malicious Security.** A series of works [21], [8], [30], [29] realize the maliciously secure multiplication protocol. Some of these works [8], [30], [29] introduce batch verification of replicated multiplication triples at sublinear cost, achieving $O(\log(N))$ communication overhead for $N$ multiplication triples. Once amortized, this overhead becomes negligible. Moreover, this batch verification technique can be directly applied to inner product computations. Some approaches [9], [8] focus on fields, while others [29] extend batch verification to rings. We use $\mathcal{F}_{\mathsf{Mult}}$ to denote an actively secure multiplication functionality for both rings and fields, with $\mathcal{F}_{\mathsf{Mult}}[p]$ for a field and $\mathcal{F}_{\mathsf{Mult}}[2^\ell]$ for a ring. In our cost analysis, we treat $\mathcal{F}_{\mathsf{Mult}}$ for multiplication and inner products in the same way as in the semi-honest setting.

**Reshare.** To ensure the randomness of secret-shared protocol outputs and to enable secure evaluation of subsequent gates, re-randomization is required after specific protocol steps. We employ the resharing technique whose functionality is illustrated in Fig 3. In our implementation, we use a PRG to generate correlated randomness for locally generating secret shares of zero, $\langle 0 \rangle$, and perform re-randomization by locally adding these zero shares to the original secret shares.

> Functionality $\mathcal{F}_{\mathsf{Reshare}}$
>
> $\mathcal{F}_{\mathsf{Mult}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.
>
> - Upon receiving $(\mathsf{Input}, \mathsf{sid}, (m_{x,k-1}, m_{x,k+1}, m_{y,k-1}, m_{y,k+1}))$ from $P_k$ for $k \in \mathbb{Z}_3$, $\mathcal{F}_{\mathsf{Reshare}}$ computes $x = m_{x,0} + m_{x,1} + m_{x,2}$;
> - picks $m_{z,0}, m_{z,1} \leftarrow \mathbb{Z}_{2^\ell}$, $m_{z,2} = x - m_{z,0} - m_{z,1} \pmod{R}$, send $(\mathsf{Output}, \mathsf{sid}, (m_{z,k-1}, m_{z,k+1}))$ to $P_k$ via *private delayed channel* for $k \in \mathbb{Z}_3$.

Fig. 3: The ideal functionality $\mathcal{F}_{\mathsf{Reshare}}$.

**Security Model.** We analyze the security of our protocols in the well-known Universal Composibility (UC) framework [11], which follows the simulation-based security paradigm. The adversary $\mathcal{A}$ is allowed to partially control the communication tapes of all uncorrupted machines, that is, it sees all the messages sent from and to the uncorrupted machines and controls the sequence in which they are delivered. Then, a protocol $\Pi$ is a secure realization of the functionality $\mathcal{F}$, if it satisfies that for every PPT adversary $\mathcal{A}$ attacking an execution of $\Pi$, there is another PPT adversary $\mathcal{S}$ (simulator) attacking the ideal process that uses $\mathcal{F}$ where the executions of $\Pi$ with $\mathcal{A}$ and that of $\mathcal{F}$ with $\mathcal{S}$ makes no difference to any PPT environment $\mathcal{Z}$.

<u>The idea world execution.</u> In the ideal world, the parties $\mathcal{P} := \{P_0, P_1, P_2\}$ only communicate with the ideal functionality $\mathcal{F}$ with the excuted function $f$. All parties send their share to $\mathcal{F}$, $\mathcal{F}$ calculate and output the result depending on the adversary $\mathcal{S}$.

<u>The real world execution.</u> In the real world, the parties $\mathcal{P} := \{P_0, P_1, P_2\}$ communicate with each other via secure channel functionality $\mathcal{F}_{\mathsf{sc}}$ for the protocol execution $\Pi$. Our protocols work in the pre-processing model, but, for simplicity, we analyze the offline and online protocols together as a whole.

**Definition 1.** We say protocol $\Pi$ UC-secure realizes functionality $\mathcal{F}$ if for all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for all PPT environment $\mathcal{Z}$ it holds:

$$\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa) \approx \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa)$$

## III. Secure Sign-bit Extraction

In this section, we propose a novel sign-bit extraction protocol $\Pi_{\mathsf{SignBit}}$. For sign-bit extraction function $z = \mathsf{sign}(x)$, protocol $\Pi_{\mathsf{SignBit}}$ outputs $\langle z \rangle$ from input $\langle x \rangle$. In Sec. IV, we apply it to the actively secure setting.

### A. Protocol Overview

Our goal is to design a low-round protocol for sign-bit extraction. Given $x = m_x + r_x$, the problem of extracting the sign bit can be reduced to a secure comparison. Since $\mathsf{sign}(x) = \mathsf{sign}(m_x + r_x)$, the sign bit of $x$ is effectively determined by the sign bits of $m_x$ and $r_x$, together with the

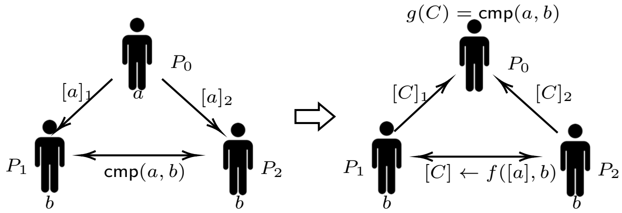



Fig. 4: The structure of our protocol.

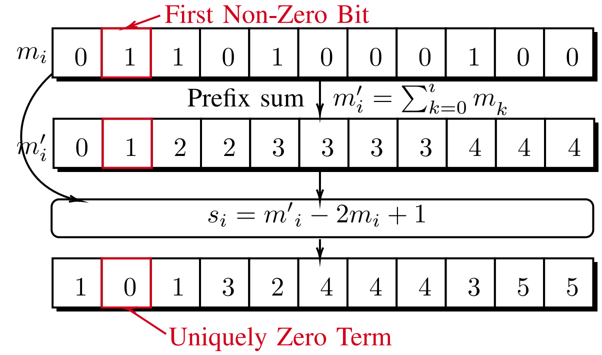



Fig. 5: Transform first non-zero bit detection to the identifying position of the uniquely zero term.

carry bit resulting from the addition of their lower bits (i.e., excluding the sign bits). Let $m_x := \mathsf{sign}(m_x)||\hat{m}_x$ and $r_x := \mathsf{sign}(r_x)||\hat{r}_x$, where $\hat{m}_x$ and $\hat{r}_x$ denote the lower $\ell-1$ bits of $m_x$ and $r_x$, respectively. The expression $(\hat{m}_x + \hat{r}_x > 2^{\ell-1})$ captures whether a carry is generated from the addition of these lower bits. Formally:

$$\mathsf{sign}(x) := \overbrace{(\hat{m_x} + \hat{r_x} \overset{?}{\gtrless} 2^{\ell-1})}^{P_0 \text{ holds } \hat{r_x}, P_1 \text{ and } P_2 \text{ hold } \hat{m_x}} \oplus \overbrace{\mathsf{sign}(r_x)}^{P_0 \text{ locally evaluate}} \oplus \overbrace{\mathsf{sign}(m_x)}^{P_1 \text{ and } P_2 \text{ locally evaluate}} \tag{1}$$

Specifically, $\mathsf{sign}(r_x)$ can be locally evaluated by $P_0$, while $\mathsf{sign}(m_x)$ can be locally evaluated by $P_1$ and $P_2$. The remaining term, $\hat{m}_x + \hat{r}_x \overset{?}{\gtrless} 2^{\ell-1}$—which, given $\hat{r}_x < 2^{\ell-1}$, is equivalent to $\hat{m}_x \overset{?}{\gtrless} 2^{\ell-1} - \hat{r}_x$—is handled via a secure comparison protocol, where $P_1$ and $P_2$ hold $\hat{m}_x$ and $P_0$ holds $2^{\ell-1} - \hat{r}_x$. For simplicity, we denote $a := 2^{\ell-1} - \hat{r}_x$ and $b := \hat{m}_x$ in the following explanation.

A straightforward approach for comparing $a$ and $b$ is for $P_0$ to secret share $a$ with $P_1$ and $P_2$, who then jointly compute the secure comparison $\mathsf{cmp}(a,b)$. However, conventional secure comparison protocols incur significant round complexity and communication overhead, even when $b$ is known to $P_1$ and $P_2$. As illustrated in Fig. 4, we leverage $P_0$ as an auxiliary server to accelerate the secure comparison $\mathsf{cmp}(a,b)$ performed by $P_1$ and $P_2$. With assistance from $P_0$, we design a protocol wherein $P_1$ and $P_2$ compute shared intermediate values, denoted as $C$ in Fig. 4, which are generated by applying a linear function $f$ to $([a],b)$. These intermediate values, termed "comparison materials", can be used to detect the result of $\mathsf{cmp}(a,b)$ via a detection function $g$, such that $g(C)$ yields the comparison outcome.

Direct evaluation of $g(C)$ over secret shares is expensive. However, due to the presence of $P_0$, we can reveal the comparison materials $C$ to $P_0$, allowing it to perform the detection locally. Since $P_0$ does not collude with $P_1$ or $P_2$, privacy is preserved as long as $C$ does not reveal any information about the private input $b$. Once $g(C)$ is computed, $P_0$ re-shares the result as replicated secret shares, producing the final output.

To improve clarity, we outline the underlying logic for constructing the comparison materials. The comparison of $a$ and $b$ is first transformed into identifying the position of the first non-zero bit in the bitwise XOR of $a$ and $b$, denoted as $m = a \oplus b$. When $a < b$, the bit at position $\zeta$ (i.e., the index of the first non-zero bit in $m$) satisfies $b_\zeta = 1$; otherwise, $b_\zeta = 0$. If $a = b$, then $m$ is an all-zero vector. To address this case, we append a 1 to $a$ and a 0 to $b$, ensuring $a \neq b$. We then transform the binary vector $m$ into another vector $s$, where the position of the first non-zero bit in $m$ corresponds to a zero in $s$, i.e., $s_\zeta = 0$ while all other positions in $s$ contain non-zero values, i.e., $s_i \neq 0$ for any other position $i$. Finally, we design an oblivious list randomization that converts $s$ and $b$ into random lists $u$. These random lists are constructed in a way that only reveals $b_\zeta$, the bit at position $\zeta$ in $b$. In the subsequent sections, we present the detailed protocol construction following the above design.

**Step 1: First Non-zero Bit Detection.** We begin by transforming the comparison problem $a \overset{?}{\lessgtr} b$ into the task of identifying the first non-zero bit position. Specifically, we aim to determine $b_\zeta$, where $\zeta \in \mathbb{Z}_\ell$ is the index of the first non-zero bit in the list $\mathcal{L}_1 := \{m_i\}_{i \in \mathbb{Z}_\ell}$. To this end, $P_0$ first performs bit-decomposition on $a$, producing the bit vector $\{a_0, \ldots, a_{\ell-1}\}$, and secret shares each bit to $P_1$ and $P_2$. Given these shares, $P_1$ and $P_2$ locally compute the bitwise XOR $m := a \oplus b$, where each bit $m_i = a_i \oplus b_i$, and $a_i$, $b_i$ denote the $i$-th bits of $a$ and $b$, respectively. It follows that the comparison result $a \overset{?}{\lessgtr} b$ is equivalent to $b_\zeta$, where $\zeta$ is the smallest index such that $m_\zeta \neq 0$.

**Setp 2: Uniquely Zero Value Identification.** To detect the first non-zero position in $m$, we transform the problem into detecting a unique zero value. This process is illustrated in Fig. 5. Let $\mathcal{L}_1 := \{m_i\}_{i \in \mathbb{Z}_\ell}$ be the bit vector obtained in Step 1. We compute a prefix sum vector $m_i' = \sum_{t=0}^{i} m_t$ for each index $i$. By construction, all $m_i' = 0$ until the first 1-bit in $m$, after which $m_i' \geq 1$. We then define a transformation $s_i = m_i' - 2m_i + 1$ for each $i$, obtaining the list $\mathcal{L}_2 := \phi(\mathcal{L}_1) := \{s_i\}_{i \in \mathbb{Z}_\ell}$. This transformation achieves the following properties:

- If $m_i = 0$ and $m_i' = 0$, then $s_i = 1$.
- If $m_i = 1$ and $m_i' = 1$, then $s_i = 0$ (identifies the first non-zero bit).
- If $m_i = 0$ and $m_i' \geq 1$, then $s_i = m_i' + 1 \geq 2$.
- If $m_i = 1$ and $m_i' \geq 2$, then $s_i = m_i' - 1 \geq 1$.

Hence, the resulting list $\mathcal{L}_2$ contains a unique zero at index $\zeta$, the first non-zero bit of $m$, and all other entries are strictly

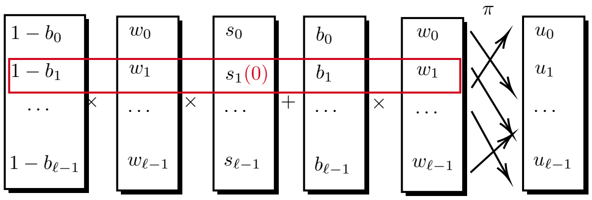



Fig. 6: Oblivious list transformation, transform $b$ and $s$ to random list $u$, where $b_\zeta$ for $s_\zeta = 0$ can be detected by predicate $b_\zeta = (u_i = 0)$.

positive. To ensure correctness under modular arithmetic and avoid unintended wrap-around, all computations are performed in a prime field $\mathbb{Z}_p$ where $p > \ell + 1$. We formally define the transformation as:

$$\mathcal{L}_2 = \phi(\mathcal{L}_1) := \left\{ \left( \sum_{t=0}^{i} m_t \right) - 2 \cdot m_i + 1 \bmod p \right\}_{i \in \mathbb{Z}_\ell}.$$

We summarize the correctness of this transformation in Theorem 1.

**Theorem 1.** *Let $\mathcal{L} := (m_0, \ldots, m_{\ell-1}) \in \{0,1\}^\ell$ be a binary vector. There exists a linear transformation $\phi$ such that $\phi(\mathcal{L}) = (s_0, \ldots, s_{\ell-1}) \in \mathbb{Z}_p^\ell$ satisfies:*

- *Let $i^* \in \mathbb{Z}_\ell$ be the index of the first non-zero bit in $\mathcal{L}$, that is, $m_{i^*} = 1 \wedge \forall i < i^* : m_i = 0$.*
- *$s_{i^*} = 0$ and $s_j \neq 0$ for all $j \neq i^*$.*

*Proof.* Cf Appendix. A □

To ensure $m_i$ operates over the field $\mathbb{Z}_p$, we let $P_0$ in **step 1** secret share $[\![a_i]\!]$ over $\mathbb{Z}_p$, and other two parties perform $[\![m_i]\!] = [\![a_i]\!] + b_i - 2b_i[\![a_i]\!]$ to calculate $m_i = a_i \oplus b_i$. It holds that $a \overset{?}{<} b := \{b_\zeta | s_\zeta = 0, s_i \in \mathcal{L}_2\}$.

**Step 3: Oblivious List Randomization.** At this point, the list $\mathcal{L}_2 := \{s_i \in \mathbb{Z}_p\}_{i \in \mathbb{Z}_\ell}$ and the bit string $b$ cannot be directly revealed to $P_0$ due to privacy concerns. To address this, we perform an oblivious randomization process that hides the information of $b$. Fig. 6 illustrates this procedure. The idea is to transform $\mathcal{L}_2$ and $b$ into a randomized list $\mathcal{L}_3 := \{u_i \in \mathbb{Z}_p\}_{i \in \mathbb{Z}_\ell}$, such that $\mathcal{L}_3$ allows detection of $b_\zeta$ through a public function $g(\mathcal{L}_3)$, while preserving the secrecy of $b$. Let $w_i \in \mathbb{Z}_p^*$ be a random nonzero scalar and $\pi : [\ell] \to [\ell]$ a random permutation. For each index $i$, we define:

$$u_{\pi(i)} = \begin{cases} w_i \cdot s_i \pmod p & b_i = 0 \\ w_i & b_i = 1 \end{cases}.$$

and its corresponding detection function is defined to return a positive result if the sequence $\mathcal{L}_3$ contains the element 0, namely,

$$g(u) = \begin{cases} 0 & \exists u_i = 0 \\ 1 & \forall u_i \neq 0 \end{cases}.$$

Intuitively, this transformation masks each $s_i$ using a random scalar $w_i$ and permutes the resulting list using $\pi$ to conceal the location of the zero. Depending on $b_i$, we output either $w_i \cdot s_i$ (preserving zero) or $w_i$ (random nonzero). In particular:

- If $b_\zeta = 0$, then $u_{\pi(\zeta)} := w_\zeta \cdot s_\zeta \pmod p = 0$ which implies $0 \in \{u_i\}_{i \in \mathbb{Z}_\ell}$ , and hence $u_{\pi(\zeta)} = 0$, resulting in $g(\mathcal{L}_3) = 0$.
- If $b_\zeta = 1$, then $u_{\pi(\zeta)} = w_\zeta \neq 0$ which implies $0 \notin \{u_i\}_{i \in \mathbb{Z}_\ell}$, so $g(\mathcal{L}_3) = 1$.

**Masking $b_\zeta$ to Prevent Leakage.** Revealing $\mathcal{L}_3 := \{u_i\}_{i \in \mathbb{Z}_\ell}$ directly to $P_0$ would disclose the comparison result. To mitigate this, a binary mask $\Delta \in \{0,1\}$ is introduced and held by $P_1$ and $P_2$. During the construction of $\mathcal{L}_3$, $P_1$ and $P_2$ replace $b_i$ with $\Delta \oplus b_i$, such that the final output satisfies:

$$a \overset{?}{<} b = g(\mathcal{L}_3) \oplus \Delta.$$

As a result, $P_0$ holds $t := g(\mathcal{L}_3)$, and $P_1$, $P_2$ each hold the bias $\Delta$, forming a Boolean secret sharing of the comparison outcome. This mechanism naturally integrates with sign-bit extraction. Given that $\mathsf{sign}(x) = \mathsf{sign}(r_x) \oplus \mathsf{sign}(m_x) \oplus b_\zeta$, each party can locally compute $\mathsf{sign}(r_x) \oplus t$ and $\mathsf{sign}(m_x) \oplus \Delta$ to obtain boolean secret sharing of the sign-bit extraction.

**The Second Round: obtaining $\langle\cdot\rangle$-shared result.** If the output is required in the form of $\langle\cdot\rangle$-secret sharing, an additional round of interaction is needed, with a communication cost of $2\ell$ bits for resharing. Let $z := b_\zeta$ and $\langle z \rangle := \{m_z, [r_z]\}$ denote the replicated share of $z$. Given that $z = t \oplus \Delta = \Delta + t - 2t \cdot \Delta$ and $r' \in \mathbb{Z}_{2^\ell}$ be a random value generated via the offline phase. Then,

$$\begin{aligned} m_z &= \Delta + t - 2t \cdot \Delta - r_z \\ &= \Delta + r' + t(1 - 2\Delta) - r' - r_z \\ &= \underbrace{\Delta + r'}_{[\cdot]\text{-shared}} + \underbrace{(t - r')}_{P_0 \text{ holds}} \cdot \underbrace{(1 - 2\Delta)}_{P_1/P_2 \text{ holds}} - \underbrace{2 \cdot \Delta \cdot r' - r_z}_{[\cdot]\text{-shared}}. \end{aligned} \tag{2}$$

During the offline phase, the parties jointly sample $[r']$ and $[r_z]$ using $\Pi_{[\cdot]}$. Then, $P_1$ and $P_2$ locally compute $\Gamma = \Delta + r' - 2 \cdot \Delta \cdot r' - r_z$ in shared form $[\cdot]$ and reconstruct $\Gamma$ without leak information about $r'$. In the online phase, $P_0$ can directly send $m' = t - r'$ to $P_1$ and $P_2$. Finally, $P_1$ and $P_2$ locally calculate $m_z = m' \cdot (1 - 2\Delta) + \Gamma$.

### B. Concrete Construction

By filling in some detailed descriptions, we complete our protocol, which is depicted in Fig. 7. Next, we will explain our protocol step by step as follows.

- In the offline phase, $P_1$ and $P_2$ generate $\Delta$ to mask the sign-bit and $\Gamma$ for the second round resharing. $P_0$ split the sign-bit of $r_x$ and the remain part $\hat{r}_x$. As mentioned before, the sign-bit $\mathsf{sign}(x)$ equal to $(\hat{m}_x + \hat{r}_x \geq 2^{\ell-1}) \oplus \mathsf{sign}(r_x) \oplus \mathsf{sign}(m_x)$. $P_0$ bit-extract $2^{\ell-1} - \hat{r}_x$ for the comparison $\hat{m}_x + \hat{r}_x \geq 2^{\ell-1}$, and share each bit in the field $\mathbb{Z}_p$.
- In steps 1-3, $P_1$ and $P_2$ set $[\![m_i]\!]^p$, where $m_i$ represents the $i$-th bit of $\hat{m}_x \oplus (2^{\ell-1} - \hat{r}_x)$. The transformation can be locally performed. Moreover, we set $\hat{m}_{x,\ell} = 1$ and $[\![r_{x,\ell}]\!] = [\![0]\!]$ to ensure that protocol output equals to 1 when $\hat{m}_x + \hat{r}_x = 2^{\ell-1}$.

**Protocol $\Pi_{\mathsf{SignBit}}(\langle x \rangle)$**

$P_j$ and $P_k$ hold the common seed $\eta_{j,k} \in \{0,1\}^\lambda$.
Input : $\langle\cdot\rangle$-shared value of $x$.
Output : $\langle\cdot\rangle$-shared value of $z = \mathsf{sign}(x)$.

**Preprocessing:**

- All parties perform $[r'], [r_z] \leftarrow \Pi_{[\cdot]}$;
- $P_i$, for $i \in \{1,2\}$ generates the same random value $\Delta \in \{0,1\}$ via PRF with seed $\eta_{1,2}$ and reveals $[\Gamma] = \Delta + [r'] - 2\Delta \cdot [r'] + [r_z]$ to each other.
- $P_0$ does:
  1) calculate $\hat{r}_x = r_x - \mathsf{sign}(r_x) \cdot 2^{\ell-1} \in \mathbb{Z}_{2^{\ell-1}}$
  2) extract $2^{\ell-1} - \hat{r}_x$ as $\{r_{x,0}, \dots, r_{x,\ell-2}\}$
  3) perform $[\![r_{x,i}]\!]^p \leftarrow \Pi^p_{[\![\cdot]\!]}(r_{x,i})$ for $i \in \mathbb{Z}_{\ell-1}$, taking the biggest prime of $p \in (\ell, 2^{\log \ell+1}]$;

**Online:**

- $P_j$, for $j \in \{1,2\}$ does:
  1) set $\hat{m}_x = m_x - \mathsf{sign}(m_x) \cdot 2^{\ell-1}$ and bitexact it as $\{\hat{m}_{x,i} \in \{0,1\}\}_{i \in \mathbb{Z}_{\ell-1}}$ while $\sum_{i=0}^{\ell-2} 2^{\ell-2-i}\hat{m}_{x,i} = \hat{m}_x$;
  2) set $\hat{m}_{x,\ell-1} = 1$ and $[\![r_{x,\ell-1}]\!] = [\![0]\!]$;
  3) set $[\![m_i]\!]^p = \hat{m}_{x,i} + [\![r_{x,i}]\!]^p - 2\hat{m}_{x,i} \cdot [\![r_{x,i}]\!]^p$ for $i \in \mathbb{Z}_\ell$.
  4) pick same random values $\{w_i\}_{i \in \mathbb{Z}_\ell} \in (\mathbb{Z}_p^*)^{2\ell}$ via PRF with seed $\eta_{1,2}$;
  5) calculate $[\![m'_i]\!]^p = \sum_{t=1}^{i} [\![m_t]\!]^p - 2 \cdot [\![m_i]\!]^p + 1$ and $[\![u_i]\!]^p = w_i \cdot [\![m'_i]\!]^p \cdot (1 \oplus \mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta) + w_i \cdot (\mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta)$ for $i \in \mathbb{Z}_\ell$;
  6) pick a random permutation $\pi$ via PRF with seed $\eta_{1,2}$ and permute the list $\{[\![\hat{u}_i]\!]^p\}_{i \in \mathbb{Z}_\ell} = \pi(\{[\![u_i]\!]^p\}_{i \in \mathbb{Z}_\ell})$;
  7) reveal $\{[\![\hat{u}_i]\!]^p\}_{i \in \mathbb{Z}_\ell}$ to $P_0$;
- $P_0$ sets $t = \mathsf{sign}(r_x)$ if $\exists \hat{u}_i = 0$ for $i \in \mathbb{Z}_\ell$ else $t = 1 \oplus \mathsf{sign}(r_x)$ to $P_j$, for $j \in \{1,2\}$; %output binary share $(\Delta, t)$
- $P_0$ sends $m' = t - r'$ to $P_1$ and $P_2$;
- $P_j$, for $j \in \{1,2\}$ sets $m_z = m' - 2\Delta \cdot m' + \Gamma$;
- All parties invoke $\mathcal{F}_{\mathsf{Reshare}}$ to re-randomize $\langle z \rangle = ([r_z], m_z)$.

Fig. 7: The Sign-bit Extraction Protocol.

- In step 5, $P_1, P_2$ transfer $[\![m_i]\!]^p$ to $[\![m'_i]\!]^p$ via the transformation $\phi$ and generate the aforementioned lists $\{u_i\}_{i \in \mathbb{Z}_\ell}$. Considering $(\hat{m}_x + \hat{r}_x \geq 2^{\ell-1}) \oplus \mathsf{sign}(r_x) \oplus \mathsf{sign}(m_x)$, we let $P_1$ and $P_2$ further XOR the sign-bit of $m_x$, such that $P_0$ will output $\mathsf{sign}(m_x) \oplus \hat{m}_{x,\zeta} \oplus \Delta$ rather than $\hat{m}_{x,\zeta} \oplus \Delta$.
- In step 6, $P_1, P_2$ random shuffle the list $\{u_i\}_{i \in \mathbb{Z}_\ell}$ with the same permutation $\pi$.
- In step 7, $P_1, P_2$ open $\{u_i\}_{i \in \mathbb{Z}_\ell}$ to $P_0$. $P_0$ can draw the conclusion based on observations of $\{u_i\}_{i \in \mathbb{Z}_\ell}$: if there exist $i$ that $u_i = 0$, then $\mathsf{sign}(m_x) \oplus \hat{m}_{x,\zeta} \oplus \Delta = 0$, otherwise $\mathsf{sign}(m_x) \oplus \hat{m}_{x,\zeta} \oplus \Delta = 1$.
- For the second round of online phase, $P_0$ further XOR $\mathsf{sign}(r_x)$ to get $\mathsf{sign}(r_x) \oplus \mathsf{sign}(m_x) \oplus \hat{m}_{x,\zeta} \oplus \Delta$ which is the masked value of sign-bit, stemming from $\mathsf{sign}(x) = \mathsf{sign}(r_x) \oplus \mathsf{sign}(m_x) \oplus \hat{m}_{x,\zeta}$. Now, $P_1$ and $P_2$ hold $\Delta$. We use the aforementioned reshare technique to transfer the XOR shared value $\{\mathsf{sign}(x) \oplus \Delta, \Delta\}$ to $\langle\cdot\rangle$-shared value, with one round and $2\ell$ communication.

**Efficiency.** Our sign-bit extraction protocol $\Pi_{\mathsf{SignBit}}$ costs 1 round with communication of $(\ell-1)\log\ell$ bits in the offline phase and requires 1 rounds with communication of $2\ell\log\ell$ bits in the online phase to output a boolean shared result; costs 1 round with communication of $(\ell-1)\log\ell + 2\ell$ bits in the offline phase and requires 2 rounds with communication of $2\ell\log\ell + 2\ell$ bits in the online phase to output $\langle\cdot\rangle$-shared result.

**Functionality $\mathcal{F}_{\mathsf{SignBit}}[\mathbb{Z}_{2^\ell}]$**

$\mathcal{F}_{\mathsf{SignBit}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.

**Input:**

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, (m_{k-1}, m_{k+1}))$ from $P_k$ for $k \in \mathbb{Z}_3$, send $(\mathsf{Input}, \mathsf{sid}, P_k)$ to $\mathcal{S}$ and record $(m_{k-1}, m_{k+1}) \in (\mathbb{Z}_{2^\ell})^2$;

**Execution:**

- Compute $z := \mathsf{sign}(m_0 + m_1 + m_2)$;
- Pick random $u_1, u_2 \leftarrow \mathbb{Z}_{2^\ell}$, set $u := u_0 + u_1$ and $u_2 := z - u$;
- Send $(\mathsf{Output}, \mathsf{sid}, (u_{k-1}, u_{k+1}))$ to $P_k$ for $k \in \mathbb{Z}_3$.

Fig. 8: The ideal functionality $\mathcal{F}_{\mathsf{SignBit}}$.

**Security.** We analyze the security of our sign-bit extraction protocol in the UC framework. We define the functionality $\mathcal{F}_{\mathsf{SignBit}}$ for our sign-bit extraction in Fig. 8.

**Theorem 2.** *The protocol $\Pi_{\mathsf{SignBit}}$ as depicted in Fig. 7 UC realizes $\mathcal{F}_{\mathsf{SignBit}}$ in the $\mathcal{F}_{\mathsf{Reshare}}$-hybrid model against semi-honest PPT adversaries who can statically corrupt up to one party.*

*Proof.* To prove Thm. 2, we construct a PPT simulator $\mathcal{S}$, such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between the ideal world and the real world. We consider the following cases:

Case 1: $P_0$ is corrupted.

Simulator: The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest $P_1$, $P_2$. $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $\{[\![r_{x,i}]\!]^p_1\}_{i \in \mathbb{Z}_{\ell-1}}, [r']_1$ form corrupted $P_0$ to $P_1$, and $\{[\![r_{x,i}]\!]^p_2\}_{i \in \mathbb{Z}_{\ell-1}}, [r']_2$ form corrupted $P_0$ to $P_2$, $\mathcal{S}$ picks random list $\{\hat{u}_i\}_{i \in \mathbb{Z}_\ell}$ as following steps:
  - Set $\hat{u}_i \leftarrow \mathbb{Z}_p^*$.
  - Pick coin $\leftarrow \{0,1\}$.
  - Pick index $\leftarrow \mathbb{Z}_\ell$.
  - If coin equal 1, set $\hat{u}_{\mathsf{index}} = 0$.
  - Send $\{\hat{u}_i\}_{i \in \mathbb{Z}_\ell}$ to $P_0$.
- Upon receiving $m'$ from corrupted $P_0$ to $P_1$ and $P_2$, $\mathcal{S}$ sends $(\mathsf{Input}, \mathsf{sid}, [r_x]_1, [r_x]_2)$ to $\mathcal{F}_{\mathsf{SignBit}}$ and receives $(\mathsf{Output}, \mathsf{sid}, [r'_z]_1, [r'_z]_2)$;
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, [r_z]_1, [r_z]_2)$ from corrupted $P_0$ to internal $\mathcal{F}_{\mathsf{Reshare}}$, $\mathcal{S}$ sends $(\mathsf{Output}, \mathsf{sid}, [r'_z]_1, [r'_z]_2)$ as output of $\mathcal{F}_{\mathsf{Reshare}}$;

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{Real}_{\Pi_{\mathsf{SignBit}}, \mathcal{A}, \mathcal{Z}}(1^\kappa)$.

Hybrid $\mathcal{H}_1$: It is same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, list $\hat{u}_i$ reveal to $P_0$ are picked as follow

- Set $\hat{u}_i \leftarrow \mathbb{Z}_p^*$.
- Pick $\mathsf{coin} \leftarrow \{0,1\}$.
- Pick $\mathsf{index} \leftarrow \mathbb{Z}_\ell$.
- If coin equal 1, set $\hat{u}_{\mathsf{index}} = 0$.

rather than $[\![u_i]\!]^p = \pi(w_i \cdot [\![m_i']\!]^p \cdot (1 \oplus \mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta) + w_i \cdot (\mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta))$.

$\mathcal{H}_1$ is same as ideal world $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa)$

**Claim 1.** *If* $\mathsf{PRF}^{(\mathbb{Z}_p)^p}$ *is the secure permutation with adversarial advantage* $\mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A})$*, then* $\mathcal{H}_1$ *and* $\mathcal{H}_0$ *are indistinguishable with advantage* $\epsilon = \mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A})$.

*Proof.* It is easy to verify the outputs of $\mathcal{H}_1$ and $\mathcal{H}_1$ are same. For the real-world list $[\![u_i]\!]^p = \pi(w_i \cdot [\![m_i']\!]^p \cdot (1 \oplus \mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta) + w_i \cdot (\mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta))$, it only contains zero when both $m_i' = 0$ and $\mathsf{sign}(m_x) \oplus \hat{m}_{x,i} \oplus \Delta = 0$, which happens with a probability of 1/2 (Since $\Delta$ is choosen uniformly at random and unknown to the adversary). Apart from the uncertain zero, every other element is selected uniformly at random from $\mathbb{Z}_p^*$. The list $\hat{u}_i$ in hybrid $\mathcal{H}_1$ keep the same distribution. The permutation $\pi$, derived from $\mathsf{PRF}^{\mathbb{Z}_p{}^p}$ in the real-world execution, is replaced by a truly random choice in the hybrid—namely, index $\leftarrow \mathbb{Z}\ell$. As a result, the overall distinguishing advantage is bounded by $\epsilon = \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p^p}}(1^\kappa, \mathcal{A})$. □

Case 2: $P_1$(or $P_2$) is corrupted.

Simulator: The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest $P_0$, $P_2$. $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- $\mathcal{S}$ generate $[r']_1$ using PRF with seed $\eta_{0,1}$.
- $\mathcal{S}$ picks $[\Gamma]_2 \leftarrow \mathbb{Z}_{2^\ell}$ and acts as $P_2$ to send it to $P_1$.
- Upon receiving $[\Gamma]_1$ from $P_1$, $\mathcal{S}$ picks $[\![r_{x,i}]\!]_1 \leftarrow \mathbb{Z}_p$ for $i \in \mathbb{Z}_\ell$ and acts as $P_0$ to send it to $P_1$.
- Upon receiving $\{[\![\hat{u}_j]\!]_1\}_{j\in\mathbb{Z}_{\ell+1}^*}$ from corrupted $P_1$ to $P_0$, $\mathcal{S}$ does.
  - Invoke PRF with $\eta_{1,2}$ to generate permutation $\pi$, $\{w_i\}_{i\in\mathbb{Z}_\ell} \in (\mathbb{Z}_p^*)^\ell$, $\Delta \in \mathbb{Z}_2$.
  - Calculate $\{[\![u_i]\!]_1\}_{i\in\mathbb{Z}_\ell} = \pi^{-}(\{[\![\hat{u}_i]\!]_1\}_{i\in\mathbb{Z}_\ell})$.
  - Calculate $\hat{m}_{x,i}'$ via $\{[\![u_i]\!]_1\}_{i\in\mathbb{Z}_{\ell+1}}$, $w_i, \Delta$ and $[\![r_{x,i}]\!]_1$
  - Set $m_x = m_{x,0}||\hat{m}_{x,1}'||\ldots||\hat{m}_{x,\ell-1}'$.
  - Act as the corrupted $P_1$ ($P_2$) to send $(\mathsf{Input}, \mathsf{sid}, m_x)$ to the external $\mathcal{F}_{\mathsf{SignBit}}$ and receive $(\mathsf{Output}, \mathsf{sid}, [r_z']_1, m_z')$;
  - Pick random $m' \leftarrow \mathbb{Z}_{2^\ell}$ and act as $P_0$ send to corrupted $P_1$.
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, [r_z]_1, m_z)$ from corrupted $P_1$ to internal $\mathcal{F}_{\mathsf{Reshare}}$, $\mathcal{S}$ sends $(\mathsf{Output}, \mathsf{sid}, [r_z']_1, m_z')$ as output of $\mathcal{F}_{\mathsf{Reshare}}$;

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{Real}_{\Pi_{\mathsf{SignBit}},\mathcal{A},\mathcal{Z}}(1^\kappa)$.

**Protocol $\Pi^k_{\mathsf{BatchRec}}(\{\|x^{(i)}\|^k\}_{i\in\mathbb{Z}_N})$**

Input : $N$ numbers of $\|\cdot\|$-shared value.
Output : $P_k$ receive $\{x^{(i)}\}_{i\in\mathbb{Z}_N}$.
**Execution:**

- $P_{k-1}$ and $P_{k+1}$ reveal $\{x^{(i)}\}_{i\in\mathbb{Z}_N}$ to $P_k$;
- $P_k$ picks $\lambda$ random value $\{w_j \in \mathbb{Z}_p^*\}_{j\in\mathbb{Z}_\lambda}$ and send them to $P_{k-1}$ and $P_{k+1}$.
- $P_{k-1}$ and $P_{k+1}$ do
  - calculate $[\![t_j]\!] = \sum_{i=0}^{N-1}(w_j)^i \cdot [\![\gamma(x^{(i)})_j]\!]$ for $j \in \mathbb{Z}_\lambda$.
  - reveal $\{t_j\}_{j\in\mathbb{Z}_\lambda}$ to $P_0$.
- $P_k$ cacluaItes $\hat{t}_j = \alpha_j \cdot \sum_{i=0}^{N-1}(w_j)^i \cdot x^{(i)}$ and abort if exist $\hat{t}_j \neq t_j$ for any $j \in \mathbb{Z}_\lambda$.
- $P_k$ outputs $\{x^{(i)}\}_{i\in\mathbb{Z}_N}$.

Fig. 9: The Batch Verifiable Reconstruction Protocol

Hybrid $\mathcal{H}_1$: It is same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, $[\![r_{x,i}]\!]_1$, $m'$ and $[\Gamma]_2$ are picked uniformly random instead of calculating from $r_{x,j}$, $t - r'$, $\Delta + [r']_2 - 2\Delta \cdot [r']_2 + [r_z]_2$.

Hybrid $\mathcal{H}_1$ is same as ideal world $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

**Claim 2.** *If* $\mathsf{PRF}^{\mathbb{Z}_p}$ *and* $\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}$ *are the secure pseudorandom functions with adversarial advantage* $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\kappa, \mathcal{A})$ *and advantage* $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}}(1^\kappa, \mathcal{A})$*, then* $\mathcal{H}_1$ *and* $\mathcal{H}_0$ *are indistinguishable with advantage* $\epsilon = \ell \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\kappa, \mathcal{A}) + 2 \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}}(1^\kappa, \mathcal{A})$.

*Proof.* It is easy to verify the outputs of the real world and ideal world are consistent. In the real world, secret share of $r_{x,j}$ is generated by $\mathsf{PRF}^{\mathbb{Z}_p}$ which is indistinguishable from $[\![r_{x,i}]\!]_1$ randomly picked by $\mathcal{S}$ with advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\kappa, \mathcal{A})$. Considering such a procedure repeat $\ell$ times, the advantage is $\ell \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\kappa, \mathcal{A})$. Similarly, $\Delta + [r']_2 - 2\Delta \cdot [r']_2 + [r_z]_2$ is calcualted by random value generated by $\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}$ which is indistinguishable from $[\Gamma]_2$ with advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}}(1^\kappa, \mathcal{A})$; $m' = t - r'$ can be view as ciphertext masked by $\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}$ output $r'$. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable with advantage $\epsilon = \ell \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\kappa, \mathcal{A}) + 2 \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}}(1^\kappa, \mathcal{A})$. □

This concludes the proof. □

**The ReLU Construction.** We construct the semi-honest ReLU protocol $\Pi_{\mathsf{ReLU}}$ based on the protocol $\Pi_{\mathsf{SignBit}}$ (see Appendix A for details). The ReLU function can be expressed as $w = x \cdot (1 - \mathsf{sign}(x)) = x - x \cdot \mathsf{sign}(x)$, which can be evaluated by combining $\Pi_{\mathsf{Mult}}$ and $\Pi_{\mathsf{SignBit}}$. However, we aim to *eliminate the extra round of invoking* $\Pi_{\mathsf{Mult}}$ by embedding its communication round into $\Pi_{\mathsf{SignBit}}$. Let $\langle z\rangle = \Pi_{\mathsf{SignBit}}(\langle x\rangle)$ and $\langle w\rangle = \Pi_{\mathsf{Mult}}(\langle x\rangle, \langle z\rangle)$, we have:

$$
\begin{aligned}
m_w &= m_x m_z + m_x r_z + m_z r_x + r_x r_z - r_w \\
&= m_x m_z + m_x r_z + (m' - 2\Delta m' + \Gamma) r_x + r_x r_z - r_w \\
&= m_x m_z + m_x r_z + (1 - 2\Delta)(m' r_x + r'') + \Gamma'
\end{aligned}
$$

where $m'$, $\Delta$, and $\Gamma$ are fresh random values introduced in $\Pi_{\mathsf{SignBit}}$, and we used the fact that $m_z = m' - 2\Delta m' + \Gamma$ as defined in $\Pi_{\mathsf{SignBit}}$. We denote $\Gamma' = \Gamma \cdot r_x - (1-2\Delta)r'' + r_x \cdot r_z - r_w$, where $r''$ is a fresh random used to protect the privacy of $r_w$. We let $P_1$ and $P_2$ calculate $[\Gamma'] = \Gamma \cdot [r_x] - (1-2\Delta)[r''] + [r_x \cdot r_z] - [r_w]$ locally in the offline phase. $P_1$ and $P_2$ reveal $[\Gamma''] = m_x \cdot [r_z] + [\Gamma']$ to each other in the first round of $\Pi_{\mathsf{SignBit}}$. For item $(1-2\Delta)(m' r_x + r'')$, $P_0$ send $m'' = m' r_x + r''$ to $P_1$ and $P_2$. Then $P_1, P_2$ locally calculate $m_w = m_x \cdot m_z + \Gamma'' + (1-2\Delta)m''$. Our ReLU protocol requires 1 rounds with $(\ell-1)\log\ell + 2\ell$ bits of communication in the preprocessing phase and requires 2 rounds with $2\ell\log\ell + 4\ell$ bits of communication in the online phase. In Appendix. A, we discuss other PPML operators constructed using our protocol $\Pi_{\mathsf{SignBit}}$.

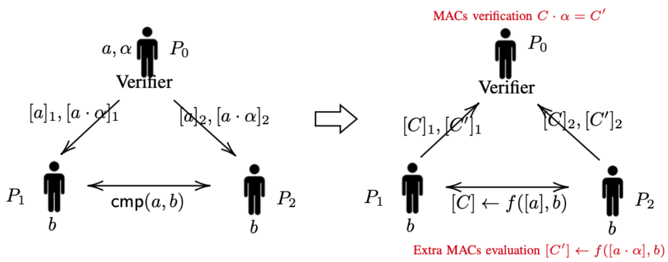



Fig. 10: Apply IT-mac in our comparison protocol.

## IV. Achieving Malicious Security

Given a sign-bit extraction pair $\{\langle x \rangle, \langle z \rangle\}$ with $z = \mathsf{sign}(x)$, a malicious adversary could potentially inject faults to cause $\mathsf{sign}(x) \neq z$. To counter such attacks, we incorporate several mechanisms into the protocol to detect malicious behavior.

Our maliciously secure protocol is illustrated in Fig. 11. We observe a clear asymmetry between $P_0$ and $P_1/P_2$ in our semi-honest protocol, the malicious version benefits from this asymmetry. Notably, $P_0$'s role facilitates efficient integrity verification by embedding IT-MACs [16] into the input layer, as shown in the overall architecture of Fig. 10.

In the passively secure version, during the offline phase, $P_0$ sends secret shares of $r_{x,i}$; during the online phase, $P_1$ and $P_2$ reveal the comparison result $u$ to $P_0$, who checks the result and sends the masked outcome $m'$ back. In the malicious setting, $P_0$ additionally shares $r_{x,i} \cdot \alpha$ together with $r_{x,i}$. Meanwhile, $P_1$ and $P_2$ compute $m' \cdot \alpha$ alongside $m'$. Upon receiving $m'$, $P_0$ can validate it by checking its MAC tag, thus ensuring correctness if $P_1$ or $P_2$ are corrupted.

The more challenging case arises when $P_0$ is malicious. In particular, the integrity of the value $t$ (the masked comparison result) becomes difficult to guarantee. While zero-knowledge proofs could theoretically validate correctness, the non-linear operations involved incur significant overhead. To address this, we adopt the *dual execution paradigm* [23], [20], which validates correctness by executing the protocol twice with switched roles. After the first execution, $P_0$ holds $t = z \oplus \Delta$, and $P_1$, $P_2$ hold $\Delta$. Then the roles of $P_0$ and $P_1$ are swapped, resulting in $P_1$ holding $t' = z \oplus \Delta'$, and $P_0$, $P_2$ holding $\Delta'$. This approach enables cross-checks to detect inconsistencies:

- If $P_0$ attempts to alter $\Delta'$, $P_2$—who also holds $\Delta'$—can detect it.
- If $P_0$ tampers with $t$, the inconsistency is caught by verifying whether $t \oplus \Delta = t' \oplus \Delta'$.
- Likewise, any manipulation of the outputs by $P_1$ or $P_2$ can also be detected through similar consistency checks.

Another point to note is that the typical dual execution protocol will introduce a one-bit leakage.

**Avoiding One-bit Leakage.** A malicious adversary can exploit probabilistic behavior to perform a selective failure attack. For example, a corrupted $P_0$ may inject an error $e$ into the input $x$ during the generation of $[\![r_{x,i}]\!]$ in Step (3) of the preprocessing phase (Fig. 7). During dual execution, the adversary can infer whether the modified input $x + e$ changes the sign-bit by observing whether the two executions yield consistent results. Similarly, a corrupted $P_1$ or $P_2$ could inject faults into $m_x$ while computing the comparison material $u_i$, affecting the output $z = \mathsf{sign}(x)$. If the protocol only verifies the correctness of the final output—without checking intermediate computations—such one-bit leakage becomes unavoidable. In our approach, we employ a step-by-step correctness check to avoid one-bit leakage.

**Step-by-step Correctness.** Below, we analyze the communication messages of the $\Pi_{\mathsf{SignBit}}$ protocol step by step, and explain how we ensure the correctness of its computation:

- $[\![r_{x,i}]\!]$: We employ an actively secure multiplication protocol $\mathcal{F}_{\mathsf{Mult}}$ to generate $[\![r_{x,i}]\!]$.
- $[\![\hat{u}_i]\!]$: We employ IT-MAC to ensure the correctness of $[\![\hat{u}_i]\!]$.
- $t$: The final output $t$ is validated using the dual execution paradigm.

It is important to emphasize that, once IT-MACs are introduced for $[\![r_{x,i}]\!]$, the MAC generation must also be carried out using a maliciously secure multiplication protocol $\mathcal{F}_{\mathsf{Mult}}$. Otherwise, a corrupted $P_0$ could inject errors into the MACs to launch a selective failure attack. Following we will provide a detailed breakdown of how the combination of $\mathcal{F}_{\mathsf{Mult}}$, IT-MACs, and dual execution ensures the correctness of each message and prevents malicious tampering throughout the protocol.

**Maliciously Secure Additively Share Generation.** We aim to generate the pair $\{[\![r_{x,i}]\!]\}_{i \in \mathbb{Z}_{\ell-1}}$, and $[r_x]$ under maliciously secure model, such that $r_x = 2^{\ell-1} - \sum_{i=0}^{\ell-1} 2^i \cdot [\![r_{x,i}]\!] + 2^{\ell-1} \cdot \mathsf{sign}(r_x)$ holds. Our strategy proceeds in three steps: (1) generate secret shares of each bit $r_{x,i}$ under both $\langle\!\langle \cdot \rangle\!\rangle$ and $\langle \cdot \rangle$; (2) use $\langle r_{x,i} \rangle$ to calculate $\hat{r}_x = 2^{\ell-1} - \sum_{i=0}^{\ell-1} 2^i \cdot [\![r_{x,i}]\!]$ and $r_x = \hat{r}_x + 2^{\ell-1} \cdot \mathsf{sign}(r_x)$; (3) convert $\langle r_x \rangle$ and $\langle\!\langle r_{x,i} \rangle\!\rangle$ into 2PC shares $[r_x]$ and $[\![r_{x,i}]\!]$ via local computation. To securely generate two types share of $r_{x,i}$, we let $P_0$ and $P_1$ locally generate a random bit $d_{1,i}$, unknown to $P_2$, by setting the shares they both hold to random bit (i.e., setting $[r_d]_1 \leftarrow \{0,1\}$ for $\langle d \rangle$) and the other shares to 0. We denote a list of such boolean value as $\{d_{1,i}\}_{i \in \mathbb{Z}_\ell}$. Similarly, $P_0$ and $P_2$ generate another random bit $d_{2,i}$, unknown to $P_1$. Next, we compute $r_{x,i} = d_{1,i} \oplus d_{2,i}$ using an actively secure protocol. Notably, the local random secret shares $d_{1,i}$ and $d_{2,i}$ can be treated as valid shares over any ring or field. If the XOR operation is computed in $\mathbb{Z}_{2^\ell}$, we obtain shares $\langle r_{x,j} \rangle$; if performed in

**Protocol $\Pi_{\mathsf{VSignBit}}(\langle x\rangle^0)$**

$P_j$ and $P_k$ hold the common seed $\eta_{j,k} \in \{0,1\}^\kappa$, $P_0$ holds the MAC keys $\alpha := (\alpha_0,\ldots,\alpha_{\lambda-1}) \in \mathbb{Z}_p^\lambda$, $P_1$ holds $\alpha' := (\alpha'_0,\ldots,\alpha'_{\lambda-1}) \in \mathbb{Z}_p^\lambda$, all parties hold $\boldsymbol{\langle}\alpha\boldsymbol{\rangle}^0$ and $\boldsymbol{\langle}\alpha'\boldsymbol{\rangle}^1$, $P_1$ and $P_2$ hold $[\![\alpha]\!]^0$, $P_0$ and $P_2$ hold $[\![\alpha']\!]^1$.
Input : $\langle\cdot\rangle$-shared value of $x$.
Output : $\langle\cdot\rangle$-shared value of $\mathsf{sign}(x)$.

**Offline:**

- $P_1$ and $P_2$ generate the same random value $\Delta \in \{0,1\}$ via PRF with seed $\eta_{1,2}$ and all parties set $\langle\Delta\rangle^0 := (\Delta,0,0)$.
- $P_0$ and $P_2$ generate the same random value $\Delta' \in \{0,1\}$ via PRF with seed $\eta_{0,2}$ and all parties set $\langle\Delta'\rangle^1 := (0,\Delta',0)$.
- $P_0$ and $P_j$ for $j \in \{1,2\}$ pick random bit list $\{d_{j,i}\}_{i\in\mathbb{Z}_\ell} \leftarrow \mathbb{Z}_2^\ell$ via PRF with seed $\eta_{0,1}$ and $\eta_{0,2}$;
- $P_1$ and $P_j$ for $j \in \{0,2\}$ pick random bit list $\{c_{j,i}\}_{i\in\mathbb{Z}_\ell} \leftarrow \mathbb{Z}_2^\ell$ via PRF with seed $\eta_{0,1}$ and $\eta_{1,2}$;
- For $i \in \mathbb{Z}_\ell$, all parties set:
  1) $\langle d_{1,i}\rangle^0 = (0,0,d_{1,i})$, $\langle d_{2,i}\rangle^0 = (0,d_{2,i},0)$, $\langle c_{0,i}\rangle^1 = (0,0,c_{0,i})$, $\langle c_{2,i}\rangle^1 = (c_{2,i},0,0)$;
  2) $\boldsymbol{\langle} d_{1,i}\boldsymbol{\rangle}^0 = (0,0,d_{1,i})$, $\boldsymbol{\langle} d_{2,i}\boldsymbol{\rangle}^0 = (0,d_{2,i},0)$, $\boldsymbol{\langle} c_{0,i}\boldsymbol{\rangle}^1 = (0,0,c_{0,i})$, $\boldsymbol{\langle} c_{2,i}\boldsymbol{\rangle}^1 = (c_{2,i},0,0)$;
- All parties invoke $\mathcal{F}_{\mathsf{Mult}}[2^\ell]$ and $\mathcal{F}_{\mathsf{Mult}}[p]$ to calculate
  1) $\boldsymbol{\langle}\hat r_{x,i}\boldsymbol{\rangle}^0 = \boldsymbol{\langle} d_{1,i}\boldsymbol{\rangle}^0 + \boldsymbol{\langle} d_{2,i}\boldsymbol{\rangle}^0 - 2\boldsymbol{\langle} d_{1,i}\boldsymbol{\rangle}^0 \cdot \boldsymbol{\langle} d_{2,i}\boldsymbol{\rangle}^0$ and $\boldsymbol{\langle}\hat r'_{x,i}\boldsymbol{\rangle}^0 = \boldsymbol{\langle} c_{0,i}\boldsymbol{\rangle}^1 + \boldsymbol{\langle} c_{2,i}\boldsymbol{\rangle}^1 - 2\boldsymbol{\langle} c_{0,i}\boldsymbol{\rangle}^1 \cdot \boldsymbol{\langle} c_{2,i}\boldsymbol{\rangle}^1$, for $i \in \mathbb{Z}_{\ell-1}$;
  2) $\boldsymbol{\langle}\gamma(\hat r_{x,i})_i\boldsymbol{\rangle}^0 = \boldsymbol{\langle}\hat r_{x,i}\boldsymbol{\rangle}^0 \cdot \boldsymbol{\langle}\alpha_i\boldsymbol{\rangle}^0$ and $\boldsymbol{\langle}\gamma(\hat r'_{x,i})_i\boldsymbol{\rangle}^1 = \boldsymbol{\langle}\hat r'_{x,i}\boldsymbol{\rangle}^1 \cdot \boldsymbol{\langle}\alpha'_i\boldsymbol{\rangle}^1$, for $i \in \mathbb{Z}_{\ell-1}$ and $i \in \mathbb{Z}_{\lambda-1}$;
  3) $\langle r_x\rangle^0 = 2^{\ell-1} - \sum_{i=0}^{\ell-2} 2^i(\langle d_{1,i}\rangle^0 + \langle d_{2,i}\rangle^0 - 2\langle d_{1,i}\rangle^0 \cdot \langle d_{2,i}\rangle^0) + 2^{\ell-1}(\langle d_{1,i}\rangle^0 + \langle d_{2,i}\rangle^0 - 2\langle d_{1,i}\rangle^0 \cdot \langle d_{2,i}\rangle^0)$;
  4) $\langle r'_x\rangle^1 = 2^{\ell-1} - \sum_{i=0}^{\ell-2} 2^i(\langle c_{0,i}\rangle^1 + \langle c_{2,i}\rangle^1 - 2\langle c_{0,i}\rangle^1 \cdot \langle c_{2,i}\rangle^1) + 2^{\ell-1}(\langle c_{1,i}\rangle^1 + \langle c_{2,i}\rangle^1 - 2\langle c_{0,i}\rangle^1 \cdot \langle c_{2,i}\rangle^1)$;
- All parties set $[r_x]^0 := ([r_{r_x}]_1^0 + m_{r_x}, [r_{r_x}]_2^0)$, $[\![r_{x,i}]\!]^0 := ([\![r_{r_{x,i}}]\!]_1^0 + m_{r_{x,i}}, [\![r_{r_{x,i}}]\!]_2^0)$ and $[\![\gamma(r_{x,i})_l]\!]^0 := ([\![r_{\gamma(r_{x,i})_l}]\!]_1^0 + m_{\gamma(r_{x,i})_l}, [\![r_{\gamma(r_{x,i})_l}]\!]_2^0)$ for $i \in \mathbb{Z}_{\ell-1}, l \in \mathbb{Z}_{\lambda-1}$, similar to $[r'_x]^1$, $[\![r'_{x,i}]\!]^1$ and $[\![\gamma(r'_{x,i})_i]\!]^1$;
- All parties set $\|r_{x,i}\|^{\lambda,0} := ([\![r_{x,i}]\!]^0, \{[\![\alpha_l]\!]^0, [\![\gamma(r_{x,i})_l]\!]^0\}_{l\in\mathbb{Z}_\lambda})$ and $\|r_{x,i}\|^{\lambda,1} := ([\![r_{x,i}]\!]^1, \{[\![\alpha_l]\!]^1, [\![\gamma(r_{x,i})_l]\!]^1\}_{l\in\mathbb{Z}_\lambda})$;

**Online:**

- $P_1$ and $P_2$ both calculate $\delta := m_x - [r'_x]_2^1$ and send it to $P_0$, $P_0$ and $P_1$ both calculate $\delta' := [r_x]_1^0 - [r'_x]_0^1$ and send it to $P_2$;
- $P_0$ sets $m'_x = [r_x]_1^0 + [r_x]_2^0 - [r'_x]_0^1 + \delta$ if received $\delta$ from $P_1$ and $P_2$ are consistant, else abort;
- $P_2$ sets $m'_x = [r_x]_2^0 + m_x - [r'_x]_2^1 + \delta'$ if received $\delta'$ from $P_0$ and $P_1$ are consistant, else abort;
- $P_j$, for $j \in \{1,2\}$ does:
  1) set $\hat m_x = m_x - \mathsf{sign}(m_x)\cdot 2^{\ell-1}$ and bitexact it as $\{\hat m_{x,i} \in \{0,1\}\}_{i\in\mathbb{Z}_{\ell-1}}$ while $\sum_{i=0}^{\ell-2} 2^{\ell-2-i}\hat m_{x,i} = \hat m_x$;
  2) set $\hat m_{x,\ell-1} = 1$ and $\|r_{x,\ell-1}\| = \|0\|^0$;
  3) set $\|m_i\|^0 = \hat m_{x,i} + \|r_{x,i}\|^0 - 2\hat m_{x,i}\cdot\|r_{x,i}\|^0$ for $i \in \mathbb{Z}_\ell$.
  4) pick same random values $\{w_i\}_{i\in\mathbb{Z}_\ell} \in (\mathbb{Z}_p^*)^{2\ell}$ via PRF with seed $\eta_{1,2}$;
  5) calculate $\|m'_i\|^0 = \sum_{t=1}^{i} \|m_t\|^0 - 2\cdot\|m_i\|^0 + 1$ and $\|u_i\|^0 = w_i \cdot \|m'_i\|^0 \cdot (1 \oplus \mathsf{sign}(m_x) \oplus \hat m_{x,i} \oplus \Delta) + w_i \cdot (\mathsf{sign}(m_x) \oplus \hat m_{x,i} \oplus \Delta)$ for $i \in \mathbb{Z}_\ell$;
  6) pick a random permutation $\pi$ via PRF with seed $\eta_{1,2}$ and permute the list $\{\|\hat u_i\|^0\}_{i\in\mathbb{Z}_\ell} = \pi(\{\|u_i\|^0\}_{i\in\mathbb{Z}_\ell})$;
  7) invoke $\Pi_{\mathsf{BatchRec}}(\|\hat u_i\|^0)$ to reconstruct list $\{\hat u_i\}_{i\in\mathbb{Z}_\ell}$ and $P_0$ holds $\hat u_i$;
- Similiar to above steps, $P_0$ and $P_2$ calculate $\|\hat u'_i\|^1$ and send to $P_1$ for reconstruction.
- $P_0$ sets $t = \mathsf{sign}(r_x)$ if $\exists \hat u_i = 0$ for $i \in \mathbb{Z}_\ell$ else $t = \mathsf{sign}(r_x) \oplus 1$; all parties invoke $[t]^0 \leftarrow \Pi_{[\cdot]}^0(t)$ and set $\langle t\rangle^0 := (0, [t]_2^0, [t]_1^0)$.
- $P_1$ sets $t' = \mathsf{sign}(r'_x)$ if $\exists \hat{u'_i} = 0$ for $i \in \mathbb{Z}_\ell$ else $t' = \mathsf{sign}(r'_x) \oplus 1$; all parties invoke $[t']^1 \leftarrow \Pi_{[\cdot]}^1(t')$, set $\langle t'\rangle^1 := ([t']_2^1, 0, [t']_0^1)$.
- All parties invoke $\mathcal{F}_{\mathsf{Mult}}$ to calculate $\langle z\rangle = \langle t\rangle + \langle\Delta\rangle - 2\langle t\rangle\cdot\langle\Delta\rangle$ and $\langle z'\rangle = \langle t'\rangle + \langle\Delta'\rangle - 2\langle t'\rangle\cdot\langle\Delta'\rangle$.
- All parties call $\langle r\rangle \leftarrow \Pi_{\langle\cdot\rangle}$ and invoke $\mathcal{F}_{\mathsf{Mult}}$ to calculate $\langle c\rangle = (\langle z\rangle - \langle z'\rangle)\cdot(2\cdot\langle r\rangle + 1)$;
- All parties reveal $c \leftarrow \Pi_{\mathsf{Rec}}(\langle c\rangle)$, abort if $c \neq 0$.
- All parties invoke $\mathcal{F}_{\mathsf{Reshare}}$ re-randomize $\langle z\rangle$.

Fig. 11: Actively secure sign-bit extraction protocol.

$\mathbb{Z}_p$, the result is a valid secret share in $\mathbb{Z}_p$. In particular, $P_0$ and $P_1$ jointly generate a random bit string $\{d_{1,i}\}_{i\in\mathbb{Z}_\ell}$ using seed $\eta_{0,1}$, $P_0$ and $P_2$ jointly generate $\{d_{2,i}\}_{j\in\mathbb{Z}_\ell}$ with seed $\eta_{0,2}$. By setting $\langle d_{1,i}\rangle := (0,0,d_{1,i})$, $\boldsymbol{\langle} d_{1,i}\boldsymbol{\rangle} := (0,0,d_{1,i})$, $\langle d_{2,i}\rangle := (0,d_{2,i},0)$ and $\boldsymbol{\langle} d_{2,i}\boldsymbol{\rangle} := (0,d_{2,i},0)$, we obtain shares of $d_{1,i}$ and $d_{2,i}$ in both $\mathbb{Z}_{2^\ell}$ and $\mathbb{Z}_p$. For $r_{x,i} = d_{1,i} \oplus d_{2,i}$, we use maliciously secure multiplication $\mathcal{F}_{\mathsf{Mult}}$, leveraging the identity $r_{x,i} = d_{1,i} + d_{2,i} - 2\cdot d_{1,i}\cdot d_{2,i}$. After that, both $P_1$ and $P_0$ add $m_x$ to $[r_x]_1$, converting $\langle r_x\rangle$ and $\boldsymbol{\langle} r_{x,i}\boldsymbol{\rangle}$ into 2PC share $[r_x]$ and $[\![r_{x,i}]\!]$.

**IT-MAC and Batch MAC Verification.** We use IT-MAC to ensure the correctness of $\hat u_i$'s calculation, while $P_0$ holds the MAC key, to verify $\hat u_i$. In our actively secure protocol, we use $\|r_{x,i}\|$ instead of $[r_{x,i}]$ to compute $\hat u_i$. After reconstructing $\|\hat u_i\|$, $P_0$ can use the MAC key it holds to verify the correctness. As previously discussed, we need to ensure the correct generation of the MAC. To achieve this, we use aforementioned actively secure additively share generation method to produce the MAC. During the setup phase, we generate MAC keys $[\alpha]^0$ and $\langle\alpha\rangle^0$ simultaneously. In the offline phase, we use $\mathcal{F}_{\mathsf{Mult}}$ to compute the MAC $\langle\gamma(r_{x,j})\rangle$ on $\langle\alpha\rangle$ and $\langle r_{x,j}\rangle$. Afterward, through local computation, we securely convert $\langle\gamma(r_{x,j})\rangle$ into the additive secret share $[\gamma(r_{x,j})]$.

In addition, we employ the batch MAC verification to reduce the communication of reconstruction. The principle is that all

**Functionality $\mathcal{F}_{\mathsf{VSignBit}}[\mathbb{Z}_{2^\ell}]$**

$\mathcal{F}_{\mathsf{VSignBit}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.

**Input:**

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, (r_1, r_2))$ from $P_0$, send $(\mathsf{Input}, \mathsf{sid}, P_0)$ to $\mathcal{S}$ and record $(r_1, r_2) \in (\mathbb{Z}_{2^\ell})^2$;
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, (m_j, r_j))$ from $P_j$, $j \in \mathbb{Z}_2$, send $(\mathsf{Input}, \mathsf{sid}, P_j)$ to $\mathcal{S}$ and record $(m_j, r_j) \in (\mathbb{Z}_{2^\ell})^2$;

**Execution:**

- Compute $z := \mathsf{sign}(m_1 - r_1 - r_2)$;
- Pick random $u_1, u_2 \leftarrow \mathbb{Z}_{2^\ell}$, set $u := u_1 + u_2$ and $w := z + u$;
- Send $(\mathsf{Output}, \mathsf{sid}, (u_1, u_2))$ to $P_0$, $(\mathsf{Output}, \mathsf{sid}, (w, u_1))$ to $P_1$, $(\mathsf{Output}, \mathsf{sid}, (w, u_2))$ to $P_2$ via *private delayed channel.*

Fig. 12: The ideal functionality $\mathcal{F}_{\mathsf{VSignBit}}$.

TABLE III: Boolean shares after dual execution.

| | P_0 | P_1 | P_2 |
|---|---|---|---|
| First Execution | $t = z \oplus \Delta$ | $\Delta$ | $\Delta$ |
| Second Execution | $\Delta'$ | $t' = z \oplus \Delta'$ | $\Delta'$ |

MAC verifications can be combined into a single message for one-time verification. In such a scenario, if the data volume is sufficiently large, the communication overhead during the MAC verification phase becomes negligible when amortized. For $N$ pairs of $\|\cdot\|$-shared value $\|x^{(0)}\|, \ldots, \|x^{(N-1)}\|$, $P_1$ and $P_2$ partially open secret value $x^{(i)}$ (without the MACs) to $P_0$. We let $P_0$ generate a public $\lambda$-dimension random list $\{w_k \in \mathbb{Z}_p\}_{k \in \mathbb{Z}_\lambda}$ and send the list to $P_1$ and $P_2$. With the random list, the $N$ pairs of MACs can be combined to $\lambda$ pairs, that is, $\|t_k\| = \sum_{i=0}^{N-1} (w_k)^i \cdot \|x^{(i)}\|$ for $k \in \mathbb{Z}_\lambda$. Instead of verifying $n$ pairs of share, $P_0$ only needs to verify $\alpha \cdot t_k = \gamma(t_k)$ for $k \in \mathbb{Z}_\lambda$, where $n \gg \lambda$. Note that, the batch MAC verification requires an additional round for the MAC opening.

**Dual Execution.** As previously discussed, we employ dual execution to detect malicious behavior from $P_0$. A naive approach is to directly treat $\langle x \rangle^0$ as $\langle x \rangle^1$ and rerun the protocol. However, since $\langle x \rangle^1$ considers the original $m_x$ as $[r_x]_2$, and it must be generated during the online phase, this shifts many operations that were previously done offline into the online phase, incurring substantial online overhead. Our solution is to reshare $\langle x \rangle^0$ into $\langle x \rangle^1$ during the online phase. Specifically, we generate $[r_x]^0$ and $[r'_x]^1$ in the offline phase. Then, during the online phase, we compute $m'_x$ for $\langle x \rangle^1$ from $m_x$, $r_x$, and $r'_x$, with the relationship $m'_x + r'_x = m_x + r_x$. In the online phase, $P_0$ and $P_2$ need to compute $m'_x = m_x + [r_x]_1^0 + [r_x]_2^0 - [r'_x]_0^1 - [r'_x]_2^1$. Since $P_0$ already holds $[r'_x]_0^1$, $[r_x]_1^0$, and $[r_x]_2^0$, we only need to send $\delta = m_x - [r'_x]_2^1$ to $P_0$. Specifically, we let $P_1$ calculate and transmits $\delta$, and $P_2$ sends the corresponding hash $H(\delta)$. $P_0$ then verify correctness and compute the correct $m'_x$. Similarly, $P_2$ already has $m_x$, $[r_x]_2^0$, and $[r'_x]_2^1$. The other parties need only send $[r_x]_1^0 - [r'_x]_0^1$ to $P_2$. The resharing introduces only a single round of $2\ell$ communication cost.

By executing the protocol on $\langle x \rangle^0$ and $\langle x \rangle^1$ separately, and after completing the checks on the list $\hat{u}_i$ , we obtain the secret-sharing states shown in Table III ($t = z \oplus \Delta$, $\Delta$, and $t' = z \oplus \Delta'$, $\Delta'$). Next, to securely convert these two binary secret shares into $\langle \cdot \rangle$-sharing, we evaluate $z = t \oplus \Delta$, $z' = t' \oplus \Delta'$, and $\beta(z - z') = 0$ under the $\langle \cdot \rangle$-sharing scheme with $\mathcal{F}_{\mathsf{Mult}}$. Note that by treating both $\langle x \rangle^0$ and $\langle x \rangle^1$ as replicated secret shares, multiplication can be performed directly using $\mathcal{F}_{\mathsf{Mult}}$.

TABLE IV: Maximum Pairwise Communication of Comparison Protocols under 64-bit with $2^{16}$ number of elements. We evaluated the overhead of Bicoptor at an error rate of $2^{-64}$.

| | Round | Communication |
|---|---|---|
| DCF [19] | 2 | 139.26MB |
| Bicoptor [46] | 2 | 134.218MB |
| Falcon [41] | 11 | 5.24MB |
| Ours | 3 | 17.39MB |

**Security.** Fig. 12 depicts the functionality of actively secure sign-bit extraction. In this functionality, we allow the adversary to abort and terminate execution through private delayed channel.

**Theorem 3.** *The protocol $\Pi_{\mathsf{VSignBit}}$ as depicted in Fig. 11 UC-realizes $\mathcal{F}_{\mathsf{VSignBit}}$ in the ($\mathcal{F}_{\mathsf{Mult}}$. $\mathcal{F}_{\mathsf{Reshare}}$)-hybrid model against malicious PPT adversaries who can statically corrupt up to one party.*

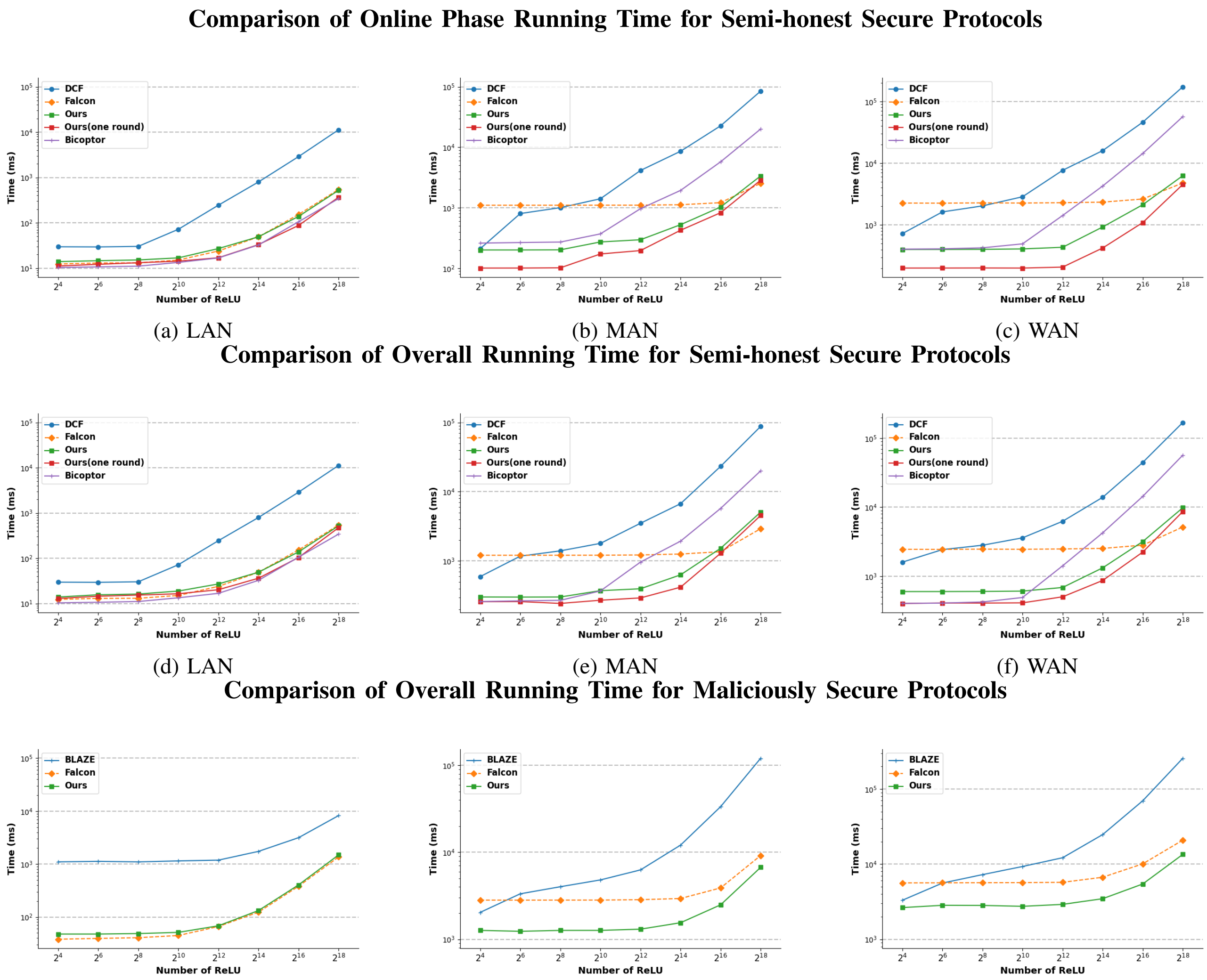
*Proof.* See Appendix B. □

**Efficiency.** Considering the amortized overhead, the MAC verification for $\Pi_{\mathsf{BatchRec}}(\|\hat{u}_i\|)$ in the online phase can be consolidated and carried out collectively during the verification phase instead of after each operation. Similarly, the zero check for $\Pi_{\mathsf{Rec}}(\langle c \rangle)$ can be merged into a single ciphertext in the verification phase (cf. Appendix A, $\Pi_{\mathsf{ZeroCheck}}$). Under amortization, our actively secure protocol requires just three rounds of $4\ell \log \ell + 10\ell$-bit communication in the online phase and $10\ell + 6\ell(\lambda+1) \log \ell$-bit communication in the offline phase (take $\lambda = 7$ for soundness error $2^{-49}$).

## V. Implementation and Benchmarks

In this section, we evaluate our protocols in both the semi-honest and malicious settings. For the semi-honest version, we compare the communication and running time of our protocol with the DCF solution [7], Falcon [41] and Bicoptor [46]. For the malicious version, we compare our protocol with Falcon[41], Edabit[17] and BLAZE [35].

**Benchmark Setting.** As a baseline, we used the open-source FSS library [3] to evaluate DCF [7] and re-implemented Bicoptor [46]. For the maliciously secure protocol BLAZE [35], we re-implemented it based on the garbled circuits from emp-toolkit [2] and incorporated support for the half-gate optimization [45]. We directly used the code provided by Falcon [4] to benchmark both its semi-honest and malicious versions. All the benchmark code [1] can be found on the anonymous GitHub repository. In our benchmark setting, we take the size of the ring $\ell = 64$. Our experiments are performed in a local area network, using software to simulate three network settings: local-area network (LAN, RTT: 1ms,




(a) LAN (b) MAN (c) WAN

(d) LAN (e) MAN (f) WAN

(g) LAN (h) MAN (i) WAN

Fig. 13: Run-time of ReLU in LAN/MAN/WAN setting. Here, “Ours” refers to our protocols; DCF refers to [19]; Falcon refers to [41]; Bicoptor refers to [46]; For the malicious setting, we take $\lambda = 7$ for our protocol with soundness error $2^{-49}$

bandwidth: 10Gbps), metropolitan-area network (MAN, RTT: 100ms, bandwidth: 1000Mbps), and wide-area network (WAN, RTT: 200ms, bandwidth: 100Mbps) and executed on a desktop with Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz running Ubuntu 18.04.2 LTS; with 48 CPUs, 128 GB Memory.

**Comparison of Semi-honest Secure Protocols.** Table. IV shows the overall communication cost with $2^\ell$ number of elements generated by different protocols during actual execution. As expected, our protocol requires 8 fewer rounds of communication compared to logarithmic-round protocols like Falcon, and the communication volume is 87% lower than that of constant-round protocols such as DCF and Bicoptor. Subfigures (a)-(c) of Fig. 13 show the performance of different protocols in the online phase under the semi-honest model. We evaluated our protocols under two settings: the two-round protocol that outputs $\langle\cdot\rangle$ secret shares and a one-round protocol that outputs Boolean secret shares. Under LAN settings, where computation dominates runtime, our protocol, Bicoptor, and Falcon exhibit similar performance. Bicoptor achieves slightly better results due to its marginal computational advantage. In MAN and WAN settings, where communication cost becomes significant, our protocol outperforms others in the online phase. Notably, among constant-round protocols, our approach is 6× faster than Bicoptor and 24× faster than DCF when the input size exceeds $2^{12}$. For our one-round version, it achieves 13× speedup over Bicoptor at an input size of $2^{14}$ under WAN. Compared to the logarithmic-round protocol Falcon [41], our protocol achieves an order-of-magnitude speedup for input sizes below $2^{12}$ in both MAN and WAN settings. However, when the input size is sufficiently large (e.g., $2^{18}$), the cost becomes communication-bound, and logarithmic-round protocols like Falcon can offer some advantage.

Subfigures (d)-(f) of Fig. 13 illustrate the overall runtime. Similar to the online phase, under LAN settings, our protocol, Bicoptor, and Falcon exhibit similar performance, while DCF

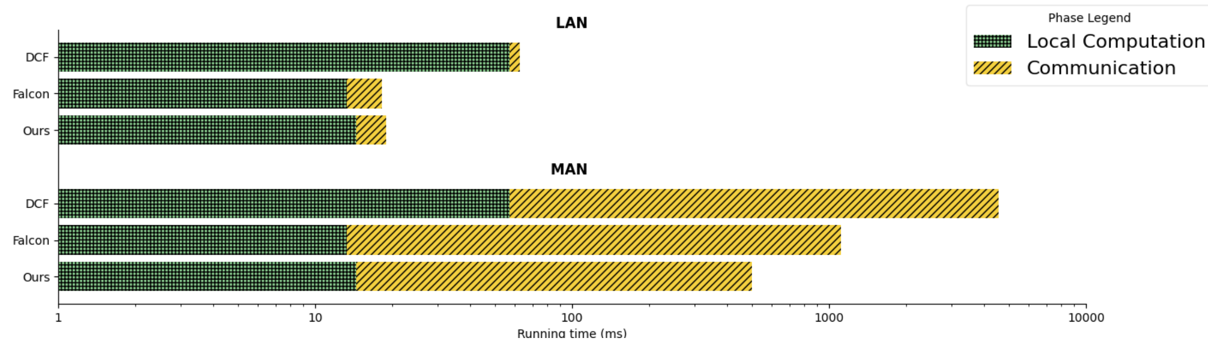



Fig. 14: Execution breakdown of comparison protocols run under LAN and MAN settings( Taking $2^{12}$ size input). The timeline on the X-axis represents the running time for each local computation or communication.

is significantly slower due to its high computational cost. Since our two-round protocol require an additional offline round compared to Bicoptor, it is slightly slower than Bicoptor in MAN and WAN settings when the input size is small. However, our one-round protocol has the same communication rounds as Bicoptor, resulting in similar runtime for small inputs. When the input size exceeds $2^{14}$, our two-round version outperforms Bicoptor by $5\times$ and DCF by $20\times$. Similarly, compared to typical logarithmic-round protocols Falcon, our protocol shows a clear advantage for smaller inputs, achieving up to $5\times$ better performance than Falcon when the input size is less than $2^{16}$.

Fig. 14 compares the computation and communication costs of DCF, Falcon, and our protocol across various network environments and It visually demonstrates how our protocol's performance changes under varying network conditions. In the LAN setting, our protocol performs slightly worse than the Bicoptor and Falcon protocols. As the network quality worsens, communication overhead becomes the primary bottleneck, significantly affecting the overall protocol performance. In contrast, DCF suffers from both extremely high computational and communication overheads, making it considerably slower than the other protocols in any network scenario. Our protocol incurs higher local computation costs compared to Falcon and Bicoptor, which may be due to our specific code implementation. Adopting more efficient computational schemes could potentially improve the performance of our protocol.

**Comparison of Maliciously Secure Protocols.** Subfigures (g)-(i) of Fig. 13 present the performance comparison of maliciously secure protocols $\Pi_{\mathsf{VSignBit}}$, Falcon and BLAZE [35] under LAN, MAN and WAN setting. In the LAN setting, our protocol is slightly slower than Falcon due to its slightly higher computational overhead. In contrast, BLAZE incurs significantly higher overhead than both Falcon and our protocol, as it requires extensive garbled circuit evaluations. In the MAN and WAN settings, our protocol significantly outperforms both Falcon and BLAZE. For small-scale datasets, our protocol achieves up to $3\times$ the performance of Falcon. Compared to BLAZE, which is also a constant-round protocol, our protocol is at least an order of magnitude faster, regardless of whether the setting is MAN or WAN. Although the performance advantage of our protocol over Falcon decreases as the input size grows, it still achieves nearly $2\times$ the performance of Falcon at a scale of $2^{18}$. Given that typical use cases for non-linear protocols—such as activation functions like ReLU— $2^{18}$ is already more than sufficient.

## VI. Related work

To evaluate non-linear functions such as ReLU and Maxpool, protocols like [33], [26], [34] employ the A2B paradigm, which is a conversion process that transforms arithmetic secret sharing into boolean secret sharing. Subsequently, they utilize this boolean secret-sharing scheme to evaluate corresponding non-linear functions. Typically, this approach need to introduce $\log \ell$ rounds of communication. Escudero [17] et al. applied a paradigm similar to A2B in the dishonest majority setting. Furthermore, in protocols such as [33], [35], [12], garbled circuits are employed for evaluating non-linear functions. The use of garbled circuits introduces a significant amount of additional communication overhead, particularly in the presence of a malicious threat model. In contrast, the protocols described in [40], [27] tackle the sign-bit extraction problem with a constant round communication overhead, while they require a substantial communication overhead of 10 rounds, which can be even larger than $\log \ell$ rounds when $\ell$ is small. In addition, Function Secret Sharing [7] provides another constant-round approach by encoding the data into correlated randomness during the offline phase, allowing the computing parties to evaluate comparisons using the correlated randomness in the online phase. On the other hand, Bicoptor [46] implements comparison through a truncation protocol. Their approach performs local truncation $\ell$ times, followed by involving a third party to verify if the result contains zero items. This scheme realizes two rounds with $\ell^2$ bits of communication. However, this approach has not been applied to malicious threat models.

## VII. Conclusion

In this work, we innovate novel semi-honest and maliciously secure sign-bit extraction protocols. The benchmark results show that our protocols have significant performance improvements over the state-of-the-art works, i.e., Bicoptot [46], Falcon [41], FSS [7].

## References

[1] Our code. https://anonymous.4open.science/r/oram_pro-91CB.

[2] emp-toolkit, 2019. https://github.com/emp-toolkit/emp-tool.git.

[3] function secret share source code, 2019. https://github.com/myl7/fss.git.

[4] Our code, 2019. https://github.com/snwagh/falcon-public.git.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[6] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.

[7] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, 2021.

[8] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. Cryptology ePrint Archive, Paper 2019/1390, 2019.

[9] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 869–886, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: Fast and robust framework for privacy-preserving machine learning. In *PoPETs*, 2020.

[11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[12] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. Astra: High throughput 3pc over rings with application to secure prediction. In *CCSW*, 2019.

[13] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *NDSS*, 2020.

[14] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-Majority Four-Party secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[15] Anders Dalskov, Daniel E. Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. *IACR Cryptol. ePrint Arch.*, 2020.

[16] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[17] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 823–852, Cham, 2020. Springer International Publishing.

[18] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[19] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in cot and dpf. In *EUROCRYPT*, 2023.

[20] Carmit Hazay, Abhi Shelat, and Muthuramakrishnan Venkitasubramaniam. Going beyond dual execution: Mpc for functions with efficient verification. In *PKC*, 2020.

[21] Eerikson Hendrik, Keller Marcel, Orlandi Claudio, Pullonen Pille, Puura Joonas, and Simkin Mark. Use your brain! arithmetic 3pc for any modulus with active security. In *ITC*, 2020.

[22] Wilko Henecka, Stefan K ögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: Tool for automating secure two-party computations. In *CCS*, 2010.

[23] Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *S&P*, 2012.

[24] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *USENIX Security 2022*, pages 809–826, 2022.

[25] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based secure training and inference with GPUs. In *S&P*, 2024.

[26] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. Swift: Super-fast and robust privacy-preserving machine learning. In *USENIX*, 2021.

[27] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *S&P*, 2020.

[28] Minghui Li, Sherman S. M. Chow, Shengshan Hu, Yuejing Yan, Chao Shen, and Qian Wang. Optimizing privacy-preserving outsourced convolutional neural network predictions. *IEEE Trans. Dependable Secur. Comput.*, 19(3):1592–1604, 2022.

[29] Yun Li, Daniel Escudero, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Sublinear distributed product checks on replicated secret-shared data over z2k without ring extensions. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 825–839, New York, NY, USA, 2024. Association for Computing Machinery.

[30] Tianpei Lu, Bingsheng Zhang, Lichun Li, and Kui Ren. The communication-friendly privacy-preserving machine learning against malicious adversaries, 2024.

[31] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. Epic: Efficient private image classification (or: Learning from the masters). In *CT-RSA*, 2019.

[32] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security 2020*, pages 2505–2522, 2020.

[33] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *CCS*, 2018.

[34] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. Aby2. 0: Improved mixed-protocol secure two-party computation. In *USENIX*, 2021.

[35] Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. In *NDSS*, 2020.

[36] Mohassel Payman and Zhang Yupeng. Secureml: A system for scalable privacy-preserving machine learning. In *S&P*, 2017.

[37] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, 2020.

[38] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS*, 2018.

[39] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO*, 2021.

[40] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. In *PoPETs*, 2019.

[41] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. In *PoPETs*, 2021.

[42] Andrew C. Yao. Protocols for secure computations. In *SFCS*, 1982.

[43] Lindell Yehuda and Nof Ariel. A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In *CCS*, 2017.

[44] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *EUROCRYPT*, 2015.

[45] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 220–250, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[46] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In *S&P*, 2023.


## Appendix

*Batch Zero Checking Protocol.* The values of $N$ secret shares $(\langle c_0\rangle, \dots, \langle c_{N-1}\rangle)$ can be checked if all equal zero using a single ciphertext. For simplicity, we represent $\langle c_i\rangle := (c_{i,0}, c_{i,1}, c_{i,2})$ where $c_{i,0} := m_{c_i}$, $c_{i,1} := [r_{c_i}]_2$ and $c_{i,2} := [r_{c_i}]_1$. $\mathcal{G}$ is a PRG and $\mathcal{K}$ is common key all parties agreed. For each share $\langle c_i\rangle := (c_{i,0}, c_{i,1}, c_{i,2})$, let $P_k$ compute $c'_{i,k} = \mathcal{G}(\mathcal{K}, c_{i,k-1} + c_{i,k+1})$, and $P_{k-1}$ and $P_{k+1}$ compute $c'_{i,k} = \mathcal{G}(\mathcal{K}, -c_{i,k})$. Then, sum the $N$ new messages as $c'_k = \sum c'_{i,k}$. By opening $c'_0, c'_1$, and $c'_2$ to each other and checking for consistency, it is possible to securely determine whether all $c_i$ are zero, even if one party behaves maliciously. For simplicity, consider the security against $P_0$ being malicious. $P_1$ and $P_2$

**Protocol $\Pi_{\mathsf{ZeroCheck}}(\langle c_0\rangle, \dots, \langle c_{N-1}\rangle)$**

For simplicity, we represent $\langle c_i\rangle := (c_{i,0}, c_{i,1}, c_{i,2})$ where $c_{i,0} := m_{c_i}$, $c_{i,1} := [r_{c_i}]_2$ and $c_{i,2} := [r_{c_i}]_1$. $\mathcal{G}$ is a PRG.
Input : $\langle\cdot\rangle$-shared value of $x$
Output : 1 if $c_0, \dots, c_{N-1}$ all equal to zero, otherwise, 0
**Execution:**

- All parties agree a common key $\mathcal{K}$
- Each party $P_k$ sets $c'_k = \sum_{i=0}^{N} \mathcal{G}(\mathcal{K}, c_{i,k-1} + c_{i,k+1})$; $P_{k-1}$ and $P_{k+1}$ set $c'_k = \sum_{i=0}^{N} \mathcal{G}(\mathcal{K}, -c_{i,k})$
- Each party send $c'_0$, $c'_1$, $c'_2$ to each others; If any message is inconsistent, output 0, else output 1.

Fig. 15: The Zero Check Protocol $\Pi_{\mathsf{ZeroCheck}}$.

can verify the consistency of $c'_1$ and $c'_2$ based on their own values, without relying on $P_0$'s input. It is worth noting that directly checking for zero by computing a linear combination over the secret shares $\langle c_i\rangle$, such as $c = \sum \beta_i c_i$, and then check $c = 0$, is insecure. This is because the ring structure may cause certain errors such as $2^{\ell-1}$ to cancel out during the linear combination, leading to incorrect results pass the verfication.

*Secure ReLU Protocol.* The ReLU of $x$ is calculated by $w = x\cdot(1-\mathsf{sign}(x)) = x - x\cdot\mathsf{sign}(x)$, which can be implemented by combining $\Pi_{\mathsf{Mult}}$ with $\Pi_{\mathsf{SignBit}}$. However, it requires an additional round for multiplication. We observe that the additional round can be eliminated by executing multiplication at the same round of sending back $m'$ in $\Pi_{\mathsf{SignBit}}$. We construct the semi-honest ReLU protocol $\Pi_{\mathsf{ReLU}}$ ( Fig. 16) from $\Pi_{\mathsf{SignBit}}$. Considering $\langle z\rangle = \Pi_{\mathsf{SignBit}}(\langle x\rangle)$ and $\langle w\rangle = \Pi_{\mathsf{Mult}}(\langle x\rangle \cdot \langle z\rangle)$, we have:

$$\begin{aligned} m_w &= m_x m_z + m_x r_z + m_z r_x + r_x r_z - r_w \\ &= m_x m_z + m_x r_z + (m' - 2\Delta m' + \Gamma) r_x + r_x r_z - r_w \\ &= m_x m_z + m_x r_z + (1 - 2\Delta)(m' r_x + r'') + \Gamma' \end{aligned}$$

$m'$, $\Delta$, $\Gamma$ are the fresh random values mentioned in $\Pi_{\mathsf{SignBit}}$ and it hold $m_z = m' - 2\Delta m' + \Gamma$ in $\Pi_{\mathsf{SignBit}}$. We denote $\Gamma' = \Gamma \cdot r_x - (1 - 2\Delta) r'' + r_x \cdot r_z - r_w$, where $r''$ is a fresh random introduced to protect the privacy of $r_w$. We let $P_1$ and $P_2$ calculate $[\Gamma'] = \Gamma\cdot[r_x] - (1-2\Delta)[r''] + [r_x \cdot r_z] - [r_w]$ locally in the offline phase. $P_1$ and $P_2$ reveal $[\Gamma''] = m_x \cdot [r_z] + [\Gamma']$ to each other in the first round of $\Pi_{\mathsf{SignBit}}$. For item $(1 - 2\Delta)(m' r_x + r'')$, $P_0$ send $m'' = m' r_x + r''$ to $P_1$ and $P_2$. Then $P_1, P_2$ locally calculate $m_w = m_x \cdot m_z + \Gamma'' + (1 - 2\Delta) m''$. Note that reveal $m''$ and $\Gamma''$ will not leak any information, since the $P_1$ and $P_2$ cannot extract additional information of $r_x$, $r_z$, $r_w$ besides of $m_w$, with the fresh random value $r''$. Our ReLU protocol requires 1 rounds and communication of $(\ell - 1)\log\ell + 2\ell$ bits in the preprocessing phase and requires 2 rounds and communication of $4\ell\log\ell + 4\ell$ bits in the online phase. The malicious version of ReLU can be achieved through verifying $\langle z\rangle = \mathsf{sign}(\langle x\rangle)$ and $\langle w\rangle = \Pi_{\mathsf{Mult}}(\langle x\rangle, \langle z\rangle)$ respectively.

*Secure Maxpool protocol.* Our Maxpool scheme is constructed by comparison $\mathsf{great}(x, y) = x \overset{?}{\geq} y$ and maximum $\mathsf{max}(x_1, \dots, x_n)$. In the case of signed numbers $x$ and $y$, $\mathsf{great}(x, y)$ can be implemented by invoking the $\Pi_{\mathsf{VSignBit}}$ three times. That is, $\mathsf{great}(x, y) = (\mathsf{sign}(x) \oplus \mathsf{sign}(y)) \cdot \mathsf{sign}(y - x) + (1 \oplus \mathsf{sign}(x) \oplus \mathsf{sign}(y)) \cdot \mathsf{sign}(y)$. For unsigned number $x$ and $y$ which $\mathsf{sign}(x) = 0$ and $\mathsf{sign}(y) = 0$, we have $\mathsf{great}(x, y) = \mathsf{sign}(y - x)$. We have observed that after applying Maxpool in the ReLU layer, the sign-bit of the data becomes 0. Therefore, we only need to calculate $\mathsf{sign}(y - x)$.

There are two approaches to evaluate $\mathsf{max}(x_1, \dots, x_n)$. One is to evaluate $\mathsf{max}(x_1, \dots, x_n)$ by $\mathsf{max}(x_1, \dots, x_n) = \sum_{i=1}^{n} (\Pi_{j=1, j\neq i}^{n} \mathsf{great}(x_i, x_j) \cdot x_i)$, which perform $\Theta(n^2)$ comparisons in the constant round. The other is to search for the maximum value through the binary tree, i.e. reduce $n$-dimension maximum to 2-dimension by expending $\mathsf{max}(x_1, \dots, x_n) = \mathsf{max}(\mathsf{max}(x_1, x_2), \dots, v(x_{n-1}, x_n))$. This method requires $\Theta(\log n)$ rounds to perform a total of $n - 1$ times 2-dimension maximum. We observe that the Maxpool procedure may re-use some comparison outcomes more than once while performing the aforementioned maximum operation, depending on the kernel shape and stride. For instance, we assume $z_{i,j}$ is the result element of performing $(2, 2)$-kernel shape and 1-stride Maxpool over an $a\times b$-dimension matrix requires where $z_{i,j} = \mathsf{max}(x_{i,j}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1})$ and $z_{i,j+1} = \mathsf{max}(x_{i,j+1}, x_{i,j+2}, x_{i+1,j+1}, x_{i+1,j+2})$. Both $z_{i,j}$ and $z_{i,j+1}$ needs the outcome of $\mathsf{great}(x_{i,j+1}, x_{i+1,j+1})$. We adopt the binary tree solution for its property to eliminate the repeated comparison due to storing the temporary comparison result.

The 2-dimension maximum $\mathsf{max}(x_i, x_j)$ can be calculated as $(x_i - x_j) \cdot \mathsf{great}(x_i, x_j) + x_j$, i.e. $(x_i - x_j) \cdot \mathsf{sign}(x_j - x_i) + x_j$. In the previous chapter, we implemented $f(x) = x \cdot \mathsf{sign}(x)$ in two rounds by introducing $2\ell$ bits of communication overhead in the online phase. We use it to evaluate $\mathsf{max}(x_i, x_j)$ by $\mathsf{max}(x_i, x_j) = x_j - f(x_j - x_i)$. We apply this approach to evaluate Maxpool, which requires $(\ell - 1)\log\ell + 2\ell$ bits of communication cost in the setup phase and $(n-1)(1\ell\log\ell + 2\ell)$ bits in the online phase. Analogously, the malicious version of Maxpool can be achieved through verifying sign-bit-exact and multiplication respectively.

### *A. The proof of Theorem 1.*

**Theorem 1.** *Let $\mathcal{L} := (L_0, \dots, L_{\ell-1}) \in \{0, 1\}^{\ell}$ be a binary vector. There exists a linear transformation $\phi$ such that $\phi(\mathcal{L}) = (L'_0, \dots, L'_{\ell-1})$ satisfies:*

- *Let $i^* \in \mathbb{Z}_\ell$ be the index of the first non-zero bit in $\mathcal{L}$, that is, $L_{i^*} = 1 \wedge \forall i < i^* : L_i = 0$.*
- *$L'_{i^*} = 0$ and $L'_j \neq 0$ for all $i \neq i^*$.*

*Proof.* Consider the transformation $\phi(\mathcal{L}) := (L'_0, \dots, L'_{\ell-1})$ such that $L'_i = \sum_{t=0}^{i} L_t - 2 \cdot L_i + 1$ for $i \in \mathbb{Z}_\ell$. Let $s_i := \sum_{t=0}^{i} L_t$ be the prefix-sum of $\mathcal{L}$ and $\mathcal{L}' = \phi(\mathcal{L}) = s_i - 2 \cdot L_i + 1$. We argue that $\mathcal{L}'$ will only contain one zero at the position $i^*$, where $L'_i \neq 0$ for all $i \neq i^*$. Indeed, it converts all the prefix zero bits of $\mathcal{L}$ to 1 (namely, if $s_i = 0 \wedge \mathcal{L}_i = 0$ then $\mathcal{L}'_i = 1$); it converts the first non-zero bit of $\mathcal{L}$ to 0 (namely, if $s_{i^*} = 1 \wedge \mathcal{L}_{i^*} = 1$ then $\mathcal{L}'_{i^*} = 0$); it converts

TABLE V: The communication cost of our protocols. (Offline.Com./Online.Com./Com.: the communication cost of offline/online and malicious phase. Rounds: the communication rounds of the online phase. $\ell$ is the ring size. $\lambda$:the statistical security parameter. $n$:the MaxPool size.)

| Operation | Execution(Semi-honest) | | | Verification | |
|---|---|---|---|---|---|
| | Offline.Com.(bit) | Rounds | Online.Com.(bit) | Rounds | Com.(bit) |
| Sign-bit Extraction | $(\ell-1)\log\ell+2\ell$ | 2 | $2\ell\log\ell+4\ell$ | 3 | $4\ell\log\ell+10\ell$ |
| ReLU | $(\ell-1)\log\ell+4\ell$ | 2 | $\ell\log\ell+4\ell$ | 3 | $4\ell\log\ell+13\ell$ |
| MaxPool | $(n-1)((\ell-1)\log\ell+2\ell)$ | $\log n$ | $(n-1)\ell(\log\ell+2)$ | $2\log n$ | $(n-1)(4\ell\log\ell+13\ell)$ |

**Protocol $\Pi_{\mathsf{ReLU}}(\langle x\rangle)$**

Input : $\langle\cdot\rangle$-shared value of $x$.
Output : $\langle\cdot\rangle$-shared values of $z=\mathsf{sign}(x)$ and $w=\mathsf{ReLU}(x)$.

**Preprocessing:**

- All parties perform $[r''],[r'],[r_z],[r_w]\leftarrow\Pi_{[\cdot]}$;
- $P_i$, for $i\in\{1,2\}$ pick $\Delta\in\{0,1\}$ and reveal $[\Gamma]=\Delta+[r']-2\Delta\cdot[r']+[r_z]$ to each other;
- $P_i$, for $i\in\{1,2\}$ calculate $[\Gamma']=\Gamma\cdot[r_x]-(1-2\Delta)[r'']+[r_x\cdot r_z]-[r_w]$;
- $P_0$ does:
  1) calculate $\hat{r}_x=-r_x-\mathsf{sign}(-r_x)\cdot 2^{\ell-1}\in\mathbb{Z}_{2^\ell}$;
  2) extract $2^{\ell-1}-1-\hat{r_x}$ as $\{r_{x,0},\dots,r_{x,\ell-2}\}$;
  3) perform $[\![r_{x,i}]\!]^p\leftarrow\Pi^p_{[\![\cdot]\!]}(r_{x,i})$ for $i\in\mathbb{Z}_{\ell-1}$, taking the biggest prime of $p\in(\ell,2^{\log\ell+1}]$;
  4) perform $[r_x\cdot r_z]\leftarrow\Pi_{[\cdot]}(r_x\cdot r_z)$;

**Online:**

- $P_j$, for $j\in\{1,2\}$ does:
  1) set $\hat{m}_x=m_x-\mathsf{sign}(m_x)\cdot 2^{\ell-1}$ and bitexact it as $\{\hat{m}_{x,i}\in\{0,1\}\}_{i\in\mathbb{Z}_\ell}$ while $\sum_{i=0}^{\ell-1}2^{\ell-1-i}\hat{m}_{x,i}=\hat{m}_x$;
  2) set $\hat{m_x}_{|\ell}=0$ and $[\![r_{x,\ell}]\!]=[\![1]\!]$;
  3) set $[\![m_i]\!]^p=\hat{m}_{x,i}+[\![r_{x,j}]\!]^p-2\hat{m}_{x,i}\cdot[\![r_{x,i}]\!]^p$ for $i\in\mathbb{Z}_\ell$.
  4) pick same random values $\{w_i,w_i'\in\mathbb{Z}_p^*\}_{i\in\mathbb{Z}_\ell}$ via PRF with seed $\eta_{1,2}$;
  5) calculate $[\![m_i']\!]^p=\sum_{t=1}^{i}[\![m_t]\!]^p-2\cdot[\![m_i]\!]^p+1$ and $[\![u_i]\!]^p=w_i\cdot[\![m_i']\!]^p\cdot(1\oplus\mathsf{sign}(m_x)\oplus\hat{m}_{x,i}\oplus\Delta)+w_i\cdot(\mathsf{sign}(m_x)\oplus\hat{m}_{x,i}\oplus\Delta)$ for $i\in\mathbb{Z}_\ell$;
  6) pick a random permutation $\pi$ via PRF with seed $\eta_{1,2}$ and permute the list $\{[\![\hat{u}_i]\!]^p\}_{i\in\mathbb{Z}_\ell}=\pi(\{[\![u_i]\!]^p\}_{i\in\mathbb{Z}_\ell})$;
  7) reveal $\{[\![\hat{u}_i]\!]^p\}_{i\in\mathbb{Z}_\ell}$ to $P_0$ and reveal $\Gamma''=m_x\cdot[r_z]+[\Gamma']$ to each other simultaneously;
- $P_0$ sets $m'=\mathsf{sign}(-r_x)-r'$ if $\exists\hat{u}_i=0$ for $i\in\mathbb{Z}_\ell$, else $m'=(1\oplus\mathsf{sign}(-r_x))-r'$;
- $P_0$ sets $m''=m'\cdot r_x+r''$;
- $P_0$ sends $m'$ and $m''$ to $P_j$, for $j\in\{1,2\}$;
- $P_j$, for $j\in\{1,2\}$ sets $m_z=m'-2\Delta\cdot m'+\Gamma$ and $m_w=m_xm_z+(1-2\Delta)m''+\Gamma''$;
- All parties output $\langle z\rangle:=([r_z],m_z)$ and $\langle w\rangle:=([r_w],m_w)$.

Fig. 16: The 2-round ReLU Protocol.

the suffix bits to non-zero values (namely, in case $\mathcal{L}_i=0$, $s_i\geq s_{i^*}+\mathcal{L}_i=1$, we have $\mathcal{L}_i'=s_i-2\mathcal{L}_i+1\geq 2$; in case $\mathcal{L}_i=1$, $s_i\geq s_{i^*}+\mathcal{L}_i=2$, we have $\mathcal{L}_i'=s_i-2\mathcal{L}_i+1\geq 1$).

This concludes our proof. □

### *B. The proof of Theorem 3.*

**Theorem 3.** *Let $\mathsf{PRF}^{\mathbb{Z}_p}$ and $\mathsf{PRF}^{\mathbb{Z}_{2^\ell}}$ be the secure pseudo-random functions. The protocol $\Pi_{\mathsf{VSignBit}}$ as depicted in Fig. 11 UC-realizes $\mathcal{F}_{\mathsf{VSignBit}}$ in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model against malicious PPT adversaries who can statically corrupt up to one party.*

*Proof.* To prove Thm. 3, we construct a PPT simulator $\mathcal{S}$, such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between the ideal world and the real world. We consider the following cases:

Case 1: $P_0$ (or $P_1$) is corrupted.

Simulator: The simulator $\mathcal{S}$ internally runs $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{Mult}}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest $P_1$, $P_2$. $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- $\mathcal{S}$ holds seeds $\eta_{0,1}$ and $\eta_{0,2}$;
- $\mathcal{S}$ generates $\{d_{1,i}\}_{i\in\mathbb{Z}_\ell}$, $\{c_{0,i}\}_{i\in\mathbb{Z}_\ell}$ with seed $\eta_{0,1}$;
- $\mathcal{S}$ generates $\Delta'$, $\{d_{2,i}\}_{i\in\mathbb{Z}_\ell}$ with seed $\eta_{0,2}$;
- Upon receiving $(\mathsf{Input},\mathsf{sid})$ from $\mathcal{F}_{\mathsf{VSignBit}}$, $\mathcal{S}$ set $\mathsf{mail}=0$.
- Pick random $[r_{\Gamma'}]_2$, play the role of $P_1$ and $P_2$ and send $[r_{\Gamma'}]_2$ to $P_0$;
- Upon receiving the messages $(d_{1,i},0)$, $(0,d_{2,i})$, $(0,c_{0,i})$ from $P_0$ send to internal $\mathcal{F}_{\mathsf{Mult}}$, check if the messages is correct, abort if not;
- $\mathcal{S}$ picks random value $c_{2,i}\leftarrow\{0,1\}$, for $i\in\mathbb{Z}_\ell$;
- $\mathcal{S}$ inputs $(0,0,d_{1,i})$, $(0,d_{2,i},0)$, $(0,0,c_{0,i})$ and $(c_{2,i},0,0)$ to internal $\mathcal{F}_{\mathsf{Mult}}$ and forword results $\langle\!\langle\hat{r}_{x,i}\rangle\!\rangle^0_0$, $\langle\!\langle\hat{r}'_{x,i}\rangle\!\rangle^0_0$, $\langle\!\langle\gamma(\hat{r}_{x,i})_i\rangle\!\rangle^0_0$, $\langle\!\langle\gamma(\hat{r}'_{x,i})_i\rangle\!\rangle^1_0$, $\langle r_x\rangle^0_0$, $\langle r'_x\rangle^1_0$ to $P_0$;
- $\mathcal{S}$ sets $[r_x]^0:=([r_x]^0_1,[r_x]^0_2)=([r_{r_x}]^0_1+m_{r_x},[r_{r_x}]^0_2)$ and sends $(\mathsf{Input},\mathsf{sid},([r_x]_1,[r_x]_2))$ to $\mathcal{F}_{\mathsf{VSignBit}}$; sets $r_x=[r_x]^0_1+[r_x]^0_2$;
- $\mathcal{S}$ picks random $\delta\leftarrow\mathbb{Z}_{2^\ell}$ and play the role of $P_1$ and $P_2$ to send it to corrupted $P_0$.
- Upon receiving $\delta'$ from $P_0$ send to $P_2$, check if it equals to $[r_x]^0_1-[r'_x]^1_1$. If $[r_x]^0_1-[r'_x]^1_1\neq\delta'$, abort.
- Upon receiving $\|\hat{u}'_i\|$ from $P_0$ send to $P_1$, check whether it is calculated by correct $m'_x$ and $\|r'_{x,j}\|_0$. Abort if checking fail.
- $\mathcal{S}$ picks random list $\{\hat{u}_i\}_{i\in\mathbb{Z}_\ell}$ as following steps:
  - Set $\hat{u}_i\leftarrow\mathbb{Z}_p^*$.
  - Pick coin $\leftarrow\{0,1\}$.
  - Pick index $\leftarrow\mathbb{Z}_\ell$.
  - If coin equal 1, set $\hat{u}_{\mathsf{index}}=0$.
  - Calculate $\gamma(\hat{u}_j)$ for each $\hat{u}_j$ using MAC keys $\alpha$;
  - Send $\{\hat{u}_j\}_{j\in\mathbb{Z}_\ell}$ and $\gamma(\hat{u}_j)$ to the corrupted $P_0$;
- Upon receiving $[t]^0_1$ and $[t]^0_2$ from corrupted $P_0$, pick random $[t']^1_0$ and play as $P_1$ to send it to $P_0$;
- $\mathcal{S}$ calculates $t=[t]^0_1+[t]^0_2$, if $t\neq\mathsf{sign}(r_x)\oplus 1\oplus(\exists\hat{u}_i=0)$, set $\mathsf{mail}=1$.

- Upon receiving $\langle t\rangle_0$, $\langle\Delta\rangle_0$, $\langle t'\rangle_0$ and $\langle\Delta\rangle'_0$ from corrupted $P_0$, check if
  - $\langle t\rangle_0 := ([t]_1^0, [t]_2^0)$;
  - $\langle\Delta\rangle_0^0 := (0, 0)$;
  - $\langle t'\rangle_0 := ([t]_0^{\prime 1}, 0)$;
  - $\langle\Delta'\rangle_0^1 := (0, \Delta')$;
- If the check fails, abort; otherwise, pick random $\langle z\rangle_0 := ([z]_1, [z]_2)$, $\langle z'\rangle_0 := ([z']_1, [z']_2)$ as the $\mathcal{F}_{\mathsf{Mult}}$ output and send them to corrupted $P_0$, pick $\langle c\rangle_0 := ([c]_1, [c]_2)$ as the $\mathcal{F}_{\mathsf{Mult}}$ output and send them to $P_0$.
- If mail $= 1$, $\mathcal{S}$ picks random number $r \leftarrow \mathbb{Z}_{2^\ell}$ and set $c = (2\cdot r + 1)(t - \mathsf{sign}(r_x) \oplus 1 \oplus (\exists \hat{u}_i = 0))$, reveals $c$ to $P_0$ and aborts,
- If mail $= 0$, $\mathcal{S}$ reveals $c = 0$ to $P_0$;
- Upon receiving $\langle c\rangle_0$ from $P_0$ to $P_1$ and $P_2$, abort if it is inconsistent with previously exchanged messages;
- Upon receiving $\langle z\rangle_0$ from $P_0$ to $\mathcal{F}_{\mathsf{Reshare}}$, abort if it is inconsistent with previously exchanged messages;
- $\mathcal{S}$ let $\mathcal{F}_{\mathsf{VSignBit}}$ output, receive $\langle\hat{z}\rangle_0$ and take as the output of $\mathcal{F}_{\mathsf{Reshare}}$ send to $P_0$;

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{Real}_{\Pi_{\mathsf{VSignBit}},\mathcal{A},\mathcal{Z}}(1^\kappa)$.

Hybrid $\mathcal{H}_1$: It is modified from $\mathcal{H}_0$ in that $\langle\!\langle \hat{r}_{x,i}\rangle\!\rangle_0^0$, $\langle\!\langle \hat{r}'_{x,i}\rangle\!\rangle_0^0$, $\langle\!\langle\gamma(\hat{r}_{x,i})_i\rangle\!\rangle_0^0$, $\langle\!\langle\gamma(\hat{r}'_{x,i})_i\rangle\!\rangle_0^1$, $\langle r_x\rangle_0^0$, $\langle r'_x\rangle_0^1$ sent to $P_0$ is calculated through $\mathcal{F}_{\mathsf{Mult}}$ using random share $(c_{2,i}, 0, 0)$ sampled by $\mathcal{S}$;

Hybrid $\mathcal{H}_2$: It is modified from $\mathcal{H}_1$ in that $\delta$ sent to $P_0$ is randomly sampled and the correctness of $\delta'$ is checked by $[r_x]_1^0 - [r'_x]_1^1 \neq \delta'$;

Hybrid $\mathcal{H}_3$: It is modified from $\mathcal{H}_2$ in that the correctness of $\|\hat{u}'_i\|_0$ is checked by $m'_x$ and $\|r'_{x,j}\|_0$ rather than MAC verification;

Hybrid $\mathcal{H}_4$: It is the same as $\mathcal{H}_3$ except that $\|\hat{u}_i\|$ sent to $P_0$ is generated as following:

- Set $\hat{u}_i \leftarrow \mathbb{Z}_p^*$.
- Pick coin $\leftarrow \{0, 1\}$.
- Pick index $\leftarrow \mathbb{Z}_\ell$.
- If coin equal 1, set $\hat{u}_{\mathsf{index}} = 0$.
- Calculate $\gamma(\hat{u}_j)$ for each $\hat{u}_j$ using MAC keys $\alpha$;
- Send $\{\hat{u}_j\}_{j\in\mathbb{Z}_\ell}$ and $\gamma(\hat{u}_j)$ to the corrupted $P_0$;

Hybrid $\mathcal{H}_5$: It hybrid differs from $\mathcal{H}_4$ in that:

1) $[t']_0^1$ sent to $P_0$ is randomly sampled;
2) set the dual execution flag mail $\in \{0,1\}$ by checking $t = \mathsf{sign}(r_x) \oplus 1 \oplus (\exists\hat{u}_i = 0)$.

Hybrid $\mathcal{H}_6$: It is modified from $\mathcal{H}_5$ in that

1) $\mathcal{S}$ performs direct validity checks on the shares $\langle t\rangle_0$, $\langle\Delta\rangle_0$, $\langle t'\rangle_0$, rather than performs $\mathcal{F}_{\mathsf{Mult}}$;
2) $\langle c\rangle_0$ sent to $P_0$ is randomly sampled by $\mathcal{S}$, rather than outputted by $\mathcal{F}_{\mathsf{Mult}}$;

Hybrid $\mathcal{H}_7$: It is modified from $\mathcal{H}_6$ in that $\mathcal{S}$ chooses whether to output $c$ as zero or a randomly sampled non-zero value through the dual execution flag mail $\in \{0,1\}$, instead of deriving $c$ via the computation $(z - z')\cdot r$.

Hybrid $\mathcal{H}_8$: It is the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}},\mathcal{S},\mathcal{Z}}(1^\kappa)$ which is modified from $\mathcal{H}_7$ in that (1) $\mathcal{S}$ performs direct validity checks on $\langle c\rangle_0$ rather than employs the replicated share verification mechanism, (2) $\mathcal{S}$ performs direct validity checks on $\langle z\rangle_0$ and outputs $\langle\hat{z}\rangle_0$ sourced from $\mathcal{F}_{\mathsf{VSignBit}}$ rather than performs $\mathcal{F}_{\mathsf{Reshare}}$.

**Claim 3.** *If* $\mathsf{PRF}^{(\mathbb{Z}_p)^p}$ *is the secure permutation with adversarial advantage* $\mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A})$*, then the ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}},\mathcal{S},\mathcal{Z}}(1^\kappa)$ *and the real world* $\mathsf{Real}_{\Pi_{\mathsf{VSignBit}},\mathcal{A},\mathcal{Z}}(1^\kappa)$ *are indistinguishable with advantage* $e = \mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A}) + \frac{1}{2^{\lambda(\log \ell + 1)}}$.

*Proof.* $\mathcal{H}_0$ **and** $\mathcal{H}_1$ **are indistinguishable.** The executions in both worlds are indistinguishable to $P_0$'s view, since the value $c_{2,i}$ is inherently uniformly random from $P_0$'s perspective.

$\mathcal{H}_1$ **and** $\mathcal{H}_2$ **are indistinguishable.** Since $[r'_x]_2^1 = c_0 \oplus c_2 - [r'_x]_0^1$ where $c_2$ is a uniformly random value unknown to $P_0$, $[r'_x]_2^1$ maintains perfect randomness from $P_0$'s perspective. $\delta = m_x - [r'_x]_2^1$ can be viewed as uniformly random to $P_0$.

$\mathcal{H}_2$ **and** $\mathcal{H}_3$ **are indistinguishable with advantage** $\frac{1}{2^{\lambda(\log \ell+1)}}$ . If corrupted $P_0$ can distinguish $\mathcal{H}_2$ and $\mathcal{H}_5$ we can construct a adversary to pass the MAC verification with introducing error. Informally, for $\lambda$ MAC keys over $\mathbb{Z}_p$, namely, $\alpha_0, \ldots, \alpha_{\lambda-1}$, and the errors $e, e_0, \ldots, e_{\lambda-1}$ over $\mathbb{Z}_p$, the probability $e\cdot(\alpha_0, \ldots, \alpha_{\lambda-1}) = (e_0, \ldots, e_{\lambda-1})$ is $p^{-\lambda}$. Taking $p \approx 2^{\log \ell + 1}$, it equals to $\frac{1}{2^{\lambda(\log \ell+1)}}$.

$\mathcal{H}_3$ **and** $\mathcal{H}_4$ **are indistinguishable with advantage** $\mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A})$ **for the secure permutation** $\mathsf{PRF}^{(\mathbb{Z}_p)^p}$**.** This part admits a proof structure similar to Theorem 2.

$\mathcal{H}_4$ **and** $\mathcal{H}_5$ **are indistinguishable.** Since $[t']_2^1$ is uniformly random to $P_0$, $[r'_x]_1^1 = t' - [r'_x]_2^1$ is also uniformly random to $P_0$.

$\mathcal{H}_5$ **and** $\mathcal{H}_6$ **are indistinguishable.** Corrupted $P_0$ submitting invalid $\langle t\rangle_0$, $\langle\Delta\rangle_0$ and $\langle t'\rangle_0$ will trigger abortion in both worlds;

$\mathcal{H}_6$ **and** $\mathcal{H}_7$ **are indistinguishable.** Considering $P_0$ introduce even error $e$ on $t$ in $\mathcal{H}_6$, it will make $c = (z - z')(2\cdot r + 1) = e(1 - 2\Delta)(2\cdot r + 1) = e(2\cdot r' + 1)$, and in such case, $\mathcal{S}$ picks random $r'$ is random in $P_0$'s view.

$\mathcal{H}_7$ **and** $\mathcal{H}_8$ **(**$\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}},\mathcal{S},\mathcal{Z}}(1^\kappa)$**) are indistinguishable.** By configuring the output of $\mathcal{F}_{\mathsf{Reshare}}$ to be the same as $\mathcal{F}_{\mathsf{VSignBit}}$'s output, the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}},\mathcal{S},\mathcal{Z}}(1^\kappa)$ achieve same output as the real world.

The overall advantage is $e = \mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A}) + \frac{1}{2^{\lambda(\log \ell+1)}}$. ☐

Case 2: $P_2$ is corrupted.

Simulator: The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from $\mathcal{Z}$ and simulates $\mathcal{F}_{\mathsf{Mult}}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest $P_0$, $P_1$. $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- $\mathcal{S}$ holds seeds $\eta_{1,2}$ and $\eta_{0,2}$, receives $m_x$ from $\mathcal{Z}$;
- $\mathcal{S}$ generates $\{d_{2,i}\}_{i\in\mathbb{Z}_\ell}$ with seed $\eta_{0,2}$;
- $\mathcal{S}$ generates $\{c_{2,i}\}_{i\in\mathbb{Z}_\ell}$ and $\Delta$ with seed $\eta_{1,2}$;

- Upon receiving the messages $(0, d_{2,i})$, $(0, c_{2,i})$ from $P_2$ send to internal $\mathcal{F}_{\mathsf{Mult}}$, check if the messages is correct, abort if not;
- $\mathcal{S}$ picks random value $c_{0,i} \leftarrow \{0,1\}$, $d_{1,i} \leftarrow \{0,1\}$, for $i \in \mathbb{Z}_\ell$;
- $\mathcal{S}$ inputs $(0, 0, d_{1,i})$, $(0, d_{2,i}, 0)$, $(0, 0, c_{0,i})$ and $(c_{2,i}, 0, 0)$ to internal $\mathcal{F}_{\mathsf{Mult}}$ and forword results $\langle \hat{r}_{x,i} \rangle_2^0$, $\langle \hat{r}'_{x,i} \rangle_2^0$, $\langle \gamma(\hat{r}_{x,i})_i \rangle_2^0$, $\langle \gamma(\hat{r}'_{x,i})_i \rangle_2^1$, $\langle r_x \rangle_2^0$, $\langle r'_x \rangle_2^1$ to $P_2$;
- $\mathcal{S}$ sets $[r_x]^0 := ([r_x]_1^0, [r_x]_2^0) = ([r_{r_x}]_1^0 + m_{r_x}, [r_{r_x}]_2^0)$;
- $\mathcal{S}$ picks random $\delta' \leftarrow \mathbb{Z}_{2^\ell}$ and play the role of $P_0$ and $P_1$ to send it to corrupted $P_2$.
- Upon receiving $\delta$ from $P_2$ send to $P_0$, check if it equals to $m_x - [r'_x]_2^1$. If $m_x - [r'_x]_2^1 \neq \delta$, abort.
- Upon receiving $\|\hat{u}_i\|$ from $P_2$ send to $P_0$, check whether it is calculated by correct $m_x$ and $\|r_{x,j}\|_2$. Abort if checking fail.
- Upon receiving $\|\hat{u}'_i\|$ from $P_2$ send to $P_1$, check whether it is calculated by correct $m'_x$ and $\|r'_{x,j}\|_2$. Abort if checking fail.
- $\mathcal{S}$ randomly samples $[t]_2^0 \leftarrow \mathbb{Z}_{2^\ell}$, $[t']_2^1 \leftarrow \mathbb{Z}_{2^\ell}$ play as $P_0$ and $P_1$ to send them to corrupted $P_2$;
- Upon receiving $\langle t \rangle_2$, $\langle \Delta \rangle_2$, $\langle t' \rangle_2$ and $\langle \Delta \rangle'_2$ from corrupted $P_2$, check if
  - $\langle t \rangle_2 := (0, [t]_2^0)$;
  - $\langle \Delta \rangle_2^0 := (0, \Delta)$;
  - $\langle t' \rangle_2 := ([t]_2'^1, 0)$;
  - $\langle \Delta' \rangle_2^1 := (0, \Delta')$;
- If the check fails, abort; otherwise, pick random $\langle z \rangle_2 := ([z]_0, [z]_1)$, $\langle z' \rangle_2 := ([z']_0, [z']_1)$ as the $\mathcal{F}_{\mathsf{Mult}}$ output and send them to corrupted $P_0$, pick $\langle c \rangle_2 := ([c]_0, [c]_1)$ as the $\mathcal{F}_{\mathsf{Mult}}$ output and send them to $P_0$.
- $\mathcal{S}$ reveals $c = 0$ to $P_0$;
- Upon receiving $\langle c \rangle_0$ from $P_0$ to $P_1$ and $P_2$, abort if it is inconsistent with previously exchanged messages;
- Upon receiving $\langle z \rangle_2$ from $P_2$ to $\mathcal{F}_{\mathsf{Reshare}}$, abort if it is inconsistent with previously exchanged messages;
- $\mathcal{S}$ let $\mathcal{F}_{\mathsf{VSignBit}}$ output, receive $\langle \hat{z} \rangle_2$ and take as the output of $\mathcal{F}_{\mathsf{Reshare}}$ send to $P_0$;

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \mathcal{H}_1$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{Real}_{\Pi_{\mathsf{VSignBit}}, \mathcal{A}, \mathcal{Z}}(1^\kappa)$.

Hybrid $\mathcal{H}_1$: It is modified from $\mathcal{H}_0$ in that $\langle \hat{r}_{x,i} \rangle_2^0$, $\langle \hat{r}'_{x,i} \rangle_2^0$, $\langle \gamma(\hat{r}_{x,i})_i \rangle_2^0$, $\langle \gamma(\hat{r}'_{x,i})_i \rangle_2^1$, $\langle r_x \rangle_2^0$, $\langle r'_x \rangle_2^1$ sent to $P_2$ is calculated through $\mathcal{F}_{\mathsf{Mult}}$ using random share $(0, 0, c_{0,i})$ and $(0, 0, c_{1,i})$ sampled by $\mathcal{S}$;

Hybrid $\mathcal{H}_2$: It is modified from $\mathcal{H}_1$ in that $\delta'$ sent to $P_2$ is randomly sampled and the correctness of $\delta$ is checked by $m_x - [r'_x]_2^1 \neq \delta$;

Hybrid $\mathcal{H}_3$: It is modified from $\mathcal{H}_2$ in that the correctness of $\|\hat{u}_i\|_2$ and $\|\hat{u}'_i\|_2$ is checked by $m_x$, $\|r_{x,j}\|_0$ and $m'_x$, $\|r'_{x,j}\|_0$ rather than MAC verification;

Hybrid $\mathcal{H}_4$: It hybrid differs from $\mathcal{H}_3$ in that $[t]_2^0$ and $[t']_2^1$ sent to $P_2$ is randomly sampled;

Hybrid $\mathcal{H}_5$: It is modified from $\mathcal{H}_4$ in that $\mathcal{S}$ performs direct validity checks on the shares $\langle t \rangle_2$, $\langle \Delta \rangle_2$, $\langle t' \rangle_2$, $\langle \Delta' \rangle_2$ and send random share $\langle z \rangle_2$, $\langle z' \rangle_2$ to $P_2$ rather than performs $\mathcal{F}_{\mathsf{Mult}}$;

Hybrid $\mathcal{H}_6$: It is modified from $\mathcal{H}_5$ in that $\mathcal{S}$ directly reveals $[c]_2 = -[c]_1 - [c]_0$ to corrupted $P_2$;

Hybrid $\mathcal{H}_7$: It is the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}}, \mathcal{S}, \mathcal{Z}}(1^\kappa)$ which is modified from $\mathcal{H}_6$ in that (1) $\mathcal{S}$ performs direct validity checks on $\langle c \rangle_2$ rather than employs the replicated share verification mechanism, (2) $\mathcal{S}$ performs direct validity checks on $\langle z \rangle_2$ and outputs $\langle \hat{z} \rangle_2$ sourced from $\mathcal{F}_{\mathsf{VSignBit}}$ rather than performs $\mathcal{F}_{\mathsf{Reshare}}$.

**Claim 4.** *If $\mathsf{PRF}^{(\mathbb{Z}_p)^p}$ is the secure permutation with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}}, \mathcal{S}, \mathcal{Z}}(1^\kappa)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{VSignBit}}, \mathcal{A}, \mathcal{Z}}(1^\kappa)$ are indistinguishable with advantage $e = \mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A}) + \frac{1}{2^{\lambda(\log \ell+1)}}$.*

*Proof.* **$\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable.** The executions in both worlds are indistinguishable to $P_0$'s view, since the value $c_{2,i}$ is inherently uniformly random from $P_0$'s perspective.

**$\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable.** Since $[r_x]_1^0$ and $[r'_x]_0^1$ are perfect randomness from $P_2$'s perspective. $\delta' = [r_x]_1^0 - [r'_x]_0^1$ can be viewed as uniformly random to $P_0$.

**$\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable with advantage $\frac{1}{2^{\lambda(\log \ell+1)}}$.** Similarly, if corrupted $P_2$ can distinguish $\mathcal{H}_4$ and $\mathcal{H}_5$ we can construct a adversary to pass the MAC verification with introducing error, which equals to $\frac{1}{2^{\lambda(\log \ell+1)}}$.

**$\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable.** Since $[t]_1^0$ and $[t']_0^1$ are uniformly random to $P_2$, $[t]_2^0 = t' - [t]_1^0$ and $[t']_2^1 = t' - [t']_0^1$ are also uniformly random to $P_2$.

**$\mathcal{H}_4$ and $\mathcal{H}_5$ are indistinguishable.** Corrupted $P_2$ submitting invalid $\langle t \rangle_2$, $\langle \Delta \rangle_2$, $\langle t' \rangle_2$ and $\langle \Delta' \rangle_2$ will trigger abortion in both worlds;

**$\mathcal{H}_5$ and $\mathcal{H}_6$ are indistinguishable.** If corrupted $P_2$ does not trigger abortion in the previous steps, it will receive $c = 0$.

**$\mathcal{H}_6$ and $\mathcal{H}_7$ ($\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}}, \mathcal{S}, \mathcal{Z}}(1^\kappa)$) are indistinguishable.** By configuring the output of $\mathcal{F}_{\mathsf{Reshare}}$ to be the same as $\mathcal{F}_{\mathsf{VSignBit}}$'s output, the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{VSignBit}}, \mathcal{S}, \mathcal{Z}}(1^\kappa)$ achieve same output as the real world.

The overall advantage is $e = \mathsf{Adv}_{\mathsf{PRF}^{(\mathbb{Z}_p)^p}}(1^\kappa, \mathcal{A}) + \frac{1}{2^{\lambda(\log \ell+1)}}$. □

□