Maliciously Secure Exact Fixed-Point Multiplication over Power-of-Two Rings for Replicated 3PC

Yutao Sun[1†], Jianguo Xie[1,2†], Guozhen Shi[1*], Jiale Han[1], Huiyan Chen[1], Rongna Xie[1]

[1*]Beijing Electronic Science and Technology Institute, Beijing, 100070,China.
[2]University of Science and Technology of China, Hefei, 230027, China.

*Corresponding author(s). E-mail(s): sgz1974@163.com;
Contributing authors: yutao_sun@foxmail.com; jgxie@mail.ustc.edu.cn;
[†]These authors contributed equally to this work.

Abstract

Exact fixed-point multiplication over $\mathbb{Z}_{2^k}$ is a fundamental primitive for secure fixed-point arithmetic. However, in the honest-majority, maliciously secure 3PC setting, no prior work simultaneously provides cross-ring compatibility, exact semantics, and malicious security within this efficient framework. In this paper, we address this gap by showing that the core cross-ring bottlenecks, namely exact signed truncation and signed extension, share a unified algebraic structure. Based on this insight, we propose a general **quotient-correction framework** that reduces complex non-linear cross-ring operations to a highly efficient **2-bit bounded-quotient extraction** problem. We instantiate this framework to construct maliciously secure protocols for exact truncation and extension. By sequentially composing these primitives with standard in-ring multiplication, we realize the first end-to-end exact fixed-point multiplication protocol that satisfies all aforementioned requirements in the replicated 3PC setting. We also present optimized variants under relaxed guarantees (e.g., 1-ULP error) that offer superior performance trade-offs. We formalize our constructions within the Universal Composability (UC) framework and provide rigorous security proofs. Theoretical analysis and experimental results demonstrate that our approach achieves practical online efficiency while maintaining exact semantics and malicious security, overcoming the limitations of prior baselines regarding security assumptions, input domains, or output precision.

Keywords: multi-party computation, fixed-point multiplication, maliciously secure, UC framework

1 Introduction

Secure multi-party computation (MPC) provides a cryptographic method for performing joint computations without revealing the private inputs of the participating parties, and has become one of the key technical approaches in fields such as privacy-preserving machine learning[1–3]. In such applications, the underlying computations typically involve decimal or real-valued quantities[4, 5]. Therefore, fixed-point arithmetic is commonly used in practical systems to represent and process these values[6]. Within this computational layer, fixed-point multiplication is particularly fundamental, because the core computation of many machine-learning tasks reduces to dot products, matrix multiplications, and the induced linear-layer and convolution operations[7–9]. Therefore, the secure and efficient implementation of fixed-point multiplication constitutes a critical foundational requirement, rather than a mere localized arithmetic challenge, for the practical deployment of higher-level secure multi-party computation-based machine learning systems.

Among the various settings for secure computation, the honest-majority maliciously secure replicated 3PC setting is an important and widely used target. In practical scenarios, it is difficult to guarantee in advance that all participating parties will strictly follow the prescribed protocol execution. Therefore, compared with security against semi-honest adversaries only, security against malicious adversaries is often the more realistic choice[10]. At the same time, honest-majority protocols typically achieve significantly lower communication cost than dishonest-majority ones[11]. As a result, honest-majority malicious security MPC has remained an important line of research that continues to receive sustained attention[11]. Under this setting, replicated secret sharing has become one of the mainstream tools in efficient MPC design, and the literature explicitly notes that it underlies many recent efficient protocols and has been used mostly in the context of three parties[12, 13]. Thus, it is meaningful to study exact fixed-point multiplication in the honest-majority maliciously secure replicated 3PC setting.

Moreover, when fixed-point multiplication crosses different precisions or rings, the computational route is also naturally determined by representation alignment and post-multiplication rescaling[14, 15]. In other words, to perform a complete fixed-point multiplication, the input is first lifted to a compatible larger ring, the multiplication is then performed in that ring, and finally, the target precision and output type are restored through truncation. In the setting considered here, this corresponds precisely to signed extension, multiplication in the product ring, and signed truncation. Accordingly, depending on the semantics induced by the truncation step, fixed-point multiplication protocols can be broadly divided into probabilistic, approximate, and exact types.

In existing works, probabilistic truncation and approximate truncation are more commonly used because they are considered sufficient for many PPML scenarios[16], and due to their non-interactive or less-interactive nature, their communication overhead is usually significantly lower than that of exact truncation[17]. However, these relaxed semantics are not always satisfactory. Once fixed-point multiplication is repeatedly called within multiplication chains, or used as an intermediate step in larger composite protocols, the semantics of truncation are directly propagated to the overall computation[18]. CrypTFlow2[19] further points out that fixed-point multiplication is necessarily followed by a truncation step, and that once correctness is sacrificed, the resulting errors can accumulate and cause the final result to deviate substantially from the expected cleartext execution. Therefore, the relevant gap is not merely the lack of fixed-point multiplication protocols, but the lack of a complete cross-ring multiplication path that simultaneously preserves exact semantics and provides malicious security.

In fact, several closely related studies have explored this target from different perspectives, but to the best of our knowledge, no work has yet fully bridged the aforementioned gap. Ditto[15] and Mosformer[20] provides efficient cross-ring fixed-point multiplication, but relies on approximate semantics and a restricted input range. SIRNN[14] supports exact cross-ring fixed-point multiplication, but only in the two-party semi-honest setting. Escudero et al.[21] provides powerful maliciously secure truncation primitives, but it does not by itself give a matching signed-extension component or a complete end-to-end fixed-point multiplication protocol. These limitations motivate a unified treatment of the cross-ring primitives required for exact fixed-point multiplication.

In the exact cross-ring setting considered in this paper, the in-ring multiplication itself already admits a standard realization in the replicated 3PC setting. The central difficulty therefore lies in the two cross-ring nonlinear primitives required to connect the input, product, and output rings, namely signed extension and exact signed truncation. Although these two functionalities serve distinct purposes, we observe that in both cases the target output can be written in the same quotient-correction form $y_{\mathrm{ex}} = \tilde{y} + q \cdot C, \quad q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0, 1, 2, 3\}$, where $\tilde{y}$ is obtained by local linear operations, C is a public correction constant, and the only nonlinear information to be recovered is the two-bit quotient q. Based on this observation, we construct a unified quotient-correction framework, and then instantiate it concretely to obtain exact signed truncation and signed extension protocols. By composing these two cross-ring functionalities with standard in-ring multiplication, we further obtain an exact fixed-point multiplication protocol over power-of-two rings in the replicated 3PC setting. In addition, to better accommodate scenarios where efficiency is prioritized over exact semantics or strongest security guarantees, we also present several more efficient variants under relaxed guarantees.

We further provided a formal proof of our protocol's security within the Universal Composability (UC) framework. Since the unified quotient-correction framework gives the two exact cross-ring primitives a common ideal-functionality interface, it also simplifies the security analysis. Under this interface, exact signed truncation and signed extension are treated as concrete instantiations of the same framework, and the

fixed-point multiplication protocol follows by sequential composition. The proof can therefore proceed through a modular analysis of the generic framework, the shared subprotocols, and the concrete instantiations, rather than through separate ad hoc arguments for each primitive. We also evaluate the proposed protocols through theoretical online communication analysis and experimental online runtime measurements at both the primitive level and the end-to-end multiplication level. Since the closest baselines differ in security model, semantic guarantee, or input-domain restriction, the results are interpreted under these qualifications and show that the proposed constructions achieve feasible online performance while providing stronger exactness and security guarantees.

1.1 Contribution

The main contributions of this paper can be summarized as follows:

1. We formulate a unified quotient-correction perspective for exact cross-ring arithmetic. Based on the insight that the non-linear bottlenecks of exact signed truncation and signed extension can both be rigorously reduced to a highly efficient **2-bit bounded-quotient extraction**, we propose a generic quotient-correction framework that captures these functionalities under a common protocol structure.
2. We instantiate this generic framework to construct maliciously secure protocols for exact signed truncation and general signed extension in the honest-majority replicated 3PC setting. By sequentially composing these two cross-ring primitives with standard in-ring multiplication, we further obtain an exact fixed-point multiplication protocol over power-of-two rings. In addition, we provide several more efficient variants under relaxed guarantees to offer practical trade-offs between efficiency and guarantees.
3. We formalize the security of the proposed framework and its instantiated protocols within the UC framework. The unified quotient-correction abstraction provides a common interface of ideal functionality, which significantly simplifies security analysis. This modularity allows the exact signed truncation, signed extension, and fixed-point multiplication protocols to be rigorously established through concise compositional proofs.
4. We provide both theoretical and experimental evaluations of the proposed protocols. The theoretical analysis quantifies the online communication overhead, while the experimental evaluation measures the online runtime at both the primitive level and the end-to-end multiplication level. The results show that the protocols proposed here achieve feasible online performance under stronger semantic and security requirements than the closest baselines.

2 Related work

The target of this work is exact fixed-point multiplication over power-of-two rings, for which previous works have been explored under different party settings, security models, and sharing domains. For this task, a natural route is to first lift the inputs to the product ring through signed extension, then perform the multiplication in the product

ring, and finally apply signed truncation to recover the desired precision and output ring. Therefore, previous works directly relevant to this target include not only end-to-end fixed-point multiplication protocols, but also the signed extension and signed truncation protocols that support this computation route. From this perspective, we organize the related works along three closely connected lines, namely fixed-point multiplication, signed extension, and signed truncation.

2.1 Fixed-point multiplication

Fixed-point multiplication has been widely studied in secure multi-party computation, with relevant results covering various party configurations, security models, sharing domains, and fixed-point semantics. Existing work includes implementations based on fields, prime rings, and power-of-two rings; it encompasses schemes designed for semi-honest scenarios as well as those aimed at malicious security; and in terms of semantic guarantees, there are approaches with varying levels of semantic guarantees, including exact, approximate, and probabilistic methods. Meanwhile, a more common setup in existing work is that the inputs are already shared over a sufficiently large field or ring, such that the exact product can be directly contained within that field or ring, and the post-multiplication processing primarily involves semantic truncation performed within the same field or ring[2, 5, 12, 13, 16, 19, 22–33]. To illustrate the overall landscape of related work, we present a comprehensive survey in Appendix A.1, with some representative results shown in Table 1.

Table 1 Comparison table for fixed-point multiplication.

Work	Parties	Security	Domain	FP Sem.	Cross-Ring
Catrina et al. 2010[5]	n-party	semi-honest	$\mathbb{Z}_p$ (used as a field)	approx.	no
ABY3 2018[26]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	prob.	no
Dalskov et al. 2020[32]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}/\mathbb{F}_p$	exact/prob.	no
Falcon 2021[33]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	exact*	no
Frederiksen et al. 2024[34]	n-party	malicious/covert	$\mathbb{Z}_{pq}$	approx.	yes
Ditto 2024[15]	3-party	semi-honest	$\mathbb{Z}_{2^k}$	approx.	yes†
Mosformer 2025[20]	3-party	malicious	$\mathbb{Z}_{2^k}$	approx.	yes†
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	exact	yes
Guo et al. 2025[9]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	exact/approx./prob.	yes†
Our Goal	**3-party**	**malicious**	$\mathbb{Z}_{2^k}$	**exact**	**yes**

Here, "prob." stands for probabilistic, where the protocol may output a completely incorrect result with small probability; "approx." stands for approximate, where the output may differ from the correct result by a small value; and "exact" denotes that the protocol guarantees an output with no deviation from the correct result.

† They are valid only when the input range is restricted.

* It claims to have implemented precise truncation, but does not specify the protocol.

At the level of end-to-end fixed-point multiplication, there is relatively little previous work that achieves the target of cross-ring fixed-point multiplication[9, 14, 15, 20, 34–36]. Most protocols proposed in previous work either remain within a single field or ring, or involve cross-ring processing only under relaxed functionality guarantees. To be more specific, in the three-party setting, the only cross-ring result achieving malicious

security is Mosformer[20], published in 2025. However, it provides only approximate semantics and, more importantly, is valid only under a restricted input range, where the absolute value of each input must not exceed one quarter of the underlying ring size. On the other hand, the cross-ring works that achieve exact semantics are all confined to the two-party semi-honest setting[9, 14, 36].

In short, there remains a significant gap in the implementation of end-to-end fixed-point multiplication in the widely used setting considered here. To the best of our knowledge, no existing work has yet proposed a protocol that simultaneously satisfies all four key requirements: cross-ring compatibility, exact output semantics, malicious-security guarantees, and support for a three-party setting. However, the absence of the complete functionality cannot be determined solely at the level of a complete multiplication protocol. If we can obtain, separately, a general signed extension protocol, an in-ring multiplication protocol, and an exact signed truncation protocol that satisfy the aforementioned requirements, then a fixed-point multiplication protocol meeting these goals can be constructed quite naturally. Among these components, maliciously secure in-ring multiplication already admits a standard realization in the replicated 3PC setting. For the other two primitives, further study is necessary to determine whether existing work is sufficient to directly assemble a fixed-point multiplication protocol that meets the aforementioned requirements.

2.2 Signed extension

The existing works related to this functionality mainly include signed extension[9, 14, 35, 36], upcasting[15, 20], and the more general notion of cross-modulus lifting[34]. For the semantics considered here, signed extension and upcasting are not essentially different. Both refer to the operation of lifting the input from $\mathbb{Z}_{2^m}$ to $\mathbb{Z}_{2^n}$, where $n > m$, while preserving the two's-complement signed semantics. For this reason, in this paper, both of them are collectively referred to as signed extension. In contrast, although cross-modulus lifting is also related to modulus transformation, its target semantics differ significantly from those required in this paper. Therefore, it can only be regarded as related work rather than a direct counterpart to the target considered here. A complete comparison of these works is given in Table 2.

Table 2 Comparison table for signed extension.

Work	Parties	Security	Input Domain	Output Domain	Extension Type
Frederiksen et al. 2024[34]	n-party	malicious/covert	$\mathbb{Z}_p$ (or $\mathbb{Z}_q$)	$\mathbb{Z}_q$ (or $\mathbb{Z}_p$)	cross-modulus lifting
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	zero/signed extension
LLAMA 2022[36]	2-party	semi-honest	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	signed extension
ORCA 2024[35]	2-party	semi-honest	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	signed extension
Guo et al. 2025[9]	2-party	semi-honest	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	signed extension†
Ditto 2024[15]	3-party	semi-honest	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	signed extension†
Mosformer 2025[20]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	signed extension†
Our Goal	**3-party**	**malicious**	$\mathbb{Z}_{2^m}$	$\mathbb{Z}_{2^n}$	**signed extension**

† They are valid only when the input range is restricted.

As shown in the table, the available results on signed extension remain limited, and most of them are still concentrated in the two-party semi-honest setting. SIRNN[14], LLAMA[36], ORCA[35], and the work of Guo et al.[9] all study signed extension over power-of-two rings, but none of them meets the requirement of malicious security in a three-party setting. Ditto[15] and Mosformer[20] consider signed extension in the three-party setting, with Mosformer further achieving malicious security; however, these results hold only within a restricted input range and therefore cannot serve as general-purpose components for arbitrary inputs. Moreover, the signed extension procedures used in these works are obtained from zero extension through translation, which further limits their applicability to the general-input setting considered here. Accordingly, under the three-party, maliciously secure, and general-input settings, a general signed extension protocol is still unavailable. Since this primitive is the very first step in the standard computation path for fixed-point multiplication, this gap remains unavoidable.

2.3 Signed truncation

Since signed truncation is one of the most fundamental primitives in fixed-point MPC, there is significantly more research on it than on signed extension. Depending on their output semantics, existing results generally fall into three semantic categories, namely probabilistic truncation[6, 9, 21, 26, 28, 29, 37], approximate truncation[5, 6, 9, 12, 13, 15, 16, 20, 22, 23, 34, 35, 35], and exact truncation[5, 6, 9, 12, 14, 17, 19, 21, 22, 36, 38]. Among them, probabilistic and approximate truncation are more common since they usually incur much lower online costs, while the resulting loss in precision is often regarded as acceptable in many PPML settings. We present a comprehensive survey in Appendix A.2, with some representative results shown in Table 3.

Table 3 Comparison table for signed truncation.

Work	Parties	Security	Input Domain	Output Domain	Truncation Type
Catrina et al. 2010[5]	n-party	semi-honest	$\mathbb{Z}_p$ (used as a field)	$\mathbb{Z}_p$ (used as a field)	exact/approx.
ABY3 2018[26]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	$\mathbb{Z}_{2^k}$	prob.
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact
Guo et al. 2025[9]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/approx./prob.†
Escudero et al. 2020[21]	n-party	semi-honest/malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/prob.
Ditto 2024[15]	3-party	malicious*	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l-k}$	approx. (truncate-reduce)
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l-k}$	exact (truncate-reduce)
Escudero et al. 2020[21]	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l-k}$	exact (truncate-reduce)**
Our Goal	**3-party**	**malicious**	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l-k}$	**exact (truncate-reduce)**

Here, "prob." stands for probabilistic, where the protocol may output a completely incorrect result with small probability; "approx." stands for approximate, where the output may differ from the correct result by a small value; and "exact" denotes that the protocol guarantees an output with no deviation from the correct result.

† Protocols in this paper are valid only when the input range is restricted.

* Ditto is presented overall in the semi-honest security model, whereas its truncation protocol itself is maliciously secure.

** The original paper only gives the version whose output remains on the original ring, while this version with output on the smaller ring follows from a straightforward adaptation of the same protocol.

As shown in the table, there are already some existing works that propose exact truncation protocols. However, these works do not directly match the goal we considered here. In a significant part of the existing works, exact truncation only realizes the semantic effect of truncation, which means that the output value is rescaled, while the secret sharing remains over the original field or ring[5, 6, 12, 19, 21, 22, 38]. At the same time, some of the protocols proposed in existing works do perform an actual reduction of the underlying sharing field or ring after truncation and are therefore functionally closer to the functionality required here[14, 17, 36]. However, these protocols only achieve semi-honest security. Overall, the existing works provide an important research foundation for the exact truncation required in this paper, but they cannot yet serve directly as the component needed here.

However, a further examination of the existing exact truncation protocols reveals a more encouraging picture. The exact truncation protocol given by Escudero et al. in CRYPTO 2020[21] was originally designed for semantic truncation that does not actually reduce the underlying sharing ring. Nevertheless, after several relatively simple modifications, this protocol can also realize the smaller-ring output semantics required here. In other words, from the viewpoint of feasibility alone, the modified protocol can indeed serve as a component for exact fixed-point multiplication. Unfortunately, we did not find a signed extension method that matches this modified protocol. This means that if we adopt this modified truncation protocol as a component, the two cross-ring primitives required for exact fixed-point multiplication would rely on different underlying logic and implementation routes. Although this is not impossible, implementing these two functions in a unified manner would be more favorable for the modular design of the overall protocol and would also be more conducive to establishing the security proof of the overall construction.

3 Technical overview

This work focuses on **maliciously secure exact fixed-point multiplication** within the honest-majority replicated 3PC framework. In MPC, fixed-point values are represented through ring encoding derived from two's-complement integers, so the protocol layer operates uniformly on the underlying ring elements. For this task, a natural procedure involves first sign-extending the inputs from the current ring to the target ring for the product, then performing the multiplication in the larger ring, and finally applying exact signed truncation to match the target precision and output ring. The bottleneck lies in the two cross-ring nonlinear primitives surrounding it, namely **signed extension** and **exact signed truncation**. Therefore, the efficiency and security of exact fixed-point multiplication hinge on low-cost and composable realizations of these two primitives.

Our key insight is that, although exact signed truncation and signed extension serve distinct purposes, the nonlinear information required for recovery in each case is significantly smaller than the full output value. In both scenarios, local linear operations initially yield a base value, while the remaining uncertainty is limited to a small correction term with only a few possible values. Thus, the challenge is not to reconstruct the entire output maliciously, but rather to recover the small nonlinear

correction term. Specifically, for exact truncation, the nonlinear core corresponds to the wrap value; for signed extension, it involves both the wrap resulting from summing the shares and the sign bit of the input. Notably, both cases can be unified as the recovery of a two-bit bounded quotient.

For the exact signed truncation, local truncation of each share already yields a linear main term, while the remaining ambiguity arises only from the carry bits introduced by the discarded local remainders. To be more specific, the sum of the three local remainders can cross the threshold 2^k zero, once, or twice. Thus, the exact result differs from the locally truncated value only by a wrap term taking values in $\{0, 1, 2\}$. Let $\tilde{y}$ denote the base value obtained by local truncation, then the exact truncation result can be computed as

$$y = \tilde{y} + c, \qquad c = \left\lfloor \frac{r_1 + r_2 + r_3}{2^k} \right\rfloor \in \{0, 1, 2\}. \tag{1}$$

Hence, the protocol only needs to determine the wrap-around count of the discarded remainders.

For the signed extension, there is a slight difference from the exact signed truncation. Summing the local shares yields a linear quantity

$$S = x_1 + x_2 + x_3, \tag{2}$$

which differs from the standard representative $\bar{x}$ of the input by a wrap value over $\{0, 1, 2\}$ induced by the summation of shares. However, recovering this wrap value alone is still insufficient to determine the correct signed extension, because under two's-complement semantics the exact result also depends on whether the input lies in the negative range, namely on $\mathrm{MSB}_m(\bar{x})$. Hence, the nonlinear core in the signed extension is the combination of the wrap value and the sign bit. By introducing a public bias $M/2$, these two quantities can be unified into a bounded quotient

$$q = \left\lfloor \frac{S + M/2}{M} \right\rfloor = \mathrm{wrap} + \mathrm{MSB}(\bar{x}) \in \{0, 1, 2, 3\}, \tag{3}$$

for which the exact result can be computed as

$$y = S - qM \pmod{N}. \tag{4}$$

So, the nonlinear information to be recovered is again not the full output, but only "wrap + msb". Thus, the signed extension can also be viewed as a bounded-quotient correction problem representable by only 2 bits.

Based on the analysis above, we unify the two core cross-ring primitives, exact signed truncation and signed extension, into a common protocol design pattern, namely **local base value computation + bounded-quotient recovery + linear correction.** The main idea is to first obtain a base value that differs from the target result only by a small correction term through purely local linear operations,

and then construct an auxiliary quantity u satisfying $u \in [0, 2^{t+2})$ to compress all nonlinear ambiguity into

$$q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0, 1, 2, 3\}. \tag{5}$$

So, these protocols only need to recover a 2-bit bounded quotient q under malicious security instead of applying a generic nonlinear computation. Then, the recovered quotient q is embedded into the target ring and is used to correct the approximate result obtained locally. The only differentiation between exact truncation and signed extension within this paradigm is the construction of u.

By mapping the principal cross-ring nonlinearities to the identical bounded-quotient correction problem, this framework substantially mitigates the heterogeneity of nonlinear operations in MPC. As a result, protocol design, security analysis, and complexity characterization can all be organized around a common maliciously secure core. It substantially reduces the heterogeneity of nonlinear primitives in the protocol and makes the dominant sources of online cost more transparent. In other words, our work identifies and exploits the shared bounded-quotient structure of these operations and uses it to construct a unified, composable, and analytically transparent framework for maliciously secure exact fixed-point multiplication.

4 Preliminaries

4.1 Notation and algebraic preliminaries

Algebraic structures. Protocols in this work primarily operate over the power-of-two rings of the form $\mathbb{Z}_{2^k}$ (for various bit-widths k), which naturally support two's-complement fixed-point arithmetic. For any integer $k > 1$, let $\mathbb{Z}_{2^k} := \mathbb{Z}/2^k\mathbb{Z}$ denote the ring of integers modulo 2^k, represented by the canonical set $\{0, 1, \ldots, 2^k - 1\}$. Additionally, we write $\mathbb{Z}_K$, where $K = 2^k$, for the same ring.

However, since ring $\mathbb{Z}_{2^k}$ contains zero divisors, some verification steps used in our malicious checks are performed after mapping elements from ring $\mathbb{Z}_{2^k}$ into a large prime field $\mathbb{Z}_p$, where p is sufficiently large for the relevant bit-width. The ring-to-field embedding is solely used to avoid pitfalls caused by zero divisors, hence such an embedding cannot preserve the underlying two's-complement semantics in general. Nevertheless, this is sufficient for our purposes since we invoke the embedding only when checking some specific algebraic identities.

Natural embedding and semantic distinction. In this work, we also need natural embedding between two rings in some verification steps. For any $m \leq n$, since $2^m \mid 2^n$, there is a natural ring homomorphism:

$$\iota_{m \to n} : \mathbb{Z}_{2^m} \to \mathbb{Z}_{2^n}, \qquad x \mapsto x \tag{6}$$

Such natural embedding can guarantee congruence, i.e., $\iota_{m \to n}(x) \equiv x \pmod{2^m}$, but it cannot preserve two's-complement signed semantics in general. In short, natural embedding $\iota_{m \to n}$ is not sign-extension. The signed-extension functionality considered later preserves the two's-complement semantics of signed values.

Comparison and absolute value. Unless specified otherwise, $|x|$ refers to the absolute value of the signed integer semantics $|\operatorname{int}_k(x)|$ under two's-complement interpretation. When writing $C < a$ for $a \in \mathbb{Z}_{2^k}$ without further explanation, we treat a as an unsigned integer in $[0, 2^k - 1]$, regardless of whether a itself has signed semantics, and compare it to the public integer constant C.

4.2 Secret sharing and audited opening

Two-party secret sharing. For a value x over either the ring $\mathbb{Z}_{2^k}$ or the prime field $\mathbb{Z}_p$, it is shared between two parties P_0 and P_1 as follows:

$$[x]^K := \left([x]_1^K, [x]_2^K\right) \in \mathbb{Z}_{2^k}^2, \qquad x = [x]_1^K + [x]_2^K \pmod{2^k} \tag{7}$$

or

$$[x]^p := \left([x]_1^p, [x]_2^p\right) \in \mathbb{Z}_p^2, \qquad x = [x]_1^p + [x]_2^p \pmod{p}. \tag{8}$$

For a bit $b \in \mathbb{Z}_2$, write

$$[b]^B := \left([b]_1^B, [b]_2^B\right) \in \mathbb{Z}_2^2, \qquad b = [b]_1^B + [b]_2^B \pmod{2}. \tag{9}$$

Here and throughout, the superscript indicates the sharing domain, while the subscript indicates the share index. Linear operations are performed component-wise. Over $\mathbb{Z}_2$, addition coincides with XOR.

Three-party replicated secret sharing. Similarly, for $x \in \mathbb{Z}_{2^k}$ or $x \in \mathbb{Z}_p$, the replicated secret sharing of x between three parties P_1, P_2 and P_3 is in the form:

$$[\![x]\!]^\star := \left([\![x]\!]_1^\star, [\![x]\!]_2^\star, [\![x]\!]_3^\star\right) \in \mathbb{Z}_\star^3, \qquad x = [\![x]\!]_1^\star + [\![x]\!]_2^\star + [\![x]\!]_3^\star \pmod{\star} \tag{10}$$

where $\star \in \{K, p\}$. The standard replicated layout is used: P_1 holds $([\![x]\!]_1^\star, [\![x]\!]_2^\star)$, P_2 holds $([\![x]\!]_2^\star, [\![x]\!]_3^\star)$, P_3 holds $([\![x]\!]_3^\star, [\![x]\!]_1^\star)$. For a bit $b \in \mathbb{Z}_2$, analogously write

$$[\![b]\!]^B = \left([\![b]\!]_1^B, [\![b]\!]_2^B, [\![b]\!]_3^B\right), \qquad b = [\![b]\!]_1^B + [\![b]\!]_2^B + [\![b]\!]_3^B \pmod{2} \tag{11}$$

with the same replicated layout.

Under the honest-majority assumption that at most one party is corrupted, replicated arithmetic sharing and Boolean sharing provide information-theoretic privacy against any single-party view. However, replicated sharing itself does not imply malicious security for arbitrary protocol execution. In our protocols, malicious behavior is detected through explicit consistency checks on the overlapping local views of the replicated representation, such as through audited openings.

Opening and audited opening. In a standard opening, a party reconstructs the secret value x and reveals it to all other parties. In contrast, during an audited opening, the parties reconstruct and reveal x while simultaneously checking the replication consistency (i.e., overlapping shares held by different parties must match). If the check fails, the execution of the current protocol aborts. Typically, audited openings are not used to reveal secret values, but rather to bind intermediate values to a well-defined replicated sharing in maliciously secure verification procedures.

4.3 Two's-complement integer and fixed-point semantics

Two's-complement integer semantics. For any $x \in \mathbb{Z}_{2^k}$, its k-bit two's-complement signed integer semantics is defined as follows:

$$\mathrm{int}_k(x) = \begin{cases} x, & 0 \le x \le 2^{k-1} - 1, \\ x - 2^k, & 2^{k-1} \le x \le 2^k - 1. \end{cases} \tag{12}$$

In particular, $\mathrm{int}_k(x) \in [-2^{k-1}, 2^{k-1}-1]$, and we further define the sign bit $\mathrm{MSB}_k(x) = \mathbf{1}\{x \ge 2^{k-1}\}$, where $\mathbf{1}\{\cdot\}$ denotes the indicator function.

Fixed-point semantics. Let $x \in \mathbb{Z}_{2^k}$ encode a signed fixed-point value with fractional precision f, where $0 \le f < k$. Its fixed-point semantics is defined by

$$\mathrm{fix}_{k,f}(x) = \frac{\mathrm{int}_k(x)}{2^f} \tag{13}$$

Thus, the ring element x represents the real value obtained by interpreting x as a k-bit two's-complement integer and scaling it by 2^{-f}.

Semantic targets of signed extension and truncation. Let $m \le n$, $x \in \mathbb{Z}_{2^m}$ and $y \in \mathbb{Z}_{2^n}$. The value y is said to be a correct signed extension of x from m-bit to n-bit if and only if $\mathrm{int}_n(y) = \mathrm{int}_m(x)$. Equivalently, if x and y are interpreted with the same fractional precision f, then signed extension preserves the fixed-point value $\mathrm{fix}_{n,f}(y) = \mathrm{fix}_{m,f}(x)$.

Let $x \in \mathbb{Z}_{2^k}$ and let t be a positive integer where $t < k$. A value $y \in \mathbb{Z}_{2^{k-t}}$ is said to be the exact signed truncation, or arithmetic right shift, of x by t bits if and only if $\mathrm{int}_{k-t}(y) = \left\lfloor \frac{\mathrm{int}_k(x)}{2^t} \right\rfloor$. Under fixed-point interpretation, this is precisely the rescaling step obtained by removing t fractional bits. More specifically, if x is interpreted with fractional precision $f + t$, then the exact truncated result y satisfies $\mathrm{fix}_{k-t,f}(y) = \mathrm{fix}_{k,f+t}(x)$.

Although integer and fixed-point values are semantically distinct, the protocol layer treats them uniformly, since all computations are performed on the underlying ring encodings and their secret sharings. Accordingly, the sequel is stated in terms of integer semantics, with the fixed-point semantics used only to interpret the induced functionality.

4.4 3PC protocol building blocks

Comparison to a public constant (LTConst). The protocol $\Pi^{B}_{\mathrm{LTConst,3HMal}}$ compares a secret k-bit value with a public k-bit constant. It takes the bitwise replicated Boolean sharings $(\llbracket x_{k-1} \rrbracket^B, \ldots, \llbracket x_0 \rrbracket^B)$ of a secret value $x \in [0, 2^k)$ and a public constant $C \in [0, 2^k)$ as the inputs, and outputs a replicated Boolean sharing $\llbracket \mathbf{1}_{\{C<x\}} \rrbracket^B$. In this work, this protocol is instantiated using the comparison machinery based on edaBits by Escudero et al.[21], with dominant online cost approximately $2k$ Boolean AND gates.

Audited opening. The protocol $\Pi^{\star}_{\mathrm{AOpen,3HMal}}$ takes a replicated sharing $\llbracket x \rrbracket^{\star}$ as the input and reconstructs and publicly reveals the secret value x. It aborts if the

overlapping local views are inconsistent to prevent malicious parties from modifying local shares. In the case of 2-out-of-3 replicated sharing, each participant needs to send their two shares to the other two participants respectively. Thus, over ring $\mathbb{Z}_{2^k}$, one invocation costs $6k$ bits, i.e., 6 ring elements.

Replicated-sharing multiplication. The protocol $\Pi^{\star}_{\mathrm{Mul,3HMal}}$ is the standard maliciously secure multiplication protocol for replicated arithmetic sharing, where $\star \in \{K, p\}$. It takes inputs $([\![x]\!]^{\star}, [\![y]\!]^{\star})$ and outputs $[\![x \cdot y]\!]^{\star}$. A Beaver-style instantiation uses one preprocessed multiplication triple and two masked openings in the online phase. When using audited openings to perform the masked openings above, its online communication is denoted by $C^{\star}_{\mathrm{Mul}} = 2C^{\star}_{\mathrm{AOpen}}$, i.e., 12 elements.

Remark. The preprocessing required by the above protocols is standard and is not a focus of this work. In particular, multiplication triples are assumed to be provided by any standard honest-majority maliciously secure backend, while daBits and strict edaBits may be provided by an honest-majority maliciously secure backend based on the construction of Escudero et al.[21]. This convention is consistent with the later use of malicious daBits in the 3PC B2A protocol and strict edaBits in the specialized high-two-bit extraction protocol.

4.5 Threat model and security goal

Threat model. We consider the standard honest-majority three-party setting with parties P_1, P_2 and P_3, where at most one party is statically corrupted. Depending on the protocol variant, the corrupted party can be either semi-honest or malicious. All parties are assumed to communicate through secure authenticated point-to-point channels.

Security goal. Unless stated otherwise, the main constructions aim for standard simulation-based malicious security with abort over 2-out-of-3 replicated sharing. Correctness is defined with respect to the semantics in Section 4.3, under which signed extension preserves the two's-complement integer semantics and exact signed truncation implements arithmetic right shift exactly. Although the corresponding proofs are carried out at the level of the integer semantics, the fixed-point semantics is induced directly by the same ring encodings and is therefore preserved as well. Some weaker variants may satisfy only relaxed guarantees, such as semi-honest security or explicit bounded-error correctness.

Assumptions and scope. The focus of this work is the design and analysis of exact cross-ring arithmetic protocols and their composition into maliciously secure fixed-point multiplication. Standard preprocessing required by the invoked black-box protocols is assumed to be available as described in Section 4.4. Denial-of-service attacks, fairness, and guaranteed output delivery are out of scope.

5 A unified quotient-correction framework

The key insight stated in Section 3 suggests that evaluating our target functionalities primarily reduces to computing a small nonlinear correction applied to a locally computable base value. By abstracting this correction process into a generic quotient-correction framework, protocols for exact signed truncation and signed extension can

be unified under a single paradigm. Consequently, the design of concrete functionalities simplifies to instantiating the base value, the auxiliary quantity, and the associated public correction constant.

5.1 The form of quotient-correction

We consider a type of functionality whose output can be divided into a locally computable base value and a small nonlinear correction term scaled by a public constant. Formally, let y_{ex} denote the target output, $\tilde{y}$ denote the locally computable base value, C denote the public constant, and q denote the small nonlinear correction coefficient. Then the output admits the representation

$$y_{ex} = \tilde{y} + qC. \tag{14}$$

Since $\tilde{y}$ is obtained by purely local linear operations and C is a public constant, all nonlinear information required to compute y_{ex} is encapsulated entirely within q. Therefore, once q is obtained, the final output follows from a linear correction. However, at this level of abstraction, the nonlinear correction coefficient q can be an arbitrary value. Only when q itself admits a more structured representation can such decomposition narrow the nonlinear part to be recovered and become algorithmically useful.

The exact constructions considered in this work belong, appropriately, to a more structured subclass in which the correction coefficient is a bounded quotient of an auxiliary quantity. Concretely, we focus on the case where

$$q = \left\lfloor \frac{u}{2^t} \right\rfloor, \qquad u \in [0, 2^{t+2}) \tag{15}$$

for a functionality-specific auxiliary quantity u. Then it follows that $q \in \{0, 1, 2, 3\}$. Here, u will be specified separately for each instantiation, and the parameter t determines the quotient granularity. Hence, for any functionality of this class, the entire nonlinear ambiguity is reduced to recovering a 2-bit bounded quotient q.

The discussion above reduces protocol design for this class of functionalities to two separate tasks. The functionality-specific task lies entirely in constructing a tuple $(\tilde{y}, u, C, t)$, while functionality-independent task involves extracting the bounded quotient q according to the auxiliary quantity u under the parameter t. This separation gives rise to a generic procedure in which the same extraction mechanism can be reused across different functionalities. In particular, both exact signed truncation and signed extension will later be shown to follow this decomposition and thus fall under the same generic protocol template.

5.2 Bounded quotient extraction and correction

As established in Section 5.1, for the structured subclass considered here, protocol design can be decomposed into a functionality-specific part and a functionality-independent part. The former consists of instantiating the tuple $(\tilde{y}, u, C, t)$ for each concrete functionality, and involves only local linear operations, thus introducing no additional distributed nonlinear complexity. By contrast, the latter contains the only

nontrivial nonlinear task shared across all such functionalities, namely the malicious extraction of the bounded quotient q from the auxiliary quantity u under the parameter t. Hence, the common difficulty of this class of constructions is entirely concentrated in a functionality-independent nonlinear layer. It is therefore natural to isolate this layer and realize it first as a unified maliciously secure component.

In the present situation, the correction coefficient satisfies $q = \left\lfloor \frac{u}{2^t} \right\rfloor$, $u \in [0, 2^{t+2})$. Therefore, the auxiliary quantity u can be decomposed as $u = u_{\text{low}} + 2^t q$, $0 \le u_{\text{low}} < 2^t$. So if $b_1, b_0 \in \{0,1\}^2$ denote the two most significant bits of the $(t+2)$-bit representation of u, q can be written as $q = 2b_1 + b_0$. Hence, bounded-quotient extraction is therefore reduced to recovering the two most significant bits of u. This motivates a dedicated protocol $\Pi^{K\to B}_{\text{Top2,3HMal}}$, which takes a shared representation of u as input and outputs replicated Boolean sharings of b_1 and b_0.

A straightforward realization of this task can be obtained by specializing the general logical-right-shift protocol proposed by Escudero et al.[21]. Their protocol is designed to recover the full shifted result over $\mathbb{Z}_{2^k}$; While their protocol recovers the fully shifted result over $\mathbb{Z}_{2^k}$, our setting only requires the two most significant bits. Therefore, we can optimize the protocol by replacing its generic second-stage unmasking procedure with a dedicated recovery step tailored to a two-bit output. The specialization yields the protocol $\Pi^{K\to B}_{\text{Top2,3HMal}}$, and the details are provided in Algorithm 1.

Correctness. The correctness of $\Pi^{K\to B}_{\text{Top2,3HMal}}$ is guaranteed by its two-stage structure. Steps 1–6 recover the low-t-bit block of the input and hence obtain the high block $u_{\text{high}} = u - u_{\text{low}} = 2^t q$, where $q = \lfloor u/2^t \rfloor \in \{0,1,2,3\}$. Steps 7–12 then open a masked version of this high block and remove the 2-bit mask locally in the Boolean domain, yielding $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ such that $q = 2b_1 + b_0$.

Online complexity. The online phase of $\Pi^{K\to B}_{\text{Top2,3HMal}}$ involves the audited opening over ring $\mathbb{Z}_{2^{t+2}}$ twice, the comparison to a public constant protocol $\Pi^{B}_{\text{LTConst,3HMal}}$ once, and the Boolean-to-arithmetic share conversion protocol $\Pi^{B\to K}_{\text{B2A,3HMal}}$ once. Accordingly, we write its abstract online communication cost as $C^t_{\text{Top2}} = 2C^{t+2}_{\text{AOpen}} + C^t_{\text{LTConst}} + C^{t+2}_{\text{B2A}}$.

The output of $\Pi^{K\to B}_{\text{Top2,3HMal}}$ is the Boolean representation of the two bits needed for the quotient correction. However, the final correction equation is evaluated in the arithmetic-sharing domain. To be more specific, let $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ be the outputs of $\Pi^{K\to B}_{\text{Top2,3HMal}}$, we should first lift them into arithmetic sharings and form

$$[\![q]\!]^K = 2 \cdot [\![b_1]\!]^K + [\![b_0]\!]^K \tag{16}$$

before executing the correction. Therefore, a Boolean-to-arithmetic lifting step is required.

This lifting task can be realized by using a standard daBit-based single-bit B2A conversion. To be more specific, daBits were introduced precisely to support conversions between arithmetic and Boolean sharings, and later mixed arithmetic-binary protocols use them as a basic tool. Therefore, we only need to instantiate a version

Algorithm 1 : Honest-majority malicious security top-2-bit extraction protocol $\Pi^{K\to B}_{\text{Top2,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated secret sharing $[\![u]\!]^{2^{t+2}}$ of an input $u \in \mathbb{Z}_{2^{t+2}}$.

Output: Parties P_1, P_2, P_3 obtain replicated Boolean sharings $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ such that $q = \lfloor u/2^t \rfloor = 2b_1 + b_0 \in \{0, 1, 2, 3\}$.

Preprocessing: Parties P_1, P_2, P_3 hold two sets of 3-party maliciously secure strict edaBits:

- $\left([\![r]\!]^{2^{t+2}}, ([\![r_i]\!]^B)_{i=0}^{t-1}\right)$ with $r = \sum_{i=0}^{t-1} 2^i r_i$ and $0 \le r < 2^t$;
- $\left([\![\rho]\!]^{2^{t+2}}, ([\![\rho_i]\!]^B)_{i=0}^{1}\right)$ with $\rho = \rho_0 + 2\rho_1$ and $0 \le \rho < 4$.

1: Parties P_1, P_2, P_3 locally compute $[\![\tau]\!]^{2^{t+2}} \leftarrow [\![u]\!]^{2^{t+2}} + [\![r]\!]^{2^{t+2}}$ and $[\![c]\!]^{2^{t+2}} \leftarrow 2^2 \cdot [\![\tau]\!]^{2^{t+2}}$.

2: Parties P_1, P_2, P_3 perform an audited opening of c. If inconsistent shares are detected, abort; otherwise set the public value $\bar{c} \leftarrow c/2^2$.

3: Parties P_1, P_2, P_3 invoke $\Pi^{t}_{\text{LTConst,3HMal}}$ on input $([\![r_i]\!]^B)_{i=0}^{t-1}$ and public value $\bar{c}$ to obtain $[\![v]\!]^B$ such that $v = \mathbf{1}\{\bar{c} < r\}$.

4: Parties P_1, P_2, P_3 invoke $\Pi^{t+2}_{\text{B2A,3HMal}}$ on input $[\![v]\!]^B$ to obtain $[\![v]\!]^{2^{t+2}}$.

5: Parties P_1, P_2, P_3 locally compute $[\![u_{\text{low}}]\!]^{2^{t+2}} \leftarrow \bar{c} - [\![r]\!]^{2^{t+2}} + 2^t[\![v]\!]^{2^{t+2}}$ and $[\![u_{\text{high}}]\!]^{2^{t+2}} \leftarrow [\![u]\!]^{2^{t+2}} - [\![u_{\text{low}}]\!]^{2^{t+2}}$.

6: Parties P_1, P_2, P_3 locally compute $[\![d]\!]^{2^{t+2}} \leftarrow [\![u_{\text{high}}]\!]^{2^{t+2}} + 2^t[\![\rho]\!]^{2^{t+2}}$.

7: Parties P_1, P_2, P_3 perform an audited opening of d. If inconsistent shares are detected, abort; otherwise compute the public value $\delta \leftarrow d/2^t \in \{0, 1, 2, 3\}$ and publicly decompose it as $\delta = \delta_0 + 2\delta_1$, where $\delta_0, \delta_1 \in \{0, 1\}$.

8: Parties P_1, P_2, P_3 locally compute $[\![b_0]\!]^B \leftarrow \delta_0 \oplus [\![\rho_0]\!]^B$ and $[\![\beta]\!]^B \leftarrow (1-\delta_0) \cdot [\![\rho_0]\!]^B$.

9: Parties P_1, P_2, P_3 locally compute $[\![b_1]\!]^B \leftarrow \delta_1 \oplus [\![\rho_1]\!]^B \oplus [\![\beta]\!]^B$.

10: Parties P_1, P_2, P_3 output $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$.

that is consistent with our replicated-3PC notation and with the subsequent correction equation. Based on this, we instantiate the protocol $\Pi^{B\to K}_{B2A,3HMal}$ for single-bit lifting and the details are provided in Algorithm 2.

Correctness. The correctness of $\Pi^{B\to K}_{\text{B2A,3HMal}}$ is guaranteed by the daBit masking structure. Steps 1–2 reveal $c = b \oplus r$, where r is the random daBit provided in preprocessing. Since c is public and r is shared in both the Boolean and arithmetic domains, Step 3 reconstructs $[\![b]\!]^K$ from $[\![r]\!]^K$ by a local affine transformation determined by the public bit c. If $c = 0$, the output is $[\![r]\!]^K$, and if $c = 1$, the output is $[\![1 - r]\!]^K$.

Online complexity. The online phase of $\Pi^{B\to K}_{\text{B2A,3HMal}}$ involves only a single audited opening in the Boolean domain, and all remaining operations are local. Accordingly, we write its abstract online communication cost as $C^K_{\text{B2A}} = C^B_{\text{AOpen}}$.

Importantly, the two protocols described above should not be viewed as disjoint contributions. Specifically, Top2 is a specialization of an established general logical right-shift mechanism, whereas the single-bit B2A protocol constitutes a direct replicated-3PC instantiation of the standard daBit-based conversion. Crucially, what matters is not the individual presentation of these subprotocols, but

Algorithm 2 : Honest-majority malicious security single-bit Boolean-to-arithmetic share conversion protocol $\Pi^{B\to K}_{\text{B2A,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated Boolean sharing $[\![b]\!]^B$ of a single bit $b \in \mathbb{Z}_2$.

Output: Parties P_1, P_2, P_3 obtain a replicated arithmetic sharing $[\![b]\!]^K$ of the same single bit b over the ring $\mathbb{Z}_{2^k}$, where $K = 2^k$.

Preprocessing: Parties P_1, P_2, P_3 hold a random daBit generated with 3-party malicious security:

- ▶ a replicated Boolean sharing $[\![r]\!]^B$ such that $r = [\![r]\!]^B_1 \oplus [\![r]\!]^B_2 \oplus [\![r]\!]^B_3$;
- ▶ a replicated arithmetic sharing $[\![r]\!]^K$ such that $r = [\![r]\!]^K_1 + [\![r]\!]^K_2 + [\![r]\!]^K_3 \pmod{2^k}$.

1: For $i \in \{1, 2, 3\}$, party P_i locally computes $[\![c]\!]^B_i \leftarrow [\![b]\!]^B_i \oplus [\![r]\!]^B_i$.
2: Parties P_1, P_2, P_3 maliciously perform an audited opening of the bit value c. If inconsistent shares are detected, abort.
3: **if** $c = 0$ **then**
4: Party P_1 outputs $([\![r]\!]^K_1, [\![r]\!]^K_2)$, party P_2 outputs $([\![r]\!]^K_2, [\![r]\!]^K_3)$, and party P_3 outputs $([\![r]\!]^K_3, [\![r]\!]^K_1)$.
5: **else**
6: Party P_1 outputs $(1 - [\![r]\!]^K_1 \pmod{2^k}, -[\![r]\!]^K_2 \pmod{2^k})$, Party P_2 outputs $(-[\![r]\!]^K_2 \pmod{2^k}, -[\![r]\!]^K_3 \pmod{2^k})$, Party P_3 outputs $(-[\![r]\!]^K_3 \pmod{2^k}, 1-[\![r]\!]^K_1 \pmod{2^k})$.
7: **end if**

rather that—collectively—they implement the same bounded-quotient interface. Consequently, bounded-quotient extraction and correction are elevated from an ad hoc collection of auxiliary subroutines to a unified malicious-security realization layer. Once this layer is instantiated, the remaining task reduces solely to specifying the tuple $(\tilde{y}, u, C, t)$ for each concrete functionality.

5.3 A generic quotient-correction framework

We now consider the structured subclass identified above. For arbitrary functionality lying in this class, the participants can locally derive, from the secret-shared input, a replicated sharing $[\![\tilde{y}]\!]^K$ of a local base value $\tilde{y}$ over the target ring $\mathbb{Z}_{2^k}$, together with a replicated sharing $[\![u]\!]^{2^{t+2}}$ of a functionality-specific auxiliary quantity u over a power-of-two ring $\mathbb{Z}_{2^{t+2}}$. There is also a public constant $C \in \mathbb{Z}_{2^k}$ and a parameter t. These quantities satisfy

$$y_{ex} = \tilde{y} + qC, \qquad q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0, 1, 2, 3\}. \tag{17}$$

Once such an instantiation is fixed, the output computation follows a uniform quotient-correction path. The concrete functionality determines only how the tuple $(\tilde{y}, u, C, t)$ is instantiated, while the bounded-quotient extraction and its transformation into the target ring remain the same.

Under such an instantiation, the framework begins with a local construction of the base value $\tilde{y}$ and the relevant auxiliary quantity u from the input, together with determination of the public constant C and the parameter t. This step introduces no additional nonlinear operation or communication between parties, and serves only to place the concrete functionality into the form of quotient-correction. Then, the parties invoke the protocol $\Pi^{K\to B}_{Top2,3HMal}$ on $[\![u]\!]^{2^{t+2}}$ and obtain the replicated Boolean sharings $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ of the bounded quotient bits, satisfying $q = 2b_1 + b_0$. After that, the parties invoke the protocol $\Pi^{B\to K}_{B2A,3HMal}$ on these two bits separately, lifting them into arithmetic sharings over the target ring to obtain $[\![b_1]\!]^K$ and $[\![b_0]\!]^K$, and locally reconstruct the replicated sharing of bounded quotient q as

$$[\![q]\!]^K = 2 \cdot [\![b_1]\!]^K + [\![b_0]\!]^K. \tag{18}$$

Finally, the parties perform a linear correction

$$[\![y_{ex}]\!]^K = [\![\tilde{y}]\!]^K + C \cdot [\![q]\!]^K, \tag{19}$$

thereby obtaining the replicated sharing of the target functionality's output.

In other words, once the local instantiation layer has been completed, the remaining protocol path is fully fixed. Different functionalities vary only in the concrete construction of $\tilde{y}$, u, C, and t, while quotient recovery, lifting, and correction proceed in the same order. Therefore, we can form a unified framework for such functionalities, as shown in Algorithm 3.

Correctness. The correctness of $\Pi^{K,t}_{\mathrm{QC,3HMal}}$ is guaranteed directly by the quotient-correction instantiation given in Step 1 and the interface guarantees of the two subprotocols. In Step 1, the parties locally construct $[\![\tilde{y}]\!]^K$ and $[\![u]\!]^{2^{t+2}}$ from the input, and determine the public constant C with the parameter t, such that $q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0,1,2,3\}$, $y_{ex} = \tilde{y} + qC$. Step 2 then invokes $\Pi^{2^{t+2}\to B}_{\mathrm{Top2,3HMal}}$ to recover Boolean sharings $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ satisfying $q = 2b_1 + b_0$. Steps 3–4 lift these two bits into the arithmetic domain through two invocations of $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$. Hence, Step 5 locally reconstructs $[\![q]\!]^K = [\![b_0]\!]^K + 2\cdot[\![b_1]\!]^K$, and Step 6 further recovers $[\![y_{ex}]\!]^K = [\![\tilde{y}]\!]^K + C\cdot[\![q]\!]^K$. Therefore, the protocol outputs the correct replicated secret sharing of the target functionality result.

Online complexity. The online phase of $\Pi^{K,t}_{\mathrm{QC,3HMal}}$ involves the Top2 protocol $\Pi^{2^{t+2}\to B}_{\mathrm{Top2,3HMal}}$ once and the Boolean-to-arithmetic share conversion protocol $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$ twice. Accordingly, its abstract online communication cost can be written as $C^{K,t}_{\mathrm{QC}} = C^t_{\mathrm{Top2}} + 2C^k_{\mathrm{B2A}}$.

Under this framework, the functionality-independent nonlinear processing has been completely determined. The design of each concrete protocol therefore lies not in reorganizing the nonlinear steps, but only in constructing a tuple $(\tilde{y}, u, C, t)$ from the concrete input, satisfying the quotient-correction form.

Algorithm 3 : Generic quotient-correction protocol template $\Pi^{K,t}_{\mathrm{QC,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold replicated secret shares of the input of a target functionality that admits a quotient-correction instantiation.

Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y_{ex}]\!]^K$ of the target functionality output over the target ring $\mathbb{Z}_{2^k}$.

Preprocessing: The template itself requires no additional preprocessing; the preprocessing required by $\Pi^{2^{t+2}\to B}_{\mathrm{Top2,3HMal}}$ and $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$ is specified in the corresponding subprotocols.

1: According to the concrete functionality, parties P_1, P_2, P_3 locally construct a replicated secret sharing $[\![\tilde{y}]\!]^K$ of the local base value and a replicated secret sharing $[\![u]\!]^{2^{t+2}}$ of the auxiliary quantity from the shared input, and simultaneously determine the public constant $C \in \mathbb{Z}_K$ and the parameter t, such that $q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0, 1, 2, 3\}, y_{ex} = \tilde{y} + qC$.

2: Parties P_1, P_2, P_3 invoke the honest-majority maliciously secure dedicated top-two-bit extraction protocol $\Pi^{2^{t+2}\to B}_{\mathrm{Top2,3HMal}}$ on input $[\![u]\!]^{2^{t+2}}$, obtaining $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$, satisfying $q = 2b_1 + b_0 = \left\lfloor \frac{u}{2^t} \right\rfloor$.

3: Parties P_1, P_2, P_3 invoke the honest-majority maliciously secure single-bit Boolean-to-arithmetic share conversion protocol $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$ on input $[\![b_1]\!]^B$, obtaining $[\![b_1]\!]^K$.

4: Parties P_1, P_2, P_3 invoke the honest-majority maliciously secure single-bit Boolean-to-arithmetic share conversion protocol $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$ on input $[\![b_0]\!]^B$, obtaining $[\![b_0]\!]^K$.

5: Parties P_1, P_2, P_3 locally compute $[\![q]\!]^K = [\![b_0]\!]^K + 2 \cdot [\![b_1]\!]^K$.

6: Parties P_1, P_2, P_3 locally compute $[\![y_{ex}]\!]^K = [\![\tilde{y}]\!]^K + C \cdot [\![q]\!]^K$.

7: Parties P_1, P_2, P_3 output $[\![y_{ex}]\!]^K$.

6 Cross-ring exact fixed-point multiplication

With the functionality-independent nonlinear quotient-recovery path established, the remaining task is to instantiate the quotient-correction form for the concrete cross-ring functionalities surrounding multiplication. The cross-ring functionalities needed for fixed-point multiplication, namely exact signed truncation and signed extension, both fit this quotient-correction form. In such cases, linear operations determine the corresponding base value and auxiliary quantity, while quotient recovery, lifting, and linear correction proceed through the same maliciously secure path specified by the framework. Furthermore, instantiating these two exact functionalities enables exact fixed-point multiplication via the sequential composition of signed extension, larger-ring multiplication, and exact signed truncation. Additionally, for practical scenarios where efficiency is more important than security or precision guarantees, we also provide several efficient trade-off variants.

6.1 Exact signed truncation

For exact signed truncation, let the input x be over the ring $\mathbb{Z}_L$, and let the output y be over the ring $\mathbb{Z}_{2^{l-k}}$, such that $\mathrm{int}_{l-k}(y) = \lfloor \mathrm{int}_l(x)/2^k \rfloor$. At the share level, local truncation determines the linear main term of the output, and the remaining deviation arises solely from the carries generated when the discarded low bits are summed across the shares. As a result, the exact truncation differs from the local truncation only by a small correction coefficient, rather than by an unrestrained nonlinear term. The following lemma gives this characterization.

Lemma 6.1 (Quotient-correction form of exact signed truncation): *Let $L = 2^l$, $x \in \mathbb{Z}_{2^l}$ with standard representative $\bar{x} \in \{0, 1, \ldots, L-1\}$, and let $x_1, x_2, x_3 \in \{0, 1, \ldots, L-1\}$ be the underlying additive shares satisfying $x \equiv x_1 + x_2 + x_3 \pmod{L}$. For a truncation by k bits, write $x_i = 2^k q_i + r_i$, $0 \le r_i < 2^k$, $i \in \{1, 2, 3\}$, and define $\tilde{y} := q_1 + q_2 + q_3 \pmod{2^{l-k}}$. If $y \in \mathbb{Z}_{2^{l-k}}$ denotes the exact signed truncation output, then*

$$y = \tilde{y} + q_{\mathrm{tr}} \pmod{2^{l-k}}, \qquad q_{\mathrm{tr}} = \left\lfloor \frac{r_1 + r_2 + r_3}{2^k} \right\rfloor \in \{0, 1, 2\}. \tag{20}$$

Proof. Define $R := r_1 + r_2 + r_3 = q_{\mathrm{tr}} \cdot 2^k + r_{\mathrm{low}}$ with $0 \le r_{\mathrm{low}} < 2^k$. Since $0 \le R < 3 \cdot 2^k$, the quotient $q_{\mathrm{tr}} = \lfloor R/2^k \rfloor$ indeed lies in $\{0, 1, 2\}$. From the equation $x_i = 2^k q_i + r_i$ and $R = r_1 + r_2 + r_3 = q_{\mathrm{tr}} \cdot 2^k + r_{\mathrm{low}}$, it follows that $x_1 + x_2 + x_3 = 2^k(q_1 + q_2 + q_3 + q_{\mathrm{tr}}) + r_{\mathrm{low}}$. On the other hand, if $\bar{y} \in \{0, 1, \ldots, 2^{l-k} - 1\}$ denotes the standard representative of the exact signed truncation output y, then by the definition of arithmetic right shift under two's-complement semantics, $\bar{x}$ can be written uniquely as $\bar{x} = 2^k \bar{y} + r'$ for some $r' \in \{0, 1, \ldots, 2^k - 1\}$. Combining this with $\bar{x} \equiv x_1 + x_2 + x_3 \pmod{L}$ and the expression above yields $r' = r_{\mathrm{low}}$ and $\bar{y} \equiv q_1 + q_2 + q_3 + q_{\mathrm{tr}} \pmod{2^{l-k}}$. Since $\tilde{y} \equiv q_1 + q_2 + q_3 \pmod{2^{l-k}}$, it follows that $y = \tilde{y} + q_{\mathrm{tr}} \pmod{2^{l-k}}$. □

Accordingly, the auxiliary quantity for exact signed truncation in the framework of Chapter 5 is taken directly to be

$$u_{\mathrm{tr}} := r_1 + r_2 + r_3, \tag{21}$$

represented over the ring $\mathbb{Z}_{2^{k+2}}$. Since $0 \le u_{\mathrm{tr}} < 3 \cdot 2^k < 2^{k+2}$, the corresponding correction coefficient satisfies $q_{\mathrm{tr}} = \lfloor \frac{u_{\mathrm{tr}}}{2^k} \rfloor$. Hence, the tuple needed for the unified framework is given by

$$(\tilde{y}, u, C, t) = (\tilde{y}, u_{\mathrm{tr}}, 1, k), \tag{22}$$

where $\tilde{y}$ is the base value obtained by local truncation, u_{tr} is the auxiliary quantity, the public correction constant is $C = 1$, and the parameter is $t = k$.

Under this instantiation, the remaining computation follows the common path already fixed in the generic quotient-correction framework. Firstly, the parties locally truncate the shares of the input they own to obtain the base value $[\![\tilde{y}]\!]^{2^{l-k}}$, and naturally embed the local remainders into the ring $\mathbb{Z}_{2^{k+2}}$ to assemble the replicated sharing $[\![u_{\mathrm{tr}}]\!]^{2^{k+2}}$ of the auxiliary quantity. Secondly, the parties invoke the specialized Top2

protocol on $[\![u_{\mathrm{tr}}]\!]^{2^{k+2}}$ to recover the Boolean bits of $q_{\mathrm{tr}} = \lfloor u_{\mathrm{tr}}/2^k \rfloor$; then invoke the B2A protocol on those bits to reconstruct the arithmetic sharing $[\![q_{\mathrm{tr}}]\!]^{2^{l-k}}$. Finally, the exact signed truncated output is obtained through the linear correction

$$[\![y]\!]^{2^{l-k}} = [\![\tilde{y}]\!]^{2^{l-k}} + [\![q_{\mathrm{tr}}]\!]^{2^{l-k}}, \tag{23}$$

and the details are provided in Algorithm 4.

Algorithm 4 : Honest-majority malicious security exact signed truncation protocol $\Pi^{L,k}_{\mathrm{Trunc,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated secret sharing $[\![x]\!]^L$ of an input $x \in \mathbb{Z}_L$, where $L = 2^l$.

Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y]\!]^{2^{l-k}}$ such that $\mathrm{int}_{l-k}(y) = \lfloor \mathrm{int}_l(x)/2^k \rfloor$.

Preprocessing: The main protocol itself requires no additional preprocessing; the preprocessing required by $\Pi^{2^{k+2}\to B}_{\mathrm{Top2,3HMal}}$ and $\Pi^{B\to 2^{l-k}}_{\mathrm{B2A,3HMal}}$ is specified in the corresponding subprotocols.

1: For $i \in \{1,2,3\}$, party P_i locally computes $[\![\tilde{y}]\!]_i^{2^{l-k}} \leftarrow \lfloor [\![x]\!]_i^L / 2^k \rfloor$, $[\![\tilde{y}]\!]^{2^{l-k}}_{(i \bmod 3)+1} \leftarrow \lfloor [\![x]\!]^L_{(i \bmod 3)+1}/2^k \rfloor$, $r_i \leftarrow [\![x]\!]_i^L \pmod{2^k}$, and $r_{(i \bmod 3)+1} \leftarrow [\![x]\!]^L_{(i \bmod 3)+1} \pmod{2^k}$.

2: For $i \in \{1,2,3\}$, party P_i naturally embeds r_i and $r_{(i \bmod 3)+1}$ into the ring $\mathbb{Z}_{2^{k+2}}$, and sets $[\![u_{\mathrm{tr}}]\!]_i^{2^{k+2}} \leftarrow r_i$ and $[\![u_{\mathrm{tr}}]\!]^{2^{k+2}}_{(i \bmod 3)+1} \leftarrow r_{(i \bmod 3)+1}$.

3: Parties P_1, P_2, P_3 invoke $\Pi^{2^{k+2}\to B}_{\mathrm{Top2,3HMal}}$ on input $[\![u_{\mathrm{tr}}]\!]^{2^{k+2}}$, obtaining $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$.

4: Parties P_1, P_2, P_3 invoke $\Pi^{B\to 2^{l-k}}_{\mathrm{B2A,3HMal}}$ on input $[\![b_1]\!]^B$, obtaining $[\![b_1]\!]^{2^{l-k}}$.

5: Parties P_1, P_2, P_3 invoke $\Pi^{B\to 2^{l-k}}_{\mathrm{B2A,3HMal}}$ on input $[\![b_0]\!]^B$, obtaining $[\![b_0]\!]^{2^{l-k}}$.

6: Parties P_1, P_2, P_3 locally compute $[\![q_{\mathrm{tr}}]\!]^{2^{l-k}} \leftarrow [\![b_0]\!]^{2^{l-k}} + 2 \cdot [\![b_1]\!]^{2^{l-k}}$.

7: Parties P_1, P_2, P_3 locally compute $[\![y]\!]^{2^{l-k}} \leftarrow [\![\tilde{y}]\!]^{2^{l-k}} + [\![q_{\mathrm{tr}}]\!]^{2^{l-k}}$.

8: Parties P_1, P_2, P_3 output $[\![y]\!]^{2^{l-k}}$.

Correctness. The correctness of $\Pi^{L,k}_{\mathrm{Trunc,3HMal}}$ is guaranteed directly by the quotient-correction instantiation above and the interface guarantees of the two subprotocols. In steps 1–2, the parties locally construct $[\![\tilde{y}]\!]^{2^{l-k}}$ and $[\![u_{\mathrm{tr}}]\!]^{2^{k+2}}$ from the input, and determine the public constant $C = 1$ with the parameter $t = k$, such that $q_{\mathrm{tr}} = \left\lfloor \frac{u_{\mathrm{tr}}}{2^k} \right\rfloor \in \{0,1,2\}$ and $y = \tilde{y} + q_{\mathrm{tr}}$. Therefore, the protocol is exactly an instantiation of the quotient-correction framework, and its correctness follows immediately from the correctness of that framework.

Online complexity. The online phase of $\Pi^{L,k}_{\mathrm{Trunc,3HMal}}$ involves the Top2 protocol $\Pi^{2^{k+2}\to B}_{\mathrm{Top2,3HMal}}$ once and the Boolean-to-arithmetic share conversion protocol $\Pi^{B\to 2^{l-k}}_{\mathrm{B2A,3HMal}}$

twice. Accordingly, its abstract online communication cost can be written as $C^{l,k}_{\text{Trunc}} = C^{k+2}_{\text{Top2}} + 2C^{l-k}_{\text{B2A}}$.

6.2 General Signed extension

For signed extension, let the input x be secret-shared over the ring $\mathbb{Z}_M$, and let the output y be secret-shared over the ring $\mathbb{Z}_N$, where $N > M$, such that $\text{int}_N(y) = \text{int}_M(x)$. Formally, if $\bar{x} \in \{0, 1, \ldots, M-1\}$ denotes the standard representative of the input x, then the correct output y of the signed extension must satisfy

$$y = \begin{cases} \bar{x}, & \bar{x} < 2^{m-1}, \\ (\bar{x} - M) \pmod N, & \bar{x} \geq 2^{m-1}. \end{cases} \tag{24}$$

At the share level, the local linear part is already given by the sum of the shares, and the remaining deviation arises solely from the carries generated when the shares are summed and the sign information of the input is considered. The correct signed extension output therefore differs from this linear quantity only by a small nonlinear correction term. The following lemma gives this characterization.

Lemma 6.2 (Quotient-correction form of signed extension): *Let $M = 2^m$ and $N = 2^n$ with $m < n$. Let $x \in \mathbb{Z}_{2^m}$ with standard representative $\bar{x} \in \{0, 1, \ldots, M-1\}$, and let $x_1, x_2, x_3 \in \{0, 1, \ldots, M-1\}$ be the underlying additive shares satisfying $x \equiv x_1 + x_2 + x_3 \pmod M$. Define a linear quantity $S := x_1 + x_2 + x_3$. If $y \in \mathbb{Z}_{2^n}$ denotes the correct signed-extension output, then*

$$y = S - q_{\text{ext}} \cdot M \pmod N, \qquad q_{\text{ext}} = \left\lfloor \frac{S + M/2}{M} \right\rfloor \in \{0, 1, 2, 3\}. \tag{25}$$

Proof. If $\bar{x}$ denotes the standard representative of x, since $x \equiv x_1 + x_2 + x_3 \pmod M$, there exists a unique $w \in \{0, 1, 2\}$ satisfying $S = \bar{x} + w \cdot M$. Then, it follows that $q_{\text{ext}} = \left\lfloor \frac{S+M/2}{M} \right\rfloor = \left\lfloor \frac{\bar{x}+(w+1/2)\cdot M}{M} \right\rfloor = \text{MSB}_m(\bar{x}) + w$, and therefore $q_{\text{ext}} \in \{0, 1, 2, 3\}$. Hence, the following equation always holds: $S - q_{\text{ext}} \cdot M = \bar{x} + w \cdot M - (\text{MSB}_m(\bar{x}) + w)M = \bar{x} - \text{MSB}_m(\bar{x}) \cdot M$. If $\bar{x} < \dfrac{M}{2}$, then $\text{MSB}_m(\bar{x}) = 0$ and therefore $y = S - q_{\text{ext}} \cdot M = \bar{x}$. Relatively, if $\bar{x} \geq \dfrac{M}{2}$, then $\text{MSB}_m(\bar{x}) = 1$ and therefore $y = S - q_{\text{ext}} \cdot M = \bar{x} - M \pmod N$. In both cases, the resulting value has the same two's-complement integer semantics as x, and therefore it is the correct output of the signed extension. □

Accordingly, the auxiliary quantity for signed extension in the framework of Chapter 5 is taken directly to be

$$u_{\text{ext}} := S + M/2, \tag{26}$$

represented over the ring $\mathbb{Z}_{2^{m+2}}$. Since $0 \le S \le 3M-3$, it follows that $0 \le u_{\text{ext}} < 3M + \frac{M}{2} < 4M = 2^{m+2}$, and the corresponding correction coefficient satisfies $q_{\text{ext}} = \left\lfloor \frac{u_{\text{ext}}}{2^m} \right\rfloor$. Hence, the tuple needed for the unified framework is given by

$$(\tilde{y}, u, C, t) = (S \pmod N, u_{\text{ext}}, -M, m), \tag{27}$$

where $\tilde{y} = S \pmod N$ is the base value obtained by naturally reinterpreting the same replicated share tuple over the target ring $\mathbb{Z}_{2^n}$, u_{ext} is the auxiliary quantity, the public correction constant is $C = -M \pmod N$, and the parameter is $t = m$.

Under this instantiation, the remainder follows the common path already fixed in the generic quotient-correction framework. Firstly, each party directly treats its share of the input as an element of the ring $\mathbb{Z}_{2^{m+2}}$, thereby constructing the replicated sharing $[\![S]\!]^{2^{m+2}}$ of the linear quantity S, and then obtains the auxiliary quantity $[\![u_{\text{ext}}]\!]^{2^{m+2}} = [\![S]\!]^{2^{m+2}} + M/2$. Secondly, the parties invoke the specialized Top2 protocol on $[\![u_{\text{ext}}]\!]^{2^{m+2}}$ to recover the Boolean bits of $q_{\text{ext}} = \lfloor u_{\text{ext}}/2^m \rfloor$, and invoke the B2A protocol on these bits to reconstruct the arithmetic sharing $[\![q_{\text{ext}}]\!]^N$. Finally, the parties naturally reinterpret the same share tuple over the ring $\mathbb{Z}_N$ to form the base value $[\![\tilde{y}]\!]^N$, corresponding to $S \pmod N$, and perform the linear correction

$$[\![y]\!]^N = [\![\tilde{y}]\!]^N - [\![q_{\text{ext}}]\!]^N \cdot M \tag{28}$$

to obtain the correct signed extension output. The details are provided in Algorithm 5.

Correctness. The correctness of $\Pi^{M,N}_{\text{SExt,3HMal}}$ is guaranteed directly by the quotient-correction instantiation above and the interface guarantees of the two subprotocols. In Steps 1–3, the parties locally construct $[\![\tilde{y}]\!]^N$ and $[\![u_{\text{ext}}]\!]^{2^{m+2}}$ from the input, and determine the public constant $C = -M \pmod N$ with the parameter $t = m$, such that $q_{\text{ext}} = \left\lfloor \frac{u_{\text{ext}}}{M} \right\rfloor \in \{0,1,2,3\}$ and $y = \tilde{y} + q_{\text{ext}} \cdot C$. Therefore, the protocol is exactly an instantiation of the quotient-correction framework, and its correctness follows immediately from the correctness of that framework.

Online complexity. The online phase of $\Pi^{M,N}_{\text{SExt,3HMal}}$ involves the Top2 protocol $\Pi^{2^{m+2}\to B}_{\text{Top2,3HMal}}$ once and the Boolean-to-arithmetic share conversion protocol $\Pi^{B\to N}_{\text{B2A,3HMal}}$ twice. Accordingly, its abstract online communication cost can be written as $C^{m,n}_{\text{SExt}} = C^{m+2}_{\text{Top2}} + 2C^{n}_{\text{B2A}}$.

6.3 Exact fixed-point multiplication

With the implementation of exact signed truncation and signed extension in the previous two subsections, all cross-ring operations required for fixed-point multiplication are now available. Assume that the two inputs x and x' are encoded over the rings $\mathbb{Z}_{M_1}$ and $\mathbb{Z}_{M_2}$, where $M_1 = 2^{m_1}$ and $M_2 = 2^{m_2}$, and that their fractional precisions are f_1 and f_2 bits, respectively. Let $P := 2^{m_1+m_2}$ be the target ring used to contain the exact product, and let τ be the truncation parameter used in the final step, where $0 \le \tau \le f_1 + f_2$. Then, the output y for fixed-point multiplication is defined over the

Algorithm 5 : Honest-majority malicious security signed extension protocol $\Pi^{M,N}_{\mathrm{SExt,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated secret sharing $[\![x]\!]^M$ of an input $x \in \mathbb{Z}_M$, where $M = 2^m$.

Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y]\!]^N$ such that $\mathrm{int}_n(y) = \mathrm{int}_m(x)$, where $N = 2^n$.

Preprocessing: The main protocol itself requires no additional preprocessing; the preprocessing required by $\Pi^{2^{m+2}\to B}_{\mathrm{Top2,3HMal}}$ and $\Pi^{B\to N}_{\mathrm{B2A,3HMal}}$ is specified in the corresponding subprotocols.

1: Publicly set $H \leftarrow 2^{m+2}$ and $\beta \leftarrow 2^{m-1}$.
2: For $i \in \{1,2,3\}$, party P_i naturally interprets $[\![x]\!]^M_i$ and $[\![x]\!]^M_{(i \bmod 3)+1}$ as elements of $\mathbb{Z}_H$, and sets $[\![S]\!]^H_i \leftarrow [\![x]\!]^M_i$ and $[\![S]\!]^H_{(i \bmod 3)+1} \leftarrow [\![x]\!]^M_{(i \bmod 3)+1}$.
3: Parties P_1, P_2, P_3 locally compute $[\![u_{\mathrm{ext}}]\!]^H \leftarrow [\![S]\!]^H + \beta$.
4: Parties P_1, P_2, P_3 invoke $\Pi^{2^{m+2}\to B}_{\mathrm{Top2,3HMal}}$ on input $[\![u_{\mathrm{ext}}]\!]^H$, obtaining $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$.
5: Parties P_1, P_2, P_3 invoke $\Pi^{B\to N}_{\mathrm{B2A,3HMal}}$ on input $[\![b_1]\!]^B$, obtaining $[\![b_1]\!]^N$.
6: Parties P_1, P_2, P_3 invoke $\Pi^{B\to N}_{\mathrm{B2A,3HMal}}$ on input $[\![b_0]\!]^B$, obtaining $[\![b_0]\!]^N$.
7: Parties P_1, P_2, P_3 locally compute $[\![q_{\mathrm{ext}}]\!]^N \leftarrow [\![b_0]\!]^N + 2 \cdot [\![b_1]\!]^N$.
8: For $i \in \{1,2,3\}$, party P_i naturally interprets $[\![x]\!]^M_i$ and $[\![x]\!]^M_{(i \bmod 3)+1}$ as elements of $\mathbb{Z}_N$, and sets $[\![\tilde{y}]\!]^N_i \leftarrow [\![x]\!]^M_i$ and $[\![\tilde{y}]\!]^N_{(i \bmod 3)+1} \leftarrow [\![x]\!]^M_{(i \bmod 3)+1}$.
9: Parties P_1, P_2, P_3 locally compute $[\![y]\!]^N \leftarrow [\![\tilde{y}]\!]^N - [\![q_{\mathrm{ext}}]\!]^N \cdot M$.
10: Parties P_1, P_2, P_3 output $[\![y]\!]^N$.

ring $\mathbb{Z}_{2^{m_1+m_2-\tau}}$ and is required to satisfy

$$\mathrm{fix}_{m_1+m_2-\tau,\, f_1+f_2-\tau}(y) = \mathrm{fix}_{m_1,f_1}(x) \cdot \mathrm{fix}_{m_2,f_2}(x'). \tag{29}$$

In concrete applications, the choice of τ is determined by the desired fractional precision of the output. If the goal is to obtain the exact product with fractional precision $f_1 + f_2$, then one sets $\tau = 0$, in which case exact signed truncation does not need to be executed. If the goal is to keep the same fractional precision as the inputs, then in the common case $f_1 = f_2 = f$, one sets $\tau = f$.

The functionality can be realized by the following sequential composition directly. First, the parties invoke the signed extension protocol to lift both inputs into the target ring $\mathbb{Z}_P$. Second, the parties perform arithmetic multiplication over $\mathbb{Z}_P$. Finally, according to the desired output precision, the product is subjected to exact signed truncation by τ bits to obtain the result in the ring $\mathbb{Z}_{2^{m_1+m_2-\tau}}$. No additional nonlinear operations are needed for the composition above, and the details are provided in Algorithm 6.

Correctness. The correctness of $\Pi^{M_1,M_2,\tau}_{\mathrm{FMul,3HMal}}$ is guaranteed directly by the correctness of $\Pi^{M_1,P}_{\mathrm{SExt,3HMal}}$, $\Pi^{M_2,P}_{\mathrm{SExt,3HMal}}$, the arithmetic multiplication protocol $\Pi^{P}_{\mathrm{Mul,3HMal}}$, and $\Pi^{P,\tau}_{\mathrm{Trunc,3HMal}}$. Since these protocols are implemented correctly, the output of

Algorithm 6 : Honest-majority malicious security exact fixed-point multiplication protocol $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold replicated secret sharings $[\![x]\!]^{M_1}$ and $[\![x']\!]^{M_2}$ of two inputs x and x' with fractional precisions f_1 and f_2, respectively, where $M_1 = 2^{m_1}$ and $M_2 = 2^{m_2}$.

Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y]\!]^{2^{m_1+m_2-\tau}}$ of an output y, where $0 \le \tau \le f_1 + f_2$, whose fixed-point semantics corresponds to the exact multiplication result after truncation by τ bits.

Preprocessing: The main protocol itself requires no additional preprocessing; the preprocessing required by $\Pi^{M_1,P}_{\text{SExt,3HMal}}$, $\Pi^{M_2,P}_{\text{SExt,3HMal}}$, $\Pi^{P}_{\text{Mul,3HMal}}$, and $\Pi^{P,\tau}_{\text{Trunc,3HMal}}$ is specified in the corresponding subprotocols.

1: Publicly set $P \leftarrow 2^{m_1+m_2}$.
2: Parties P_1, P_2, P_3 invoke $\Pi^{M_1,P}_{\text{SExt,3HMal}}$ on input $[\![x]\!]^{M_1}$, obtaining $[\![x]\!]^{P}$.
3: Parties P_1, P_2, P_3 invoke $\Pi^{M_2,P}_{\text{SExt,3HMal}}$ on input $[\![x']\!]^{M_2}$, obtaining $[\![x']\!]^{P}$.
4: Parties P_1, P_2, P_3 invoke $\Pi^{P}_{\text{Mul,3HMal}}$ on inputs $[\![x]\!]^{P}$ and $[\![x']\!]^{P}$, obtaining a replicated sharing $[\![p]\!]^{P}$ of the product.
5: Parties P_1, P_2, P_3 invoke $\Pi^{P,\tau}_{\text{Trunc,3HMal}}$ on input $[\![p]\!]^{P}$, obtaining $[\![y]\!]^{2^{m_1+m_2-\tau}}$.
6: Parties P_1, P_2, P_3 output $[\![y]\!]^{2^{m_1+m_2-\tau}}$.

$\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$ is the correct replicated secret sharing of a value y that satisfies $\text{fix}_{m_1+m_2-\tau,\, f_1+f_2-\tau}(y) = \text{fix}_{m_1,f_1}(x) \cdot \text{fix}_{m_2,f_2}(x')$.

Online complexity. The online phase of $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$ invokes the signed extension protocol twice, namely $\Pi^{M_1,P}_{\text{SExt,3HMal}}$ and $\Pi^{M_2,P}_{\text{SExt,3HMal}}$, the arithmetic multiplication protocol $\Pi^{P}_{\text{Mul,3HMal}}$ once, and the exact signed truncation protocol $\Pi^{P,\tau}_{\text{Trunc,3HMal}}$ once. Accordingly, its abstract online communication cost can be written as $C^{m_1,m_2,\tau}_{\text{FMul}} = C^{m_1,m_1+m_2}_{\text{SExt}} + C^{m_2,m_1+m_2}_{\text{SExt}} + C^{m_1+m_2}_{\text{Mul}} + C^{m_1+m_2,\tau}_{\text{Trunc}}$. Specifically, when the bit widths of the two inputs are the same and both are equal to m, the above abstract overhead simplifies to $C^{m,m,\tau}_{\text{FMul}} = 2C^{m,2m}_{\text{SExt}} + C^{2m}_{\text{Mul}} + C^{2m,\tau}_{\text{Trunc}}$. More specifically, when an exact output is required (i.e., $\tau = 0$), the abstract overhead further simplifies to $C^{m,m,0}_{\text{FMul}} = 2C^{m,2m}_{\text{SExt}} + C^{2m}_{\text{Mul}}$.

6.4 Variants under relaxed guarantees

The protocols instantiated under the quotient-correction framework above achieve exact semantics under honest-majority malicious security but incur substantially higher online communication and computation costs. In many practical deployment scenarios, efficiency takes precedence over exact semantics or strongest-security guarantees. This motivates relaxing security requirements, precision guarantees, or applicability conditions to yield more efficient protocol designs. Accordingly, we provide in turn variants that relax security or precision.

For signed extension, a lightweight variant may be employed when efficiency is prioritized, semi-honest security is sufficient, and the input satisfies the bound $|x| < \frac{M}{3}$.

Building on the efficient method for signed extension in the two-party setting over this restricted domain proposed by Guo et al.[9], we adapt the ABY3[26] technique of temporarily collapsing replicated sharing into a two-party additive form before applying Guo's method. However, constrained to two participants, Guo's scheme implements AND via Correlated Oblivious Transfer (COT), rendering it relatively expensive. In contrast, our setting natively provides a third party. Therefore, we modify their method by involving the third party in the AND computation, which yields a lighter three-party variant. The modified AND protocol $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ is given in Appendix B.1, and the modified MW protocol $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ is given in Appendix B.2. The details of the resulting protocol are provided in Algorithm 7.

Algorithm 7 : Three-party semi-honest signed extension protocol $\Pi^{M,N}_{\mathrm{SExt,3Semi}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated secret sharing $[\![x]\!]^M$ of an input $x \in \mathbb{Z}_M$, where $M = 2^m$, and the input satisfies $|x| < M/3$.

Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y]\!]^N$ such that $\mathrm{int}_N(y) = \mathrm{int}_M(x)$, where $N = 2^n > M$.

Preprocessing: The main protocol itself requires that parties P_2 and P_3 hold a common random value $r \in \mathbb{Z}_N$; the preprocessing required by $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ is specified in the corresponding subprotocol.

1: Party P_1 locally sets $[x]^M_1 \leftarrow [\![x]\!]^M_1$.
2: Party P_2 locally sets $[x]^M_2 \leftarrow [\![x]\!]^M_2 + [\![x]\!]^M_3 \pmod M$.
3: Parties P_1, P_2 invoke $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ with the support of P_3 on input $[x]^M$, obtaining $[MW]^{2^{n-m}}$.
4: Party P_1 locally computes $y_1 \leftarrow [x]^M_1 + N - [MW]^{2^{n-m}}_1 \cdot M$, and sends y_1 to P_3.
5: Party P_2 locally computes $y_2 \leftarrow [x]^M_2 - [MW]^{2^{n-m}}_2 \cdot M$, then sets $y'_2 \leftarrow (y_2 - r) \pmod N$, and sends y'_2 to P_1.
6: Party P_1 outputs (y_1, y'_2), party P_2 outputs (y'_2, r), and party P_3 outputs (r, y_1).

Correctness. The correctness of $\Pi^{M,N}_{\mathrm{SExt,3Semi}}$ is guaranteed directly by the correctness of $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ and the final resharing step. In Steps 1–3, the parties locally transform the replicated secret sharing $[\![x]\!]^M$ into the two-party additive secret sharing $[x]^M$, and recover $[MW]^{2^{n-m}}$ under the restriction $|x| < \frac{M}{3}$. Hence, Steps 4–5 further recover $y = x - MW \cdot M \pmod N$, and reassemble it into the replicated secret sharing $[\![y]\!]^N$. Therefore, the protocol outputs the correct replicated secret sharing of the signed extension result on the restricted domain $|x| < \frac{M}{3}$.

Online complexity. The online phase of $\Pi^{M,N}_{\mathrm{SExt,3Semi}}$ involves the modified MW protocol $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ once and two transmissions of ring elements in $\mathbb{Z}_N$ for the resharing step. Accordingly, its abstract online communication cost can be written as $C^{m,n}_{\mathrm{SExt,3Semi}} = C^{m,\langle 1/3\rangle}_{\mathrm{MW,3Semi}} + 2n$.

For signed truncation, when a small loss in precision is acceptable, a variant with at most 1 ULP error may be used. The preceding analysis shows that if $\tilde{y}$ denotes the local truncation value and y_{ex} denotes the exact signed truncation value, then it follows that $y_{ex} - \tilde{y} = c$, $c \in \{0, 1, 2\}$. In other words, the local truncation value may differ from the exact value by 0, 1, or 2. Moreover, the probabilities of these three cases under the standard 3PC replicated-sharing distribution are computed in Appendix B.3, and it shows that unless a substantial fraction of the inputs have their low k bits all equal to 1, the case $c = 1$ is statistically the most likely one. Therefore, we choose a fixed correction "+1" and output $y = \tilde{y} + 1$. This approach maximizes the probability of matching the exact result among all fixed-constant corrections while strictly bounding the resulting error to 1 ULP. The details for this variant are provided in Algorithm 8.

Algorithm 8 : Honest-majority malicious security signed truncation protocol with at most 1ULP error $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}(P_1, P_2, P_3)$

Input: Parties P_1, P_2, P_3 hold a replicated secret sharing $[\![x]\!]^L$ of an input $x \in \mathbb{Z}_L$, where $L = 2^l$.
Output: Parties P_1, P_2, P_3 obtain a replicated secret sharing $[\![y]\!]^{2^{l-k}}$ whose value differs from the exact signed truncation result by at most 1 ULP.
Preprocessing: The protocol itself requires no preprocessing.

1: For $i \in \{1, 2, 3\}$, party P_i locally computes $[\![\tilde{y}]\!]^{2^{l-k}}_i \leftarrow \lfloor [\![x]\!]^L_i / 2^k \rfloor$ and $[\![\tilde{y}]\!]^{2^{l-k}}_{(i \bmod 3)+1} \leftarrow \lfloor [\![x]\!]^L_{(i \bmod 3)+1} / 2^k \rfloor$.
2: Parties P_1, P_2, P_3 locally compute $[\![y]\!]^{2^{l-k}} \leftarrow [\![\tilde{y}]\!]^{2^{l-k}} + 1$.
3: Parties P_1, P_2, P_3 output $[\![y]\!]^{2^{l-k}}$.

Correctness. The correctness of $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$ is guaranteed directly by the relation $y_{ex} - \tilde{y} = c$ together with $c \in \{0, 1, 2\}$. The protocol outputs $y = \tilde{y} + 1$. Hence, $y - y_{ex} = 1 - c \in \{-1, 0, 1\}$. Therefore, the output differs from the exact signed truncation result by at most 1 ULP.
Online complexity. The online phase of $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$ only involves local truncation and local constant addition. Accordingly, its online communication cost is 0.

7 UC security proofs

The security of the protocols proposed in this paper is established under the standard UC real/ideal simulation paradigm, and the common notation $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$ is used to denote the indistinguishability between the real-world execution and the ideal-world execution. By the completeness of the trivial adversary, it is sufficient to construct a simulator only for this adversary. At the same time, the previously known protocols invoked as black boxes together with the corresponding preprocessing resources are assumed to be UC-secure in the required models. The

remaining task is therefore to establish the security of the newly introduced subprotocols in the unified framework, as well as of the exact constructions derived from that framework.

Under the conventions above, the security of the exact constructions is captured by the ideal functionalities associated with the unified framework and its instantiations. The quotient-correction framework provides a common security interface in the appropriate hybrid model, while exact signed truncation, signed extension, and exact fixed-point multiplication correspond to concrete instantiations and sequential compositions of that interface. Together with the security of the two newly introduced subprotocols invoked in the framework above, namely $\Pi^{B\to K}_{\mathrm{B2A,3HMal}}$ and $\Pi^{2^{t+2}\to B}_{\mathrm{Top2,3HMal}}$, the security of the fully instantiated exact protocols is completed by the UC composition theorem. The variants in Section 6.4 follow different design routes and satisfy only relaxed guarantees, so their security analyses are formulated separately under the corresponding models.

7.1 UC security of the exact constructions

As described in Section 5.3, the generic quotient-correction framework applies to a class of target functionalities that admit a properly structured quotient-correction form. For any functionality belonging to this class, the parties can locally construct a replicated sharing $[\![\tilde{y}]\!]^K$ of a base value over the target ring and a replicated sharing $[\![u]\!]^{2^{t+2}}$ of an auxiliary quantity, while simultaneously determining a public constant C and a parameter t, and it follows that

$$y_{ex} = \tilde{y} + qC, \qquad q = \left\lfloor \frac{u}{2^t} \right\rfloor \in \{0, 1, 2, 3\}. \tag{30}$$

Accordingly, let $\mathcal{F}^{K,t}_{\mathrm{QC}}$ described in Fig.1 denote the ideal quotient-correction functionality corresponding to this framework. For any target functionality that admits such an instantiation, $\mathcal{F}^{K,t}_{\mathrm{QC}}$ takes as input the replicated arithmetic sharing of the target-functionality input, and outputs a replicated arithmetic sharing $[\![y_{ex}]\!]^K$ of the target-functionality result over the target ring. In the protocol template $\Pi^{K,t}_{\mathrm{QC,3HMal}}$, the local instantiation layer and the final linear correction only involve local computations, and the non-local part is carried out through top-two-bit extraction and Boolean-to-arithmetic share conversion. This yields the hybrid-model security statement in Theorem 7.1.

Theorem 7.1. *The protocol $\Pi^{K,t}_{\mathrm{QC,3HMal}}$ UC-securely realizes the ideal quotient-correction functionality $\mathcal{F}^{K,t}_{\mathrm{QC}}$ in the $(\mathcal{F}^{2^{t+2}\to B}_{\mathrm{Top2}}, \mathcal{F}^{B\to K}_{\mathrm{B2A}})$-hybrid model against static malicious corruption of at most one party.*

Proof. Since the corruption threshold in the honest majority 3PC setting is one, there are four possible corruption patterns, namely no corruption and the three single-party corruptions P_1, P_2, and P_3. Since the three parties play completely symmetric roles in the protocol $\Pi^{K,t}_{\mathrm{QC,3HMal}}$, it is sufficient to consider the case where P_1 is corrupted.

Functionality $\mathcal{F}_{\mathrm{QC}}^{K,t}$

The functionality receives from each party P_i, where $i \in \{1,2,3\}$, its local arithmetic view in the replicated arithmetic sharing of the input, denoted by $(\llbracket x \rrbracket_i, \llbracket x \rrbracket_{(i \bmod 3)+1})$.

- $\mathcal{F}_{\mathrm{QC}}^{K,t}$ computes, according to the concrete target functionality, a replicated arithmetic sharing $\llbracket \tilde{y} \rrbracket^K$ of the local base value over the target ring, denoted by $\llbracket \tilde{y} \rrbracket_i^K$, $i \in \{1,2,3\}$, and a replicated arithmetic sharing $\llbracket u \rrbracket^{2^{t+2}}$ of the auxiliary quantity u over $\mathbb{Z}_{2^{t+2}}$, denoted by $\llbracket u \rrbracket_i^{2^{t+2}}$, $i \in \{1,2,3\}$. It also determines the public constant $C \in \mathbb{Z}_{2^k}$ and the parameter t only according to the type of the concrete target functionality. These quantities implicitly satisfy $y_{ex} = \tilde{y} + qC$, where $q = \lfloor u/2^t \rfloor \in \{0,1,2,3\}$ and y_{ex} is the underlying secret value of the output.
- $\mathcal{F}_{\mathrm{QC}}^{K,t}$ reconstructs u from $\llbracket u \rrbracket_i^{2^{t+2}}$, computes $q = \lfloor u/2^t \rfloor$, and samples a fresh replicated arithmetic sharing $\llbracket q \rrbracket^K$ of q over the target ring $\mathbb{Z}_{2^k}$, denoted by $\llbracket q \rrbracket_i^K$, $i \in \{1,2,3\}$.
- $\mathcal{F}_{\mathrm{QC}}^{K,t}$ computes the replicated arithmetic sharing $\llbracket y_{ex} \rrbracket^K$, denoted by $\llbracket y_{ex} \rrbracket_i^K$, $i \in \{1,2,3\}$, coordinatewise by setting $\llbracket y_{ex} \rrbracket_i^K := \llbracket \tilde{y} \rrbracket_i^K + C \cdot \llbracket q \rrbracket_i^K \pmod{2^k}$ where $i \in \{1,2,3\}$.
- $\mathcal{F}_{\mathrm{QC}}^{K,t}$ outputs to each party P_i the local arithmetic view of $\llbracket y_{ex} \rrbracket^K$, namely $(\llbracket y_{ex} \rrbracket_i^K, \llbracket y_{ex} \rrbracket_{(i \bmod 3)+1}^K)$, where $i \in \{1,2,3\}$.

Fig. 1 The ideal functionality $\mathcal{F}_{\mathrm{QC}}^{K,t}$ of the generic quotient-correction framework.

The cases where P_2 or P_3 is corrupted can be proved in the same way, and the case of no corruption is obvious.

Let S be the simulator in the ideal execution with $\mathcal{F}_{\mathrm{QC}}^{K,t}$. The simulator acts as follows:

1. The environment $\mathcal{Z}$ provides the replicated input share of the corrupted party P_1 to the simulator S. Then, S forwards the replicated input share of P_1 to the ideal functionality $\mathcal{F}_{\mathrm{QC}}^{K,t}$. Once the output of $\mathcal{F}_{\mathrm{QC}}^{K,t}$ is available, S records the local arithmetic view of P_1 in the replicated arithmetic sharing $\llbracket y_{ex} \rrbracket^K$ of the target-functionality output, denoted by $(\llbracket y_{ex} \rrbracket_1^K, \llbracket y_{ex} \rrbracket_2^K)$.
2. The simulator S locally computes the local arithmetic view of P_1 in $\llbracket \tilde{y} \rrbracket^K$ and $\llbracket u \rrbracket^{2^{t+2}}$, denoted by $(\llbracket \tilde{y} \rrbracket_1^K, \llbracket \tilde{y} \rrbracket_2^K)$ and $(\llbracket u \rrbracket_1^{2^{t+2}}, \llbracket u \rrbracket_2^{2^{t+2}})$, respectively, according to $\Pi_{\mathrm{QC,3HMal}}^{K,t}$ Step 1.
3. The simulator S samples fresh local Boolean views of P_1 in the replicated Boolean sharings $\llbracket b_1 \rrbracket^B$ and $\llbracket b_0 \rrbracket^B$, denoted by $(\llbracket b_1 \rrbracket_1^B, \llbracket b_1 \rrbracket_2^B)$ and $(\llbracket b_0 \rrbracket_1^B, \llbracket b_0 \rrbracket_2^B)$, respectively. Then S delivers them to the environment $\mathcal{Z}$ as the local outputs that the corrupted party would obtain from $\mathcal{F}_{\mathrm{Top2}}^{2^{t+2} \to B}$ in the hybrid execution.

4. The simulator S computes the set $\mathcal{Q}_1$ of all values a satisfying $[\![y_{ex}]\!]_1^K = [\![\tilde{y}]\!]_1^K + C \cdot a$, and the set $\mathcal{Q}_2$ of all values b satisfying $[\![y_{ex}]\!]_2^K = [\![\tilde{y}]\!]_2^K + C \cdot b$, where the equations are over the target ring. Then, S samples $[\![q]\!]_1^K$ uniformly from $\mathcal{Q}_1$ and independently samples $[\![q]\!]_2^K$ from $\mathcal{Q}_2$, and uses $([\![q]\!]_1^K, [\![q]\!]_2^K)$ as the local arithmetic view of P_1 in $[\![q]\!]^K$.
5. The simulator S randomly and uniformly samples r_1 and r_2 from $\mathbb{Z}_{2^k}$, and set $([\![b_1]\!]_1^K, [\![b_1]\!]_2^K) := (r_1, r_2)$ as the local arithmetic view of P_1 in $[\![b_1]\!]^K$. Then S computes and sets $([\![b_0]\!]_1^K, [\![b_0]\!]_2^K) := ([\![q]\!]_1^K - 2r_1, [\![q]\!]_2^K - 2r_2)$ as the local arithmetic view of P_1 in $[\![b_0]\!]^K$.
6. The simulator S delivers to the environment, in the same order as in the hybrid execution, the local arithmetic view of P_1 in $[\![b_1]\!]^K$ from the first invocation of $\mathcal{F}_{\mathrm{B2A}}^{B \to K}$, the local arithmetic view of P_1 in $[\![b_0]\!]^K$ from the second invocation of $\mathcal{F}_{\mathrm{B2A}}^{B \to K}$, and finally the local output view of P_1 in $[\![y_{ex}]\!]^K$.

The above simulator S works with the ideal functionality $\mathcal{F}_{\mathrm{Top2}}^{2^{t+2} \to B}$ and $\mathcal{F}_{\mathrm{B2A}}^{B \to K}$, defined below, whose outputs are specified as fresh replicated sharings. The simulator locally reconstructs from the corrupted input share the local arithmetic views of $[\![\tilde{y}]\!]^K$ and $[\![u]\!]^{2^{t+2}}$ that the corrupted party should hold, and generates the local views returned by Top2 and the two invocations of B2A with the same distribution as in the hybrid execution. At the same time, the simulator S samples $[\![q]\!]_1^K$ uniformly from $\mathcal{Q}_1$ and independently samples $[\![q]\!]_2^K$ from $\mathcal{Q}_2$, where $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the sets of all values consistent with the two coordinate equations induced by $[\![y_{ex}]\!]^K = [\![\tilde{y}]\!]^K + C \cdot [\![q]\!]^K$. This matches exactly the conditional distribution of the corrupted party's local arithmetic view in $[\![q]\!]^K$ induced by the ideal functionality. Since all remaining steps of $\Pi_{\mathrm{QC,3HMal}}^{K,t}$ are purely local computations, the entire local view seen by the corrupted party in the ideal execution is identically distributed to that in the $(\mathcal{F}_{\mathrm{Top2}}^{2^{t+2} \to B}, \mathcal{F}_{\mathrm{B2A}}^{B \to K})$-hybrid execution. Therefore, for every well-behaved environment $\mathcal{Z}$, it follows that

$$\left| \Pr\left[\mathrm{Exec}\left[\Pi_{\mathrm{QC,3HMal}}^{K,t}(\mathcal{F}_{\mathrm{Top2}}^{2^{t+2} \to B}, \mathcal{F}_{\mathrm{B2A}}^{B \to K}), A_{\mathrm{triv}}, \mathcal{Z} \right] = 1 \right] - \Pr\left[\mathrm{Exec}\left[\mathcal{F}_{\mathrm{QC}}^{K,t}, S, \mathcal{Z} \right] = 1 \right] \right| = 0.$$

□

As shown by the concrete instantiations in Section 6.1 and Section 6.2, exact signed truncation and signed extension are both the direct instantiations of the generic quotient-correction framework. The functionality-specific difference lies only in the construction of the tuple $(\tilde{y}, u, C, t)$, while quotient recovery, lifting, and linear correction follow the same protocol path. Therefore, Theorem 7.1 directly gives rise to Corollary 7.1 and Corollary 7.2, which state the hybrid-model security of these two exact cross-ring functionalities, respectively. Moreover, at an intuitive level, the implementation of exact fixed-point multiplication is merely a sequential composition of two signed extensions, one arithmetic multiplication, and one exact signed truncation. Therefore, once these component protocols UC-securely realize their corresponding ideal functionalities in the required hybrid model, the overall protocol should likewise be UC-secure by the UC composition theorem. The formal statement is given in Corollary 7.3, while the complete proof is deferred to Appendix C.

Corollary 7.1. *The protocol* $\Pi^{L,k}_{\text{Trunc,3HMal}}$ *UC-securely realizes the ideal exact signed truncation functionality* $\mathcal{F}^{L,k}_{\text{Trunc}}$ *in the* $(\mathcal{F}^{2^{k+2}\to B}_{\text{Top2}}, \mathcal{F}^{B\to 2^{l-k}}_{\text{B2A}})$*-hybrid model against static malicious corruption of at most one party.*

Corollary 7.2. *The protocol* $\Pi^{M,N}_{\text{SExt,3HMal}}$ *UC-securely realizes the ideal signed extension functionality* $\mathcal{F}^{M,N}_{\text{SExt}}$ *in the* $(\mathcal{F}^{2^{m+2}\to B}_{\text{Top2}}, \mathcal{F}^{B\to N}_{\text{B2A}})$*-hybrid model against static malicious corruption of at most one party.*

Corollary 7.3. *Let* $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$ *denote the ideal exact fixed-point multiplication functionality and let* $P = 2^{m_1+m_2}$*. Then the protocol* $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$ *UC-securely realizes* $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$ *in the* $(\mathcal{F}^{M_1,P}_{\text{SExt}}, \mathcal{F}^{M_2,P}_{\text{SExt}}, \mathcal{F}^{P}_{\text{Mul}}, \mathcal{F}^{P,\tau}_{\text{Trunc}})$*-hybrid model against static malicious corruption of at most one party.*

The hybrid-model security corollaries of protocol $\Pi^{L,k}_{\text{Trunc,3HMal}}$ and $\Pi^{M,N}_{\text{SExt,3HMal}}$ above reduce the security of exact signed truncation, signed extension and exact fixed-point multiplication to the two subprotocols, namely $\Pi^{B\to K}_{\text{B2A,3HMal}}$ and $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$. Accordingly, let $\mathcal{F}^{B\to K}_{\text{B2A}}$ denote the ideal single-bit Boolean-to-arithmetic share-conversion functionality. For any single bit $b \in \mathbb{Z}_2$, $\mathcal{F}^{B\to K}_{\text{B2A}}$ takes the replicated Boolean sharing of b as the input, and outputs a fresh sampled replicated arithmetic sharing $[\![b]\!]^K$ of the same bit over the target ring $\mathbb{Z}_{2^k}$. In the protocol $\Pi^{B\to K}_{\text{B2A,3HMal}}$, the non-local part only involves the Boolean audited opening of the masked bit $c = b \oplus r$ and the invocation of the random daBit preprocessing resource. This yields the hybrid-model security statement in Theorem 7.2.

Theorem 7.2. *The protocol* $\Pi^{B\to K}_{\text{B2A,3HMal}}$ *UC-securely realizes the ideal single-bit Boolean-to-arithmetic share-conversion functionality* $\mathcal{F}^{B\to K}_{\text{B2A}}$ *in the* $(\mathcal{F}^{B}_{\text{AOpen}}, \mathcal{F}_{\text{daBit}})$*-hybrid model against static malicious corruption of at most one party.*

Proof. Since the corruption threshold in honest majority 3PC setting is one, there are four possible corruption patterns, namely no corruption and the three single-party corruptions P_1, P_2, and P_3. Although the output rules of $\Pi^{B\to K}_{\text{B2A,3HMal}}$ differ slightly across the three parties when $c = 1$, this difference is purely syntactic and does not affect the security proof. It is therefore sufficient to consider the case where P_1 is corrupted. The cases where P_2 or P_3 is corrupted can be proved in the same way, and the case of no corruption is obvious.

Let S be the simulator in the ideal execution with $\mathcal{F}^{B\to K}_{\text{B2A}}$. The simulator acts as follows:

1. The environment $\mathcal{Z}$ provides the replicated input share of the corrupted party P_1 to the simulator S. Then, S forwards the replicated input share of P_1 to the ideal functionality $\mathcal{F}^{B\to K}_{\text{B2A}}$. Once the output of $\mathcal{F}^{B\to K}_{\text{B2A}}$ is available, S records the local arithmetic view of P_1 in the replicated arithmetic sharing $[\![b]\!]^K$ of the target-functionality output, denoted by $([\![b]\!]^K_1, [\![b]\!]^K_2)$.
2. The simulator S samples a fresh local Boolean view of P_1 in the replicated Boolean sharings $[\![r]\!]^B$, denoted by $([\![r]\!]^B_1, [\![r]\!]^B_2)$.

3. The simulator S samples a random bit $c \in \mathbb{Z}_2$.
4. If $c = 0$, the simulator S computes and sets $(\llbracket r \rrbracket_1^K, \llbracket r \rrbracket_2^K) := (\llbracket b \rrbracket_1^K, \llbracket b \rrbracket_2^K)$ as the local arithmetic view of P_1 in $\llbracket r \rrbracket^K$; otherwise, the simulator S computes and sets $(\llbracket r \rrbracket_1^K, \llbracket r \rrbracket_2^K) := (1 - \llbracket b \rrbracket_1^K \pmod{2^k}, -\llbracket b \rrbracket_2^K \pmod{2^k})$ as the local arithmetic view of P_1 in $\llbracket r \rrbracket^K$.
5. The simulator S delivers to the environment $\mathcal{Z}$, in the same order as in the hybrid execution, first the local preprocessing view of P_1 in the daBit resource, namely the local Boolean view $(\llbracket r \rrbracket_1^B, \llbracket r \rrbracket_2^B)$ and the local arithmetic view $(\llbracket r \rrbracket_1^K, \llbracket r \rrbracket_2^K)$, then the public bit c as the audit opened value, and finally the local arithmetic output view of P_1 in $\llbracket b \rrbracket^K$.

The validity of this simulation is immediate. In the hybrid execution, the non-local transcript visible at the protocol level only involves the local preprocessing view of a random daBit and the public bit opened through $\mathcal{F}_{\mathrm{AOpen}}^B$. Since at most one party can be corrupt, the corrupted party's local Boolean view of the daBit is distributed independently of the underlying random bit r, so the freshly sampled $(\llbracket r \rrbracket_1^B, \llbracket r \rrbracket_2^B)$ have exactly the same distribution as the local Boolean view of P_1 in the hybrid execution. By contrast, for each fixed opened bit c, the local output rule of $\Pi_{\mathrm{B2A,3HMal}}^{B \to K}$ is an affine bijection between the local arithmetic view of P_1 in $\llbracket r \rrbracket^K$ and the final local arithmetic output view of P_1 in $\llbracket b \rrbracket^K$. Therefore, the simulator reconstructs the local arithmetic view of P_1 in $\llbracket r \rrbracket^K$, denoted by $(\llbracket r \rrbracket_1^K, \llbracket r \rrbracket_2^K)$, by applying the inverse affine map, based on the output results of the ideal functionality $\mathcal{F}_{\mathrm{B2A}}^{B \to K}$ and the opened bit c in reverse. Then the local arithmetic view of P_1 in $\llbracket r \rrbracket^K$ reconstructed by the simulator has the same distribution as in the hybrid execution. Moreover, since the hidden bit r is uniformly random, the bit $c = b \oplus r$ opened through $\mathcal{F}_{\mathrm{AOpen}}^B$ is uniformly distributed over $\mathbb{Z}_2$ and is independent of both the underlying secret bit b and the corrupted party's local preprocessing view. Therefore, the simulated Boolean preprocessing view, the reconstructed arithmetic preprocessing view, the opened bit c, and the final local arithmetic output view have the same joint distribution as in the hybrid execution. Hence, with $\mathcal{F}_{\mathrm{AOpen}}^B$ and $\mathcal{F}_{\mathrm{daBit}}$ treated as ideal functionalities, the simulator described above introduces no additional distinguishing probability, which means that for every well-behaved environment $\mathcal{Z}$, it follows that

$$\left|\Pr\left[\mathrm{Exec}\left[\Pi_{\mathrm{B2A,3HMal}}^{B \to K}(\mathcal{F}_{\mathrm{AOpen}}^B, \mathcal{F}_{\mathrm{daBit}}), A_{\mathrm{triv}}, \mathcal{Z}\right] = 1\right] - \Pr\left[\mathrm{Exec}\left[\mathcal{F}_{\mathrm{B2A}}^{B \to K}, S, \mathcal{Z}\right] = 1\right]\right| = 0.$$

□

Furthermore, since the Boolean audited opening protocol and the daBit resource are assumed to UC-securely realize $\mathcal{F}_{\mathrm{AOpen}}^B$ and $\mathcal{F}_{\mathrm{daBit}}$, respectively, according to the UC composition theorem, for every well-behaved environment $\mathcal{Z}$,

$$\begin{aligned}\left|\Pr\left[\mathrm{Exec}\left[\Pi_{\mathrm{B2A,3HMal}}^{B \to K}, A_{\mathrm{triv}}, \mathcal{Z}\right] = 1\right] - \Pr\left[\mathrm{Exec}\left[\mathcal{F}_{\mathrm{B2A}}^{B \to K}, S', \mathcal{Z}\right] = 1\right]\right| \\ \leq \varepsilon_{\mathrm{AOpen}}(\lambda) + \varepsilon_{\mathrm{daBit}}(\lambda)\end{aligned} \tag{31}$$

, where both $\varepsilon_{\text{AOpen}}(\lambda)$ and $\varepsilon_{\text{daBit}}(\lambda)$ are negligible. Hence, $\Pi^{B\to K}_{\text{B2A,3HMal}}$ UC-securely realizes the ideal functionality $\mathcal{F}^{B\to K}_{\text{B2A}}$.

With the security of the protocol $\Pi^{B\to K}_{\text{B2A,3HMal}}$ established in Theorem 7.2, the only remaining gap in security certification of the protocol $\Pi^{L,k}_{\text{Trunc,3HMal}}$ and $\Pi^{M,N}_{\text{SExt,3HMal}}$ is the security of protocol $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$. Accordingly, let $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$ denote the ideal top-two-bit extraction functionality. For any quantity $u \in \mathbb{Z}_{2^{t+2}}$, $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$ takes the replicated arithmetic sharing of u as the input, and outputs fresh sampled replicated Boolean sharings $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ such that $\lfloor u/2^t \rfloor = 2b_1 + b_0 \in \{0,1,2,3\}$. In the protocol $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$, the non-local part only involves the audited openings of two public values, the invocations of the LTConst protocol, the Boolean-to-arithmetic share-conversion protocol, and the strict edaBit preprocessing resource. This yields the hybrid-model security statement in Theorem 7.3.

Theorem 7.3. *The protocol $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$ UC-securely realizes the ideal top-two-bit extraction functionality $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$ in the $(\mathcal{F}_{\text{strict-edaBit}}, \mathcal{F}^{2^{t+2}}_{\text{AOpen}}, \mathcal{F}^{t}_{\text{LTConst}}, \mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}})$-hybrid model against static malicious corruption of at most one party.*

Proof. Since the corruption threshold in honest majority 3PC setting is one, there are four possible corruption patterns, namely no corruption and the three single-party corruptions P_1, P_2, and P_3. Although the output rules of $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$ differ slightly across the three parties when $\delta_0 = 1$ or $\delta_1 = 1$, this difference is purely syntactic and does not affect the security proof. It is therefore sufficient to consider the case where P_1 is corrupted. The cases where P_2 or P_3 is corrupted can be proved in the same way, and the case of no corruption is obvious.

Let S be the simulator in the ideal execution with $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$. The simulator acts as follows:

1. The environment $\mathcal{Z}$ provides the replicated input share of the corrupted party P_1 to the simulator S. Then, S forwards the replicated input share of P_1 to the ideal functionality $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$. Once the output of $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$ is available, S records the local Boolean view of P_1 in the replicated Boolean sharing $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$ of the target-functionality output, denoted by $([\![b_1]\!]^B_1, [\![b_1]\!]^B_2)$ and $([\![b_0]\!]^B_1, [\![b_0]\!]^B_2)$, respectively.
2. The simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![r]\!]^{2^{t+2}}$, denoted by $([\![r]\!]^{2^{t+2}}_1, [\![r]\!]^{2^{t+2}}_2)$, and a set of local Boolean views of P_1 in the replicated Boolean sharing $([\![r_i]\!]^B)_{i=0}^{t-1}$, denoted by $([\![r_i]\!]^B_1, [\![r_i]\!]^B_2)_{i=0}^{t-1}$.
3. The simulator S samples a random public value $\bar{c} \in \mathbb{Z}_{2^t}$, and sets $c := 2^2 \cdot \bar{c} \in \mathbb{Z}_{2^{t+2}}$.
4. The simulator S samples a fresh local Boolean view of P_1 in the replicated Boolean sharing $[\![v]\!]^B$, denoted by $([\![v]\!]^B_1, [\![v]\!]^B_2)$, as the local output that P_1 would obtain from $\mathcal{F}^{t}_{\text{LTConst}}$ in the hybrid execution.
5. The simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![v]\!]^{2^{t+2}}$, denoted by $([\![v]\!]^{2^{t+2}}_1, [\![v]\!]^{2^{t+2}}_2)$, as the local output that P_1 would obtain from $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$ in the hybrid execution.

6. The simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![\rho]\!]^{2^{t+2}}$, denoted by $([\![\rho]\!]_1^{2^{t+2}}, [\![\rho]\!]_2^{2^{t+2}})$, as part of the second set of strict edaBits preprocessing view.
7. The simulator S samples a random public value $\delta \in \{0,1,2,3\}$, publicly decomposes it as $\delta = \delta_0 + 2\delta_1$, where $\delta_0, \delta_1 \in \{0,1\}$, and sets $d := 2^t \cdot \delta$.
8. The simulator S reconstructs the local Boolean view of P_1 in $[\![\rho_0]\!]^B$ by applying the inverse local output rule of P_1 in Step 8 of $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$. Specifically, if $\delta_0 = 0$, S computes and sets $([\![\rho_0]\!]_1^B, [\![\rho_0]\!]_2^B) := ([\![b_0]\!]_1^B, [\![b_0]\!]_2^B)$ and $([\![\beta]\!]_1^B, [\![\beta]\!]_2^B) := ([\![\rho_0]\!]_1^B, [\![\rho_0]\!]_2^B)$; otherwise, S computes and sets $([\![\rho_0]\!]_1^B, [\![\rho_0]\!]_2^B) := (1 \oplus [\![b_0]\!]_1^B, [\![b_0]\!]_2^B)$ and $([\![\beta]\!]_1^B, [\![\beta]\!]_2^B) := (0,0)$.
9. The simulator S reconstructs the local Boolean view of P_1 in $[\![\rho_1]\!]^B$ by applying the inverse local output rule of P_1 in Step 9 of $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$. Specifically, if $\delta_1 = 0$, S computes and sets $([\![\rho_1]\!]_1^B, [\![\rho_1]\!]_2^B) := ([\![b_1]\!]_1^B \oplus [\![\beta]\!]_1^B, [\![b_1]\!]_2^B \oplus [\![\beta]\!]_2^B)$; otherwise, S computes and sets $([\![\rho_1]\!]_1^B, [\![\rho_1]\!]_2^B) := (1 \oplus [\![b_1]\!]_1^B \oplus [\![\beta]\!]_1^B, [\![b_1]\!]_2^B \oplus [\![\beta]\!]_2^B)$.
10. The simulator S delivers to the environment $\mathcal{Z}$, in the same order as in the hybrid execution, first the local preprocessing view of P_1 in the first set of strict edaBits resource, namely the local arithmetic view in $[\![r]\!]^{2^{t+2}}$ and the local Boolean views in $([\![r_i]\!]^B)_{i=0}^{t-1}$; second the local preprocessing view of P_1 in the second set of strict edaBits resource, namely the local arithmetic view in $[\![\rho]\!]^{2^{t+2}}$ and the local Boolean views in $[\![\rho_0]\!]^B$ and $[\![\rho_1]\!]^B$; then the public value c as the output of the first audited opening; next the local Boolean view of P_1 in $[\![v]\!]^B$ and the local arithmetic view of P_1 in $[\![v]\!]^{2^{t+2}}$, as the local outputs of $\mathcal{F}^t_{\text{LTConst}}$ and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$, respectively; then the public value d as the output of the second audited opening; and finally the local Boolean output views of P_1 in $[\![b_1]\!]^B$ and $[\![b_0]\!]^B$.

The validity of this simulation is immediate. In the hybrid execution, the non-local transcript visible at the protocol level only involves the local preprocessing views of two sets of random strict edaBits, the two public values opened through $\mathcal{F}^{2^{t+2}}_{\text{AOpen}}$, and the local outputs returned by $\mathcal{F}^t_{\text{LTConst}}$ and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$. Since at most one party can be corrupt, the corrupted party's local arithmetic and Boolean preprocessing views in the first set of strict edaBits are distributed independently of the underlying random value r, so the fresh sampled $([\![r]\!]_1^{2^{t+2}}, [\![r]\!]_2^{2^{t+2}})$ and $([\![r_i]\!]_1^B, [\![r_i]\!]_2^B)_{i=0}^{t-1}$ has exactly the same distribution as in the hybrid execution. Similarly, since the outputs of $\mathcal{F}^t_{\text{LTConst}}$ and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$ are defined to be fresh replicated sharings, the fresh sampled local Boolean and arithmetic views of $[\![v]\!]^B$ and $[\![v]\!]^{2^{t+2}}$ also have exactly the same distribution as in the hybrid execution. By contrast, for each fixed public value δ, the local output rules of $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$ define an affine bijection between the local Boolean views of P_1 in $([\![\rho_0]\!]^B, [\![\rho_1]\!]^B)$ and the final local Boolean output views of P_1 in $([\![b_0]\!]^B, [\![b_1]\!]^B)$. Therefore, the simulator reconstructs the local Boolean views of P_1 in $[\![\rho_0]\!]^B$ and $[\![\rho_1]\!]^B$ by applying the inverse affine map, based on the output results of the ideal functionality $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$ and the public value δ. Moreover, since the hidden values r and ρ are both uniformly random, the public value $\bar{c} = u + r \pmod{2^t}$ is uniformly distributed over $\mathbb{Z}_{2^t}$, and the public value $\delta = \lfloor u/2^t \rfloor + \rho \pmod 4$ is

uniformly distributed over $\{0,1,2,3\}$, respectively. Also, both of them are independent of the underlying secret input u and the corrupted party's local preprocessing views. Therefore, the simulated preprocessing views of two sets of strict edaBits, the opened values $\bar{c}$ and d, the local outputs of $\mathcal{F}^{t}_{\text{LTConst}}$ and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$, and the final local Boolean output views have the same joint distribution as in the hybrid execution. Hence, with $\mathcal{F}_{\text{strict-edaBit}}$, $\mathcal{F}^{2^{t+2}}_{\text{AOpen}}$, $\mathcal{F}^{t}_{\text{LTConst}}$, and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$ treated as ideal functionalities, the simulator described above introduces no additional distinguishing probability, which means that for every well-behaved environment $\mathcal{Z}$, it follows that

$$\Big|\Pr\Big[\text{Exec}\Big[\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}(\mathcal{F}_{\text{strict-edaBit}},\mathcal{F}^{2^{t+2}}_{\text{AOpen}},\mathcal{F}^{t}_{\text{LTConst}},\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}),A_{\text{triv}},\mathcal{Z}\Big]=1\Big] - \Pr\Big[\text{Exec}\Big[\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}},S,\mathcal{Z}\Big]=1\Big]\Big| = 0.$$

□

Therefore, since the Boolean audited opening protocol, the strict edaBits resource, the less-than-constant protocol, and the Boolean-to-arithmetic share-conversion protocol are assumed to UC-securely realize $\mathcal{F}^{2^{t+2}}_{\text{AOpen}}$, $\mathcal{F}_{\text{strict-edaBit}}$, $\mathcal{F}^{t}_{\text{LTConst}}$, and $\mathcal{F}^{B\to 2^{t+2}}_{\text{B2A}}$, respectively, according to the UC composition theorem, for every well-behaved environment $\mathcal{Z}$,

$$\begin{aligned}\Big|\Pr\Big[\text{Exec}\Big[\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}},A_{\text{triv}},\mathcal{Z}\Big]=1\Big] - \Pr\Big[\text{Exec}\Big[\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}},S,\mathcal{Z}\Big]=1\Big]\Big| \\ \le \varepsilon_{\text{strict-edaBit}}(\lambda)+\varepsilon_{\text{AOpen}}(\lambda)+\varepsilon_{\text{LTConst}}(\lambda)+\varepsilon_{\text{B2A}}(\lambda).\end{aligned} \tag{32}$$

where $\varepsilon_{\text{strict-edaBit}}(\lambda)$, $\varepsilon_{\text{AOpen}}(\lambda)$, $\varepsilon_{\text{LTConst}}(\lambda)$, and $\varepsilon_{\text{B2A}}(\lambda)$ are all negligible. Hence, $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$ UC-securely realizes the ideal functionality $\mathcal{F}^{2^{t+2}\to B}_{\text{Top2}}$.

As a consequence of the hybrid-model security statements established above for $\Pi^{L,k}_{\text{Trunc,3HMal}}$, $\Pi^{M,N}_{\text{SExt,3HMal}}$, and $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$, together with the UC security of the underlying subprotocols $\Pi^{B\to K}_{\text{B2A,3HMal}}$ and $\Pi^{2^{t+2}\to B}_{\text{Top2,3HMal}}$, the corresponding fully instantiated protocols inherit the same security guarantees by the UC composition theorem. Therefore, since the protocols and preprocessing functionalities used to instantiate audited opening, public-constant comparison, replicated-sharing multiplication, malicious daBit preprocessing, and malicious strict edaBits preprocessing are assumed to be UC-secure in the corresponding models, the UC security of the fully instantiated exact signed truncation, signed extension, and exact fixed-point multiplication protocols follows immediately. The formal statement is given in Theorem 7.4.

Theorem 7.4. *Assume that the protocols and preprocessing functionalities used to instantiate the ideal audited-opening functionality, the ideal public-constant comparison functionality, the ideal replicated-sharing multiplication functionality, malicious daBit preprocessing, and malicious strict edaBits preprocessing are UC-secure in the corresponding models. Then the fully instantiated protocols* $\Pi^{L,k}_{\text{Trunc,3HMal}}$, $\Pi^{M,N}_{\text{SExt,3HMal}}$, *and* $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$ *UC-securely realize the ideal functionalities* $\mathcal{F}^{L,k}_{\text{Trunc}}$, $\mathcal{F}^{M,N}_{\text{SExt}}$, *and* $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$, *respectively, against static malicious corruption of at most one party.*

7.2 Security of the variants under relaxed guarantees

Beyond exact constructions, three variants are further considered in Section 6.4 along different design routes. These variants relax the applicability domain, the precision guarantee, or the adversarial model in exchange for lower online cost. Accordingly, their security claims no longer lie on the same hybrid-model security chain established for the exact constructions, which is proved above, but are instead matched to their respective guarantees and models. Since these variants are introduced as efficient trade-off alternatives rather than as the main exact line of the paper, we only provide security theorem statements together with proof sketches rather than fully detailed UC proofs, with the supporting auxiliary materials given in Appendix B.

The protocol $\Pi^{M,N}_{\mathrm{SExt,3Semi}}$ realizes signed extension under the restricted domain $|x| < M/3$ by temporarily transforming the two-out-of-three replicated sharing into a two-party additive form, while its non-local part only involves the invocation of the modified MW protocol together with the final resharing step. Accordingly, let $\mathcal{F}^{M,N,\langle 1/3\rangle}_{\mathrm{SExt}}$ denote the ideal restricted-domain signed-extension functionality. For any input $x \in \mathbb{Z}_{2^m}$ satisfying $|x| < M/3$, $\mathcal{F}^{M,N,\langle 1/3\rangle}_{\mathrm{SExt}}$ takes the replicated arithmetic sharing of x as the input, and outputs a replicated arithmetic sharing $[\![y]\!]^N$ of the correct signed-extension over the target ring $\mathbb{Z}_{2^n}$. This yields the security statement in Theorem 7.5.

Theorem 7.5. *Assume that the protocol $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ is UC-secure in the corresponding models. Then the protocol $\Pi^{M,N}_{\mathrm{SExt,3Semi}}$ UC-securely realizes the restricted-domain signed-extension functionality $\mathcal{F}^{M,N,\langle 1/3\rangle}_{\mathrm{SExt}}$ for inputs satisfying $|x| < M/3$, against static semi-honest corruption of at most one party.*

Proof sketch. We first consider the hybrid protocol $\Pi^{M,N,*}_{\mathrm{SExt,3Semi}}$, in which the call to $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ is replaced by the ideal functionality $\mathcal{F}^{M,\langle 1/3\rangle}_{\mathrm{MW}}$. In Step 1–2, the replicated secret sharing $[\![x]\!]^M$ is only locally transformed into the two-party additive sharing $[x]^M$, no additional information is revealed. Then, if party P_1 is corrupted, the only non-local steps are the call to the ideal functionality $\mathcal{F}^{M,\langle 1/3\rangle}_{\mathrm{MW}}$ and the receipt of y'_2, where y'_2 is exactly part of the final local output view of P_1 in $[\![y]\!]^N$. If party P_2 is corrupted, besides the call to the ideal functionality $\mathcal{F}^{M,\langle 1/3\rangle}_{\mathrm{MW}}$, the remaining values involved in the resharing step only form the final local output view of P_2 in $[\![y]\!]^N$. If party P_3 is corrupted, then it has neither input nor output in the call to $\mathcal{F}^{M,\langle 1/3\rangle}_{\mathrm{MW}}$, and the only non-local value it receives is y_1, which together with the random value r constitutes its final local output view in $[\![y]\!]^N$. Hence, in all cases, the corrupted party learns nothing beyond the information allowed by the ideal functionality $\mathcal{F}^{M,N,\langle 1/3\rangle}_{\mathrm{SExt}}$ in the hybrid protocol. By the subprotocol composition theorem, together with the assumed UC security of $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$, the original protocol $\Pi^{M,N}_{\mathrm{SExt,3Semi}}$ also UC-securely realizes $\mathcal{F}^{M,N,\langle 1/3\rangle}_{\mathrm{SExt}}$.

The protocol $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$ realizes signed truncation with at most 1 ULP error by locally computing the truncated base value and then applying the fixed correction $+1$, and its entire execution consists only of local operations. Accordingly, let $\mathcal{F}^{L,k}_{\text{Trunc-1ULP}}$ denote the ideal one-ULP-error signed-truncation functionality. For any input $x \in \mathbb{Z}_{2^l}$, $\mathcal{F}^{L,k}_{\text{Trunc-1ULP}}$ takes the replicated arithmetic sharing of x as the input, and outputs a replicated arithmetic sharing $[\![y]\!]^{2^{l-k}}$ of a signed-truncation result whose deviation from the exact signed-truncation result is bounded by at most 1 ULP. This yields the security statement in Theorem 7.6.

Theorem 7.6. *The protocol $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$ UC-securely realizes the ideal one-ULP-error signed-truncation functionality $\mathcal{F}^{L,k}_{\text{Trunc-1ULP}}$ against static malicious corruption of at most one party.*

Proof sketch. In protocol $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$, the parties only locally compute the truncated base value $[\![\tilde{y}]\!]^{2^{l-k}}$ and then apply the public correction $+1$ to obtain $[\![y]\!]^{2^{l-k}}$, so no additional information is revealed. If any one of the three parties is corrupted, then its entire view consists only of its initial local view in $[\![x]\!]^L$ together with the final local output view in $[\![y]\!]^{2^{l-k}}$. Hence, in all cases, the corrupted party learns nothing beyond the information allowed by the ideal functionality $\mathcal{F}^{L,k}_{\text{Trunc-1ULP}}$. Therefore, the protocol $\Pi^{L,k}_{\text{Trunc-1ULP,3HMal}}$ UC-securely realizes $\mathcal{F}^{L,k}_{\text{Trunc-1ULP}}$.

8 Performance evaluation

Performance evaluation in this paper is carried out from both theoretical and experimental perspectives, where the former compares online communication overhead and the latter compares online runtime. For each perspective, the evaluation is further organized into the primitive level and the end-to-end level. The primitive level focuses on signed extension and exact signed truncation, whereas the end-to-end level focuses on the overall performance of exact fixed-point multiplication.

8.1 Theoretical online communication analysis

The theoretical comparison is given uniformly in terms of online communication overhead. The compared baselines are taken from the nearest relevant works to the best of our knowledge, although most of them are not fully identical in semantic target, security guarantee, or input domain. For the protocols proposed here, the reported costs start from the abstract online costs derived above and are further instantiated as bit-level communication. For the compared baselines, the reported online communication costs are adopted directly when they are explicitly available in the original works; otherwise, the corresponding costs are reconstructed under the same online-only convention from the original protocol descriptions whenever sufficient information is available.

To enable precise quantification of bit-level online communication overhead for both the protocol proposed in this paper and the baseline protocols, we first standardize the communication costs of all invoked subprotocols. For subprotocols employed as black-box components, we adopt the standard realization, where available, as the basis for communication overhead accounting. While in cases where multiple realizations exist, we select, to the best of our knowledge, the realization that minimizes online communication while remaining fully compatible with the prevailing security model and semantic requirements.

For the audited opening, under 2-out-of-3 replicated sharing, each party sends its two locally held shares to the other two parties; therefore, the online communication cost of the invocation audited opening over ring $\mathbb{Z}_{2^k}$ is standardized as $C^k_{\mathrm{AOpen}} = 6k$ bits. Specially, for audited opening over the Boolean domain $\mathbb{Z}_2$, the online communication cost is given as $C^B_{\mathrm{AOpen}} = 6$ bits. For the standard multiplication on a fixed ring, standard implementations are required under two different security models, namely semi-honest security and malicious security. In semi-honest model, each party forms one additive share of the product locally from its view in the replicated sharing of the input and then sends this share to the previous party to complete the resharing step. Therefore, the online communication cost of the semi-honest multiplication over ring $\mathbb{Z}_{2^k}$ is standardized as $C^k_{\mathrm{Mul,3Semi}} = 3k$ bits. While in malicious model, the multiplication is standardized as the Beaver-style masked multiplication, whose online phase consists of two masked openings. Therefore, the online communication cost of the malicious multiplication over ring $\mathbb{Z}_{2^k}$ is standardized as $C^k_{\mathrm{Mul,3mal}} = 2C^k_{\mathrm{AOpen}} = 12k$ bits. For comparison to a public constant, we adopt the core part of edaBits-based comparison machinery of Escudero et al.[21], whose online operations are $2k$ Boolean AND gates. And we instantiate these AND gates by the malicious 3PC binary-circuit proposed by Li et al.[39], which costs 3 bits per AND gate in total. Therefore, the online communication cost of the LTConst over ring $\mathbb{Z}_{2^k}$ is standardized as $C^k_{\mathrm{LTConst}} = 2k \cdot C^B_{\mathrm{AND}} = 6k$ bits.

On top of the black-box components above, we further construct several subprotocols for our main protocols in Section 5 and Appendix B. For the 3PC B2A protocol, the abstract online communication cost is $C^K_{\mathrm{B2A}} = C^B_{\mathrm{AOpen}}$ and is therefore standardized as $C^K_{\mathrm{B2A}} = 6$ bits. For the Top2 protocol, the abstract online communication cost is $C^t_{\mathrm{Top2}} = 2C^{t+2}_{\mathrm{AOpen}} + C^t_{\mathrm{LTConst}} + C^{t+2}_{\mathrm{B2A}}$, and is therefore standardized as $C^t_{\mathrm{Top2}} = 2 \cdot 6(t+2) + 6t + 6 = 18t + 30$ bits. Meanwhile, for the modified MW protocol, its concrete online communication cost has already been standardized as $C^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}} = 2 \cdot C^{B\to K}_{\mathrm{AND,3Semi}} = 28$ bits.

To place all instantiated communication costs under a unified convention, two additional instantiation rules are fixed throughout the following comparisons. For any compared protocol whose online communication cost depends on the security parameter λ, the instantiated communication cost is reported with $\lambda = 128$. Moreover, for each type of semantics being compared, in order to visually demonstrate the differences in communication overhead among different works, three sets of representative parameter settings are selected for instantiation.

For signed extension, the compared baselines include SIRNN [14], the two variants of Guo et al.[9] under the respective input restrictions $|x| < M/4$ and $|x| < M/3$,

and the protocol of Ditto[15] under the input restrictions $|x| < M/4$. The comparison results are summarized in Table 4. As shown in the table, lower communication is generally achieved either under weaker security guarantees or under more restrictive input domains. Ditto attains the lowest communication in the 3-party setting, but its signed extension is essentially realized from zero extension through a shifting procedure, so the condition $|x| < M/4$ is intrinsic to this route and cannot be further relaxed in the same manner. By contrast, the restricted variant we proposed in this paper retains low communication under the broader condition $|x| < M/3$, while the main protocol further attains full-domain exact signed extension under malicious security.

Table 4 Theoretical online communication comparison for signed extension.

					Comm. (bits)		
Work	Parties	Security	Restriction	Online Comm. Expr.	Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	semi-honest	$M/4$	$3m + n$	40	80	160
SIRNN[14]	2-party	semi-honest	**None**	$< \lambda(m + 1) + 13m + n$	< 1272	< 2416	< 4704
Guo et al.[9]	2-party	semi-honest	$M/4$	$\lambda + n - m$	136	144	160
Guo et al.[9]	2-party	semi-honest	$M/3$	$2(\lambda + n - m)$	272	288	320
Ours-semi	3-party	semi-honest	$M/3$	$2n + 28$	60	92	156
Ours-mal	3-party	**malicious**	**None**	$18m + 42$	186	330	618

The upper part of the table reports semi-honest protocols, while the lower part reports maliciously secure protocols.

[1]Instantiation 1: the input is an 8-bit signed secret-shared value, and the output is a 16-bit signed secret-shared value, i.e., $m = 8$ and $n = 16$.

[2]Instantiation 2: the input is a 16-bit signed secret-shared value, and the output is a 32-bit signed secret-shared value, i.e., $m = 16$ and $n = 32$.

[3]Instantiation 3: the input is a 32-bit signed secret-shared value, and the output is a 64-bit signed secret-shared value, i.e., $m = 32$ and $n = 64$.

For signed truncation, we compared results under both approximate and exact semantics. The baselines for approximate signed truncation is Ditto[15], whereas the baselines for exact signed truncation include SIRNN[14], the exact variant of Guo et al.[9], and the protocol of Escudero et al.[21] together with its straightforward adapted version whose output is directly produced on the smaller ring. The comparison results are summarized in Table 5. As shown in the table, the communication overhead of approximate signed truncation is significantly lower than that of exact signed truncation, while the exact truncation protocol in a two-party setting incurs particularly high overall overhead. Specifically, Ditto achieves zero online communication in a 3-party setup, while the 1 ULP error variant proposed in this paper is an optimization of this approach based on an analysis of its output error distribution; it typically produces error-free results with a much higher probability while maintaining zero online communication. On the other hand, the original protocol by Escudero et al. only provides a version where the output remains on the original ring; however, our analysis shows that this protocol can be easily modified to produce a version where the output is produced on the reduced ring. This modified version is the only baseline that directly aligns with the semantic target of the main protocol in this paper, while our exact protocol achieves significantly lower communication overhead under the same

semantic conditions. For typical application scenarios, the online phase communication overhead of our semantically equivalent protocol is less than 50% of that of the baseline.

Table 5 Theoretical online communication comparison for signed truncation.

							Comm. (bits)		
Work	Parties	Security	Out-Ring	Sem.	Restriction	Online Comm. Expr.	Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	malicious*	reduced	approx.	None	0	0	0	0
Ours-1ULP	3-party	malicious	reduced	approx.	None	0	0	0	0
SIRNN[14]	2-party	semi-honest	reduced	exact	None	$\lambda(k+1)+\ell+13k$	708	1288	2448
Guo et al.[9]	2-party	semi-honest	original	exact	$L/4$	$< \lambda(k+2)+\ell+15k$	< 844	< 1432	< 2608
Escudero et al.[21]	3-party	malicious	original	exact	None	$18\ell+12$	300	588	1164
Escudero et al.[21]	3-party	malicious	reduced**	exact	None	$12\ell+6k+6$	222	438	870
Ours-exact	3-party	malicious	reduced	exact	None	$18k+42$	114	186	330

The upper part of the table reports approximate signed truncation protocols, while the lower part reports exact signed truncation protocols.

* Ditto is presented overall in the semi-honest security model, whereas its truncation protocol itself is maliciously secure.

** The original paper only gives the version whose output remains on the original ring, while this version with output on the smaller ring follows from a straightforward adaptation of the same protocol.

[1] Instantiation 1: the input is a 16-bit signed fixed-point secret-shared value, and 4 fractional bits are truncated, i.e., $\ell = 16$ and $k = 4$.

[2] Instantiation 2: the input is a 32-bit signed fixed-point secret-shared value, and 8 fractional bits are truncated, i.e., $\ell = 32$ and $k = 8$.

[3] Instantiation 3: the input is a 64-bit signed fixed-point secret-shared value, and 16 fractional bits are truncated, i.e., $\ell = 64$ and $k = 16$.

For fixed-point multiplication, the compared baselines include SIRNN[14], the exact route of Guo et al.[9], and Ditto[15]. The comparison results are summarized in Table 6. As shown in the table, none of the compared baselines directly matches the semantic target of the main protocol proposed here. Ditto attains the lowest communication cost, but only under approximate semantics, weaker security guarantees, and the restricted input domain $M/4$, and therefore does not provide a general baseline for exact fixed-point multiplication. The remaining exact baselines, namely SIRNN and the exact route of Guo et al., are both given in the 2-party semi-honest setting, while the latter does not directly output the final truncation result on the smaller ring. In contrast, the main protocol Ours-mal proposed in this paper attains general exact fixed-point multiplication under malicious security without such restrictions. Moreover, although its online communication cost is higher than that of Ditto, it remains substantially lower than those of the two exact 2-party baselines. In addition, Ours-tradeoff shows that, under relaxed guarantees, combining our semi-honest signed-extension variant, semi-honest in-ring multiplication, and our 1-ULP truncation variant yields communication close to Ditto while supporting a broader input range, and providing a higher probability of coinciding with the exact truncation result.

8.2 Experimental evaluation

The experimental comparison is reported in terms of online runtime. All experiments run on a workstation with an AMD Ryzen 9 9950X CPU and 128 GB of RAM.

Table 6 Theoretical online communication comparison for fixed-point multiplication.

Work	Parties	Security	Sem.	Restriction	Online Comm. Expr.	Comm. (bits)		
						Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	semi-honest	approx.	$M/4$	$6m + 5n$	128	256	512
SIRNN[14]*	2-party	semi-honest	exact	None	$< 3m^2 + 4\lambda m + \lambda(\tau + 1) + 34m + 13\tau$	< 5252	< 10760	< 22928
Guo et al.[9]*	2-party	semi-honest	exact**	$M/4$	$< \lambda(2m + \tau + 14) + 3m^2 + 9m + n + 15\tau$	< 4692	< 7969	< 15696
Ours-tradeoff	3-party	semi-honest	approx.	$M/3$	$7n + 56$	168	280	504
Ours-mal	3-party	**malicious**	exact	None	$36m + 12n + 18\tau + 126$	678	1230	2334

* The fixed-point multiplication protocols of SIRNN and Guo et al. do not follow the conventional route of first extending the operands to a larger ring and then multiplying. Instead, they directly multiply operands already given over non-uniform rings.

** Unlike the other compared protocols, the exact route of Guo et al. does not directly output the final truncation result on the smaller ring.

[1]Instantiation 1: two 8-bit signed fixed-point secret-shared values with 4 fractional bits are multiplied, and 4 fractional bits are truncated from the product, i.e., $m = 8$, $n = 16$ and $\tau = 4$.

[2]Instantiation 2: two 16-bit signed fixed-point secret-shared values with 8 fractional bits are multiplied, and 8 fractional bits are truncated from the product, i.e., $m = 16$, $n = 32$ and $\tau = 8$.

[3]Instantiation 3: two 32-bit signed fixed-point secret-shared values with 16 fractional bits are multiplied, and 16 fractional bits are truncated from the product, i.e., $m = 32$, $n = 64$ and $\tau = 16$.

Network latency is emulated with RTT set to 10 ms and 100 ms, corresponding to LAN and WAN settings, respectively. For each functionality, namely signed extension, signed truncation, and exact fixed-point multiplication, three representative parameter settings are used. These settings are consistent with those used in Section 8.1. For each parameter configuration, we execute 1000 independent single-instance runs and report the mean across five independent trials. Moreover, to better reflect the latency of the protocol itself, we do not use cross-instance batching, vectorization, or pipelining to amortize the online cost; only the parallelism naturally available within a single protocol instance is used.

It is important to note that the original implementation frameworks of Ditto and Escudero et al., namely MP-SPDZ and spu, respectively, do not directly support the instantiations required in our evaluation. Therefore, we reimplemented the relevant protocols of Ditto and Escudero et al. under the same experimental interface as our own. Unlike the theoretical comparison, Guo et al.'s work is excluded from the experimental evaluation. Although their paper indicates that the implementation has been integrated into a public GitHub repository, the corresponding source code was not publicly accessible at the time of this study. Moreover, some of the low-level operations required by their protocols are not specified in sufficient implementation detail, which makes it difficult for us to reimplement a benchmark that faithfully reflects the original performance of Guo et al.'s protocols.

For signed extension, the experimental baselines include SIRNN[14] and Ditto[15], and the results are summarized in Table 7. The runtime results should be interpreted together with the underlying security model and applicability domain. In particular, Ours-mal targets malicious security on the full domain, whereas the other baselines provide only semi-honest security, and some further require restricted input ranges. Its higher runtime is therefore expected and reflects the cost of stronger guarantees

rather than a directly comparable slowdown. Nevertheless, the experimental results demonstrate that our protocol still achieves practical runtime under malicious security.

Table 7 Experimental online runtime comparison for signed extension.

				Runtime (ms)					
				LAN (RTT = 10 ms)			WAN (RTT = 100 ms)		
Work	Parties	Security	Restriction	Ins I[1]	Ins II[2]	Ins III[3]	Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	semi-honest	$M/4$	16.25	16.26	16.27	151.26	151.26	151.27
SIRNN[14]	2-party	semi-honest	None	31.57	41.54	41.57	303.60	404.48	404.61
Ours-semi	3-party	semi-honest	$M/3$	26.79	26.80	26.77	252.22	252.22	252.23
Ours-mal	3-party	**malicious**	None	37.76	43.03	48.45	353.12	403.55	453.98

The upper part of the table reports semi-honest protocols, while the lower part reports maliciously secure protocols.

[1]Instantiation 1: the input is an 8-bit signed secret-shared value, and the output is a 16-bit signed secret-shared value, i.e., $m = 8$ and $n = 16$.

[2]Instantiation 2: the input is a 16-bit signed secret-shared value, and the output is a 32-bit signed secret-shared value, i.e., $m = 16$ and $n = 32$.

[3]Instantiation 3: the input is a 32-bit signed secret-shared value, and the output is a 64-bit signed secret-shared value, i.e., $m = 32$ and $n = 64$.

For signed truncation, the experimental baselines include Ditto[15] for the approximate semantic, SIRNN[14], Escudero et al.'s work[21], and its modified version for the exact route. The comparison results are summarized in Table 8. As shown in the table, our approximate protocol with 1 ULP error and Ditto exhibit nearly identical runtime, but the analysis in Appendix B.3 shows that, in most cases, the fixed correction used in our approximate variant yields a higher probability of coinciding with the exact result. On the other hand, our exact signed truncation protocol is only slightly slower than the modified Escudero-based implementation. This gap mainly comes from the two additional B2A invocations in our protocol. Nevertheless, as already shown in the theoretical comparison, our protocol achieves substantially lower online communication overhead. This indicates that our exact protocol attains a favorable trade-off between runtime and communication.

For fixed-point multiplication, the experimental baselines include Ditto[15] and SIRNN[14], and the comparison results are summarized in Table 9. At present, to the best of our knowledge, there are no prior implementations that realize the same functionality considered here. In particular, Ditto provides only a semi-honest protocol under restricted inputs, and its output is not exact, while SIRNN is a two-party semi-honest protocol. Therefore, their runtime should be regarded only as a reference rather than as a directly comparable target. Nevertheless, the main protocol Ours-mal achieves practical runtime while providing malicious security, exact semantics, and no input-domain restriction. In particular, although Ours-mal is slower than Ditto, which relies on weaker guarantees and approximate semantics, it is consistently faster than SIRNN in all tested settings despite providing stronger security guarantees. In addition, although Our-tradeoff is still somewhat slower than Ditto, it supports a broader input range, and its truncation has a higher probability of coinciding with the exact result in typical cases.

Table 8 Experimental online runtime comparison for signed truncation.

					Runtime					
					LAN (RTT = 10 ms)			WAN (RTT = 100 ms)		
Work	Parties	Security	Out-Ring	Sem.	Ins I[1]	Ins II[2]	Ins III[3]	Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	malicious*	reduced	approx.	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$
Ours-1ULP	3-party	malicious	reduced	approx.	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$	$\ll 1$
SIRNN[14]	2-party	semi-honest	reduced	exact	20.79	31.48	41.51	202.23	303.67	404.53
Escudero et al.[21]	3-party	malicious	original	exact	53.93	64.56	75.39	504.22	605.07	705.89
Escudero et al.[21]	3-party	malicious	reduced**	exact	26.98	32.25	37.58	252.12	302.53	353.06
Ours-exact	3-party	malicious	reduced	exact	32.49	37.76	43.07	302.70	353.13	403.56

The upper part of the table reports approximate signed truncation protocols, while the lower part reports exact signed truncation protocols.

* Ditto is presented overall in the semi-honest security model, whereas its truncation protocol itself is maliciously secure.

** The original paper only gives the version whose output remains on the original ring, while this version with output on the smaller ring follows from a straightforward adaptation of the same protocol.

[1]Instantiation 1: the input is a 16-bit signed fixed-point secret-shared value, and 4 fractional bits are truncated, i.e., $\ell = 16$ and $k = 4$.

[2]Instantiation 2: the input is a 32-bit signed fixed-point secret-shared value, and 8 fractional bits are truncated, i.e., $\ell = 32$ and $k = 8$.

[3]Instantiation 3: the input is a 64-bit signed fixed-point secret-shared value, and 16 fractional bits are truncated, i.e., $\ell = 64$ and $k = 16$.

Table 9 Experimental online runtime comparison for fixed-point multiplication.

					Runtime (ms)					
					LAN (RTT = 10 ms)			WAN (RTT = 100 ms)		
Work	Parties	Security	Sem.	Restriction	Ins I[1]	Ins II[2]	Ins III[3]	Ins I[1]	Ins II[2]	Ins III[3]
Ditto[15]	3-party	semi-honest	approx.	$M/4$	21.51	21.54	21.50	201.76	201.70	201.70
SIRNN[14]	2-party	semi-honest	exact	None	94.52	125.25	135.64	911.88	1295.70	1618.12
Ours-tradeoff	3-party	semi-honest	approx.	$M/3$	32.13	32.14	32.22	302.76	302.75	302.71
Ours-mal	3-party	**malicious**	exact	None	75.71	86.44	97.35	706.13	806.96	907.86

The compared baselines are reported only for reference, since none of them realizes the same functionality considered here.

[1]Instantiation 1: two 8-bit signed fixed-point secret-shared values with 4 fractional bits are multiplied, and 4 fractional bits are truncated from the product, i.e., $m = 8$, $n = 16$, and $\tau = 4$.

[2]Instantiation 2: two 16-bit signed fixed-point secret-shared values with 8 fractional bits are multiplied, and 8 fractional bits are truncated from the product, i.e., $m = 16$, $n = 32$, and $\tau = 8$.

[3]Instantiation 3: two 32-bit signed fixed-point secret-shared values with 16 fractional bits are multiplied, and 16 fractional bits are truncated from the product, i.e., $m = 32$, $n = 64$, and $\tau = 16$.

9 Conclusion

This work investigates maliciously secure, exact fixed-point multiplication over power-of-two rings within the honest-majority replicated 3PC framework. The central challenge of this functionality arises from two cross-ring nonlinear primitives required for multiplication: exact signed truncation and signed extension. The efficiency, security, and composability of these primitives fundamentally govern the practical viability of exact fixed-point arithmetic in this setting. A key insight is that, although these functionalities serve distinct algorithmic roles, the nonlinear information that must be securely reconstructed in each case is substantially smaller than the full output.

Specifically, local linear operations already determine a base value, and the residual ambiguity reduces to a small, bounded quotient. Leveraging this observation, we identify a unified computational primitive, namely bounded-quotient extraction and correction, and demonstrate that both exact signed truncation and signed extension arise as direct, efficient instantiations thereof. Composing these instantiations sequentially yields a complete exact fixed-point multiplication protocol. Beyond the concrete constructions, this unification delivers conceptual and analytical benefits: the resulting protocols admit a common UC-style abstraction; their security proofs follow a uniform compositional argument; and their dominant online communication and computation costs are governed by a shared analytical model. The theoretical analysis and experimental results further show that the proposed approach achieves meaningful efficiency while providing exact semantics and malicious security in a setting where prior baselines generally rely on weaker guarantees, restricted domains, or non-identical semantic targets. Thus, the principal contribution of this work is not merely a set of isolated protocols, but a unified, composable, and analytically transparent framework for cross-ring exact arithmetic. We posit that this framework establishes a principled foundation for building more sophisticated exact numeric protocols in replicated 3PC.

References

[1] Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., Maaten, L.: CrypTen: Secure Multi-Party Computation Meets Machine Learning. In: Advances in Neural Information Processing Systems, vol. 34, pp. 4961–4973. Curran Associates, Inc., Red Hook, NY, USA (2021). https://proceedings.neurips.cc/paper/2021/hash/2754518221cfbc8d25c13a06a4cb8421-Abstract.html Accessed 2026-04-19

[2] Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 19–38. IEEE, San Jose, CA, USA (2017). https://doi.org/10.1109/SP.2017.12 . http://ieeexplore.ieee.org/document/7958569/ Accessed 2026-03-20

[3] Rathee, D., Bhattacharya, A., Gupta, D., Sharma, R., Song, D.: Secure Floating-Point Training, pp. 6329–6346 (2023). https://www.usenix.org/conference/usenixsecurity23/presentation/rathee Accessed 2026-04-19

[4] Yuan, B., Yang, S., Zhang, Y., Ding, N., Gu, D., Sun, S.-F.: MD-ML: Super Fast Privacy-Preserving Machine Learning for Malicious Security with a Dishonest Majority, pp. 2227–2244 (2024). https://www.usenix.org/conference/usenixsecurity24/presentation/yuan Accessed 2026-04-19

[5] Catrina, O., Saxena, A.: Secure Computation with Fixed-Point Numbers. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell,

J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Sion, R. (eds.) Financial Cryptography and Data Security, vol. 6052, pp. 35–50. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_6 . Series Title: Lecture Notes in Computer Science. http://link.springer.com/10.1007/978-3-642-14577-$3_6 Accessed 2026-03-20$

[6] Harth-Kitzerow, C., Suresh, A., Carle, G.: SoK: Truncation Untangled: Scaling Fixed-Point Arithmetic for Privacy-Preserving Machine Learning to Large Models and Datasets. In: Proceedings on Privacy Enhancing Technologies (2025). https://www.petsymposium.org/popets/2025/popets-2025-0135.php Accessed 2026-03-20

[7] Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation, pp. 2165–2182 (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/patra Accessed 2026-04-19

[8] Ma, J., Zheng, Y., Feng, J., Zhao, D., Wu, H., Fang, W., Tan, J., Yu, C., Zhang, B., Wang, L.: SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning, pp. 17–33 (2023). https://www.usenix.org/conference/atc23/presentation/ma Accessed 2026-04-19

[9] Guo, H., Peng, L., Xue, H., Peng, L., Liu, W., Liu, Z., Hu, L.: Improved Secure Two-party Computation from a Geometric Perspective. In: 34th USENIX Security Symposium (USENIX Security 25), pp. 4957–4974 (2025). https://www.usenix.org/conference/usenixsecurity25/presentation/guo-hao-improved Accessed 2026-03-20

[10] Furukawa, J., Lindell, Y.: Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. CCS '19, pp. 1557–1571. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3339811 . https://dl.acm.org/doi/10.1145/3319535.3339811 Accessed 2026-04-19

[11] Harth-Kitzerow, C., Suresh, A., Wang, Y., Yalame, H., Carle, G., Annavaram, M.: High-Throughput Secure Multiparty Computation with an Honest Majority in Various Network Settings. Proceedings on Privacy Enhancing Technologies (2025). Accessed 2026-04-19

[12] Dalskov, A., Escudero, D., Keller, M.: Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 2183–2200 (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov Accessed 2026-03-20

[13] Liu, F., Xie, X., Yu, Y.: Scalable multi-party computation protocols for machine learning in the honest-majority setting. In: Proceedings of the 33rd USENIX Conference on Security Symposium. SEC '24, pp. 1939–1956. USENIX Association, USA (2024)

[14] Rathee, D., Rathee, M., Kiran Goli, R.K., Gupta, D., Sharma, R., Chandran, N., Rastogi, A.: SiRnn: A Math Library for Secure RNN Inference. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1003–1020 (2021). https://doi.org/10.1109/SP40001.2021.00086 . ISSN: 2375-1207. https://ieeexplore.ieee.org/document/9519413 Accessed 2026-03-20

[15] Wu, H., Fang, W., Zheng, Y., Ma, J., Tan, J., Wang, L.: Ditto: Quantization-aware Secure Inference of Transformers upon MPC. In: Proceedings of the 41st International Conference on Machine Learning, pp. 53346–53365. PMLR, Online (2024). https://proceedings.mlr.press/v235/wu24d.html Accessed 2026-03-20

[16] Huang, Z., Lu, W.-j., Hong, C., Ding, J.: Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 809–826 (2022). https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong Accessed 2026-03-20

[17] Zhou, L., Song, Q., Zhang, S., Wang, Z., Wang, X., Li, Y.: Bicoptor 2.0: Addressing Challenges in Probabilistic Truncation for Enhanced Privacy-Preserving Machine Learning. arXiv. arXiv:2309.04909 [cs] (2024). https://doi.org/10.48550/arXiv.2309.04909 . http://arxiv.org/abs/2309.04909 Accessed 2026-03-20

[18] Agarwal, A., Peceny, S., Raykova, M., Schoppmann, P., Seth, K.: Communication-Efficient Secure Logistic Regression. In: 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P), pp. 440–467 (2024). https://doi.org/10.1109/EuroSP60621.2024.00031 . ISSN: 2995-1356. https://ieeexplore.ieee.org/document/10628532 Accessed 2026-04-19

[19] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow2: Practical 2-Party Secure Inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 325–342. ACM, Virtual Event USA (2020). https://doi.org/10.1145/3372297.3417274 . https://dl.acm.org/doi/10.1145/3372297.3417274 Accessed 2026-03-20

[20] Cheng, K., Xia, Y., Song, A., Fu, J., Qu, W., Shen, Y., Zhang, J.: Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. CCS '25, pp. 4124–4138. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3719027.3765028 . https://dl.acm.org/doi/10.1145/3719027.3765028 Accessed 2026-03-20

[21] Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved Primitives

for MPC over Mixed Arithmetic-Binary Circuits. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, pp. 823–852. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_29

[22] Catrina, O., Dragulin, C.: Multiparty Computation of Fixed-Point Multiplication and Reciprocal. In: 2009 20th International Workshop on Database and Expert Systems Application, pp. 107–111. IEEE, Linz, Austria (2009). https://doi.org/10.1109/DEXA.2009.84 . http://ieeexplore.ieee.org/document/5337405/ Accessed 2026-03-20

[23] Zhang, S., Qu, G., Zhang, Z., Huang, M., Jin, H., Yang, L.: Efficient and secure multi-party computation protocol supporting deep learning. Cybersecurity **8**(1), 46 (2025) https://doi.org/10.1186/s42400-024-00343-4 . Accessed 2026-03-20

[24] Keller, M., Sun, K.: Secure Quantized Training for Deep Learning. In: Proceedings of the 39th International Conference on Machine Learning, pp. 10912–10938. PMLR, Online (2022). https://proceedings.mlr.press/v162/keller22a.html Accessed 2026-03-20

[25] Badrinarayanan, S., Lee, E., Miao, P., Rindal, P.: Improved Multi-Party Fixed-Point Multiplication. Cryptology ePrint Archive. Publication info: Preprint. (2024). https://eprint.iacr.org/2024/1047 Accessed 2026-03-20

[26] Mohassel, P., Rindal, P.: ABY^3: A Mixed Protocol Framework for Machine Learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 35–52. ACM, Toronto Canada (2018). https://doi.org/10.1145/3243734.3243760 . https://dl.acm.org/doi/10.1145/3243734.3243760 Accessed 2026-03-20

[27] Chaudhari, H., Rachuri, R., Suresh, A.: Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In: Proceedings 2020 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2020). https://doi.org/10.14722/ndss.2020.23005 . https://www.ndss-symposium.org/wp-content/uploads/2020/02/23005.pdf Accessed 2026-03-20

[28] Patra, A., Suresh, A.: BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In: Proceedings 2020 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2020). https://doi.org/10.14722/ndss.2020.24202 . https://www.ndss-symposium.org/wp-content/uploads/2020/02/24202.pdf Accessed 2026-03-20

[29] Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 2651–2668 (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/koti Accessed 2026-03-20

[30] Koti, N., Patra, A., Rachuri, R., Suresh, A.: Tetrad: Actively Secure 4PC for Secure Training and Inference. In: Proceedings 2022 Network and Distributed System Security Symposium. Internet Society, San Diego, CA, USA (2022). https://doi.org/10.14722/ndss.2022.24058 . https://www.ndss-symposium.org/wp-content/uploads/2022-58-paper.pdf Accessed 2026-03-20

[31] Koti, N., Patil, S., Patra, A., Suresh, A.: MPClan: Protocol Suite for Privacy-Conscious Computations. Journal of Cryptology **36**(3), 22 (2023) https://doi.org/10.1007/s00145-023-09469-z . Accessed 2026-03-20

[32] Dalskov, A., Escudero, D., Keller, M.: Secure Evaluation of Quantized Neural Networks. In: Proceedings on Privacy Enhancing Technologies, vol. 2020, pp. 355–375 (2020). https://doi.org/10.2478/popets-2020-0077 . https://petsymposium.org/popets/2020/popets-2020-0077.php Accessed 2026-03-20

[33] Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. In: Proceedings on Privacy Enhancing Technologies, vol. 2021, pp. 188–208 (2021). https://doi.org/10.2478/popets-2021-0011 . https://petsymposium.org/popets/2021/popets-2021-0011.php Accessed 2026-03-20

[34] Frederiksen, T.K., Lindstrøm, J., Madsen, M.W., Spangsberg, A.D.: A New Approach to Efficient and Secure Fixed-Point Computation. In: Pöpper, C., Batina, L. (eds.) Applied Cryptography and Network Security, vol. 14583, pp. 58–87. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-54770-6_3 . Series Title: Lecture Notes in Computer Science. https://link.springer.com/10.1007/978-3-031-54770-6$_3 Accessed 2026 - 03 - 20

[35] Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: FSS-based Secure Training and Inference with GPUs. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 597–616 (2024). https://doi.org/10.1109/SP54263.2024.00063 . ISSN: 2375-1207. https://ieeexplore.ieee.org/document/10646833 Accessed 2026-03-20

[36] Gupta, K., Kumaraswamy, D., Chandran, N., Gupta, D.: LLAMA: A Low Latency Math Library for Secure Inference. In: Proceedings on Privacy Enhancing Technologies, vol. 2022, pp. 274–294 (2022). https://doi.org/10.56553/popets-2022-0109 . https://petsymposium.org/popets/2022/popets-2022-0109.php Accessed 2026-03-20

[37] Zbudila, M., Pohle, E., Abidin, A., Preneel, B.: MaSTer: Maliciously Secure Truncation for Replicated Secret Sharing Without Pre-processing. In: Kohlweiss, M., Di Pietro, R., Beresford, A. (eds.) Cryptology and Network Security, pp. 49–73. Springer, Singapore (2025). https://doi.org/10.1007/978-981-97-8013-6_3

[38] Zou, H., Xiao, Y., Zhang, R.: Semi-Honest 2-Party Faithful Truncation from Two-Bit Extraction. In: Wang, D., Yung, M., Liu, Z., Chen, X. (eds.) Information and Communications Security, pp. 215–234. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-7356-9_13

[39] Li, Y., Duan, Y., Huang, Z., Hong, C., Zhang, C., Song, Y.: Efficient 3PC for Binary Circuits with Application to Maliciously-Secure DNN Inference. In: 32nd USENIX Security Symposium (USENIX Security 23), pp. 5377–5394 (2023). https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun Accessed 2026-03-20

Appendix A Comprehensive comparison tables

This appendix contains a comprehensive comparison table corresponding to the discussion of related work in Chapter 2, while the main text focuses on identifying the most relevant prior works for this study. We list here the works that, to the best of our knowledge, explicitly provide relevant semantic mappings; works that only present high-level frameworks are excluded from our comparison. Since there is relatively little work on signed extensions, the complete comparison table has already been provided in Section 2.2 and will not be repeated here. Specifically, Appendix A.1 provides the complete comparison table for fixed-point multiplication, and Appendix A.2 provides the complete comparison table for signed truncation.

A.1 Fixed-point multiplication

The complete comparison for fixed-point multiplication is summarized in Table A1, covering the previous works known to us that are relevant to our target in the semantic, security, and applicability dimensions considered in this paper.

A.2 Signed truncation

The complete comparison for signed truncation is summarized in Table A2, covering the previous works known to us that are relevant to our target in the semantic, security, and applicability dimensions considered in this paper.

Appendix B Auxiliary materials for the relaxed variants

Due to space limitations, the supporting materials underlying the relaxed variants in Section 6.4 are not included in the main text. To keep the presentation there focused on the high-level ideas, protocol statements, and trade-offs of the variants, we collect these auxiliary components in this appendix.

Table A1 Comprehensive comparison table for fixed-point multiplication.

Work	Parties	Security	Domain	FP Sem.	Cross-Ring
Catrina et al. 2009[22]	n-party	not explicit*	$\mathbb{Z}_p$ (used as a field)	exact / approx.	no
Catrina et al. 2010[5]	n-party	semi-honest	$\mathbb{Z}_p$ (used as a field)	approx.	no
Liu et al. 2024[13]	n-party	semi-honest	$\mathbb{F}_p$	approx.	no
Zhang et al. 2025[23]	n-party	semi-honest	$\mathbb{F}_p$ (mapped to $\mathbb{Z}_{p,k}$)	approx.	no
Keller et al. 2022[24]	n-party	semi-honest**	$\mathbb{Z}_p$	approx.	no
Badrinarayanan et al. 2024[25]	n-party	semi-honest	$\mathbb{F}_p$	approx.	no
SecureML 2017[2]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	prob.	no
ABY3 2018[26]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	prob.	no
Trident 2020[27]	4-party	malicious	$\mathbb{Z}_{2^k}$	prob.	no
BLAZE 2020[28]	3-party	malicious	$\mathbb{Z}_{2^k}$	prob.	no
SWIFT 2021[29]	4-party	malicious	$\mathbb{Z}_{2^k}$	prob.	no
SWIFT 2021[29]	3-party	malicious	$\mathbb{Z}_{2^k}$	prob.	no
Tetrad 2022[30]	4-party	malicious	$\mathbb{Z}_{2^k}$	prob.	no
MPClan 2023[31]	$(2t+1)$-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	prob.	no
Keller et al. 2022[24]	3-party	semi-honest**	$\mathbb{Z}_{2^k}$	approx.	no
Badrinarayanan et al. 2024[25]	n-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	approx.	no
Cheetah 2022[16]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	approx.	no
Fantastic Four 2021[12]	4-party	malicious	$\mathbb{Z}_{2^k}$	approx.	no
Dalskov et al. 2020[32]	2-party	semi-honest/malicious	$\mathbb{Z}_{2^k}/\mathbb{F}_p$	exact/prob.	no
Dalskov et al. 2020[32]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}/\mathbb{F}_p$	exact/prob.	no
CrypTFlow2 2020[19]	2-party	semi-honest	$\mathbb{Z}_{2^k}/\mathbb{Z}_p$	exact	no
Falcon 2021[33]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^k}$	exact***	no
Frederiksen et al. 2024[34]	n-party	malicious/covert	$\mathbb{Z}_{pq}$	approx.	yes
ORCA 2024[35]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	approx.	yes
Ditto 2024[15]	3-party	semi-honest	$\mathbb{Z}_{2^k}$	approx.	yes†
Mosformer 2025[20]	3-party	malicious	$\mathbb{Z}_{2^k}$	approx.	yes†
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	exact	yes
LLAMA 2022[36]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	exact	yes
Guo et al. 2025[9]	2-party	semi-honest	$\mathbb{Z}_{2^k}$	exact/approx./prob.	yes†
Our Goal	**3-party**	**malicious**	$\mathbb{Z}_{2^k}$	**exact**	**yes**

Some works provide multiple substantially different relevant settings; they may therefore occupy multiple rows in the table.

Here, "prob." stands for probabilistic, where the protocol may output a completely incorrect result with small probability; "approx." stands for approximate, where the output may differ from the correct result by a small value; and "exact" denotes that the protocol guarantees an output with no deviation from the correct result.

† They are valid only when the input range is restricted.

* It satisfies statistical privacy for the truncation subprotocol.

** This paper benchmarks additional malicious backends, but the concrete lower-level protocols described are semi-honest.

*** It claims to have implemented precise truncation, but does not specify the protocol.

B.1 A three-party semi-honest AND protocol with arithmetic output

This protocol serves as the AND subprotocol invoked within the modified MW protocol used by the semi-honest signed extension variant in Section 6.4. In particular, only parties P_1 and P_2 hold the input bits $u, v \in \mathbb{Z}_2$, while the protocol outputs a two-party arithmetic additive sharing $[w]^K$ such that $w = u \wedge v$, where $K = 2^k$. The details of this protocol are provided in Algorithm 9.

Correctness. The correctness of $\Pi^{B\to K}_{\text{AND,3Semi}}$ is guaranteed by the following equations. Let $m_i = u_i v_i \oplus u_{(i \bmod 3)+1} v_i \oplus u_i v_{(i \bmod 3)+1}$, then $m_1 \oplus m_2 \oplus m_3 = u \wedge v$. Moreover, since $w_i = m_i \oplus r_i \oplus r_{(i \bmod 3)+1}$, the random masks cancel pairwise in the XOR sum,

Table A2 Comprehensive comparison table for signed truncation.

Work	Parties	Security	Input Domain	Output Domain	Truncation Type
Catrina et al. 2009[22]	n-party	not explicit*	$\mathbb{Z}_p$ (used as a field)	$\mathbb{Z}_p$	exact/approx.
Catrina et al. 2010[5]	n-party	semi-honest	$\mathbb{Z}_p$ (used as a field)	$\mathbb{Z}_p$	exact/approx.
Liu et al. 2024[13]	n-party	semi-honest	$\mathbb{F}_p$	$\mathbb{F}_p$	approx.
Zhang et al. 2025[23]	n-party	semi-honest	$\mathbb{F}_p$ (mapped to $\mathbb{Z}_{p,k}$)	$\mathbb{F}_p$	approx.
Frederiksen et al. 2024[34]	n-party	malicious/covert	$\mathbb{Z}_{pq}$	$\mathbb{Z}_{pq}$	approx.
ABY3 2018[26]	3-party	semi-honest/malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	prob.
BLAZE 2020[28]	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	prob.
SWIFT 2021[29]	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	prob.
SWIFT 2021[29]	4-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	prob.
MaSTer 2024[37]	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	prob.
Cheetah 2022[16]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	approx.
ORCA 2024[35]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	approx.
Fantastic Four 2021[12]	4-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/approx.
CrypTFlow2 2020[19]	2-party	semi-honest	$\mathbb{Z}_{2^l}/\mathbb{Z}_p$	$\mathbb{Z}_{2^l}/\mathbb{Z}_p$	exact
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact
Guo et al. 2025[9]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/approx./prob.†
Zou et al. 2023[38]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact
Harth-Kitzerow et al. 2025[6]	3-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/approx./prob.†
Harth-Kitzerow et al. 2025[6]	4-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/approx./prob.†
Escudero et al. 2020[21]	n-party	semi-honest/malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^l}$	exact/prob.
ORCA 2024[35]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	approx. (truncate-reduce)
Ditto 2024[15]	3-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	approx. (truncate-reduce)
Mosformer 2025[20]	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	approx. (truncate-reduce)
SIRNN 2021[14]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	exact (truncate-reduce)
LLAMA 2022[36]	2-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	exact (truncate-reduce)
Bicoptor 2.0 2024[17]	3-party	semi-honest	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	exact (truncate-reduce)
Escudero et al. 2020[21]**	3-party	malicious	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	exact (truncate-reduce)
Our Goal	**3-party**	**malicious**	$\mathbb{Z}_{2^l}$	$\mathbb{Z}_{2^{l-k}}$	**exact (truncate-reduce)**

Some works provide multiple substantially different relevant settings; they may therefore occupy multiple rows in the table.

Here, "prob." stands for probabilistic, where the protocol may output a completely incorrect result with small probability; "approx." stands for approximate, where the output may differ from the correct result by a small value; and "exact" denotes that the protocol guarantees an output with no deviation from the correct result.

† They are valid only when the input range is restricted.

* It satisfies the statistical privacy for truncation subprotocol.

** The original paper only gives the version whose output remains on the original ring, while this version with output on the smaller ring follows from a straightforward adaptation of the same protocol.

and hence $w_1 \oplus w_2 \oplus w_3 = u \wedge v$. Therefore, Step 6 gives a two-party Boolean additive sharing $[w]^B$ of $u \wedge v$. Finally, Steps 7–9 convert $[w]^B$ into a two-party arithmetic additive sharing $[w]^K$. Hence, the protocol correctly outputs an arithmetic sharing of $u \wedge v$ over $\mathbb{Z}_K$.

Online complexity. The online phase of $\Pi^{B\to K}_{\text{AND,3Semi}}$ exclusively involves bit communication and does not require invoking any subprotocols. In Step 1–2, party P_1 and P_2 each perform replicated secret sharing on the secret bits they held, sending a total of 8 bits. In Step 4, each party sends a random mask bit for a total of 3 bits. In Step 5, party P_3 sends a single-bit to party P_2. Finally, in Step 8, party P_1 and P_2 open a masked bit, sending a total of 2 bits. Hence, the online communication cost is $C^{B\to K}_{\text{AND,3Semi}} = 8 + 3 + 1 + 2 = 14$ bits.

Algorithm 9 : A three-party semi-honest AND protocol with arithmetic output $\Pi^{B\to K}_{\mathrm{AND},3\mathrm{Semi}}(P_1, P_2, P_3)$

Input: Party P_1 holds a bit $u \in \mathbb{Z}_2$, and party P_2 holds a bit $v \in \mathbb{Z}_2$.

Output: Parties P_1 and P_2 obtain a two-party arithmetic additive sharing $[w]^K$ of $w = u \wedge v$ over $\mathbb{Z}_K$, where $K = 2^k$.

Preprocessing: Parties P_1 and P_2 hold a random daBit consisting of a Boolean additive sharing $[R]^B$ and an arithmetic additive sharing $[R]^K$ of the same random bit $R \in \{0, 1\}$.

1: Party P_1 samples $u_2, u_3 \in \mathbb{Z}_2$ uniformly at random and sets $u_1 \leftarrow u \oplus u_2 \oplus u_3$. It then sends (u_2, u_3) to P_2 and (u_3, u_1) to P_3. At this point, the parties hold a replicated Boolean sharing $[\![u]\!]^B = (u_1, u_2, u_3)$ of u, where P_1 holds (u_1, u_2), P_2 holds (u_2, u_3), and P_3 holds (u_3, u_1).

2: Party P_2 samples $v_1, v_2 \in \mathbb{Z}_2$ uniformly at random and sets $v_3 \leftarrow v \oplus v_1 \oplus v_2$. It then sends (v_1, v_2) to P_1 and (v_3, v_1) to P_3. At this point, the parties hold a replicated Boolean sharing $[\![v]\!]^B = (v_1, v_2, v_3)$ of v, where P_1 holds (v_1, v_2), P_2 holds (v_2, v_3), and P_3 holds (v_3, v_1).

3: For $i \in \{1, 2, 3\}$, party P_i locally computes $m_i \leftarrow u_i v_i \oplus u_{(i \bmod 3)+1} v_i \oplus u_i v_{(i \bmod 3)+1}$.

4: For $i \in \{1, 2, 3\}$, party P_i samples a random bit $r_i \in \mathbb{Z}_2$ and sends r_i to $P_{((i-2) \bmod 3)+1}$.

5: For $i \in \{1, 2, 3\}$, after receiving $r_{(i \bmod 3)+1}$, party P_i locally computes $w_i \leftarrow m_i \oplus r_i \oplus r_{(i \bmod 3)+1}$ and party P_3 sends w_3 to P_2.

6: Party P_1 locally sets $[w]^B_1 \leftarrow w_1$, and party P_2 locally sets $[w]^B_2 \leftarrow w_2 \oplus w_3$. Thus, $[w]^B$ forms a two-party Boolean additive sharing of w.

7: For $i \in \{1, 2\}$, party P_i locally computes $[c]^B_i \leftarrow [w]^B_i \oplus [R]^B_i$.

8: Parties P_1 and P_2 then open the masked bit c.

9: **if** $c = 0$ **then**

10: Party P_1 outputs $[R]^K_1$ and party P_2 outputs $[R]^K_2$.

11: **else**

12: Party P_1 outputs $1 - [R]^K_1 \pmod K$ and party P_2 outputs $-[R]^K_2 \pmod K$.

13: **end if**

Security. The protocol $\Pi^{B\to K}_{\mathrm{AND},3\mathrm{Semi}}$ UC-securely realizes the arithmetic-output AND functionality $\mathcal{F}^{B\to K}_{\mathrm{AND}}$ in the $\mathcal{F}_{\mathrm{daBit}}$-hybrid model against static semi-honest corruption of at most one party.

Proof sketch. If party P_1 or P_2 is corrupted, every message the corrupted party received before the final opening is either a uniformly random share of the other input or a value one-time masked by fresh random bits unknown to the corrupted party, and the opened bit $c = w \oplus R$ is masked by the random daBit bit R, and therefore reveals no additional information about $w = u \wedge v$. If party P_3 is corrupted, then it has neither input nor output, and its view consists only of random replicated-share components of the inputs together with one-time masked values. Hence, in all cases, the corrupted party learns nothing beyond the information allowed by the ideal function $\mathcal{F}^{B\to K}_{\mathrm{AND}}$.

B.2 Modified MW protocols for semi-honest signed extension

This protocol serves as the MW subprotocol invoked by the semi-honest signed extension variant in Section 6.4. It is directly adapted from Guo's two-party MW protocol[9]. The main modification of this protocol is that we use the three-party AND protocol with arithmetic output provided in Appendix B.1, replacing the original bit multiplication protocol for efficiency. The details of this protocol are provided in Algorithm 10.

Algorithm 10 : A modified MW protocol for semi-honest signed extension $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}(P_1, P_2, P_3)$

Input: Parties P_1 and P_2 hold a two-party additive sharing $[x]^M$ of an input $x \in \mathbb{Z}_M$, where $M = 2^m$ and $|x| < M/3$.
Output: Parties P_1 and P_2 obtain a two-party arithmetic additive sharing $[MW]^K$ of $MW = \mathrm{wrap} + \mathrm{MSB}_m(\bar{x})$ over $\mathbb{Z}_K$, where $K = 2^k$ and $k \geq 2$.
Preprocessing: The preprocessing required by $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ is specified in the corresponding subprotocol.

1: Party P_1 locally computes $a_1 = \mathbf{1}\{[x]^M_1 < M/3\}$ and $a_2 = \mathbf{1}\{[x]^M_1 \geq 2M/3\}$.
2: Party P_2 locally computes $b_1 = \mathbf{1}\{[x]^M_2 < M/3\}$ and $b_2 = \mathbf{1}\{[x]^M_2 \geq 2M/3\}$.
3: Parties P_1, P_2 invoke $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ with the support of P_3 on inputs a_1 and b_1, obtaining $[c_1]^K$, where $c_1 = a_1 \wedge b_1$.
4: Parties P_1, P_2 invoke $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ with the support of P_3 on inputs a_2 and b_2, obtaining $[c_2]^K$, where $c_2 = a_2 \wedge b_2$.
5: Parties P_1 and P_2 locally compute $[MW]^K = 1 - [c_1]^K + [c_2]^K$.
6: Parties P_1 and P_2 output $[MW]^K$.

Correctness. The correctness of $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ is guaranteed directly by Guo's original two-party MW protocol and the correctness of the substituted AND protocol $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$.

Online complexity. The online phase of $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ involves the three-party AND protocol $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ twice, each of which incurs a constant communication cost. Accordingly, the online communication cost of this protocol is $C^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}} = 2 \cdot C^{B\to K}_{\mathrm{AND,3Semi}} = 2 \times 14 = 28$ bits.

Security. The protocol $\Pi^{M,\langle 1/3\rangle}_{\mathrm{MW,3Semi}}$ UC-securely realizes the calculating MW functionality $\mathcal{F}^{M,\langle 1/3\rangle}_{\mathrm{MW}}$ with restricted input in the $\mathcal{F}^{B\to K}_{\mathrm{AND}}$-hybrid model against static semi-honest corruption of at most one party.

Proof sketch. We first consider the hybrid protocol $\Pi^{M,\langle 1/3\rangle *}_{\mathrm{MW,3Semi}}$, in which the two calls to $\Pi^{B\to K}_{\mathrm{AND,3Semi}}$ are replaced by the ideal functionality $\mathcal{F}^{B\to K}_{\mathrm{AND}}$. If party P_1 or P_2 is corrupted, the only non-local steps are the two calls to the ideal functionality $\mathcal{F}^{B\to K}_{\mathrm{AND}}$. If party P_3 is corrupted, then it has neither input nor output in this protocol, and its entire view is already captured by the two ideal calls. Hence, in all cases, the corrupted party learns nothing beyond the information allowed by the

ideal functionality $\mathcal{F}_{\mathrm{MW}}^{M,\langle 1/3\rangle}$ in the hybrid protocol. By the subprotocol composition theorem, together with the fact that $\Pi_{\mathrm{AND,3Semi}}^{B\to K}$ UC-securely realizes $\mathcal{F}_{\mathrm{AND}}^{B\to K}$, the original protocol $\Pi_{\mathrm{MW,3Semi}}^{M,\langle 1/3\rangle}$ is also UC-secure.

B.3 Distributional motivation for the constant correction +1

Recall that $y_{ex} - \tilde{y} = c$, where $c = \lfloor (r_1 + r_2 + r_3)/K \rfloor \in \{0, 1, 2\}$. The three possible deviations $c \in \{0, 1, 2\}$ between local truncation and exact truncation occur with different probabilities under the standard 3PC replicated-sharing distribution, and these probabilities can be computed as follows.

Let $K = 2^k$, and let $x_{\mathrm{low}} := x \pmod K$ denote the low-k bits of the input. Under the standard three-party replicated-sharing setting, one may view x_1 and x_2 as independent and uniformly distributed over $\mathbb{Z}_{2^l}$, while $x_3 = x - x_1 - x_2 \pmod{2^l}$. Hence, $r_1 = x_1 \pmod K$ and $r_2 = x_2 \pmod K$ are independent and uniform over $\{0, 1, \ldots, K-1\}$, whereas r_3 is uniquely determined by $r_3 = x_{\mathrm{low}} - r_1 - r_2 \pmod K$.

Define $u := r_1 + r_2$. Then u follows a discrete triangular distribution with probability mass function

$$\Pr[u = t] = \begin{cases} \dfrac{t+1}{K^2}, & 0 \le t \le K-1, \\ \dfrac{2K-1-t}{K^2}, & K \le t \le 2K-2. \end{cases} \tag{B1}$$

Meanwhile, since $r_3 = x_{\mathrm{low}} - r_1 - r_2 \pmod K$, the event $c = 0$ is equivalent to $u \le x_{\mathrm{low}}$, while the event $c = 2$ is equivalent to $u > x_{\mathrm{low}} + K$; the remaining case corresponds to $c = 1$. Therefore, the probabilities of each discrepancy between the local-truncated result and the exact-truncated result can be calculated as follows:

$$\Pr[c = 0 \mid x] = \Pr[u \le x_{\mathrm{low}}] = \sum_{t=0}^{x_{\mathrm{low}}} \frac{t+1}{K^2} = \frac{(x_{\mathrm{low}}+1)(x_{\mathrm{low}}+2)}{2K^2}, \tag{B2}$$

$$\Pr[c = 2 \mid x] = \Pr[u > x_{\mathrm{low}}+K] = \sum_{t=x_{\mathrm{low}}+K+1}^{2K-2} \frac{2K-1-t}{K^2} = \frac{(K-x_{\mathrm{low}}-1)(K-x_{\mathrm{low}}-2)}{2K^2}, \tag{B3}$$

$$\Pr[c = 1 \mid x] = 1 - \Pr[c = 0 \mid x] - \Pr[c = 2 \mid x]. \tag{B4}$$

The expressions above already suffice to compare the three fixed-constant corrections. In particular, if one outputs "local truncation $+a$" for some fixed $a \in \{0, 1, 2\}$, then the output coincides with the exact truncation result if and only if $c = a$. Accordingly, the success probabilities of the three fixed-constant choices are precisely $\Pr[c = 0 \mid x]$, $\Pr[c = 1 \mid x]$, and $\Pr[c = 2 \mid x]$.

Moreover, by directly comparing the conditional probabilities in (B2)–(B4), one obtains

$$\Pr[c = 1 \mid x] - \Pr[c = 2 \mid x] = \frac{4Kx_{\mathrm{low}} + 6K - 3x_{\mathrm{low}}^2 - 9x_{\mathrm{low}} - 6}{2K^2}, \tag{B5}$$

which is positive for all $x_{\text{low}} \in \{0, 1, \ldots, K-1\}$. Similarly, one obtains

$$\Pr[c = 1 \mid x] - \Pr[c = 0 \mid x] = \frac{K^2 + 2Kx_{\text{low}} + 3K - 3x_{\text{low}}^2 - 9x_{\text{low}} - 6}{2K^2}, \tag{B6}$$

which is positive for all $x_{\text{low}} \in \{0, 1, \ldots, K-2\}$, and is negative only at the single boundary value $x_{\text{low}} = K - 1$.

Therefore, among the three fixed-constant corrections $\{0, 1, 2\}$, the correction "+1" has the highest probability of matching the exact truncation result for every low-k-bit pattern except the only upper-boundary value $x_{\text{low}} = K - 1$, where "+0" is preferred. However, in typical cases, there is no reason to expect that the lower k bits of a large number of inputs will all be 1. Accordingly, from a statistical viewpoint, outputting "local truncation +1" should have the highest probability of coinciding with the exact truncation result among fixed-constant choices. And at the same time, it also guarantees that the final output differs from the exact value by at most 1 ULP.

Appendix C UC Proof of Corollary 7.3

Proof. Since the corruption threshold in honest majority 3PC setting is one, there are four possible corruption patterns, namely no corruption and the three single-party corruptions P_1, P_2, and P_3. Since three parties play completely symmetric roles in the protocol $\Pi^{M_1,M_2,\tau}_{\text{FMul,3HMal}}$, it is sufficient to consider the case where P_1 is corrupted. The cases where P_2 or P_3 is corrupted can be proved in the same way, and the case of no corruption is obvious. And the protocol itself is a sequential composition of two signed extensions, one arithmetic multiplication, and one exact signed truncation.

Let S be the simulator in the ideal execution with $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$. The simulator acts as follows:

1. The environment $\mathcal{Z}$ provides the replicated input shares of the corrupted party P_1 in $[\![x]\!]^{M_1}$ and $[\![x']\!]^{M_2}$ to the simulator S. Then, S forwards the replicated input shares of P_1 to the ideal functionality $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$. Once the output of $\mathcal{F}^{M_1,M_2,\tau}_{\text{FMul}}$ is available, S records the local arithmetic view of P_1 in the replicated arithmetic sharing $[\![y]\!]^{2^{m_1+m_2-\tau}}$ of the target-functionality output, denoted by $([\![y]\!]_1^{2^{m_1+m_2-\tau}}, [\![y]\!]_2^{2^{m_1+m_2-\tau}})$.
2. The simulator S locally computes the local arithmetic view of P_1 in the corresponding base value $[\![\tilde{x}]\!]^{2^{m_1+m_2}}$ that the corrupted party should hold under the quotient-correction instantiation of signed extension, denoted by $([\![\tilde{x}]\!]_1^{2^{m_1+m_2}}, [\![\tilde{x}]\!]_2^{2^{m_1+m_2}})$. To be more specific, this is obtained by naturally interpreting the same local tuple of P_1 over the target ring $\mathbb{Z}_{2^{m_1+m_2}}$. Then, the simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![q_x]\!]^{2^{m_1+m_2}}$ of the bounded quotient q_x, denoted by $([\![q_x]\!]_1^{2^{m_1+m_2}}, [\![q_x]\!]_2^{2^{m_1+m_2}})$. After that, the simulator S computes and sets $([\![\hat{x}]\!]_1^{2^{m_1+m_2}}, [\![\hat{x}]\!]_2^{2^{m_1+m_2}}) := ([\![\tilde{x}]\!]_1^{2^{m_1+m_2}} - M_1 \cdot [\![q_x]\!]_1^{2^{m_1+m_2}}, [\![\tilde{x}]\!]_2^{2^{m_1+m_2}} - M_1 \cdot [\![q_x]\!]_2^{2^{m_1+m_2}})$ as the local arithmetic output view that P_1 would obtain from $\mathcal{F}^{M_1,2^{m_1+m_2}}_{\text{SExt}}$ on input $[\![x]\!]^{M_1}$ in the hybrid execution.

3. The simulator S simulates the call of $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$ in the same way, and delivers the resulting local arithmetic output view of P_1 in $[\![\hat{x}']\!]^{2^{m_1+m_2}}$, denoted by $([\![\hat{x}']\!]_1^{2^{m_1+m_2}}, [\![\hat{x}']\!]_2^{2^{m_1+m_2}})$.
4. The simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![p]\!]^{2^{m_1+m_2}}$, denoted by $([\![p]\!]_1^{2^{m_1+m_2}}, [\![p]\!]_2^{2^{m_1+m_2}})$, as the local arithmetic output view that P_1 would obtain from the call of $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$ in the hybrid execution.
5. The simulator S delivers to the environment, in the same order as in the hybrid execution, the local arithmetic view of P_1 in $[\![\hat{x}]\!]^{2^{m_1+m_2}}$ from the invocation of $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$, the local arithmetic view of P_1 in $[\![\hat{x}']\!]^{2^{m_1+m_2}}$ from the invocation of $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$, the local arithmetic view of P_1 in $[\![p]\!]^{2^{m_1+m_2}}$ from the invocation of $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$, and finally the local output view of P_1 in $[\![y]\!]^{2^{m_1+m_2-\tau}}$.

The above simulator S works with the ideal functionality $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$, and $\mathcal{F}_{\mathrm{Trunc}}^{2^{m_1+m_2},\tau}$. In the hybrid execution, the non-local transcript visible at the protocol level only involves the local outputs returned by $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$, and $\mathcal{F}_{\mathrm{Trunc}}^{2^{m_1+m_2},\tau}$. For the invocation of $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$, the simulator S locally reconstructs the local arithmetic view of P_1 in the corresponding base value $[\![\tilde{x}]\!]^{2^{m_1+m_2}}$ from the replicated input shares of P_1 in $[\![x]\!]^{M_1}$. Then, since at most one party can be corrupt, the corrupted party's local arithmetic view in $[\![q_x]\!]^{2^{m_1+m_2}}$ is distributed independently of the underlying actual quotient value, the simulator S samples a fresh local arithmetic view of P_1 in the replicated arithmetic sharing $[\![q_x]\!]^{2^{m_1+m_2}}$ of the bounded quotient q_x. Therefore, after applying the same local linear correction rule as in the quotient-correction instantiation of signed extension, the resulting local arithmetic output view of P_1 in $[\![\hat{x}]\!]^{2^{m_1+m_2}}$ has exactly the same distribution as the local output view that P_1 would obtain from the call of $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$ in the hybrid execution. Naturally, the same argument still holds for the call to $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$, and therefore the simulated local arithmetic output view of P_1 in $[\![\hat{x}']\!]^{2^{m_1+m_2}}$ also has exactly the same distribution as in the hybrid execution.

For the invocation of $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$, since the output of $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$ is a fresh replicated arithmetic sharing of the product, the fresh sampled local arithmetic view of P_1 in the replicated arithmetic sharing $[\![p]\!]^{2^{m_1+m_2}}$ has exactly the same distribution as the local output view that P_1 would obtain in the hybrid execution. For the invocation of $\mathcal{F}_{\mathrm{Trunc}}^{2^{m_1+m_2},\tau}$, the local arithmetic output view of P_1 is exactly the final local arithmetic output view of P_1 in $[\![y]\!]^{2^{m_1+m_2-\tau}}$, and the simulator therefore directly delivers this view.

For any fixed local arithmetic view of P_1 in the multiplication output $[\![p]\!]^{2^{m_1+m_2}}$, denoted by $([\![p]\!]_1^{2^{m_1+m_2}}, [\![p]\!]_2^{2^{m_1+m_2}})$, the local arithmetic view of P_1 in the corresponding base-value $[\![\tilde{y}]\!]^{2^{m_1+m_2-\tau}}$, denoted by $([\![\tilde{y}]\!]_1^{2^{m_1+m_2-\tau}}, [\![\tilde{y}]\!]_2^{2^{m_1+m_2-\tau}})$, is also fixed. However, since exact signed truncation is a direct instantiation of the quotient-correction functionality with public correction constant $C = 1$, and its output shares follow that $[\![y]\!]^{2^{m_1+m_2-\tau}} = [\![\tilde{y}]\!]^{2^{m_1+m_2-\tau}} + [\![q_{\mathrm{tr}}]\!]^{2^{m_1+m_2-\tau}}$. Hence, for any fixed local

arithmetic views of P_1 in $[\![\tilde{y}]\!]^{2^{m_1+m_2-\tau}}$ and $[\![y]\!]^{2^{m_1+m_2-\tau}}$, there is a unique local arithmetic view $([\![q_{\mathrm{tr}}]\!]_1^{2^{m_1+m_2-\tau}}, [\![q_{\mathrm{tr}}]\!]_2^{2^{m_1+m_2-\tau}})$ such that the above relationship holds. Moreover, for each fixed value q_{tr}, the local arithmetic view of P_1 in a fresh replicated arithmetic sharing of q_{tr} is uniformly distributed over $\mathbb{Z}_{2^{m_1+m_2-\tau}}$, and is therefore independent of the actual probability mass function of q_{tr}. Consequently, the conditional distribution is exactly the same between the local arithmetic view of P_1 in $[\![p]\!]^{2^{m_1+m_2}}$ and the final local arithmetic output view of P_1 in $[\![y]\!]^{2^{m_1+m_2-\tau}}$.

Therefore, the simulated local arithmetic output views of P_1 in $[\![\hat{x}]\!]^{2^{m_1+m_2}}$, $[\![\hat{x}']\!]^{2^{m_1+m_2}}$, $[\![p]\!]^{2^{m_1+m_2}}$, and $[\![y]\!]^{2^{m_1+m_2-\tau}}$ have the same joint distribution as in the hybrid execution. Hence, with $\mathcal{F}_{\mathrm{SExt}}^{M_1,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{SExt}}^{M_2,2^{m_1+m_2}}$, $\mathcal{F}_{\mathrm{Mul}}^{2^{m_1+m_2}}$, and $\mathcal{F}_{\mathrm{Trunc}}^{2^{m_1+m_2},\tau}$ treated as ideal functionalities, the simulator described above introduces no additional distinguishing probability, which means that for every well-behaved environment $\mathcal{Z}$, it follows that

$$\left|\Pr\left[\mathrm{Exec}\left[\Pi_{\mathrm{FMul,3HMal}}^{M_1,M_2,\tau}(\mathcal{F}_{\mathrm{SExt}}^{M_1,P}, \mathcal{F}_{\mathrm{SExt}}^{M_2,P}, \mathcal{F}_{\mathrm{Mul}}^{P}, \mathcal{F}_{\mathrm{Trunc}}^{P,\tau}), A_{\mathrm{triv}}, \mathcal{Z}\right] = 1\right]\right.$$
$$\left. - \Pr\left[\mathrm{Exec}\left[\mathcal{F}_{\mathrm{FMul}}^{M_1,M_2,\tau}, S, \mathcal{Z}\right] = 1\right]\right| = 0.$$

□