Actively Secure MPC with $O(|C|)$ Computation and Communication via CRT

Alexander Bienstock[1], Daniel Escudero[2], Antigoni Polychroniadou[1]

[1] JPMorgan AI Research & AlgoCRYPT CoE
[2] TACEO

Abstract. Secure multiparty computation (MPC) allows n parties to compute a function of their private inputs, so that nothing beyond the output of the function is revealed. In the sub-optimal honest majority setting in which the number of corrupted parties $t < (1/2-\varepsilon)n$, the works of Goyal *et al.* (CRYPTO'21 and CRYPTO'22), achieved $O(|C|)$ communication even against active adversaries, but with $\Omega(n \cdot |C|)$ computation, where C is the arithmetic circuit computed by the MPC. Recent work by Garg *et al.* (CRYPTO'24) showed that both $O(|C|)$ communication *and* computation can be achieved in this regime, however, only against passive adversaries. In this work, we achieve the best-of-both-worlds by obtaining MPC with $O(|C|)$ communication *and* computation against active corruption of $t < (1/2-\varepsilon)n$ parties. To do this, we introduce novel techniques for actively-secure MPC constructed from Chinese Remainder Theorem based secret sharing.

1 Introduction

Secure multiparty computation (MPC) enables a set of n parties to compute a function on private inputs without leaking anything beyond the output of the computation itself. Indeed, this should hold even if there is an adversary that corrupts t of the parties in the computation. This adversary can either be *passive* or *active*. In the former case, the adversary can view the messages that the t corrupted parties receive from the honest parties during the execution of the protocol, but must follow the description of the protocol itself when sending messages to the honest parties. In the latter case, the adversary, in addition to viewing the messages that the t corrupted parties receive from the honest parties, can send any messages of its choosing to the honest parties, in an attempt to infer information about honest parties' inputs, or cause them to output the incorrect values. While it is important to study both passive and active adversaries, constructing protocols for active adversaries is of course more challenging and general than those that are just passively-secure.

It is also important to choose a corruption threshold on the number of parties t that the adversary can corrupt. In this work, we focus on the *honest majority* setting, and in particular, $t < (1/2 - \varepsilon)n$, for some constant $0 < \varepsilon < 1/2$. There has been a long line of works in this setting developing both passively and actively secure MPC protocols, beginning with [BGW88; CCD88;

RB89], to name a few. Protocols in this setting can achieve *statistical* security against *unbounded* adversaries (rather than weaker computational security against bounded adversaries), where λ is the statistical security parameter. Statistical security is thus typically the goal.

When developing MPC protocols, a main goal is to ensure that they are efficient. Most statistically-secure MPC protocols view the function to be computed as an arithmetic circuit C over some finite field $\mathbb{F}$ with addition and multiplication gates. Two primary efficiency measures that are typically considered are communication and computational complexity, both in terms of the number of gates in $|C|$. There have been major efforts in trying to bring down the communication complexity, in particular, of statistically-secure MPC protocols, culminating in recent protocols with $O(|C|)$ communication [Bec+21; DIK10; Esc+23; FY92; GIP15; GPS21; GPS22; GSY21].[3] Indeed, it is known how to achieve this communication complexity even against active adversaries, with abort. I.e., the honest parties must output the correct answer, but they can abort if the corrupted parties do not follow the protocol description (stronger guaranteed output delivery with this communication complexity remains open, as far as we are aware). However, all of these protocols suffer from $\Omega(n|C|)$ computational overhead for general circuits.

To fill this gap, the recent work by Garg *et al.* [Gar+24] achieves *both* $O(|C|)$ communication *and* computation. However, their protocol only achieves passive security. This leads to the following conundrum:

Can we achieve a statistically-secure with abort MPC protocol against active adversaries with both $O(|C|)$ communication and computation?

1.1 Our Contribution

In this work, we answer the above question affirmatively. In particular, by developing novel techniques for actively-secure MPC, we are able to extend the work by Garg *et al.* [Gar+24] to achieve active security with abort, while maintaining $O(|C|)$ communication and computation. As their protocol only achieves the stated complexity for fields $\mathbb{F}$ where $\log|\mathbb{F}| = \Omega(n^2 \log n)$, the same restriction applies for ours.

Theorem 1 (Informal). [4] *Let n be the number of parties. Then for any constant $0 < \varepsilon < 1/2$ and any field $\mathbb{F}$ such that $\log|\mathbb{F}| = \Omega(n^2 \log n)$, there is a statistically-secure with abort MPC protocol against $t < (1/2 - \varepsilon)n$ active corruptions that can securely compute any arithmetic circuit C over $\mathbb{F}$ with $O(|C| \log|\mathbb{F}|)$ bits of communication and bit computations.*

The main insight that Garg *et al.* [Gar+24] has for their passively-secure protocol is to use Chinese Remainder Theorem (CRT) based secret sharing [AB83;

[3] In fact, this communication is with respect to only multiplication gates, since addition gates can typically be handled without any interaction.

[4] We ignore λ, $\log n$, and $\log\log|\mathbb{F}|$ factors throughout.

Gar+23b; GRS99; Mig83], in which a *single* secret in $\mathbb{F}$ is shared such that the sum of the sizes of the shares of the parties is $O(\log|\mathbb{F}|)$. Therefore, there is no overhead with respect to the size of the secret, and thus performing operations for the MPC on these shares introduces no (asymptotic) overhead over just computing the circuit in the clear. However, most of the statistically-secure MPC literature to date does not use such secret sharing, and thus how to achieve active security is unclear. Our main technical contribution, overviewed in Section 2, is thus to introduce several novel techniques to achieve active security for CRT-based MPC.

MPC Over Large Fields. As with the work of Garg *et al.* [Gar+24], we require that $\log|\mathbb{F}| = \Omega(n^2 \log n)$. However, as discussed in [Gar+24], there are numerous applications in which such a field size is realistic. One primary example is the private outsourcing of computationally-intensive cryptographic algorithms, such as the generation of zero-knowledge succinct non-interactive arguments (zkSNARGs) [Chi+23; Fan+25; Gar+23a; Gar+25; Hu+25; Liu+24; Liu+25; OB22; Yan+25]. In this application, the underlying field is usually large, specifically $\log|\mathbb{F}|$ is proportional to the (computational) security parameter, which would allow for the use of our protocol with tens of servers. See the discussion in [Gar+24] for more details.

1.2 Related Work

The works of [BGW88; CCD88; RB89] initiated the study of statistically-secure MPC in the honest majority setting. These early works required large complexity, typically at least n^2 total communication and computation per gate. For the *optimal* honest majority setting of $t < n/2$, [DN07] was able to achieve statistical security with abort against active adversaries with $O(n|C|)$ total communication. The work of [DLN19] showed that this communication is necessary if t can be large enough such that $n = 2t + 1$. Through the use of packed secret sharing [FY92], a line of works was able to reduce this overhead to $O(|C|)$ communication, for the *sub-optimal* regime, $t < (1/2 - \varepsilon)n$, but with $\Omega(n|C|)$ computation [Bec+21; DIK10; Esc+23; FY92; GIP15; GPS21; GPS22; GSY21]. These works only achieve active security with abort; guaranteed output delivery with this communication complexity remains open, as far as we are aware. Garg *et al.* [Gar+24] were able to achieve $O(|C|)$ communication and computation against passive adversaries for $t < (1/2 - \varepsilon)n$.

As in [Gar+24], we rely on CRT-based secret sharing [AB83; Gar+23b; GRS99; Mig83] to obtain our results. CRT-based secret sharing was also used by [Gar+23b] for *weighted* MPC, i.e., each party has an associated weight, and the corruption threshold is determined by the sum of the weights of the corrupted parties. They show that the use of CRT-based secret sharing is better than typical linear secret sharing schemes for achieving this notion. The work of [Gar+24] (which we build upon) builds on this work.

2 Technical Overview

We present an overview of the techniques we use to achieve statistically-secure with abort MPC of an arithmetic circuit C over the field $\mathbb{F}_{p_0}$ with $O(|C| \cdot \log p_0)$ bits of communication and bit-level operations via CRT secret sharing. In this work, we use the *signed* representation of any given ring $\mathbb{Z}_p$, denoted by $\mathbb{Z}_p^{\pm} := [\lfloor -(p-1)/2 \rfloor, \lfloor (p-1)/2 \rfloor]$; at times we will omit the $^{\pm}$, when it is inconsequential to the techniques.[5] We will let $|x|$ represent the absolute value of an element $x \in \mathbb{Z}_p^{\pm}$, i.e., $|x| := x$ if $x \geq 0$ and $|x| := -x$ if $x < 0$. Let n be the number of parties and t the number of corruptions. Let λ be the statistical security parameter.

2.1 Recap of Passively-Secure Protocol [Gar+24]

To explain our techniques to achieve active security, we start from the passively-secure protocol of [Gar+24] and then in the next section observe how an active adversary can cause problems. The main achievement of [Gar+24] is achieving passively-secure MPC with not only $O(|C| \cdot \log p_0)$ bits of communication, but also *computation*. Although a line of previous works [Bec+21; DIK10; FY92; GIP15; GPS21; GSY21] was able to achieve this communication using *packed secret sharing* [FY92], they all required $\Omega(n|C| \log p_0)$ computation for general circuits C. The main reason for the high computational overhead in these works is the use of packed secret sharing. Packed secret sharing allows $\Omega(n)$ secrets in $\mathbb{F}_{p_0}$ to be shared amongst the n parties at once, with shares of size $\log p_0$. Therefore, the amortized cost of sharing a secret is indeed $O(1)$, which allows for the efficient communication in these protocols. However, since many secrets are shared at once, one must use packed secret sharing in the MPC to evaluate *many* multiplication gates at once. The high computational cost is due to the need to properly arrange the secrets in a given packed secret sharing so that multiplying with another packed secret sharing actually corresponds to real multiplication gates in the circuit C.

"Unpacked" CRT Secret Sharing. A key insight of [Gar+24] is to swap out packed secret sharing for CRT secret sharing [AB83; Gar+23b; GRS99; Mig83], which one can view as "unpacked" secret sharing. Indeed, CRT secret sharing allows a single secret in $\mathbb{F}_{p_0}$ to be shared amongst the n parties, where the *sum* of the share sizes is $\log(p_0)$.[6] I.e., the secret is "unpacked" to the n parties. Thus, the cost of sharing a secret via CRT secret sharing indeed maintains $O(1)$ overhead and the need to arrange the vector of secrets for evaluating multiplication gates is removed.

CRT-based secret sharing works as follows. Let $p_1, \ldots, p_n$ be numbers that are pairwise coprime (also to p_0). To share a secret $x \in \mathbb{F}_{p_0}^{\pm}$, one samples $u \leftarrow_{\$} \mathbb{Z}_L^{\pm}$,

[5] [Gar+24] does not use signed representation, but it is more convenient for us.

[6] This is one reason why sub-optimal corruption threshold $t < (1/2 - \varepsilon)n$ is used—if it were the case that $t + 1$ shares determine the secret but t shares reveal nothing, one would need shares of size $\log(p_0)$ instead of $\log(p_0)/n$.

for some large L, and then computes the integer $X \leftarrow x + u \cdot p_0$. Note that clearly $X \bmod p_0 \equiv x$. Then, the secret share of party $\mathcal{P}_i$ is $x^i \leftarrow X \bmod p_i$. To reconstruct the secret via the shares x^i, one uses the Chinese Remainder Theorem to compute the integer X and then $x \leftarrow X \bmod p_0$ (followed by converting to signed representation). In more detail, for correctness, it must be the case that $\prod_{i=1}^{n} p_i \geq 2|X|$. This is because during reconstruction, the computed integer X is really $X \in \mathbb{Z}^{\pm}_{\prod_{i=1}^{n} p_i}$. For privacy, L must be big enough to mask the secret x if the adversary obtains at most t shares x^i (see Section 3.1 for more details). However, one can set L so that privacy is achieved but still $\sum_{i=1}^{n} \log p_i = O(\log p_0)$, so that sharing indeed has $O(1)$ overhead with respect to the secret size.[7] We will refer to a CRT secret sharing of a secret x via integer X as $[\![X]\!]$.

Passively-Secure MPC via CRT Secret Sharing. Garg *et al.* [Gar+24] continue to build MPC via CRT secret sharing using the standard "king" multiplication technique of [DN07] commonly used through information-theoretically secure MPC. First, [Gar+24] observe that although CRT secret sharing does not have linear reconstruction (unlike Shamir and packed secret sharing), it does satisfy a "local homomorphism". Indeed, given sharings $[\![X]\!]$ and $[\![Y]\!]$, of secrets x and y, respectively, if the parties locally add their shares of $[\![X]\!]$ and $[\![Y]\!]$ together (resp. multiply their shares together), then they will obtain shares of $x + y$ (resp. $x \cdot y$). We denote this addition of shares as $[\![X]\!] \oplus [\![Y]\!]$ (resp. $[\![X]\!] \odot [\![Y]\!]$), where $\oplus$ (resp. $\odot$) represents each party performing the addition (resp. multiplication) modulo their prime p_i.

However, recall that the integer X of a CRT secret sharing must satisfy that $2|X| \leq \prod_{i=1}^{n} p_i$. If not, then CRT reconstruction may result in an element $\tilde{X} \in \mathbb{Z}_{\prod_{i=1}^{n} p_i}$, in which $\tilde{X} \bmod p_0 \not\equiv x$, due to overflow modulo $\prod_{i=1}^{n} p_i$. Thus, to obtain correctness in their MPC protocol, [Gar+24] use an "integer-reduction" step after multiplying $[\![X]\!] \odot [\![Y]\!]$, akin to the "degree-reduction" step needed in Shamir-based "king" multiplication technique [DN07].

In more detail, the parties first jointly sample random sharings $[\![R]\!]$ and $\langle \overline{R} \rangle$ (see below for explanation of new notation) such that $R \bmod p_0 \equiv \overline{R}$. A main contribution of [Gar+24] is showing how to do this by extending the typical Vandermonde matrix randomness extraction procedure [DN07] to distributions *over the integers*; a side-effect of this technique is the requirement that $\log p_0 = \Omega(n^2 \log n)$ (see Lemma 3 and Section 3.4 for more details). The parties then compute $\langle D \rangle \leftarrow [\![X]\!] \odot [\![Y]\!] \oplus \langle \overline{R} \rangle$ and open $\langle D \rangle$ to the "king", party $\mathcal{P}_1$. Since the size of the integer representing $[\![X]\!] \odot [\![Y]\!]$ is much larger after the multiplication of shares, we need to mask it with the random sharing $\langle \overline{R} \rangle$ whose underlying integer $\overline{R}$ is larger than usual sharings, hence the new notation $\langle \cdot \rangle$. The king $\mathcal{P}_1$ then reconstructs $d \leftarrow D \bmod p_0$ and redistributes $d_i \leftarrow d \bmod p_i$ to each other party $\mathcal{P}_i$. We denote such a "sharing" of a public value (without a random mask) as $[d]$. Finally, the parties compute $[\![Z]\!] \leftarrow [d] \oplus (-[\![R]\!])$. Since $R \bmod p_0 \equiv \overline{R}$, if the adversary is passive, it is clear that $Z \bmod p_0 \equiv x \cdot y$.

[7] As [Gar+24] observe, $\sum_{i=1}^{n} \log p_i$ grows with n and thus the field size $|\mathbb{F}_{p_0}|$ must grow with 2^n for any "unpacked" secret sharing scheme.

The parties continue performing multiplication gates using the above procedure, and addition gates via local addition of shares ("integer reduction" is not needed after additions since the growth of the underlying integer is small in this case), until they obtain shares of the outputs, which they reconstruct.

2.2 Our Techniques to Achieve Active Security

We now show how the protocol of [Gar+24] is insufficient for active security and then our techniques to transform it into a protocol that is actively secure, while maintaining $O(|C| \cdot \log p_0)$ bits of communication and computation.

Error Detection for CRT Secret Sharing. In secret-sharing based MPC, it is crucial that given a sharing $[\![X]\!]$, the parties can reconstruct the correct secret x. This is important not just for correctness, but also privacy (selective-failure attacks can allow the adversary to learn information about private values in the MPC if it is able to change the secrets of sharings).

Let $\mathcal{H}$ be the indices of the honest parties in the MPC and $\mathcal{T} = [n] \setminus \mathcal{H}$ be the indices of corrupted parties. In [Gar+24], the shares of all parties are required to reconstruct X from $[\![X]\!]$. Indeed, it may be the case that the underlying integer X is such that $2|X| > p_{\mathcal{H}} \coloneqq \prod_{i \in \mathcal{H}} p_i$. Therefore, the shares of the honest parties do not define the underlying secret $x \leftarrow X \bmod p_0$. Thus, an active adversary can easily change the shares $x^i \in \mathbb{Z}_{p_i}$ of corrupted parties $\mathcal{P}_i$ in $\mathcal{T}$ to arbitrary values $\tilde{x}^i \in \mathbb{Z}_{p_i}$, and since CRT establishes an isomorphism $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_n} \cong \mathbb{Z}_{\prod_{i=1}^n p_i}$, it is clear that performing CRT reconstruction with the changed corrupted parties' shares $\tilde{x}^i$ will result in a different $\tilde{X} \not\equiv X \in \mathbb{Z}_{\prod_{i=1}^n p_i}$. Thus, a different $\tilde{x} \not\equiv x \in \mathbb{Z}_{p_0}$ may be reconstructed.

Therefore, we need to alter the CRT sharing of [Gar+24] to obtain *error-detection*. That is, we need the honest parties to be able to discern whether the element $\tilde{x} \in \mathbb{Z}_{p_0}$ they reconstruct from a sharing is correct, and if not, abort. The first ingredient we need for error-detection is to ensure that the primes p_i of the honest parties satisfy $p_{\mathcal{H}} > 2|X|$ for the underlying integer X of secret sharings $[\![X]\!]$. Indeed, letting $I = \{i_1, i_2, \ldots, i_{n-t}\}$ be any set of $n - t$ indices, such as those of the honest parties, we will ensure via the isomorphism

$$\mathbb{Z}_{p_{i_1}} \times \mathbb{Z}_{p_{i_2}} \times \cdots \times \mathbb{Z}_{p_{i_{n-t}}} \cong \mathbb{Z}_{\prod_{i \in I} p_i} \tag{1}$$

between the share space of the parties in set I and the ring $\mathbb{Z}_{\prod_{i \in I} p_i}$, that any such set I of shares defines all honestly generated sharing integers X. Indeed, we show in Section 3.1 that if we make each prime p_i slightly bigger to obtain this property, then still privacy of the sharing scheme holds, i.e., $\prod_{j \in \mathcal{T}} p_j$ is still too small to obtain any useful information about X from the corrupted parties' shares x_j.

Once we have the above property, we show that error-detection is possible. Indeed, an honest party $\mathcal{P}$ reconstructing $[\![X]\!]$ from the shares $x^1, \ldots, x^n$ can take the first $n - t$ shares $x^1, \ldots, x^{n-t}$ to obtain $\tilde{X} \in \mathbb{Z}_{\prod_{i=1}^{n-t} p_i}$. Then, $\mathcal{P}$ can obtain $\tilde{x}^i \leftarrow \tilde{X} \bmod p_i$ and compare them to the received x^i for each $i \in [n - t + 1, n]$.

If any are different, then $\mathcal{P}$ aborts; otherwise, $\mathcal{P}$ outputs the correct secret $x \leftarrow \tilde{X} \bmod p_0$. Note that this procedure does not affect the asymptotic overhead of reconstruction—computing both directions of the CRT isomorphism costs $O(\log n \cdot \log p_0)$ bit operations, both of which are performed just once.

First, assuming that the adversary did not modify any corrupted shares, it is clear that the honest parties will reconstruct the correct secret x and not abort. On the other hand, letting $\mathcal{H} = \{i_1, i_2, \ldots, i_{n-t}\}$, via two applications of the CRT isomorphism in Equation 1, we see that the share space of the honest parties, $\mathbb{Z}_{p_{i_1}} \times \cdots \times \mathbb{Z}_{p_{i_{n-t}}}$, is isomorphic to the share space of the first $n-t$ parties, $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_{n-t}}$. Thus, there is a one-to-one mapping between the honest parties' shares and the first $n-t$ shares. So, if any of the first $n-t$ shares are modified, then the shares of the honest parties $\tilde{x}^i$ that party $\mathcal{P}$ obtains in the error-detection procedure above will not be the same as those of the received shares x^i from the honest parties.[8]

Note that for "larger" sharings $\langle X \rangle$, we do not obtain error-detection. This is because, e.g., the underlying integer Z that results from multiplying shares $[\![X]\!]$ and $[\![Y]\!]$ becomes bigger than $p_{\mathcal{H}}$ and thus the honest parties' shares do not define Z (if we were to make each p_i even bigger to achieve this, then privacy of "smaller" sharings $[\![X]\!]$ would no longer hold). This is akin to the "high-degree" sharings used in Shamir secret sharing based honest majority MPC, e.g., [DN07]. Indeed, in that case, up to n shares might be needed to reconstruct such sharings, which clearly includes some corrupted shares. Later on, we show how to still obtain active security, even without error-detection for larger sharings $\langle X \rangle$.

Integer Bound Checking. As stated above, it is important that elements X underlying sharings $[\![X]\!]$ of circuit wire values never overflow modulo $p_{\mathcal{H}}$. However, corrupted parties are responsible for sharing many values throughout the computation, and could share an arbitrary element X such that $2|X|$ is as big as $p_{\mathcal{H}} - 1$, which would immediately overflow after any additional operations. Therefore, we must require parties to prove that the elements they share throughout the computation are properly bounded. We accomplish this in Section 4.3, by borrowing some techniques from the Lattice-based Zero-Knowledge literature [BL17; BN20; GHL22; LNS20]. In particular, for any vector $\vec{X} \in (\mathbb{Z}^{\pm}_{p_{\mathcal{H}}})^m$, let $||\vec{X}||_\infty \coloneqq \max_{i \in [m]} |X_i|$. We use the following result from [BL17]:

Lemma 1 (Informal Version of Corollary 2.4 in [BL17]). *Let $\vec{X} \in (\mathbb{Z}^{\pm}_{p_{\mathcal{H}}})^m$ and $\vec{R} \in (\mathbb{Z}^{\pm}_{p_{\mathcal{H}}})^\lambda$ be any vectors generated by a malicious party. Then with all-but-negligible probability over randomly chosen matrix $\boldsymbol{C} \leftarrow_{\$} \{0,1\}^{\lambda \times m}$,*

$$||\boldsymbol{C} \cdot \vec{X} + \vec{R}||_\infty \geq \frac{1}{2} \cdot ||\vec{X}||_\infty.$$

Let $\mathcal{B}$ be the *ideal* bound on every shared integer X based on the normal CRT secret sharing procedure. With the above lemma in hand, the parties can use the

[8] Note that there are some additional subtleties regarding the size of the honest parties' primes p_i compared to the first $n-t$ primes, which we cover in Section 3.1.

following protocol to ensure that the sharings of a given party $\mathcal{P}$ are (almost) bounded by $\mathcal{B}$: After party $\mathcal{P}$ has shared some values $[\![X_1]\!], \ldots, [\![X_m]\!]$, they generate λ random masking sharings $[\![R_1]\!], \ldots, [\![R_\lambda]\!]$. Next, all of the parties will jointly sample $\lambda \cdot m$ random bits $\{c_{ij}\}_{i\in[\lambda], j\in[m]}$,[9] then compute and open to one another $[\![Y_i]\!] \leftarrow \bigoplus_{j=1}^{m}(c_{ij} \odot [\![X_j]\!]) \oplus [\![R_i]\!]$, for each $i \in [\lambda]$. If party $\mathcal{P}$ is honest, then the random sharings $[\![R_1]\!], \ldots, [\![R_\lambda]\!]$ will maintain the privacy of $[\![X_1]\!], \ldots, [\![X_m]\!]$, even given $[\![Y_1]\!], \ldots, [\![Y_\lambda]\!]$. Moreover, we see that each $|Y_i| \leq (m+1) \cdot \mathcal{B}$ for honest $\mathcal{P}$. Thus, the honest parties will abort only if any $|Y_i| > (m+1) \cdot \mathcal{B}$. Indeed, Lemma 1 guarantees that with all-but-negligible probability, if a corrupted party $\mathcal{P}$ shares any integer X_i such that $|X_i| > 2(m+1) \cdot \mathcal{B}$, then some $|Y_i| > (m+1) \cdot \mathcal{B}$ and thus the parties will abort. Recall that no matter what the corrupted parties share, the honest parties' shares will define some $X_1, \ldots, X_m$ and $R_1, \ldots, R_\lambda$ in $\mathbb{F}_{p_\mathcal{H}}$, and thus some $Y_1, \ldots, Y_\lambda$ in $\mathbb{F}_{p_\mathcal{H}}$; the honest parties will either output these exact $Y_1, \ldots, Y_\lambda$, or abort if error-detection catches adversarial tampering of shares.

Therefore, with this protocol, the parties can ensure that all elements shared by corrupted parties are bounded by $2(m+1) \cdot \mathcal{B}$. We will show that even with this small slack, we can ensure that the shared circuit wire values will never overflow modulo $p_\mathcal{H}$, while maintaining security and efficiency. Note that this bound check protocol does not introduce any additional asymptotic efficiency overhead. Indeed, if $m \geq \lambda$ (which will be the case for all relevant checks), then sharing λ additional sharings $[\![R_i]\!]$ has no impact and computing then opening the resulting $[\![Y_i]\!]$ can be done with just a λ-factor overhead.

Verifying Multiplications. Recall that [Gar+24] use the "king" technique of [DN07] to perform secure multiplication over secret shared values. For any MPC-based protocol using this style of multiplication, there are a couple of ways in which an adversary can misbehave. First, if the king party is corrupted, then instead of reconstructing $\langle D \rangle \leftarrow [\![X]\!] \odot [\![Y]\!] \oplus \langle \overline{R} \rangle$ and sending shares of $d \leftarrow D \bmod p_0$ to the rest of the parties, the adversary can send shares of some other $d' \in \mathbb{F}_{p_0}$ to the parties. Indeed, even if the king party is honest, since there is no error-detection for the sharing $\langle D \rangle$, the adversary can force the king party to reconstruct some incorrect D' and thus send shares of incorrect $d' \leftarrow D' \bmod p_0$ to the rest of the parties. Note that for this reason, checking the size of $\langle \overline{R} \rangle$ is useless. We will show that both of these attacks can be reduced to an *additive attack* [Gen+14], in which the simulator can extract some value $\delta \in \mathbb{F}_{p_0}$ (independent of secrets x and y) such that $d' = d + \delta$, and therefore $[\![Z]\!] = [\![X]\!] \odot [\![Y]\!] + \delta$.

[9] This can be accomplished efficiently with standard techniques.

We employ *triple sacrificing* [Dam+12] with the standard "multiple king" opening technique of [DN07], using the help of *parity-check matrices* [BEP24],[10] all of which preserve our desired efficiency.[11]

See Section 6.4 for more details.

Complete Overview of Actively-Secure Protocol. We are now ready to describe the overview of our complete actively-secure MPC protocol. First, for any opening of a secret sharing $[\![X]\!]$ throughout the protocol, the parties use the error-detection procedure of Section 4.2. During the input phase, each party $\mathcal{P}_i$ secret shares their input $[\![X_i]\!]$ and the parties use the bound checking protocol of Section 4.3 to ensure that the underlying elements X_i are not too big. For the execution phase, each addition gate is evaluated locally; for multiplication gates, the parties run the same "king" multiplication protocol of [Gar+24], except the parties use the bound checking protocol to ensure that the elements that the king reshares are not too big. At the end of the execution phase, the parties use the triple sacrificing procedure of Section 6.4 to ensure that no additive attacks were successful. Once this check is finished, the parties can safely open the outputs of the circuit; since the bound checking protocol and error detection is used throughout, the parties indeed output the correct values.

3 Preliminaries

We let $\mathcal{T} \subseteq [n]$ be the set of corrupted parties and $\mathcal{H} = [n] \setminus \mathcal{T}$ be the set of honest parties. We focus on the case of submaximal honest majority $t < n(1/2 - \varepsilon)$, with $\varepsilon > 0$.

Prior works deal with negative values by representing $-x$ as $p_0 - x$ (see *e.g.* [Gar+23b, Fig. 1]). However, this makes the bound analysis more cumbersome. This is worsened by the fact that we have to subtract in many places, not just subtraction gates; for instance, we do this for verifying multiplication triples. Instead, we use a *signed* representation which enables addition, multiplication and *subtraction* gates, which enables a much cleaner bound analysis. Given an integer $N \geq 2$, we will let

$$\left\{ \left\lfloor -\frac{N-1}{2} \right\rfloor, \ldots, -1, 0, 1, \ldots, \left\lfloor \frac{N-1}{2} \right\rfloor \right\} = \left[\left\lfloor -\frac{N-1}{2} \right\rfloor, \left\lfloor \frac{N-1}{2} \right\rfloor \right] = \mathbb{Z}_N^{\pm} \subseteq \mathbb{Z}$$

be the canonical representation of the ring of integers $\mathbb{Z}_N$ (or $\mathbb{F}_N^{\pm}$ be the canonical representation of field $\mathbb{F}_N$ in case N is prime), instead of $\{0, 1, \ldots, N-1\}$.[12] In

[10] An avid reader may note that we cannot use parity-check matrices directly over CRT sharings, since they do not constitute a proper error-correcting code. However, we use these matrices for actual *linear* Reed-Solomon codes, which are defined across shared values. This is done thanks to the local homomorphism property of CRT sharings.

[11] Other methods for achieving active security exist, but we chose triple sacrificing since it seems to have the smallest effect on the growth of the underlying secret shared integers in the MPC, which impacts the efficiency of CRT-sharing.

[12] Note: $-(x) \in [-\lfloor \frac{N-1}{2} \rfloor, -1] \equiv (-x) \mod N$, where $(-x)$ is the additive inverse of x, so this is a valid representation (isomorphism).

this case, for some $x \in \mathbb{Z}_N^{\pm} = [\lfloor -(N-1)/2 \rfloor, \lfloor (N-1)/2 \rfloor]$, we let $|x|$ denote the absolute value of x, where $|x| \coloneqq x$ if $x \geq 0$ and $|x| \coloneqq -x$ if $x < 0$. Note that if N is odd then $\mathbb{Z}_N^{\pm} = [-(N-1)/2, (N-1)/2]$, so $x \in \mathbb{Z}_N^{\pm}$ iff $|x| \leq (N-1)/2$. Given a vector of elements $\vec{x} \in (\mathbb{Z}_N^{\pm})^m$, we let $||\vec{x}||_\infty \coloneqq \max_{i \in [m]} |x_i|$.

3.1 CRT-Based Secret-Sharing

Let $p_0, p_1, \ldots, p_n$ be pairwise coprime *integers*. We assume all p_i's are odd for simplicity. Also, it will be convenient to assume that $p_1 \leq p_2 \leq \cdots \leq p_n$. To share $s \in \mathbb{Z}_{p_0}^{\pm} \cong \mathbb{Z}_{p_0}$, we use the following procedure:

Procedure 1: $\pi_{\mathsf{Share}}(s)$

1. Sample $u \in \mathbb{Z}_L^{\pm}$ and let $S = s + u \cdot p_0$.
2. Compute and distribute the i-th share $s_i = S \bmod p_i$, for $i \in [n]$.

To reconstruct the secret from a set of shares $\{s_i\}_{i \in A}$ indexed by a set $A \subseteq \{1, \ldots, n\}$:

Procedure 2: $\pi_{\mathsf{Recons}}(\{s_i\}_{i \in A})$

1. Apply CRT to get an integer S, that is, find the unique $S \in \mathbb{Z}_{p_A}^{\pm}$ such that $S \equiv s_i \bmod p_i$, where $p_A \coloneqq \prod_{i \in A} p_i$.
2. Return $S \mod p_0$ (as a representative in $\mathbb{Z}_{p_0}^{\pm}$)

For correctness, we require that the integer S reconstructed above is the same as the original integer S. This requires $S = s + u \cdot p_0$ to be in $\mathbb{Z}_{p_A}^{\pm}$, or in other words $|S| \leq (p_A - 1)/2$. The privacy of CRT-sharing yields that the view of any corrupt set of shares, and in fact the shares corresponding to the maximum product of p_i's, given by p_{max}, is (p_{max}/L)-close to uniform [Gar+24, Lemma 3].[13] Indeed, for statistical security, it must be that $p_{\mathsf{max}}/L \leq 2^{-\lambda}$, which is equivalent to $p_{\mathsf{max}} \cdot 2^{\lambda} \leq L$.

We denote by $[\![S]\!]$ the sharing of the integer S, that is, each party holds $s_i = S \bmod p_i$ and the secret is $s = S \bmod p_0$, where privacy holds against t corruptions and correctness allows any set of at least $\lceil n/2 \rceil$ parties to reconstruct the secret. This requires $|S| \leq (p_{\mathsf{min}} - 1)/2$, where $p_{\mathsf{min}} = \min_{\substack{A \subseteq \{1,\ldots,n\} \\ |A| = \lceil n/2 \rceil}} (\prod_{i \in A} p_i) = p_1 \cdots p_{\lceil n/2 \rceil}$.[14] We also use $\langle S \rangle$ to denote a sharing of a *bigger* integer S, which requires, for reasons that will be made clear later (see Section 3.4), a bigger

[13] It is easy to adapt the privacy proof of [Gar+24, Lemma 3] to the setting of signed values.

[14] Note that an actively corrupt dealer may distribute shares $[\![S]\!]$ with $|S| > (p_{\mathsf{min}} - 1)/2$, and in this case, different subsets of shares may end up reconstructing different secrets. To account for this, when we write $[\![S]\!]$, we think of S as being the secret integer *defined by the honest parties' shares*, that is, $S \in \mathbb{Z}_{\prod_{i \in \mathcal{H}} p_i}$ is the result of applying CRT to the shares $\{s_i\}_{i \in \mathcal{H}}$.

masking integer, in this case denoted as L_{2t}. However, correctness for $\langle S \rangle$ only requires that *all* parties can reconstruct, which imposes the weaker requirement $|S| \leq (p_1 \cdot p_2 \cdots p_n - 1)/2$; therefore, even though L_{2t} is bigger, the collection of shares do not need to be bigger to reconstruct $\langle S \rangle$. We will refer to the masking integer for $[\![\cdot]\!]$ sharings as L_t.

When adding sharings together locally, we use $[\![R]\!] \oplus [\![S]\!]$, to represent each party $\mathcal{P}_i$ adding their corresponding shares together mod p_i. Similarly, we use $\odot$ for local multiplication of shares mod p_i.

Given two sharings $[\![R]\!]$ and $[\![S]\!]$, the parties can perform local operations on their shares, and via CRT this results in the same operations on R and S. However, taking mod p_0 will only correspond to the same operations on the associated secrets $r, s \in \mathbb{Z}_{p_0}$, if the resulting integer T does not exceed the "correctness range": $|T| \leq (p_{\mathsf{min}} - 1)/2$ for $[\![\cdot]\!]$-sharings, and $|T| \leq (p_1 \cdots p_n - 1)/2$ for $\langle \cdot \rangle$-sharings. We will set parameters so that additions and multiplication by constants (of certain size) can be done locally on $[\![\cdot]\!]$ and $\langle \cdot \rangle$-sharings. Also, we will set parameters so that parties can locally compute $[\![X]\!] \odot [\![Y]\!]$ and obtain $\langle XY \rangle$, which requires $|X||Y| \leq (p_1 \cdots p_n - 1)/2$.

Finally, given a public value $s \in \mathbb{Z}_{p_0}^{\pm}$, often we will let the parties obtain "shares" of it but without privacy, that is, $\mathcal{P}_i$ sets $s_i = s \bmod p_i$. We denote this by $[s]$, and we will set parameters so that parties can locally multiply $[s] \odot [\![R]\!]$ to obtain $[\![s \cdot R]\!]$.

3.2 Super-Invertible Vandermonde Matrices and their Parity Check Matrices

As in [Gar+24], we use super-invertible matrices for randomness extraction.

Definition 1. *A matrix $\boldsymbol{V} \in \mathbb{Z}_{p_0}^{r \times c}$ with $c > r$ is* super-invertible *if for any column index set $C \subseteq \{1, \ldots, c\}$ such that $|C| = r$, the sub-matrix $\boldsymbol{V}^C$ is invertible.*

Over fields with p_0 some large enough prime, the transpose of a Vandermonde matrix, where each entry $\boldsymbol{V}_{i,j}^{\intercal} = x_j^i$ for distinct x_j's, is super-invertible. This is because for any column index set C, $\det(\boldsymbol{V}^C) = \prod_{i,j \in C, i>j}(x_i - x_j)$, which is non-zero if all x_i's are distinct. In this case, if $x_j = j$, each element of the $n \times (n-t)$ matrix (of relevance for MPC) $\boldsymbol{V}^{\intercal}$, viewed over the integers, can be upper bounded by n^{n-t-1}. The determinant of any square sub-matrix of this Vandermonde matrix over the integers can be upper bounded by $\prod_{1 \leq i < j \leq n}(j - i) \leq n^{n^2}$.

Note that if $\boldsymbol{V}^{\intercal}$ is super-invertible, then also every square sub-matrix of the original Vandermonde matrix, $\boldsymbol{V}$ will be invertible. We will use this property for detecting adversarial misbehavior in certain parts of our protocol.

We will also take advantage of the so-called $t \times n$ *parity check* matrix, $\boldsymbol{H}$, corresponding to a given $n \times (n-t)$ Vandermonde matrix, $\boldsymbol{V}$, defined over $\mathbb{F}_{p_0}$. Indeed, a Vandermonde matrix can actually be seen as the generator matrix for a Reed-Solomon error-correcting code. All error-correcting codes have a corresponding parity check matrix $\boldsymbol{H}$, in which $\boldsymbol{H} \cdot \mathbf{y} = 0$ if and only if $\mathbf{y}$ corresponds to a valid codeword generated by the matrix $\boldsymbol{V}$. Indeed, if $\mathbf{y}$ is

a valid codeword and $\mathbf{y}'$ is a modification of $\mathbf{y}$ in at most t coordinates, then $\boldsymbol{H} \cdot \mathbf{y}' \neq 0$. The linear operations of $\boldsymbol{H}$ can be seen as first applying the inverse of the first $n-t$ rows of the Vandermonde matrix $\boldsymbol{V}_{[n-t]}^{-1}$ to the first $n-t$ coordinates of $\mathbf{y}$ to recover the original message $\mathbf{x}'$ that was ostensibly encoded, then applying the rest of the Vandermonde $\boldsymbol{V}_{[n-t+1,n]}$ to $\mathbf{x}'$ to obtain $\mathbf{y}'$, and finally subtracting $\mathbf{y}'$ by the last t coordinates of $\mathbf{y}$.

3.3 Adjugate Vandermonde Matrices

A key observation that [Gar+24] makes is that super-invertibility over $\mathbb{F}_{p_0}$ is necessary, but not sufficient for masking private vectors in CRT MPC. Thus, we need to consider the behavior of the Vandermonde matrix (with entries mod p_0), *over the integers*.

Let I denote the identity matrix. While a matrix $\boldsymbol{M}$ over a ring (such as the integers) may not be invertible, the adjugate matrix $\mathsf{adj}(\boldsymbol{M})$ is always defined. This matrix satisfies

$$\boldsymbol{M} \cdot \mathsf{adj}(\boldsymbol{M}) = \mathsf{adj}(\boldsymbol{M}) \cdot \boldsymbol{M} = \det(\boldsymbol{M}) \cdot I.$$

In particular, the (i,j)-th entry of $\mathsf{adj}(\boldsymbol{M})$ is $(-1)^{i+j} \cdot \det(\boldsymbol{M}_{-j,-i})$, where $\boldsymbol{M}_{-j,-i}$ is the sub-matrix of $\boldsymbol{M}$ obtained by removing the j-th row and i-th column. Each entry in the adjugate of a square $(n-t) \times (n-t)$ sub-matrix of the above-defined Vandermonde matrix $\boldsymbol{V}$ corresponding to a field can be upper bounded by $(n-t-1)! \cdot (n^{n-t-1})^{n-t-1} \leq n^{n^2}$.

3.4 High-Dimensional Integer Masking Lemma

Equipped with the Vandermonde and adjugate matrices above, we can finally discuss masking private vectors in CRT MPC.

We start with masking a single integer. For positive integer N, let U_N be the uniform distribution over $\{0, 1, 2, \ldots, N-1\}$ and U_N^d be d independent samples from U_N.

Lemma 2 (Integer Masking Lemma [Gar+23b]). *Let $0 \leq r < P$, and $\Delta \geq 0$ be any integers. Let N also be an arbitrary integer. Then*

$$\mathsf{SD}((\Delta + U_N \mid U_N \mod P \equiv r), (U_N \mid U_N \mod P \equiv \Delta + r)) \leq \frac{2P}{N} + \frac{|\Delta|}{N}.$$

Lemma 3 (High-Dimensional Integer Masking Lemma (Inspired by [Gar+24])). *Let P be an integer, $\boldsymbol{r} = (r_1, \ldots, r_n) \in [0, P-1]^n$, $\boldsymbol{V} \in \mathbb{Z}^{n \times n}$, and $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n) \in \mathbb{Z}_{\geq 0}^n$ such that $\det(\boldsymbol{V}) \mid \Delta_i$ for every $i \in [n]$. Let N also be an arbitrary integer. Then*

$$\mathsf{SD}\Big((\boldsymbol{\Delta} + \boldsymbol{V} \cdot U_N^n \mid U_N^n \mod P \equiv \boldsymbol{r}), \Big(\boldsymbol{V} \cdot U_N^n \mid U_N^n \mod P \equiv \boldsymbol{r} + \frac{1}{\det(\boldsymbol{V})}(\mathsf{adj}(\boldsymbol{V}) \cdot \boldsymbol{\Delta})\Big)\Big)$$

$$\leq n \cdot \left(\frac{2P}{N} + \frac{||\mathsf{adj}(\boldsymbol{V}) \cdot \boldsymbol{\Delta} / \det(\boldsymbol{V})||_\infty}{N} \right),$$

where $\mathsf{adj}(\boldsymbol{V})$ *is the adjugate matrix of* $\boldsymbol{V}$*.*

We provide a modified proof of the above in the full version. Given the bounds on the entries of $\mathsf{adj}(\boldsymbol{V})$ from Section 3.3, we can bound the above distance by $n \cdot \left(\frac{2P}{N} + \frac{\max_\Delta \cdot n^{n^2}}{N} \right)$, where $\max_\Delta \coloneqq \max_i \{\Delta_i / \det(\boldsymbol{V})\}$.

Reformatting Secrets. In order to use Lemma 3, we see that it must be that $\det(\boldsymbol{V}) \mid \Delta_i$, for each $i \in [n]$. Therefore, we will need to ensure that any secret sharing integer that is masked-and-opened is divisible by ℓ, the LCM of determinants of the $(n-t) \times (n-t)$ sub-matrices of the $(n-t) \times n$ transpose Vandermonde matrix. The approach in [Gar+24] is to enforce the invariant that, for all internal wires of the computation, the corresponding secret shared integer is divisible by ℓ. This is done with the help of a procedure π_{reformat}, which dealers use to turn an element $x \in \mathbb{Z}_{p_0}^{\pm}$ into $\overline{x} = \ell\ell^{-1} \cdot x \bmod p_0$, where $\ell^{-1} \in \mathbb{Z}_{p_0}$ is the inverse of ℓ modulo p_0. This satisfies $\overline{x} \equiv x \bmod p_0$, i.e., reconstruction will still work, and $\overline{x} \equiv 0 \bmod \ell$, i.e., $\overline{x}$ is a multiple of ℓ. Then, this new secret $\overline{x} \in \mathbb{Z}_{p_0 \cdot \ell}$ is secret-shared.

In [Gar+24], this reformatting happens locally at the dealer's level, when distributing shares. However, certain complications arise when handling active security, since a corrupt dealer may not reformat the secret correctly. Instead, we take a slightly different approach that allows this reformatting to happen locally, *after* a secret $s \in \mathbb{Z}_{p_0}$ has been dealt. Consider $[\![S]\!]$, where $S = s + u \cdot p_0$. The parties locally compute $\ell\ell^{-1} \odot [\![S]\!] = [\![\ell\ell^{-1} \cdot S]\!]$, and these are the values we compute over. This has almost the same impact as the previous approach in terms of the parameters needed for the secret-sharing, while avoiding the issue of achieving security when reformatting happens at the dealers' end.

3.5 Computational Complexity of Key Components

In this section, we discuss the computational complexity of some key components used throughout our protocol. We focus on the complexities at the *bit level* to compare operations over different fields fairly.

Already, [Gar+24] discussed the computational complexity of several key components, which we briefly recap here (see more references therein):

- **Field Operations.** Each party will perform arithmetic operations over $\mathbb{F}_p$, for some different p. Addition and subtraction can easily be performed in $O(\log p)$ time, while multiplication can be performed in quasilinear, e.g., $O(\log p \cdot \log\log p \cdot \log\log\log p)$ time. Throughout, we will ignore the lower-order $\log\log p$ terms.
- **CRT Sharing/Reconstruction.** Let p_i be the prime used for party $\mathcal{P}_i$ in CRT Sharing and Reconstruction. Then, these operations will take $O(\log n \sum_{i=1}^{n} \log p_i)$ time.

– **Vandermonde Matrix Multiplication.** In our protocol, we will need to compute structured matrix multiplications of the form $\boldsymbol{V} \cdot \mathbf{x}$ and $\boldsymbol{V}^\intercal \cdot \mathbf{y}$, where $\mathbf{x}, \mathbf{y}$ are vectors and $\boldsymbol{V}$ is a Vandermonde matrix. One can use standard FFT techniques to compute both of these operations over any field $\mathbb{F}_p$ with $O(n \log^2 n)$ field arithmetics. In particular, since each party performs these multiplications over their own field $\mathbb{F}_{p_i}$, the i-th party takes $O(n \log^2 n \cdot \log p_i)$ bit operations.

The only other key operations present in our work and not in [Gar+24] are inverse Vandermonde matrix multiplication and parity-check matrix multiplication. For the former, we can also use standard inverse FFT techniques to take $O(n \log^2 n \cdot \log p_i)$ bit operations for each party $\mathcal{P}_i$. For the latter, as explained in Section 3.2, multiplication of the parity check matrix can simply be seen as first applying the inverse of a sub-matrix of the Vandermonde (which is itself a Vandermonde) on a $O(n)$-length prefix of the vector, followed by applying the rest of the Vandermonde, and finally $O(n)$ subtractions. These operations can therefore clearly be performed in $O(n \log^2 n \cdot \log p_i)$ bit operations for each party $\mathcal{P}_i$.

4 Tools for Active Security

4.1 Random Coins

First, we need a way to generate random coins:

Functionality 1: $\mathcal{F}_{\mathsf{Coin}}$

1. Sample a random element $\rho \leftarrow_\$ [0, 2^\lambda - 1] \subseteq \mathbb{Z}_{p_0}$.
2. Send ρ to the ideal adversary and:
   – If the adversary replies continue, send ρ to the honest parties.
   – Otherwise, send abort to the honest parties.

One can generate random coins $r \leftarrow_\$ \mathbb{F}_{p_0}$ using standard techniques (via CRT sharing or packed Shamir sharing) with $O(\log p_0)$ amortized communication and $O(\log p_0)$ computation per random value r. To get random coins $\rho \leftarrow_\$ [0, 2^\lambda - 1] \subseteq \mathbb{F}_{p_0}$ and realize the above functionality, one can take a random coin $r \in \mathbb{F}_{p_0}$ generated as above, view it as a bit string, and split it into λ-length substrings to get $\log p_0 / \lambda$ random coins $\rho \in [0, 2^\lambda - 1] \subseteq \mathbb{F}_{p_0}$. Thus, each coin costs $O(n\lambda)$ bits of communication and $O(\log^2 n \cdot \lambda)$ computation.[15]

[15] Note: we need to reject random values $r \geq 2^{\lfloor \log_2(p_0) \rfloor}$, which will be at most half of those sampled, and throw out the top bit (which will always be 0 after rejection).

4.2 Error Detection for CRT-based Secret Sharing

We now show that error detection can be achieved for CRT-based secret sharing. Consider a shared secret $[\![S]\!]$.[16] Given all shares $(s_1, \ldots, s_n)$, where t of them may be corrupt, the parties can use the first $\lceil n/2 \rceil$ shares $s_1, \ldots, s_{\lceil n/2 \rceil}$ to obtain an integer $S' \in \mathbb{Z}_{p_1 \cdots p_{\lceil n/2 \rceil}} = \mathbb{Z}_{p_{\mathsf{min}}}$, and then check that this is consistent with the remaining shares, that is, $S' \bmod p_i = s_i$ for $i = \lceil n/2 \rceil + 1, \ldots, n$. If this is not the case, the parties abort. Else, the parties accept $S' \bmod p_0$ as the reconstructed secret. Note that the computational complexity is asymptotically the same as one reconstruction and one sharing operation (which are asymptotically the same).

We claim that S' is exactly equal to the integer S, which is—by definition—the integer in $\mathbb{Z}_{\prod_{i \in \mathcal{H}} p_i}$ reconstructed via CRT from the honest parties' shares $\{s_i\}_{i=1}^{\lceil n/2 \rceil}$. To see this, note that since $S' \bmod p_i = s_i$ for $i = \lceil n/2 \rceil + 1, \ldots, n$, uniqueness of CRT ensures that $S \equiv S' \bmod \prod_{i \in \mathcal{H}} p_i$. Furthermore, we know that $|S'| \leq (p_1 \cdots p_{\lceil n/2 \rceil} - 1)/2$, and since $p_1 \leq \cdots \leq p_n$, this implies that $|S'| \leq (\prod_{i \in \mathcal{H}} p_i - 1)/2$, so in particular we have integer equality $S = S'$.

As an important remark, note that reconstruction via error detection always results in an integer over $\mathbb{Z}_{p_{\mathsf{min}}}$.

4.3 Checking Bounds for Sharing

As pointed out earlier, when a party secret-shares a value $[\![X]\!]$ (or $[x]$), we need to make sure that $|X| \leq B$, for some bound B. If a corrupt party violates this requirement, it may be that the value V of some circuit wire during the computation might overflow modulo p_{min} resulting in some $V' \in \mathbb{Z}_{p_{\mathsf{min}}}$ such that $V \bmod p_0 \neq V' \bmod p_0$, affecting correctness (or even integrity, depending on where in the protocol this wrap-around happens).

We will address this problem by ensuring that corrupt parties cannot share values $[\![X]\!]$, such that $|X|$ is much bigger than B (relative to p_{min}). We use the following result from [BL17; LNS20]:

Lemma 4 (Corollary 2.4 in [LNS20]). *Let $\vec{X} \in (\mathbb{Z}^{\pm}_{p_{\mathsf{min}}})^m$ and $\vec{R} \in (\mathbb{Z}^{\pm}_{p_{\mathsf{min}}})^\lambda$. Then*

$$\Pr_{\boldsymbol{C} \leftarrow_{\$} \{0,1\}^{\lambda \times m}} \left[||\boldsymbol{C} \cdot \vec{X} + \vec{R}||_\infty < \frac{1}{2} \cdot ||\vec{X}||_\infty \right] \leq 2^{-\lambda}.$$

The lemma can be interpreted as follows: with all-but-negligible probability, many random binary linear combinations of a vector $\vec{X}$ mod p_{min} each cannot become very small—even if an arbitrary value R_i is added to every i-th linear combination—unless the initial vector was relatively small to begin with.

We will use this lemma to implement the functionality $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ below. There, we see that corrupt parties can share values $[\![X]\!]$ such that $|X| \leq B' = 2(Bm + \overline{B})$, without getting caught.[17] To ensure that no integers representing

[16] Recall S here means $S \in \mathbb{Z}_{\prod_{i \in \mathcal{H}} p_i}$, the value defined by the honest parties' shares $\{s_i\}_{i \in \mathcal{H}}$.

[17] For notational purposes, if $\overline{B}$ is not specified, then $\overline{B} = B$.

circuit wires overflow mod p_{min}, we will have to bound the size of such integers based on B', rather than B.

Functionality 2: $\mathcal{F}_{\text{Check-Bounds}}[\mathcal{P}, B, \overline{B}]$

Parameter: This functionality is parameterized by a party $\mathcal{P}$ that owns the input sharings to be checked, a bound B on each shared integer, and a bound $\overline{B}$ on the size of a "masking" integer.

1. Let $[\![X_1]\!], \ldots, [\![X_m]\!]$ be party $\mathcal{P}$'s input sharings to be checked. $\mathcal{F}_{\text{Check-Bounds}}$ receives from the honest parties their shares. If party $\mathcal{P}$ is corrupted, $\mathcal{F}_{\text{Check-Bounds}}$ sends to the adversary the honest parties' shares.
2. $\mathcal{F}_{\text{Check-Bounds}}$ recovers each entire sharing $[\![X_i]\!]$ based on the honest parties' shares and sends to the adversary the corrupted parties' shares.
3. Then it checks if any $|X_i| > 2(B \cdot m + \overline{B})$. If so, $\mathcal{F}_{\text{Check-Bounds}}$ sends abort to all parties.
4. Otherwise, $\mathcal{F}_{\text{Check-Bounds}}$ may receive abort from the adversary, in which case it sends abort to all parties. If no abort was sent, $\mathcal{F}_{\text{Check-Bounds}}$ sends continue to the parties.

Protocol 3: $\Pi_{\text{Check-Bounds}}[\mathcal{P}, B, \overline{B}]$

1. The parties have as input m sharings $[\![X_1]\!], \ldots, [\![X_m]\!]$ from party $\mathcal{P}$. (They are supposed to satisfy $|X_i| \leq B$, which is precisely what we will check)
2. Party $\mathcal{P}$ first samples $R_1, \ldots, R_\lambda \in U_{\overline{B}}$, and distributes sharings $[\![R_1]\!], \ldots, [\![R_\lambda]\!]$. (Recall U_N is the uniform distribution over $\{0, \ldots, N-1\}$. For an honest $\mathcal{P}$, these values satisfy $|R_i| \leq \overline{B}$)
3. Then the parties invoke $\mathcal{F}_{\text{Coin}}$ and receive random bits $\{c_{jk}\}_{j\in[\lambda],k\in[m]} \leftarrow_{\$} \{0,1\}$.
4. Next, the parties compute $[\![Y_j]\!] = \bigoplus_{k=1}^{m} (c_{jk} \odot [\![X_k]\!]) \oplus [\![R_j]\!]$ for $j \in [\lambda]$. (We have that $|Y_j| \leq m \cdot B + \overline{B}$)
5. Finally, the parties send to each other the shares of all $[\![Y_j]\!]$, reconstruct (performing error detection as described in Section 4.2), and abort if $|Y_j| > B \cdot m + \overline{B}$ for any $j \in [\lambda]$. (Correctness requires $m \cdot B + \overline{B} \leq (p_{\mathsf{min}} - 1)/2$)

This protocol requires

$$m \cdot B + \overline{B} \leq (p_{\mathsf{min}} - 1)/2, \tag{2}$$

where m is the number of total inputs to $\Pi_{\text{Check-Bounds}}$. As we will see throughout the following, $\Pi_{\text{Check-Bounds}}$ is called the following times, which helps us determine the values of m for each such call:

Offline calls – From π_{Mult} (from circuit computation and sacrifice): $2\frac{\mathcal{C}_{\text{Mult}}}{n-t}$ offline calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_i, \mathcal{B}_{\text{Rand-Share-t}}, \overline{\mathcal{B}}_{\text{Rand-Share-t}}]$, for $i = 1, \ldots, n$. The bound guaranteed here is $\mathcal{B}_{\text{Rand-Share-t-slack}} := 2(\mathcal{B}_{\text{Rand-Share-t}} \cdot \frac{2\mathcal{C}_{\text{Mult}}}{n-t} + \overline{\mathcal{B}}_{\text{Rand-Share-t}})$.

- From $\pi_{\text{Sacrifice}}$: $\frac{2\mathcal{C}_{\text{Mult}}}{n-t}$ calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_i, \overline{\mathcal{B}}_{\text{Rand-Share-t}}, \overline{\overline{\mathcal{B}}}_{\text{Rand-Share-t}}]$, for $i = 1, \ldots, n$. The bound guaranteed here is $\overline{\mathcal{B}}_{\text{Rand-Share-t-slack}} := 2(\overline{\mathcal{B}}_{\text{Rand-Share-t}} \cdot \frac{2\mathcal{C}_{\text{Mult}}}{n-t} + \overline{\overline{\mathcal{B}}}_{\text{Rand-Share-t}})$.
- From $\pi_{\text{Sacrifice}}$: $\mathcal{C}_{\text{Mult}}$ offline calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_1, p_0/2]$. The bound guaranteed here is $p_0 \cdot (\mathcal{C}_{\text{Mult}} + 1)$.

Online calls
- From π_{Input}: $\mathcal{C}_{\text{Input}}$ online calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_i, \mathcal{B}_{\text{Input}}, \overline{\mathcal{B}}_{\text{Rand-Share-t}}]$, for $i = 1, \ldots, n$. The bound guaranteed in each call is then $\mathcal{B}_{\text{Input-slack}} := 2(\mathcal{B}_{\text{Input}} \cdot \mathcal{C}_{\text{Input}} + \overline{\mathcal{B}}_{\text{Rand-Share-t}})$.
- From π_{Mult} (from the circuit computation): $\mathcal{C}_{\text{Width}}$ online calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_1, p_0/2]$, for every layer of the circuit. The bound guaranteed in each such batched call is then $2\frac{p_0}{2}(\mathcal{C}_{\text{Width}} + 1) = p_0(\mathcal{C}_{\text{Width}} + 1)$.
- From $\pi_{\text{Open-Redist}}$ in $\pi_{\text{Sacrifice}}$: $\frac{\mathcal{C}_{\text{Mult}}}{n-t}$ calls to $\Pi_{\text{Check-Bounds}}[\mathcal{P}_i, p_0/2]$ for $i = 1, \ldots, n$. The bound guaranteed here is $p_0 \cdot (\frac{\mathcal{C}_{\text{Mult}}}{n-t} + 1)$.

We make use of Lemma 4 to prove the following lemma in the full version.

Lemma 5. *Let L be the range from which the masking element u from π_{Share} is sampled for each $[\![R_j]\!]$. Protocol $\Pi_{\text{Check-Bounds}}$ UC-realizes $\mathcal{F}_{\text{Check-Bounds}}$ in the $\mathcal{F}_{\text{Coin}}$-hybrid model with error at most $2^{-\lambda} + \lambda \cdot (2p_{\max} + m \cdot B)/(L \cdot p_0)$.*

4.4 Check Zero

A common tool in MPC is checking that several sharings all share 0. We present functionality $\mathcal{F}_{\text{Check-Zero}}$ for checking $n - t$ such sharings at once.

Functionality 3: $\mathcal{F}_{\text{Check-Zero}}$

1. Let $[\![X_1]\!], \ldots, [\![X_{n-t}]\!]$ be the input sharings to be checked. $\mathcal{F}_{\text{Check-Zero}}$ receives from the honest parties their shares and recovers the entire sharings, sending them to the adversary.
2. Next, $\mathcal{F}_{\text{Check-Zero}}$ checks that each $X_i \bmod p_0 \equiv 0$; if not $\mathcal{F}_{\text{Check-Zero}}$ sends abort to all parties.
3. Otherwise, $\mathcal{F}_{\text{Check-Zero}}$ may receive abort from the adversary, in which case it sends abort to all parties. If no abort was sent, $\mathcal{F}_{\text{Check-Zero}}$ sends continue to all parties.

We now present our protocol $\Pi_{\text{Check-Zero}}$ for doing so. It uses the classical "multiple king" technique of [DN07], which uses error-detection via a linear code defined by a Vandermonde matrix. Note: the code is defined across the shared values (through linear homomorphism of the sharings; see Section 3.1), and is separate from the error detection for each CRT sharing.

Protocol 4: $\Pi_{\text{Check-Zero}}$

Parameters: This protocol is parameterized by the $n \times (n - t)$ Vandermonde matrix $\boldsymbol{V}$.

1. The parties have as input $n-t$ sharings $[\![X_1]\!], \ldots, [\![X_{n-t}]\!]$.
2. The parties apply $\boldsymbol{V}$ to $([\![X_1]\!], \ldots, [\![X_{n-t}]\!])^\intercal$ to obtain $([\![Y_1]\!], \ldots, [\![Y_n]\!])^\intercal$. ($|Y_i| \leq |\boldsymbol{V}|(n-t)\max_j |X_j|$)
3. The parties open $[\![Y_i]\!]$ to party $\mathcal{P}_i$ for each $i \in [n]$. (Error detection requires $|Y_i| \leq (p_{\mathsf{min}}-1)/2$, for which it suffices that $|\boldsymbol{V}|(n-t)\max_j |X_j| < (p_{\mathsf{min}}-1)/2$)
4. Each party $\mathcal{P}_i$ reconstructs Y_i (with error detection) and checks whether $Y_i \bmod p_0 = 0$. If not, it sends abort to all parties.

This requires

$$|\boldsymbol{V}|(n-t)\max_j |X_j| \leq (p_{\mathsf{min}}-1)/2 \tag{3}$$

We prove the following lemma in the full version.

Lemma 6. *$\Pi_{\mathsf{Check\text{-}Zero}}$ UC-realizes $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$.*

4.5 Robust Open and Redistribute

Now, we show in $\pi_{\mathsf{Open\text{-}Redist}}$ how the parties can efficiently and robustly reshare a batch of (non-private) sharings $[\![X_1]\!], \ldots, [\![X_{n-t}]\!]$ such that $X_i \bmod p_0 \equiv x_i$ into sharings of the same values, with smaller integers, $[x_1], \ldots, [x_{n-t}]$. The importance of this procedure to obtain smaller sharings of the same values is that we will later multiply these sharings by other $[\![\cdot]\!]$ sharings, and we will still desire error-detection. Note that $\pi_{\mathsf{Open\text{-}Redist}}$ will always be run several times, thus we use $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ across invocations.

As in $\Pi_{\mathsf{Check\text{-}Zero}}$, $\pi_{\mathsf{Open\text{-}Redist}}$ uses error-detection and parity checking via a linear code defined by a Vandermonde matrix. Again, this code is defined across the shared values (through linear homomorphism of the sharings; see Section 3.1), and is separate from the error detection for each CRT sharing.

Procedure 5: $\pi_{\mathsf{Open\text{-}Redist}}$

Parameters: This procedure is parameterized by the $n \times (n-t)$ Vandermonde matrix $\boldsymbol{V}$ and the $t \times n$ parity check matrix $\boldsymbol{H}$.

1. The parties have as input sharings $[\![X_1]\!], \ldots, [\![X_{n-t}]\!]$.
2. The parties apply $\boldsymbol{V}$ to $([\![X_1]\!], \ldots, [\![X_{n-t}]\!])^\intercal$ to obtain $([\![Y_1]\!], \ldots, [\![Y_n]\!])^\intercal$. ($|Y_i| \leq |\boldsymbol{V}|(n-t)\max_j |X_j|$)
3. The parties open $[\![Y_i]\!]$ to party $\mathcal{P}_i$ for each $i \in [n]$. (For error detection this requires $|Y_i| \leq (p_{\mathsf{min}}-1)/2$, implied by $|\boldsymbol{V}|(n-t)\max_j |X_j| \leq (p_{\mathsf{min}}-1)/2$.)
4. Each party $\mathcal{P}_i$ reconstructs every corresponding $y_i \bmod p_0$ above (with error detection) and sends to each party $\mathcal{P}_j$ their "shares" $[y_i]$.
5. The parties invoke $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, p_0/2]$ on each party $\mathcal{P}_i$'s sharings $[y_i]$. (From Section 4.3, we see this only guarantees $|y_i| \leq p_0(\frac{C_{\mathsf{Mult}}}{n-t}+1)$.)
6. Then the parties apply $\boldsymbol{H}$ to $([y_1], \ldots, [y_n])^\intercal$ to obtain $([z_1], \ldots, [z_t])^\intercal$. ($|z_j| \leq |\boldsymbol{H}|n \cdot \max_j |y_j| \leq |\boldsymbol{H}|n \cdot p_0(\frac{C_{\mathsf{Mult}}}{n-t}+1)$)
7. The parties invoke $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$ on $[z_1], \ldots, [z_t]$.[a] (This requires $|\boldsymbol{V}|(n-t)\max_j |z_j| < (p_{\mathsf{min}}-1)/2$, implied by $|\boldsymbol{V}|(n-t)\left(|\boldsymbol{H}|n \cdot p_0(\frac{C_{\mathsf{Mult}}}{n-t}+1)\right) < (p_{\mathsf{min}}-1)/2$)

8. If no abort occurs, the parties apply $\boldsymbol{V}^{-1}_{[1:n-t]}$ to $[y_1], \ldots, [y_{n-t}]$ to obtain $[x_1], \ldots, [x_{n-t}]$. ($|x_i| \leq |\boldsymbol{V}^{-1}_{[1:n-t]}|(n-t)\max_j |y_j|$. This implies $|x_i| \leq \mathcal{B}_{\mathsf{OpenRedis}} := |\boldsymbol{V}^{-1}_{[1:n-t]}|(n-t)p_0(\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}+1)$)

[a] Note that $\Pi_{\mathsf{Check\text{-}Zero}}$ is defined on $[\![\cdot]\!]$-sharings, but it works with $[\cdot]$-"sharings" since these are syntactically the same, just with different bounds.

For $n-t$ openings, $\pi_{\mathsf{Open\text{-}Redist}}$ requires one call to $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, p_0/2]$, for every $i = 1, \ldots, n$, and n calls to $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$. Hence, m openings, amortized, require $\frac{m}{n-t}$ calls to $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, p_0/2]$, for every $i = 1, \ldots, n$, and $\frac{m \cdot t}{n-t}$ calls to $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$.

This requires

$$|\boldsymbol{V}|(n-t)\max_j |X_j| \leq (p_{\mathsf{min}}-1)/2 \tag{4}$$

$$|\boldsymbol{V}|(n-t)\left(|\boldsymbol{H}|n \cdot p_0\left(\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}+1\right)\right) < (p_{\mathsf{min}}-1)/2 \tag{5}$$

The following lemma follows immediately from error-detection of $[\![\cdot]\!]$ sharings, correctness of $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ and $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$, and the properties of the Vandermonde and parity-check matrices.

Lemma 7. *In $\pi_{\mathsf{Open\text{-}Redist}}$, if any corrupted party redistributes an incorrect value y_i, then the parties abort. Otherwise, the parties output shares of the original $x_1, \ldots, x_{n-t}$.*

5 Offline Phase

We now present the offline phase of our protocol, which will consist of first generating *random double sharings* as in [DN07]. These random double sharings will be used for multiplication gates in the execution phase, as in [Gar+24]. However, we will also sample some to generate random multiplication triples [Bea92], $[\![A]\!], [\![B]\!], [\![C]\!]$ such that $A \odot B \bmod p_0 \equiv C$ in the offline phase. At the end of the protocol, these will be used to check that the multiplication gates of the execution phase were performed correctly.

5.1 Random Double Sharings Generation

We start with $\pi_{\mathsf{Rand\text{-}Double}}$, which generates random double sharings. Note that $\pi_{\mathsf{Rand\text{-}Double}}$ will always be run several times in parallel, thus we use $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ across all such invocations.

Procedure 6: $\pi_{\mathsf{Rand\text{-}Double}}$

Parameters: This protocol is parameterized by the $(n-t) \times n$ transpose of a Vandermonde matrix $\boldsymbol{V}^\intercal$.

1. Each $\mathcal{P}_i$ samples random $s_i \leftarrow_\$ \mathbb{Z}^{\pm}_{p_0}$ and corresponding sharings $[\![S_i]\!]$ and $\langle \overline{S}_i \rangle$ of s_i. They distribute the shares to other parties. (We let $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}$ and $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}$ be such that, for honest parties, $|S_i| \le \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}$ and $|\overline{S}_i| < \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}$. We specify these values later.)
2. The parties invoke $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$ for each $i \in [n]$ on $\{[\![S_i]\!]\}$ (in parallel across invocations of $\pi_{\mathsf{Rand\text{-}Double}}$). (From Section 4.3, this only ensures $|S_i| \le \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}} := 2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}} \cdot \frac{4\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.)
3. The parties compute and output

$$([\![R_1]\!], \ldots, [\![R_{n-t}]\!])^\intercal \leftarrow \boldsymbol{V}^\intercal \odot ([\![S_1]\!], \ldots, [\![S_n]\!])^\intercal$$
$$(\langle \overline{R}_1 \rangle, \ldots, \langle \overline{R}_{n-t} \rangle)^\intercal \leftarrow \boldsymbol{V}^\intercal \odot (\langle \overline{S}_1 \rangle, \ldots, \langle \overline{S}_n \rangle)^\intercal$$

(We have $|R_t| \le \mathcal{B}_{\mathsf{Rand\text{-}t}} := \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}}|\boldsymbol{V}|n$ and $|R_{2t}| \le \mathcal{B}_{\mathsf{Rand\text{-}2t}} := \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}|\boldsymbol{V}|n$.)

π_{Rand} is defined in the same way, but without the $\langle \overline{R} \rangle$ sharings and the range L'_t from which the masking integer u is sampled for the sharings is $L'_t \leftarrow L^c_t$, for some constant c to be determined later in Section 7.

Generating $n-t$ double sharings takes one call to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$. Hence, amortized, generating m double sharings takes $\frac{m}{n-t}$ calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$. Note that, as we will see in Section 6.3, we do not (for now) require the shares $\overline{S}_i \equiv S_i \bmod p_0$ and moreover do not need to check the size of $\overline{S}_i$.[18] We will detect this later in $\pi_{\mathsf{Sacrifice}}$.

5.2 Multiplication Triples Generation

Now we present π_{Triple} for multiplication triples generation.

Procedure 7: π_{Triple}

1. The parties run π_{Rand} to obtain random sharings $[\![A]\!]$ (resp. $[\![B]\!]$). ($|A|, |B| \le \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}$)
2. Parties call π_{Mult} on input $[\![A]\!]$ and $[\![B]\!]$ to obtain $[\![C]\!]$. (From Section 6.3, we can bound $|C| \le p_0(\mathcal{C}_{\mathsf{Mult}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$. Also, π_{Mult} requires $|\ell| \frac{p_0}{2} \overline{\mathcal{B}}^2_{\mathsf{Rand\text{-}t}} + \mathcal{B}_{\mathsf{Rand\text{-}2t}} \le (p_1 \cdots p_n - 1)/2$)

In our work we will need $\mathcal{C}_{\mathsf{Mult}}$ triples. $\mathcal{C}_{\mathsf{Mult}}$ calls to π_{Triple} need $2\mathcal{C}_{\mathsf{Mult}}$ calls to π_{Rand} and $\mathcal{C}_{\mathsf{Mult}}$ calls to π_{Mult}. In Section 6.3 we analyze the cost of the $\mathcal{C}_{\mathsf{Mult}}$

[18] This is akin to not having any guarantees on degree-$2t$ Shamir secret shares, which is ultimately handled by some verification of the multiplication gates.

calls to π_{Mult}. Regarding the $2\mathcal{C}_{\mathsf{Mult}}$ calls to π_{Rand}, plugging in the analysis from Section 5.1, we see that this costs:

– $\frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t}$ *offline calls* to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$.

6 Online Phase

Finally, we present the online phase of our protocol. It proceeds by executing addition gates locally, and using the random double sharings from the offline phase to perform multiplications, as in [Gar+24]. At the end, the parties sacrifice [Dam+12] the random multiplication triples from the offline phase to ensure that the multiplication gates were executed correctly.

6.1 Input Phase

We begin with a simple procedure for sharing inputs. Each party uses CRT sharing to share their inputs, and then the parties collectively use $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ to ensure that the underlying integers are of the appropriate size.

Procedure 8: π_{Input}

$\mathcal{P}$ has input $x \in \mathbb{Z}^{\pm}_{p_0}$.

1. $\mathcal{P}$ distributes shares $[\![X]\!]$, where $X \equiv x \bmod p_0$. (For an honest $\mathcal{P}$, it should hold that $|X| \leq \mathcal{B}_{\mathsf{Input}}$)
2. The parties call $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}, \mathcal{B}_{\mathsf{Input}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$ on input $[\![X]\!]$ (in parallel for all inputs of $\mathcal{P}$). (This only ensures that $|X| \leq \mathcal{B}_{\mathsf{Input\text{-}slack}} := 2(\mathcal{B}_{\mathsf{Input}} \cdot \mathcal{C}_{\mathsf{Input}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$)

Let $\mathcal{C}_{\mathsf{Input}}$ be the (maximum) number of input gates each party has. π_{Input} calls $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}, \mathcal{B}_{\mathsf{Input}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$ a total of $\mathcal{C}_{\mathsf{Input}}$ times, per party, so its output $[\![X]\!]$ satisfies $|X| \leq \mathcal{B}_{\mathsf{Input\text{-}slack}} := 2\mathcal{B}_{\mathsf{Input}}(\mathcal{C}_{\mathsf{Input}} + 1)$.

6.2 Addition and Multiplication-by-Constant Gates

To add two shared values $[\![X]\!]$ and $[\![Y]\!]$, the parties add their shares locally $[\![X]\!] \oplus [\![Y]\!]$, and to multiply a constant $c \in \mathbb{Z}^{\pm}_{M}$, the parties multiply locally $c \cdot [\![X]\!]$. Let $\mathcal{C}_{\mathsf{LinComb}}$ be a bound such that every input to a multiplication gate—or to an output gate—is a linear combination of at most $\mathcal{C}_{\mathsf{LinComb}}$ outputs of previous multiplication and input gates. Let $\mathcal{B}_{\mathsf{Mult}}$ be a bound $|Z| \leq \mathcal{B}_{\mathsf{Mult}}$ on the output of multiplications $[\![Z]\!]$—we determine what this bound is later. Finally, let $\mathcal{B}_{\mathsf{Wire}} = \max(\mathcal{B}_{\mathsf{Mult}}, \mathcal{B}_{\mathsf{Input\text{-}slack}})$ so that, for every shared value $[\![Z]\!]$ that is either an input wire or the output of a multiplication gate, it holds that $|Z| \leq \mathcal{B}_{\mathsf{Wire}}$. Now, let $[\![X]\!]$ be an input to a multiplication gate. Then

$$|X| \leq p_0 \cdot \mathcal{C}_{\mathsf{LinComb}} \cdot \mathcal{B}_{\mathsf{Wire}} \tag{6}$$

By letting $\mathcal{B}_{\mathsf{Circuit}} := p_0 \cdot \mathcal{C}_{\mathsf{LinComb}} \cdot \mathcal{B}_{\mathsf{Wire}}$, we have that every intermediate shared value $[\![X]\!]$ satisfies $|X| \leq \mathcal{B}_{\mathsf{Circuit}}$.

6.3 Multiplication Gates

We describe a generic multiplication protocol π_{Mult} which we use to handle multiplication gates, but as we will see, also to perform a check that these gates are correctly executed. Recall that Lemma 3 requires all integers of wire value sharings to be divisible by the determinant $\det(\boldsymbol{V}_{\mathcal{H}}^{\intercal})$ of the sub-matrix of the transpose Vandermonde matrix corresponding to the honest parties. Thus, we multiply the input sharings $[\![X]\!], [\![Y]\!]$ by $\ell\ell^{-1}$, where ℓ is the LCM of determinants of $(n-t)\times(n-t)$ sub-matrices of the transpose Vandermonde and ℓ^{-1} is its inverse, modulo p_0.

Procedure 9: π_{Mult}

Parameters: This procedure is parameterized by ℓ, the LCM of determinants of the $(n-t)\times(n-t)$ submatrices of the $(n-t)\times n$ transpose of a Vandermonde matrix $\boldsymbol{V}^{\intercal}$. Let ℓ^{-1} be the inverse of ℓ modulo p_0. (Note that $|\ell^{-1}| \leq p_0/2$)

Input: Two sharings $[\![X]\!], [\![Y]\!]$.

1. The parties run $\pi_{\mathsf{Rand\text{-}Double}}$ to obtain random sharings $[\![U]\!], \langle \overline{U} \rangle$. ($|U| \leq \mathcal{B}_{\mathsf{Rand\text{-}t}}$ and $|\overline{U}| \leq \mathcal{B}_{\mathsf{Rand\text{-}2t}}$)
2. The parties compute $\langle \overline{D} \rangle \leftarrow (\ell\ell^{-1} \cdot [\![X]\!] \odot [\![Y]\!]) \oplus \langle \overline{U} \rangle$ and open $\langle \overline{D} \rangle$ to party $\mathcal{P}_1$. ($|\overline{D}| \leq |\ell||\ell^{-1}||X||Y| + |\overline{U}| \leq |\ell|\frac{p_0}{2}|X||Y| + \mathcal{B}_{\mathsf{Rand\text{-}2t}}$. For reconstruction to be correct it is required then $|\ell|\frac{p_0}{2}|X||Y| + \mathcal{B}_{\mathsf{Rand\text{-}2t}} \leq (p_1 \cdots p_n - 1)/2$)
3. Party $\mathcal{P}_1$ reconstructs D and computes $d \leftarrow D \bmod p_0$, and sends to each party $\mathcal{P}_i$ their "share" $[d]$ (that is, $d \bmod p_i$ to $\mathcal{P}_i$).
4. Parties call $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$ on $[d]$. (As seen in Section 4.3, this only ensures $|d| \leq p_0 \cdot (m+1)$, where m is the number of parallel calls to $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$.)
5. All parties $\mathcal{P}_i$ compute and output $[\![Z]\!] \leftarrow [d] \oplus (-[\![U]\!])$. ($|Z| \leq p_0(m+1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$)

Recall that at this point, it may not be the case (due to adversarial behavior) that $\overline{U} \equiv U \bmod p_0$, and in fact $\overline{U}$ could be any value in $\mathbb{Z}_{p_1 \cdots p_n}$. This is fine, ultimately the reshared value $[d]$ will be some value in $\mathbb{Z}_{p_{\mathsf{min}}}$; if it is not equivalent to $X \cdot Y + U$ modulo p_0, then $\pi_{\mathsf{Sacrifice}}$ will catch the adversary (Lemma 8).

One call to π_{Mult} needs one offline call to $\pi_{\mathsf{Rand\text{-}Double}}$, and one online call to $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$. Overall, plugging in the costs of $\pi_{\mathsf{Rand\text{-}Double}}$, we see that m calls to π_{Mult} need (first item is the cost of $\pi_{\mathsf{Rand\text{-}Double}}$):

- $\frac{m}{n-t}$ offline calls to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$
- m online calls to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$.

π_{Mult} is used in two places. First, we use it to securely compute secret-shared values during the circuit computation. Here, it is called $\mathcal{C}_{\mathsf{Width}}$ times per layer of the circuit, so it needs:

- $\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}$ offline calls[19] to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$.
  The bound guaranteed by these calls is $2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}} \frac{\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.

[19] Technically it is $\mathcal{C}_{\mathsf{Width}}$ calls per layer, but since these calls are offline, we batch them together and approximate them as $\mathcal{C}_{\mathsf{Mult}}$ calls in total.

– $\mathcal{C}_{\mathsf{Width}}$ online calls to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$, for every layer of the circuit. The bound guaranteed by these calls is $p_0(\mathcal{C}_{\mathsf{Width}} + 1)$.

In this context, the bound $|Z| \leq \mathcal{B}_{\mathsf{Mult}}$ guaranteed for the output of multiplication gates $[\![Z]\!]$ can be taken as $\mathcal{B}_{\mathsf{Mult}} = p_0 \cdot (\mathcal{C}_{\mathsf{Width}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$. Furthermore, in this context it is required that

$$|\ell|\frac{p_0}{2}\mathcal{B}^2_{\mathsf{Circuit}} + \mathcal{B}_{\mathsf{Rand\text{-}2t}} \leq (p_1 \cdots p_n - 1)/2 \tag{7}$$

Second, we use π_{Mult} to generate $\mathcal{C}_{\mathsf{Mult}}$ triples in an *offline phase*, which are used to verify these products were done correctly. This requires:

– $\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}$ offline calls to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i = 1, \ldots, n$. The bound guaranteed by these calls is $2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.
– $\mathcal{C}_{\mathsf{Mult}}$ *offline* calls to $\Pi_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$ The bound guaranteed by these calls is $p_0(\mathcal{C}_{\mathsf{Mult}} + 1)$.

Here, the output $[\![C]\!]$ of π_{Triple} satisfies $|C| \leq p_0 \cdot (\mathcal{C}_{\mathsf{Mult}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$. Finally, in this second use of π_{Mult} it is required that

$$|\ell|\frac{p_0}{2}\overline{\mathcal{B}}^2_{\mathsf{Rand\text{-}t}} + \mathcal{B}_{\mathsf{Rand\text{-}2t}} \leq (p_1 \cdots p_n - 1)/2 \tag{8}$$

6.4 Verification Phase

Lastly, we present the classical triple sacrifice procedure $\pi_{\mathsf{Sacrifice}}$ [Dam+12], which sacrifices triples from the offline phase to ensure that batches of multiplication gates were performed correctly. This procedure uses $\pi_{\mathsf{Open\text{-}Redist}}$ for robust openings of values and $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$ to check that a batch of sharings share 0.

Procedure 10: $\pi_{\mathsf{Sacrifice}}$

The procedure takes in a batch of $n - t$ triples computed during the circuit evaluation: $\{([\![X_i]\!], [\![Y_i]\!], [\![Z_i]\!])\}_{i\in[n-t]}$. $\pi_{\mathsf{Sacrifice}}$ will be called on a total of $\mathcal{C}_{\mathsf{Mult}}$ inputs. ($|X_i|, |Y_i|, |Z_i| \leq \mathcal{B}_{\mathsf{Circuit}}$.)

1. Call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\alpha \in \mathbb{Z}^{\pm}_{p_0}$
2. Call π_{Triple} to get $\{([\![A_i]\!], [\![B_i]\!], [\![C_i]\!])\}_{i\in[n-t]}$. ($|A_i|, |B_i| \leq \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}$, and $|C_i| \leq p_0(\mathcal{C}_{\mathsf{Mult}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$)
3. Compute locally $[\![D_i]\!] \leftarrow \alpha\ell\ell^{-1} \odot [\![X_i]\!] \oplus (-[\![A_i]\!])$ and $[\![E_i]\!] \leftarrow \ell\ell^{-1} \cdot [\![Y_i]\!] \oplus (-[\![B_i]\!])$. ($|D_i|, |E_i| \leq \frac{p_0}{2} \cdot |\ell| \cdot \frac{p_0}{2} \cdot \mathcal{B}_{\mathsf{Circuit}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}} = (\frac{p_0}{2})^2 |\ell|\mathcal{B}_{\mathsf{Circuit}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}$)
4. Call $\pi_{\mathsf{Open\text{-}Redist}}$ on input $\{[\![D_i]\!], [\![E_i]\!]\}_{i\in[n-t]}$, so the parties obtain $\{[d_i], [e_i]\}_{i\in[n-t]}$. ($|d_i|, |e_i| \leq \mathcal{B}_{\mathsf{OpenRedis}}$. This call requires $|\boldsymbol{V}|(n - t)\max_j |D_j| \leq (p_{\mathsf{min}} - 1)/2$, and same for $|E_j|$. This is implied by $|\boldsymbol{V}|(n - t)\left((\frac{p_0}{2})^2 |\ell|\mathcal{B}_{\mathsf{Circuit}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}\right) \leq (p_{\mathsf{min}} - 1)/2$)
5. The parties compute $[\![W_i]\!] \leftarrow ([d_i] \odot [e_i]) \oplus ([d_i] \odot [\![B_i]\!]) \oplus ([e_i] \odot [\![A_i]\!]) \oplus [\![C_i]\!] \oplus (-\alpha \cdot [\![Z_i]\!])$, for $i \in [n-t]$. ($|W_i| \leq \mathcal{B}^2_{\mathsf{OpenRedis}} + 2\mathcal{B}_{\mathsf{OpenRedis}}\overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}} + (p_0(\mathcal{C}_{\mathsf{Mult}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}) + \frac{p_0}{2}\mathcal{B}_{\mathsf{Circuit}}$)
6. The parties call $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$ on $[\![W_1]\!], \ldots, [\![W_{n-t}]\!]$, and abort if this aborts. (This requires the bound above to be

$|\boldsymbol{V}|(n-t)\max_j|W_j| \leq (p_{\mathsf{min}}-1)/2$. This is implied by $|\boldsymbol{V}|(n-t)\left(\mathcal{B}^2_{\mathsf{OpenRedis}} + 2\mathcal{B}_{\mathsf{OpenRedis}}\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}} + (p_0(\mathcal{C}_{\mathsf{Mult}}+1)+\mathcal{B}_{\mathsf{Rand\text{-}t}}) + \frac{p_0}{2}\mathcal{B}_{\mathsf{Circuit}}\right) \leq (p_{\mathsf{min}}-1)/2)$

We call $\pi_{\mathsf{Sacrifice}}$ for a total of $\mathcal{C}_{\mathsf{Mult}}$ multiplication gates, which requires $\mathcal{C}_{\mathsf{Mult}}$ offline calls to π_{Triple}, $\mathcal{C}_{\mathsf{Mult}}/(n-t)$ online calls to $\pi_{\mathsf{Open\text{-}Redist}}$, and $\mathcal{C}_{\mathsf{Mult}}/(n-t)$ online calls to $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$ (on $[\![\cdot]\!]$-sharings). Now, $\mathcal{C}_{\mathsf{Mult}}$ offline calls to π_{Triple}, taking into account the analyses from Sections 5.2 and 6.3, cost:

- $\frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t}$ offline calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i=1,\ldots,n$ (these are the calls to π_{Rand} to get $[\![A]\!], [\![B]\!]$). The bound guaranteed by these calls is $2(\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}\frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.
- $\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}$ offline calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}, \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}]$, for $i=1,\ldots,n$ (these are the calls to $\pi_{\mathsf{Rand\text{-}Double}}$ to get $[\![R]\!]$ needed for multiplication of $[\![A]\!], [\![B]\!]$). This can be merged with the calls for the online phase. The bound guaranteed by these calls is $2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}\frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.
- $\mathcal{C}_{\mathsf{Mult}}$ offline calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_1, p_0/2]$. The bound guaranteed by these calls is $p_0(\mathcal{C}_{\mathsf{Mult}}+1)$.

The $\mathcal{C}_{\mathsf{Mult}}$ online calls to $\pi_{\mathsf{Open\text{-}Redist}}$ cost:

- $\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}$ parallel online calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Bounds}}[\mathcal{P}_i, p_0/2]$, for every $i=1,\ldots,n$,
- $\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t}$ parallel online calls to $\mathit{\Pi}_{\mathsf{Check\text{-}Zero}}$ (on $[\cdot]$-sharings).

These calls to $\pi_{\mathsf{Open\text{-}Redist}}$ require from Eq. (4):

$$|\boldsymbol{V}|(n-t)\left(\left(\frac{p_0}{2}\right)^2|\ell|\mathcal{B}_{\mathsf{Circuit}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}\right) \leq (p_{\mathsf{min}}-1)/2 \tag{9}$$

Finally, the $\mathcal{C}_{\mathsf{Mult}}$ calls (on $[\![\cdot]\!]$-sharings) to $\mathcal{F}_{\mathsf{Check\text{-}Zero}}$ have the requirement from Eq. (3) (replacing the bound for $|W_i|$) that

$$|\boldsymbol{V}|(n-t)\left(\mathcal{B}^2_{\mathsf{OpenRedis}} + 2\mathcal{B}_{\mathsf{OpenRedis}}\overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}} + (p_0(\mathcal{C}_{\mathsf{Mult}}+1)+\mathcal{B}_{\mathsf{Rand\text{-}t}}) + \frac{p_0}{2}\mathcal{B}_{\mathsf{Circuit}}\right) \leq (p_{\mathsf{min}}-1)/2 \tag{10}$$

We prove the following Lemma in the full version.

Lemma 8. *If for any of the input triple sharings $[\![X_i]\!], [\![Y_i]\!], [\![Z_i]\!]$, $X_i \cdot Y_i \not\equiv Z_i \bmod p_0$, then the parties abort with probability at least $1-2^{-\lambda}$.*

6.5 Output Phase

For every output wire $[\![X]\!]$, the parties send the shares of X to each other and perform error detection. We have $|X| \leq \mathcal{B}_{\mathsf{Circuit}}$, so error detection demands

$$\mathcal{B}_{\mathsf{Circuit}} \leq (p_{\mathsf{min}}-1)/2, \tag{11}$$

but note that this is implied by Eq. (9).

6.6 Putting all the Bounds Together

Throughout all of our protocols we have defined different bound requirements that ultimately determine several parameters of our construction, such as share sizes and some others. First, we recall some of the notation we have introduced to represent some bounds:

- $\mathcal{B}_{\mathsf{Input}}$: the bound $|S| \leq \mathcal{B}_{\mathsf{Input}}$ for honestly generated shares $[\![S]\!]$, for the input gates.
- $\mathcal{B}_{\mathsf{Input\text{-}slack}}$: as above, but guaranteed for corrupt parties. $\mathcal{B}_{\mathsf{Input\text{-}slack}} = 2(\mathcal{B}_{\mathsf{Input}} \cdot \mathcal{C}_{\mathsf{Input}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$, where $\mathcal{C}_{\mathsf{Input}}$ is a bound on the number of input gates each party has.
- $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}$ and $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}$: the bound $|S| \leq \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}$ and $|\overline{S}| \leq \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}$ for honestly generated shares $[\![S]\!]$ and $\langle \overline{S} \rangle$, for the calls to π_{Rand} and $\pi_{\mathsf{Rand\text{-}Double}}$.
- $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}}$: same bound as above, but guaranteed for corrupt parties. This is $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}} := 2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}} \cdot \frac{4\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.
- $\mathcal{B}_{\mathsf{Rand\text{-}t}}$ and $\mathcal{B}_{\mathsf{Rand\text{-}2t}}$: bounds $|S| \leq \mathcal{B}_{\mathsf{Rand\text{-}t}}$ and $|\overline{S}| \leq \mathcal{B}_{\mathsf{Rand\text{-}2t}}$ on the output $([\![S]\!], \langle \overline{S} \rangle)$ of $\pi_{\mathsf{Rand\text{-}Double}}$. $\mathcal{B}_{\mathsf{Rand\text{-}t}} := \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}}|\boldsymbol{V}|n = \left(2(\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}} \cdot \frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})\right) \cdot |\boldsymbol{V}|n$, and $\mathcal{B}_{\mathsf{Rand\text{-}2t}} := \mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}} \cdot |\boldsymbol{V}|n$.
- $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}$ and $\overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}}$: the bound $|S| \leq \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}$ for honestly generated shares $[\![S]\!]$, for the calls to π_{Rand} in π_{Triple}. Same for $\overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}}$ for calls to $\Pi_{\mathsf{Check\text{-}Bounds}}$.
- $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}}$: same bound as above, but guaranteed for corrupt parties. This is $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}} := 2(\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}} \cdot \frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}})$.
- $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}$: bounds $|S| \leq \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}}$ on the output $[\![S]\!]$ of π_{Rand}. $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}} := \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t\text{-}slack}}|\boldsymbol{V}|n = \left(2(\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}} \cdot \frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} + \overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}})\right) \cdot |\boldsymbol{V}|n$.
- $\mathcal{B}_{\mathsf{OpenRedis}}$: bound $|c| \leq \mathcal{B}_{\mathsf{OpenRedis}}$ in the output $[c]$ of $\pi_{\mathsf{Open\text{-}Redist}}$. $\mathcal{B}_{\mathsf{OpenRedis}} := |\boldsymbol{V}^{-1}_{[1:n-t]}|(n-t)p_0(\frac{\mathcal{C}_{\mathsf{Mult}}}{n-t} + 1)$
- $\mathcal{B}_{\mathsf{Mult}}$: bound $|Z| \leq \mathcal{B}_{\mathsf{Mult}}$ for the output $[\![Z]\!]$ of π_{Mult}, when used to compute the circuit. $\mathcal{B}_{\mathsf{Mult}} = p_0 \cdot (\mathcal{C}_{\mathsf{Width}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}$
- $\mathcal{B}_{\mathsf{Wire}}$: bound $|S| \leq \mathcal{B}_{\mathsf{Wire}}$ for every share $[\![S]\!]$ that is either an input gate, or the output of a multiplication gate. We had set $\mathcal{B}_{\mathsf{Wire}} = \max(\mathcal{B}_{\mathsf{Mult}}, \mathcal{B}_{\mathsf{Input\text{-}slack}})$, but for simplicity we set instead the looser bound

$$\begin{aligned}\mathcal{B}_{\mathsf{Wire}} &= \mathcal{B}_{\mathsf{Mult}} + \mathcal{B}_{\mathsf{Input\text{-}slack}} \\ &= (p_0 \cdot (\mathcal{C}_{\mathsf{Width}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}) + 2(\mathcal{B}_{\mathsf{Input}} \cdot \mathcal{C}_{\mathsf{Input}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}).\end{aligned}$$

- $\mathcal{B}_{\mathsf{Circuit}}$: bound $|S| \leq \mathcal{B}_{\mathsf{Circuit}}$ guaranteed for every shared wire value $[\![S]\!]$.

$$\begin{aligned}\mathcal{B}_{\mathsf{Circuit}} &= p_0 \cdot \mathcal{C}_{\mathsf{LinComb}} \cdot \mathcal{B}_{\mathsf{Wire}} \\ &= p_0 \cdot \mathcal{C}_{\mathsf{LinComb}} \cdot \big((p_0 \cdot (\mathcal{C}_{\mathsf{Width}} + 1) + \mathcal{B}_{\mathsf{Rand\text{-}t}}) + 2(\mathcal{B}_{\mathsf{Input}} \cdot \mathcal{C}_{\mathsf{Input}} + \overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}})\big)\end{aligned}$$

  The difference with $\mathcal{B}_{\mathsf{Wire}}$ is that $\mathcal{B}_{\mathsf{Circuit}}$ accounts for addition/mult-by-constant gates.

The "free variables" are $\mathcal{B}_{\mathsf{Input}}$, $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}t}}$, $\mathcal{B}_{\mathsf{Rand\text{-}Share\text{-}2t}}$, $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}Share\text{-}t}}$, and $\overline{\overline{\mathcal{B}}}_{\mathsf{Rand\text{-}Share\text{-}t}}$, and the requirements are summarized in equations (5), (7), (8), (9) and (10). Note that each call to $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ also has its own requirement (equation (2)), but these are subsumed by the other requirements. This is because the guarantee of $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ is that each input is at most some integer, which is bigger than that of equation (2), and these inputs will later be opened in the protocol.

From these requirements one can derive *exactly* what the parameters should be, but instead we focus on the *asymptotics*. First note that equation (5) is subsumed by equation (9). Observe that the matrices $\boldsymbol{V}, \boldsymbol{H}$ are defined over p_0 and thus each element of them and their inverses are at most p_0. Also, we assume that $\mathcal{C}_{\mathsf{LinComb}} = \Theta(\mathcal{C}_{\mathsf{Mult}})$ and $\mathcal{C}_{\mathsf{Mult}} \gg \mathcal{C}_{\mathsf{Input}}, \mathcal{C}_{\mathsf{Width}}$. From this we have that:

- $\mathcal{B}_{\mathsf{Rand\text{-}t}} = O(p_0^2 \cdot L_t^c \cdot n)$.
- $\mathcal{B}_{\mathsf{Circuit}} = O(p_0 \cdot \mathcal{B}_{\mathsf{Rand\text{-}t}}) = O(p_0^3 \cdot L_t^c \cdot n)$.
- $\mathcal{B}_{\mathsf{Rand\text{-}2t}} = O(p_0^2 \cdot L_{2t} \cdot n)$.
- $\overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}} = O(p_0^2 \cdot L_t^{c'} \cdot n)$
- $\mathcal{B}_{\mathsf{OpenRedis}} = O(p_0^2 \cdot \mathcal{C}_{\mathsf{Mult}})$.

Thus, remembering that $|\ell| = O(n^{n^2})$:

- The left hand side of Equation (7) is bounded by $O(n^{n^2} \cdot p_0^7 \cdot L_t^{2c} + p_0^2 \cdot L_{2t} \cdot n)$.
- The left hand side of Equation (8) is bounded by $O(n^{n^2} \cdot p_0^5 \cdot L_t^{2c'} + p_0^2 \cdot L_{2t} \cdot n)$.
- The left hand side of Equation (9) is bounded by $O(n^{n^2} \cdot p_0^6 \cdot L_t^c + n^2 \cdot p_0^3 \cdot L_t^{c'})$.
- The left hand side of Equation (10) is bounded by $O(np_0 \cdot (p_0 \cdot \mathcal{B}_{\mathsf{Circuit}} + \mathcal{B}_{\mathsf{OpenRedis}} \overline{\mathcal{B}}_{\mathsf{Rand\text{-}t}})) = O(n^2 p_0^5 \cdot L_t^c + n^2 p_0^5 \mathcal{C}_{\mathsf{Mult}} \cdot L_t^{c'})$.

Thus we have the new asymptotic requirements:

$$p_{\mathsf{min}}/2 = \Omega(n^{n^2} \cdot p_0^6 \cdot L_t^c + n^2 p_0^5 \cdot \mathcal{C}_{\mathsf{Mult}} \cdot L_t^{c'}).$$

and

$$\frac{1}{2} \cdot \prod_{i=1}^{n} p_i = \Omega(n^{n^2} \cdot p_0^7 \cdot L_t^{2c'} + L_{2t} \cdot p_0^2 n).$$

Put together, assuming that $\log p_1 \approx \log p_2 \approx \cdots \approx \log p_n$,[20] we need:

$$(n-t) \cdot \log p_1 \geq n^2 \log n + 6 \log p_0 + c \log L_t \tag{12}$$
$$(n-t) \cdot \log p_1 \geq 2 \log n + 5 \log p_0 + \log \mathcal{C}_{\mathsf{Mult}} + c' \log L_t, \tag{13}$$
$$n \log p_1 \geq n^2 \log n + 7 \log p_0 + 2c' \log L_t, \tag{14}$$
$$n \log p_1 \geq \log L_{2t} + 2 \log p_0 + \log n. \tag{15}$$

7 Main Protocol

We summarize the entire protocol in Π_{Main}.

[20] By the Prime Number Theorem, they all can be within additive constants for large enough n.

Protocol 11: Π_{Main}

Input Phase: For each input x of party $\mathcal{P}$, all parties execute $\pi_{\mathsf{Input}}(x)$ to obtain $[\![X]\!]$, where $X \equiv x \bmod p_0$.

Online Phase:

1. For addition and multiplication-by-constant gates, the parties locally perform them as described in Section 6.2.
2. For multiplication gates, the parties use π_{Mult}.

Verification Phase: The parties run $\pi_{\mathsf{Sacrifice}}$ on batches of $n-t$ multiplication gates at a time, aborting if any checks fail.

Output Phase: If they do not abort, the parties send all shares of output gates $[\![Z]\!]$ to each other, as in Section 6.5.

We prove the following theorem in the full version.

Theorem 2. *Π_{Main} UC-realizes $\mathcal{F}_{\mathsf{MPC}}$ in the $(\mathcal{F}_{\mathsf{Check\text{-}Zero}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Check\text{-}Bounds}})$-hybrid model with $p_{\mathsf{max}}/L_t + \delta_1 + \delta_2 + \delta_3 + \delta_4$ error, where*

$$\delta_1 \leq \mathcal{C}_{\mathsf{Mult}} \cdot \frac{2p_{\mathsf{max}} + \mathcal{B}^2_{\mathsf{Circuit}} \cdot n^{n^2}}{L_{2t} \cdot p_0},$$

$$\delta_2 \leq \mathcal{C}_{\mathsf{Mult}} \cdot \frac{2p_{\mathsf{max}} + \overline{\mathcal{B}}^2_{\mathsf{Rand\text{-}t}} \cdot n^{n^2}}{L_{2t} \cdot p_0},$$

$$\delta_3 \leq 2\mathcal{C}_{\mathsf{Mult}} \cdot \frac{2p_{\mathsf{max}} + \mathcal{B}_{\mathsf{Circuit\text{-}no\text{-}slack}} \cdot n^{n^2}}{L_t^c \cdot p_0},$$

$$\delta_4 \leq \mathcal{C}_{\mathsf{Output}} \cdot \frac{2p_{\mathsf{max}} + (\mathcal{C}_{\mathsf{LinComb}} \cdot p_0 + p_0 \cdot p_{\mathsf{max}}) \cdot n^{n^2}}{L_t \cdot p_0},$$

where $\mathcal{B}_{\mathsf{Circuit\text{-}no\text{-}slack}} = O(p_0^3 \cdot \mathcal{C}_{\mathsf{LinComb}} \cdot n \cdot L_t)$.

We also have from Lemma 5, error $\lambda \cdot \frac{2\mathcal{C}_{\mathsf{Mult}}}{n-t} \cdot \frac{1}{L_t^{c'-c}}$ (from the application of $\mathcal{F}_{\mathsf{Check\text{-}Bounds}}$ for the sampling of the $[\![A]\!], [\![B]\!]$ parts of triples). Thus for negligible error, we need:[21]

$$L_t \geq 2^\lambda \cdot \left(\left(2 \cdot \lambda \cdot \frac{\mathcal{C}_{\mathsf{Mult}}}{n-t} \right)^{1/(c'-c)} + (2 \cdot \mathcal{C}_{\mathsf{Mult}} \cdot \mathcal{B}_{\mathsf{Circuit\text{-}no\text{-}slack}} \cdot n^{n^2}/p_0)^{1/c} + \mathcal{C}_{\mathsf{Output}} \cdot n^{n^2} \cdot p_{\mathsf{max}} \right)$$

and

$$L_{2t} \geq 2^\lambda \cdot n^{n^2} \cdot \mathcal{C}_{\mathsf{Mult}}/p_0 \cdot (\overline{\mathcal{B}}^2_{\mathsf{Rand\text{-}t}} + \mathcal{B}^2_{\mathsf{Circuit}}).$$

[21] Note that the LHS of this first inequality indeed is exponentially larger in L_t than the RHS.

Taking logarithms yields (ignoring $\log \lambda$ factors):

$$\log L_t \geq \lambda + \log p_{\mathsf{max}}, \tag{16}$$

$$\frac{c-1}{c} \cdot \log L_t \geq \frac{1}{c'-c} \cdot \log \mathcal{C}_{\mathsf{Mult}} + \frac{1}{c} \cdot (2 \log \mathcal{C}_{\mathsf{Mult}} + 2 \log p_0 + \log n) + n^2 \log n + \mathcal{C}_{\mathsf{Output}}, \tag{17}$$

$$\log L_{2t} \geq \lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + 3 \log p_0 + 2c' \log L_t + 2 \log n, \tag{18}$$

$$\log L_{2t} \geq \lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + 5 \log p_0 + 2c \log L_t + 2 \log n. \tag{19}$$

Now, to obtain our setting of $\log p_1$ (and thus asymptotically all $\log p_i$): Applying Equations 16 and 17 to Equation 12:

$$(n - t - ct) \log p_1 = \Omega(\lambda + n^2 \log n + \log p_0 + \log \mathcal{C}_{\mathsf{Mult}} + \log \mathcal{C}_{\mathsf{Output}}),$$

applying Equations 16 and 17 to Equation 13:

$$(n - t - c't) \log p_1 = \Omega(\lambda + n^2 \log n + \log p_0 + \log \mathcal{C}_{\mathsf{Mult}} + \log \mathcal{C}_{\mathsf{Output}}),$$

applying Equations 16 and 17 to Equation 14:

$$(n - 2c't) \log p_1 = \Omega(\lambda + n^2 \log n + \log p_0 + \log \mathcal{C}_{\mathsf{Mult}} + \log \mathcal{C}_{\mathsf{Output}}),$$

applying Equations 16, 17, and 18 to Equation 15:

$$(n - 2c't) \log p_1 = \Omega(\lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + \log p_0 + \log \mathcal{C}_{\mathsf{Output}}),$$

applying Equations 16, 17, and 19 to Equation 15:

$$(n - 2ct) \log p_1 = \Omega(\lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + \log p_0 + \log \mathcal{C}_{\mathsf{Output}}),$$

Therefore, recalling that $t < (1/2 - \varepsilon)n$, we need

$$\begin{aligned}
1 < c &< \frac{1}{1/2 - \varepsilon} - 1, \\
1 < c' &< \frac{1}{1/2 - \varepsilon} - 1, \\
1 < c &< \frac{1}{1 - 2\varepsilon}, \\
1 < c' &< \frac{1}{1 - 2\varepsilon}, \\
c &< c',
\end{aligned}$$

which are all possible for $\varepsilon > 0$.

7.1 Efficiency.

We see that in order for $\sum_{i=1}^{n} \log p_i = \Theta(\log p_0)$, we need $\sum_{i=1}^{n} \log p_i = \Theta(n \log p_1) = \Theta(\lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + \log \mathcal{C}_{\mathsf{Output}} + \log p_0) = \Theta(\log p_0)$, and thus $\log p_0 = \Theta(\lambda + n^2 \log n + \log \mathcal{C}_{\mathsf{Mult}} + \log \mathcal{C}_{\mathsf{Output}})$, which is the same as in [Gar+24]. With this:

- The cost of all $\mathcal{F}_{\text{Check-Bounds}}$ invocations amortizes away since it only requires λ openings and new sharings.
- $\Pi_{\text{Check-Zero}}$ costs n openings for $\Theta(n)$ values, so it has constant overhead.
- $\pi_{\text{Open-Redist}}$ thus has $O(n)$ openings for $\Theta(n)$ values, so it has constant overhead.
- π_{Rand} and $\pi_{\text{Rand-Double}}$ both create $\Theta(n)$ random sharings for the cost of n sampled random sharings, so they have constant overhead.
- π_{Mult} costs 1 opening, thus it has constant overhead. (So too does π_{Triple})
- $\pi_{\text{Sacrifice}}$ just uses amortized calls to $\pi_{\text{Open-Redist}}$ and $\mathcal{F}_{\text{Check-Zero}}$, so it has constant overhead.

Thus, in total, Π_{Main} has constant overhead.

Disclaimer

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan") and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

[AB83] C. Asmuth and J. Bloom. "A modular approach to key safeguarding". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 208–210. DOI: 10.1109/TIT.1983.1056651.

[Bea92] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology — CRYPTO '91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432. ISBN: 978-3-540-46766-3.

[Bec+21] Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. "Order-C Secure Multiparty Computation for Highly Repetitive Circuits". In: 2021, pp. 663–693. DOI: 10.1007/978-3-030-77886-6_23.

[BEP24] Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. "Perfectly Secure Fluid MPC with Abort and Linear Communication Complexity". In: 1.4 (2024), p. 28. DOI: 10.62056/aesg89n4e.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[BL17] Carsten Baum and Vadim Lyubashevsky. *Simple Amortized Proofs of Shortness for Linear Relations over Polynomial Rings*. Cryptology ePrint Archive, Report 2017/759. 2017. URL: https://eprint.iacr.org/2017/759.

[BN20] Carsten Baum and Ariel Nof. "Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography". In: 2020, pp. 495–526. DOI: 10.1007/978-3-030-45374-9_17.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty Unconditionally Secure Protocols (Extended Abstract)". In: 1988, pp. 11–19. DOI: 10.1145/62212.62214.

[Chi+23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. "Eos: Efficient Private Delegation of zkSNARK Provers". In: 2023, pp. 6453–6469. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. "Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography". In: 2010, pp. 445–465. DOI: 10.1007/978-3-642-13190-5_23.

[DLN19] Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. "Communication Lower Bounds for Statistically Secure MPC, With or Without Preprocessing". In: 2019, pp. 61–84. DOI: 10.1007/978-3-030-26951-7_3.

[DN07] Ivan Damgård and Jesper Buus Nielsen. "Scalable and Unconditionally Secure Multiparty Computation". In: 2007, pp. 572–590. DOI: 10.1007/978-3-540-74143-5_32.

[Esc+23] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. "SuperPack: Dishonest Majority MPC with Constant Online Communication". In: 2023, pp. 220–250. DOI: 10.1007/978-3-031-30617-4_8.

[Fan+25] Zhiyong Fang, Sanjam Garg, Bhaskar Roberts, Wenxuan Wu, and Yupeng Zhang. *Collaborative zkSNARKs with Sublinear Prover Time and Constant Proof Size*. Cryptology ePrint Archive, Paper 2025/1388. 2025. URL: https://eprint.iacr.org/2025/1388.

[FY92] Matthew K. Franklin and Moti Yung. "Communication Complexity of Secure Computation (Extended Abstract)". In: 1992, pp. 699–710. DOI: 10.1145/129712.129780.

[Gar+23a] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. "zkSaaS: Zero-Knowledge SNARKs as a Service". In: 2023, pp. 4427–4444. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/garg.

[Gar+23b] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. "Cryptography with Weights: MPC, Encryption and Signatures". In: 2023, pp. 295–327. DOI: 10.1007/978-3-031-38557-5_10.

[Gar+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, and Mingyuan Wang. "Scalable Multiparty Computation from Non-linear Secret Sharing". In: 2024, pp. 384–417. DOI: 10.1007/978-3-031-68397-8_12.

[Gar+25] Sanjam Garg, Aarushi Goel, Abhishek Jain, Bhaskar Roberts, and Sruthi Sekar. "Malicious Security in Collaborative zk-SNARKs: More than Meets the Eye". In: 2025, pp. 396–428. DOI: 10.1007/978-3-032-01907-3_13.

[Gen+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. "Circuits resilient to additive attacks with applications to secure computation". In: 2014, pp. 495–504. DOI: 10.1145/2591796.2591861.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties". In: 2022, pp. 458–487. DOI: 10.1007/978-3-031-06944-4_16.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. "Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits". In: 2015, pp. 721–741. DOI: 10.1007/978-3-662-48000-7_35.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "Unconditional Communication-Efficient MPC via Hall's Marriage Theorem". In: 2021, pp. 275–304. DOI: 10.1007/978-3-030-84245-1_10.

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing". In: 2022, pp. 3–32. DOI: 10.1007/978-3-031-15985-5_1.

[GRS99] Oded Goldreich, Dana Ron, and Madhu Sudan. "Chinese Remaindering with Errors". In: 1999, pp. 225–234. DOI: 10.1145/301250.301309.

[GSY21] S. Dov Gordon, Daniel Starin, and Arkady Yerukhimovich. "The More the Merrier: Reducing the Cost of Large Scale MPC". In: 2021, pp. 694–723. DOI: 10.1007/978-3-030-77886-6_24.

[Hu+25] Yuncong Hu, Pratyush Mishra, Xiao Wang, Jie Xie, Kang Yang, Yu Yu, and Yuwen Zhang. "DFS: Delegation-friendly zkSNARK and Private Delegation of Provers". In: *USENIX Security 2025*. 2025.

[Liu+24] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. *Scalable Collaborative zk-SNARK: Fully Distributed*

Proof Generation and Malicious Security. Cryptology ePrint Archive, Report 2024/143. 2024. URL: https://eprint.iacr.org/2024/143.

[Liu+25] Xuanming Liu, Shelei Zhou, Yinghao Wang, Yanxin Pang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. "Scalable Collaborative zk-SNARK and Its Application to Fully Distributed Proof Delegation". In: *USENIX Security 2025*. 2025.

[LNS20] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. "Practical Lattice-Based Zero-Knowledge Proofs for Integer Relations". In: 2020, pp. 1051–1070. DOI: 10.1145/3372297.3417894.

[Mig83] Maurice Mignotte. "How to Share a Secret?" In: 1983, pp. 371–375. DOI: 10.1007/3-540-39466-4_27.

[OB22] Alex Ozdemir and Dan Boneh. "Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets". In: 2022, pp. 4291–4308. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir.

[RB89] Tal Rabin and Michael Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: 1989, pp. 73–85. DOI: 10.1145/73007.73014.

[Yan+25] Yunbo Yang, Yuejia Cheng, Kailun Wang, Xiaoguo Li, Jianfei Sun, Jiachen Shen, Xiaolei Dong, Zhenfu Cao, Guomin Yang, and Robert H. Deng. "Siniel: Distributed Privacy-Preserving zkSNARK". In: 2025.