Sublinear GMW-Style Compiler for MPC with Preprocessing

Elette Boyle[1], Niv Gilboa[2], Yuval Ishai[3], and Ariel Nof[3]

[1] IDC Herzliya, Israel.
eboyle@alum.mit.edu
[2] Ben-Gurion University, Israel.
gilboan@bgu.ac.il
[3] Technion, Israel.
{yuvali,ariel.nof}@cs.technion.ac.il

Abstract. We consider the efficiency of protocols for secure multiparty computation (MPC) with a dishonest majority. A popular approach for the design of such protocols is to employ *preprocessing*. Before the inputs are known, the parties generate correlated secret randomness, which is consumed by a fast and possibly "information-theoretic" online protocol. A powerful technique for securing such protocols against malicious parties uses *homomorphic MACs* to authenticate the values produced by the online protocol. Compared to a baseline protocol, which is only secure against semi-honest parties, this involves a significant increase in the size of the correlated randomness, by a factor of up to a statistical security parameter. Different approaches for partially mitigating this extra storage cost come at the expense of increasing the online communication.

In this work we propose a new technique for protecting MPC with preprocessing against malicious parties. We show that for circuit evaluation protocols that satisfy mild security and structural requirements, that are met by many standard protocols with semi-honest security, the extra *additive* storage and online communication costs are both *logarithmic* in the circuit size. This applies to Boolean circuits and to arithmetic circuits over fields or rings, and to both information-theoretic and computationally secure protocols. Our protocol can be viewed as a sublinear information-theoretic variant of the celebrated "GMW compiler" that applies to natural protocols for MPC with preprocessing.

Our compiler makes a novel use of the techniques of Boneh et al. (Crypto 2019) for sublinear distributed zero knowledge, which were previously only used in the setting of *honest-majority* MPC.

1 Introduction

Protocols for secure computation [30, 23, 3, 14] enable a set of parties with private inputs to compute a joint function of their inputs while revealing nothing but the output. Secure computation protocols provide a general-purpose tool for computing on sensitive data while eliminating single points of failure, and their asymptotic and concrete optimization has been the subject of a significant body of research.

A popular approach for the design of such protocols is to employ *preprocessing*. Before the inputs are known, the parties generate correlated secret randomness, which is consumed by a lightweight and typically "information-theoretic" online protocol. This model, known also as the offline/online model, is in particular appealing when no honest majority can be guaranteed, since it allows to push the heavy "cryptographic" part of the protocol to the offline phase, minimizing the cost of the online protocol. It also enables modular analysis, where security of the online protocol can be treated independently given access to an idealized "dealer" who delivers the correlated randomness from the offline phase. The dealer can then be emulated by the parties via a secure preprocessing protocol for generating the correlated randomness. Alternatively, the dealer can be directly realized by an external party or by trusted hardware, both of which are only used before the protocol's execution.

Originating from the work of Beaver [1], who showed how to use "multiplication triples" for secure arithmetic computation with no honest majority, many protocols for secure computation make extensive use of correlated randomness [4, 19, 25, 28, 16, 20, 18, 15, 9]. In particular, a powerful technique for securing such protocols against malicious parties uses *homomorphic MACs* to authenticate the values produced by the online protocol [4, 19].

Efficiency of MPC protocols with security against malicious parties is typically measured with respect to the costs of the best known protocols with a "minimal" level of security, namely security against semi-honest parties, who act as prescribed by the protocol but try to learn additional information from messages they receive. In the case of MPC with preprocessing, two primary efficiency metrics are:

i. overhead to the *online communication* cost; and
ii. overhead to the *correlated randomness* consumed by the online protocol.

Indeed, communication and storage costs (as opposed to computation) typically dominate the online cost of concretely efficient MPC protocols in the preprocessing model. Minimizing both of these measures simultaneously is instrumental for achieving a fast and scalable online protocol.

However, current MPC with preprocessing protocols exhibit a trade-off between these two efficiency goals. For the case of evaluating an arithmetic circuit C with $|C|$ multiplication gates, some protocols [4, 28, 19, 15] succeed to minimize the online communication cost, but with a large correlated randomness overhead of $O(|C|)$ field elements over large fields, or $O(|C| \cdot \kappa)$ for Boolean circuits or circuits over rings of any size, where κ is a statistical security parameter. Other protocols [20, 13] manage to achieve $O(|C|)$ correlated randomness size for Boolean circuits (which asymptotically improves the storage cost), but at the expense of substantially increasing the online communication cost and relying on algebraic geometric codes that hurt concrete efficiency.

This raises the following question about MPC with preprocessing:

Can we achieve malicious security with sublinear (in $|C|$) additive overhead in <u>both</u> the online communication and amount of correlated randomness?

Further, *can this be done without introducing any new assumption?*

1.1 Our Contribution

In this work, we answer the above question in the affirmative. We present a complier from any MPC with preprocessing protocol for arithmetic circuits that satisfies mild security and structural requirements (met by most standard protocols with semi-honest security), to one achieving standard security against malicious adversaries, where the extra *additive* storage and online communication costs are both *logarithmic* in the circuit size. This applies to Boolean circuits and to arithmetic circuits over fields or rings, and to both information-theoretic and computationally secure protocols. In particular, our compiler introduces no additional assumptions. Our compiler can be viewed as an information-theoretic variant of the "GMW compiler" [23] that applies to the setting of MPC with preprocessing.

The compiler requires two properties from the underlying semi-honest secure protocol. First, the protocol must be secure *up to additive attacks*. Such a protocol guarantees not only standard semi-honest security, but further that the actions of a malicious adversary reduce to the ability to inject additive errors to the circuit wires (independent of secret values). This notion was formulated by [21], who showed that many semi-honest protocols that are based on secret sharing (both in the honest- and the dishonest-majority setting), satisfy this requirement. This in particular is true for standard semi-honest protocols in the preprocessing model, which is what interests us in this work.

Our second requirement is a structural robustness property we refer to as *"star-compliance."* We observe that most natural semi-honest protocols with preprocessing exhibit the following structure. The correlated randomness includes additive shares of a random mask r_w for each wire w within the circuit being evaluated; then, in the online phase, the parties iteratively compute the *masked* wire values $(x_w - r_w)$.[4] Effectively, after an honest execution, each wire value x_w is held in a particular secret-shared form, which can be linearly reconstructed either by all parties together by adding to $(x_w - r_w)$ their shares of r_w, or by any individual party together with the dealer who knows r_w—thus forming a "star" structure.

Recall that the dealer is a physical or virtual entity that generates correlated randomness for the online protocol. One of the ideas of this work, as we will see later, is that the dealer itself can act as an additional honest party in the system, with the restriction that its actions must be fully done before the start of the online phase.

Our main result is summarized by the following theorem, which assumes only point-to-point communication except for a final broadcast (of one bit) to enable security with unanimous abort.

[4] Note that x_w may not be the *correct* wire value following an additive attack by the adversary. This is not an issue.

Theorem 1.1 (Sublinear GMW-style compiler, informal). *Let C be an arithmetic circuit of size $|C|$ (counting multiplication gates, inputs and outputs) over a ring R, where R is either a finite field $\mathbb{F}$ or the ring $\mathbb{Z}_{2^k}$. Then, every n-party MPC protocol Π in the preprocessing model that computes C with additive security and is star-compliant can be compiled into a protocol Π' that computes C with security against malicious parties and the following efficiency features.*

- Correlated randomness: *Π' uses the correlated randomness of Π and additional $O(n \cdot \log |C| \cdot \kappa)$ elements of R per party for a statistical security parameter κ;*
- Online communication: *In addition to the online communication of Π, each party in Π' communicates $O(\log |C| \cdot \kappa)$ elements of R.*

Furthermore, if Π has information-theoretic security then so does Π'.

We use this theorem to derive concretely efficient protocols with malicious security, by applying our compiler to semi-honest secure protocols based on multiplication triples [1]. Using circuit-dependent preprocessing (where the correlated randomness can depend on the choice of the circuit C), we obtain a protocol where each party sends $(2 - \frac{2}{n})$ elements per multiplication gate, and the correlated randomness includes $|C| + O(n \cdot \log |C| \cdot \kappa)$ ring elements given to one of the parties and $O(n \cdot \log |C| \cdot \kappa)$ elements given to the remaining $n - 1$ parties (in addition to seeds to a pseudorandom generator). Beginning with a semi-honest protocol with circuit-independent preprocessing (where the correlated randomness depends on the size of C, but not its topology), we obtain a protocol with the same amount of correlated randomness but with slightly higher communication, namely, $(4 - \frac{4}{n})$ elements per multiplication gate per party.

The (logarithmic size) extra correlated randomness introduced by our compiler *does* depend on the structure of the circuit, and thus the resulting protocols in both cases are in the circuit-dependent preprocessing model. However, we address both versions, as the semi-honest portion of the correlated randomness is a dominating cost that can be generated more efficiently in the circuit-independent case (including recent techniques for concretely efficient generation with sublinear communication via pseudorandom correlation generators (PCGs) [7]).[5]

Corollary 1.1 (Efficient MPC with preprocessing, informal). *Let C be an arithmetic circuit of size $|C|$ (counting multiplication gates, inputs and outputs) over ring R, where R is either a finite field or the ring $\mathbb{Z}_{2^k}$. Then there exist n-party MPC protocols in the preprocessing model with security against malicious parties and the following efficiency features.*

- Correlated randomness*: $4 \cdot |C| + O(n \cdot \log |C| \cdot \kappa)$ R-elements per party, where κ is a statistical security parameter. Settling for computational security*

[5] We remark that efficient PCG constructions also exist for more complex correlations, including circuit-dependent multiplication triples, as well as authenticated multiplication triples [8]; however, these constructions rely on stronger tools and do not extend effectively beyond the 2-party setting.

and making a black-box use of a pseudorandom generator, this can be compressed to $|C|+O(n\cdot\log|C|\cdot\kappa)$ *R-elements to one party and* $O(n\cdot\log|C|\cdot\kappa)$ *to the other* $n-1$ *parties.*

- Online communication*:*
  - $(2-\frac{2}{n})$ *R-elements per party per gate (circuit-dependent preprocessing);*
  - $(4-\frac{4}{n})$ *R-elements per party per gate (circuit-independent preprocessing).*

More concretely, the correlated randomness in the above protocols consists of shared multiplication triples (i.e., additive shares of random $a, b \in R$, and $a \cdot b$, where the shares of random values are directly compressible via black-box use of a pseudorandom generator), together with additional $O(n\cdot\log|C|\cdot\kappa)$ R-elements resulting from our compiler.

Note that our improvement is particularly significant when the computation is carried out over small fields or rings. For example, for Boolean circuits we are able to eliminate the $O(|C|\cdot\kappa)$ additive storage overhead completely, without increasing online communication as done in previous works.

A PCG-based compression. As noted above, by using a PCG to compress the multiplication triples we can get the *total* storage complexity to be sublinear in $|C|$. In particular, for secure 2-party computation of Boolean circuits, each triple can be locally generated using 2 random OT correlations, where the latter can be efficiently compressed using fast PCGs for OT [7, 6, 29]. For concretely efficient PCG-based protocols for $n \geq 3$ parties, one can use a PCG for OLE [8] for arithmetic circuits over big fields or a PCG for OT for Boolean circuits, though the latter incurs an $O(n^2)$ multiplicative overhead to the online communication.

Distributing the dealer. In Section 4 we discuss the cost of emulating the dealer in our protocols by a secure preprocessing protocol involving the parties. Concretely, we show that given the multiplication triples required by the semi-honest protocol, generating the (sublinear) extra correlated randomness can be done using roughly $4|C| + 2n|C|$ secure multiplications.

1.2 Our Techniques

Fully linear proofs. The main technical building block in our compiler is a *fully linear proof system* [5], enabling information-theoretic zero-knowledge proofs with sublinear communication, on secret-shared input statements. In this setting, there is a prover who wishes to prove some statement over an input x to a verifier. In each round of the protocol, the prover produces a proof which can be queried by the verifier using linear queries only. Moreover, the verifier is allowed to also make linear queries to the input x (this is what makes the proof system *fully* linear). The main observation of [5] is that for low-degree languages (i.e., languages for which membership can be checked using a degree-d polynomial), there exist fully linear proof systems with communication which is only logarithmic in the size of the input.

A central motivating application of such proof systems is to proofs on input statements which are distributed or linearly secret-shared between two or more verifiers. If the prover also shares the proof in the same way, then the parties can query their shares of the proof and the input, and then reconstruct the answers. This is particularly useful for achieving malicious security in MPC protocols, since it allows the parties to prove honest behavior, given the data being shared across the parties. Indeed, this tool was used by [5, 10, 11] to compile semi-honest protocols to malicious security with sublinear communication cost in the honest majority setting, by observing that the statement to be proven in MPC protocols can be represented by a low-degree polynomial.

All of these works crucially relies on the fact that in the honest majority setting, the secret sharing is *robust*, meaning that the shares held by the honest parties determine all the other shares. This fact is what prevents the corrupted parties from cheating in the proof, since even if the prover colludes with some of the verifiers, they cannot change the answers to the queries without being caught by the honest verifiers.

Thus, at first glance, it seems that fully linear proof systems *cannot* be used in the setting where an honest majority is not guaranteed, without adding some kind of authentication to all sharings held by the parties during the execution, thereby increasing significantly the amount of correlated randomness. Our main observation towards overcoming this challenge is that, in the preprocessing model, we can view the star-sharing scheme discussed above, as a robust secret sharing, since any honest party together with the trusted dealer determine the shares held by the corrupted parties! Leveraging this property, we view the dealer as one of the verifiers in the fully linear proof.

Our main technical contribution is a novel protocol with sublinear communication to verify the correctness of a semi-honest computation, which builds upon the fully linear proof system. In each step of the protocol, we carefully make sure that each piece of information along the way is robustly shared across the parties and the dealer using the star-sharing scheme, which is what eventually guarantees that any cheating will be detected. Finally, we observe that all messages sent by the dealer during the verification protocol are a function of random data, and so we can let the dealer precompute all its messages and commit to them before the start of the computation. When distributing the role of the dealer, this amounts to having the parties securely compute the dealer's messages, and then output an authenticated secret sharing of each message, which can be later reconstructed by the parties. The main and final point here is that the proof size and the public randomness in the verification protocol are both *logarithmic* in the size of the computed circuit. This follows directly from the efficiency features of fully linear proof systems for simple languages [5]. Thus, the amount of correlated randomness the dealer needs to generate is also logarithmic in the size of the circuit, thereby achieving our main result.

We believe that our technique is quite broadly applicable and will open the door to new applications of fully linear proof systems in the dishonest majority setting, which is something that has not been done prior to this work.

1.3 Related Work

A long line of works have used an authenticated variant of Beaver's protocol [1] to achieve malicious security [4, 19, 16, 15, 26, 27], without increasing the online communication cost beyond that of the semi-honest protocol. These protocols use *authenticated multiplication triples* of the form $(a\cdot\Delta, b\cdot\Delta, ab\cdot\Delta)$ for a random secret Δ. The parties receives additive shares of each value in the authenticated triple as well as shares of Δ (and of course shares of a, b and c, which are required for the semi-honest protocol). These are used to authenticate the opening of the actual values. Over a field $\mathbb{F}$, the cheating probability is $\frac{1}{|\mathbb{F}|}$. Thus, over a large field this method doubles the amount of correlated randomness compared to that of the semi-honest protocol. When working over a small field, the triples should be produced over a larger field, thus increasing the size of correlated randomness. The situation is worse for rings, where the cheating probability is $1/2$ regardless of the size of the ring. Naively, this implies an overhead of $|C|\cdot\kappa$ for some statistical parameter κ. This is indeed the case for the TinyOT protocol [28] for Boolean circuits.

However, some improvements were suggested over the years. The MiniMac protocol [20] (optimized and implemented in [17]) focuses on reducing overall computation costs for circuits over small fields (including preprocessing correlated randomness size) at the expense of greater online communication. Their idea is to batch the authentication via linear error-correcting codes (ECC). However, the ECC being used requires good minimal distance for security within multiplications of batched vectors. Achieving this requires smaller rate, translating to greater communication overhead. A recent work by [13] has suggested an alternative to linear ECC of MiniMAC, via "reverse multiplication friendly embeddings" for embedding $(\mathbb{F}_q)^k$-vector mults into a single $F_{q^{k'}}$ field mult. However, the gap between k and k' yields overheads. While this construction reduces the online work, it requires generating extra correlated randomness in the preprocessing phase. The MiniMac protocol and its followers offer a trade-off between the amount of correlated randomness and online communication for computation over Boolean circuits. Their batching ideas remove the κ multiplicative factor, but increase the online communication. In any way, both the correlated randomness and the online cost do not match those of the underlying semi-honest protocols, which we are able to achieve.

Over a large ring, the SPDZ-2k protocol [15] introduced a way to reduce the extra correlated randomness, without increasing communication. Specifically, they require *adding* κ bits to the size of the authenticated triples instead of multiplying the size by κ. For large rings, this amounts to doubling the size of the correlated randomness compared to fields.

Finally, a different approach for 2-party computation was suggested in the TinyTable protocol [18], based on generating a permuted version of its truth table. The overhead of this protocol is $O(|C|)$ for both communication and the correlated randomness.

As can be seen from the above, we are the first to achieve sublinear overhead for both the communication cost and the amount of correlated randomness.

2 Preliminaries

Notation. Let $P_1, \ldots, P_n$ be the parties participating in the protocol. We use $[n]$ to denote the set $\{1, \ldots, n\}$. Let R be a ring which is either a finite field $\mathbb{F}$ or the ring $\mathbb{Z}_{2^k}$ and let $|R|$ be its size. Finally, let κ be the security parameter.

2.1 MPC with Preprocessing

In our setting, there is a set of n parties who wish to jointly run some computation. We assume that all parties are connected via point-to-point channels, which enable them to send private messages to each other.

We begin with defining the meaning of an n-party protocol to compute any functionality in the preprocessing model.

Definition 2.1 (MPC with preprocessing). *Let $\mathcal{F}$ be a family on n-party functionalities and let $f \in \mathcal{F}$ be a function description. A protocol Π to compute $\mathcal{F}$ consists of the PPT algorithm* NextMsg*, which given $(1^\kappa, f, j, i, x_i, r_i, \boldsymbol{m})$ outputs a vector of messages sent by P_i in round j, based on its input x_i, randomness r_i and vector $\boldsymbol{m}$ of messages sent to P_i in previous rounds. If the output of* NextMsg *to P_i is of the form (out, y), then P_i outputs y and halts.*

We say that Π is a protocol with function-dependent preprocessing*, if in addition to* NextMsg*, it consists of a PPT algorithm $\mathcal{D}$ (called "the dealer"), which receives 1^κ and f as an input, and outputs correlated random strings $r_1, \ldots, r_n$. We say that Π is a protocol with* function-independent preprocessing*, if $\mathcal{D}$ receives only a bound 1^S on the size of f as an input instead of f.*

A protocol $\pi = (\mathsf{NextMsg}, \mathcal{D})$ computes any arithmetic circuit, when $\mathcal{F}$ is the class of arithmetic circuits and f is a description of a ring R and a circuit C over R, with the size S being a description of the ring and the number of output wires and multiplication gates in C.

To define what it means to securely compute a functionality, we follow the standard ideal-world vs. real-world paradigm of MPC [22, 12]. Let $\mathcal{A}$ be an adversary who chooses a set of parties before the beginning of the execution and corrupts them. There are two main types of adversaries which are usually considered in the literature. A *semi-honest* adversary follow the protocol instructions, but sees the input and randomness of the corrupted parties, and all the messages they receive in the execution. A *malicious* adversary can also choose the messages sent by the corrupted parties. We assume that the adversary is *rushing*, meaning that it first receives the messages sent by the honest parties in each round, and only then determines the corrupted parties' messages in this round.

To formally define security, let $\text{REAL}_{\Pi,\mathcal{A},T}(1^\kappa, f, \boldsymbol{v})$ be a random variable that consists of the view of the adversary A controlling a set of parties T, and the honest parties' outputs, following an execution of Π over a vector of inputs $\boldsymbol{v}$ to compute f with security parameter κ. Similarly, we define an ideal-world execution with an ideal-world adversary $\mathcal{S}$, where $\mathcal{S}$ and the honest parties interact with a trusted party who computes f for them. We consider secure computation

with selective abort, meaning that $\mathcal{S}$ is allowed to send the trusted party computing f a special command abort. Specifically, $\mathcal{S}$ can send an abort command instead of handing the corrupted parties' inputs to the trusted party (causing all parties to abort the execution), or, hand the inputs and then, after receiving the corrupted parties' outputs from the trusted party, send abort_j for an honest party P_j, preventing it from receiving its outputs[6]. We denote by $\text{IDEAL}_{\mathcal{F},\mathcal{S},T}(1^\kappa, f, \boldsymbol{v})$, the random variable that consists of the output of $\mathcal{S}$'s and the honest parties in an ideal execution to compute f, over a vector of inputs $\boldsymbol{v}$, where $\mathcal{S}$ controls a set of parties T. The security definition states that a protocol Π securely computes f with statistical error ε, if for every real-world adversary there exists an ideal-world adversary, such that the statistical distance between the two random variables is less than ε.

Definition 2.2 (Statistically-secure MPC with preprocessing). *Let $\mathcal{F}$ be a family of n-party functionalities and $\varepsilon = \varepsilon(\kappa, f)$ be a statistical error bound. We say that a protocol $\Pi = (\mathsf{NextMsg}, \mathcal{D})$ ε-securely computes $\mathcal{F}$ with abort in the preprocessing model, if for every real-world malicious adversary $\mathcal{A}$ controlling a set of parties T with $|T| \leq n-1$, there exists an ideal-world adversary $\mathcal{S}$, such that for every $f \in \mathcal{F}$, every κ and every vector of inputs $\boldsymbol{v}$ it holds that*

$$SD\left(\text{REAL}_{\Pi,\mathcal{A},T}(f, \boldsymbol{v}), \text{IDEAL}_{\mathcal{F},\mathcal{S},T}(f, \boldsymbol{v})\right) \leq \varepsilon$$

where $SD(X, Y)$ is the statistical distance between X and Y.

Secure computation of circuits with additive attacks [21]. In this work, our protocol computes arithmetic circuits, which are defined in a natural way, using addition and multiplication gates. We next define a weaker notion of security for computing arithmetic circuits, called "security-up-to-additive-attack", which was introduced by Genkin et al [21]. In this model, we also allow the ideal-world adversary $\mathcal{S}$ to add an error to the value on some of the wires of the circuit. Specifically, we allow additive attacks on *input wires to multiplication gates and on the circuit's output wires*. The trusted party then determines the output of the honest parties by computing the circuit over the parties' inputs and the additive errors. We denote by $\text{IDEAL}^{add}_{\mathcal{F},\mathcal{S},T}(1^\kappa, C, \boldsymbol{v})$ the random variable consists of $\mathcal{S}$'s and honest parties' outputs in such an execution. Given this new model of ideal-world execution, security is defined similarly to Definition 2.2.

Definition 2.3 (Secure MPC with additive security). *Let $\mathcal{F}$ be the class of n-party functionalities represented by an arithmetic circuit C and let $\varepsilon = \varepsilon(\kappa, C)$ be a statistical error bound. We say that a protocol $\Pi = (\mathsf{NextMsg}, \mathcal{D})$ ε-securely computes $\mathcal{F}$ with abort and with additive security, in the pre-processing model, if for every real-world malicious adversary $\mathcal{A}$ controlling a set of parties T with $|T| \leq n-1$, there exists an ideal-world adversary $\mathcal{S}$, such that*

[6] It easy to modify our protocol so that the honest parties unanimously abort by running a single Byzantine agreement at the end of the protocol. For simplicity, we omit the details from the description of our protocols.

for every circuit $C \in \mathcal{F}$, every κ, and every vector of inputs $\boldsymbol{v}$ it holds that $SD\left(\text{REAL}_{\Pi,\mathcal{A},T}(1^\kappa, C, \boldsymbol{v}), \text{IDEAL}^{add}_{\mathcal{F},\mathcal{S},T}(1^\kappa, C, \boldsymbol{v})\right) \leq \varepsilon$.

The hybrid model. We use the hybrid model to prove security of our protocols. In this model, the parties run a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition theorem of [12] states that it is possible to replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. When the subfunctionality is g, we say that the protocol works in the g-hybrid model.

Instantiations. Many standard semi-honest protocols in the preprocessing model used in the literature are in fact, or can easily be converted into being additively-secure. Most notably, a semi-honest protocol which uses the well-known Beaver's method [1] to compute multiplication gates via random triples satisfies this definition. For completeness, in Appendix A.1 we present the version of this method which relies on circuit-dependent preprocessing (due to [9] and [2]), and in Appendix A.1, the standard circuit-independent version.

2.2 Fully Linear Proof Systems

A main technical building block in our protocols is a *fully linear* proof system [5], enabling information-theoretic sublinear-communication zero-knowledge proofs on secret-shared input statements. More concretely, we can use any (public-coin) *zero-knowledge fully linear interactive oracle proof* (zk-FLIOP), as defined in Definition 2.4. In a nutshell, a zk-FLIOP is an information-theoretic proof system in which a prover P wishes to prove that some statement about an input x to a verifier V. In each round of the protocol, P produces a proof which, together with x, can be queried by V using *linear queries* only. Then, a public random challenge is generated and the parties proceed to the next round. At the end, the verifier V accepts or rejects based on the answers it received to its queries.

Definition 2.4 (Public-coin zk-FLIOP [5]). *A public-coin fully linear interactive proof system over R with ρ-round and ℓ-query and message length $(u_1, \ldots, u_\rho) \in \mathbb{N}^t$, consists of a randomized prover algorithm P and a deterministic verifier algorithm V. Let the input to P be $x \in R^m$ and let $r_0 = \perp$. In each round $i \in [\rho]$:*

1. *P outputs a proof $\pi_i \in R^{u_1}$, computed as a function of x, $r_1, \ldots, r_{i-1}$ and $\pi_1, \ldots, \pi_{i-1}$.*
2. *A random public challenge r_i is chosen uniformly from a finite set S_i.*
3. *ℓ linear oracle queries $q^i_1, \ldots, q^i_\ell \in R^{m+u_i}$ are determined based on $r_1, \ldots, r_i$. Then, V receives ℓ answers $(\langle q^i_1, x||\pi_i\rangle, \ldots, \langle q^i_\ell, x||\pi_i\rangle)$.*

At the end of round ρ, V outputs accept *or* reject *based on the random challenges and all the answers to the queries.*

Let $\mathcal{L} \subseteq R^m$ be an efficiently recognizable language. We say that ρ-round ℓ-query interactive fully linear protocol $(\mathsf{P}_{\mathsf{FLIOP}}, \mathsf{V}_{\mathsf{FLIOP}})$ over R is zero-knowledge fully linear interactive oracle proof *system for $\mathcal{L}$ with soundness error ϵ if it satisfies the following properties:*

- Completeness: *If $x \in \mathcal{L}$, then $\mathsf{V}_{\mathsf{FLIOP}}$ always outputs* accept
- Soundness: *If $x \notin \mathcal{L}$, then for all P^*, the probability that $\mathsf{V}_{\mathsf{FLIOP}}$ outputs* accept *is at most $2^{-\epsilon}$.*
- Zero-knowledge: *There exists a simulator $\mathcal{S}_{\mathsf{FLIOP}}$ such that for all $x \in \mathcal{L}$ it holds that $\mathcal{S}_{\mathsf{FLIOP}} \equiv \mathsf{view}_{[\mathsf{P}_{\mathsf{FLIOP}}(x), \mathsf{V}_{\mathsf{FLIOP}}]}(\mathsf{V}_{\mathsf{FLIOP}})$ (where the verifier's view $\mathsf{view}_{[\mathsf{P}_{\mathsf{FLIOP}}(x), \mathsf{V}_{\mathsf{FLIOP}}]}(\mathsf{V}_{\mathsf{FLIOP}})$ consists of $\{r_i\}_{i\in[\rho]}$ and $\{(q_1^i, \ldots, q_\ell^i)\}_{i\in[\rho]}$).*

In this paper, we will use this tool for degree-d languages. That is, languages for which membership can be checked using a degree-d polynomial. The following theorem, which will be used by us, states that for degree-d languages, there are zk-FLIOP protocols with sublinear communication and rounds in the size of the input and number of monomials.

Theorem 2.1 ([5]). *Let $q : R^m \to R$ be a polynomial of degree-d with M monomials, and let $\mathcal{L}_q = \{x \in R^m \mid q(x) = 0\}$. Let ϵ be the required soudness error. Then, there is a zk-FLIOP for $\mathcal{L}_q$ with the following properties:*

- Constant rounds, $d = 2$: *It has 1 round, proof length $O(\eta\sqrt{m})$, challenge length $O(\eta)$ and the number of queries is $O(\sqrt{m})$, where $\eta = \log_{|R|}\left(\frac{\sqrt{m}}{\epsilon}\right)$ when R is a finite field, and $\eta = \log_2\left(\frac{\sqrt{m}}{\epsilon}\right)$ when $R = \mathbb{Z}_{2^k}$. The computational complexity is $\tilde{O}(M)$.*
- Logarithmic rounds, $d \geq 2$: *It has $O(\log M)$ rounds, proof length $O(d\eta \log M)$, challenge length $O(\eta \log M)$ and the number of queries is $O(d + \log M)$, where $\eta = \log_{|R|}\left(\frac{d \log m}{\epsilon}\right)$ when R is a finite field, and $\eta = \log_2\left(\frac{d \log m}{\epsilon}\right)$ when $R = \mathbb{Z}_{2^k}$. The computational complexity is $O(dM)$.*

2.3 Ideal Functionalities

We now describe two ideal functionalities that will be used in our construction. We stress that both of them are called sublinear number of times (in the size of the computed circuit), and so any way to implement them will suffice.

Honest dealer commitment with selective abort. We denote by $F_{\mathsf{com}}^{\mathsf{dealer}}$ an ideal functionality which allows an honest dealer to commit to a value which is revealed to parties at a later stage. Upon receiving a secret from the dealer, the functionality $F_{\mathsf{com}}^{\mathsf{dealer}}$ stores it. Then, upon receiving a request from the honest

parties to reveal it to parties in a set J, it lets the adversary decide for each party in J, whether to send each party P_j in J the secret or the command abort_j.

To implement it with information-theoretic security we can use information-theoretic MACs as in [4, 19]. Specifically, each party will hold an additive sharing of the secret x, and in addition, will hold an additive sharing of a information-theoretic MAC over x computed with each party's key. Then, when opening the secret towards a party P_i, all parties send it their additive shares of x and their additive shares of the MAC computed using P_i's key. Since P_i knows its own key, it can use it to check the correctness of x. If any party tries to cheat, then over a field $\mathbb{F}$, it will succeed without being caught with probability of $\frac{1}{|\mathbb{F}|}$. Over a small field or a ring, we can have the MAC over an extension field or ring, to achieve a sufficiently small error.

Broadcast with selective abort. Throughout the paper, when we say that a party broadcasts x to the other parties, it means that it uses an ideal functionality $\mathcal{F}_{\mathsf{bc}}$ which allows sending a message to all parties, while, as before, giving the adversary the ability to cause any party to abort. This can be implemented by having each party sending x to all other parties and then having all parties echo-broadcast the message they received to the other parties. It is possible to batch the second-round check for many messages together, by taking a random linear combination of all received messages. The random coefficients can be derived from a single random element r, by taking $r, r^2 \ldots$ and so on. If the parties check m messages together, then the random linear combination yields a polynomial of degree m, which is evaluated on a random point r. Thus, the cheating probability in this case when working over a field $\mathbb{F}$ is, by the Schwartz-Zippel Lemma, $\frac{m}{|\mathbb{F}|}$. As before, to obtain a sufficiently small error over small fields or over rings, this check should be run over a suitable extension field or ring.

3 The General Framework

In this section, we present a protocol to compute any arithmetic circuit with malicious security and dishonest majority. Our protocol works by first computing the circuit using a secure-up-to-additive-attack protocol, and then running a light verification step, where the parties verify the correctness of the computation and abort if cheating was detected. Our protocol is statistically secure in the preprocessing model, i.e., it relies on a trusted dealer $\mathcal{D}$ which provides correlated randomness to the parties. We will discuss how to securely distribute the dealer in the next section.

Before proceeding, we define an additional property that will be required from our protocol. Specifically, we require the parties to maintain an invariant over wires which we call "star-sharing".

Definition 3.1 (Star-sharing). *We say that $x \in R$ is* star-shared *across a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and a trusted dealer $\mathcal{D}$, if there exists $\hat{x}, (r_1, \ldots, r_n)$, such that each party P_i holds the pair $(\hat{x}, r_i)$, where $\hat{x} = x - r$, $r = \sum_{i=1}^{n} r_i$, and $\mathcal{D}$ holds $\{r_i\}_{i=1}^{n}$.*

The main feature of this sharing scheme is that it is *robust*, in the sense that an honest party and the dealer alone determine all the other values, and in particular the values held by the corrupted parties. In addition, as we will see later, given star-sharing of x and star-sharing of y, this scheme allows local conversion to an *additive* sharing of $x \cdot y$. These two features will play an important role in our constructions.

We next define what it means for a protocol to be "star-sharing compliant".

Definition 3.2 (Star-sharing compliance). *Let $\Pi = (\mathsf{NextMsg}, \mathcal{D})$ be a protocol with preprocessing to compute any arithmetic circuit C, and let W denote the set of output wires and input wires to multiplication gates in C. We say that Π is* star sharing compliant *if the following holds: if all parties follow the protocol's instructions, then the parties hold a star-sharing of the value on each wire $w \in W$.*

Note that if a protocol is both secure-up-to-additive-attack and star-sharing compliant, then it implies that the parties hold on each wire $w \in W$ a star-sharing of either the correct value or of a different value determined by the adversary's additive attack.

3.1 Verifying Correctness via zk-FLIOP

In this section, we present our protocol to verify the correctness of the values the parties hold on the circuit's wires. Recall that we allow the adversary to add errors to wires of the circuit. The protocol we describe in this section aims to detect such cheating. Let W be the set of the circuit's output wires and multiplication gates' input wires. For each wire $w \in W$, the parties need to verify that they hold a sharing of the correct value on w, given the sharings they hold on wires that feed w. Specifically, let G_w be the set of multiplication gates that feed w (i.e., that between their output wire and w there are no other multiplication gates). For each $g \in G_w$, let x_1^g, x_2^g be the two input wires to g. The parties wish thus to verify for each $w \in W$ that $\phi(x_w, \{x_1^g, x_2^g\}_{g \in G_w}) = x_w - \sum_{g \in G_w} x_1^g \cdot x_2^g = 0$. Recall that the parties hold $\hat{x}_w = x_w - r_w$, $\hat{x}_1^g = x_1^g - r_1^g$, $\hat{x}_2^g = x_2^g - r_2^g$ on each wire, as well as additive shares $r_{w,i}$, $r_{1,i}^g$ and $r_{2,i}^g$ for each party P_i. The trusted dealer $\mathcal{D}$ knows the additive shares of all parties and so knows the mask on each wire. Now, in the protocol, instead of checking equality to 0 for each equation separately, the parties will batch all the checks together, by taking a random linear combination of all $\phi(x_w, \{x_1^g, x_2^g\}_{g \in G_w})$ for each $w \in W$. That is, the parties will check that

$$p(W) = \sum_{w \in W} \alpha_w \cdot \phi\left(x_w, \{x_1^g, x_2^g\}_{g \in G_w}\right) = 0.$$

Next, for each multiplication gate g_ℓ, let W^{g_ℓ} be the set of wires w for which $g_\ell \in G_w$ (i.e., that g_ℓ's output feed these wires). Then, let $\gamma_\ell = \sum_{w \in W^{g_\ell}} \alpha_w$.

Letting mult be the set of all multiplication gates, we can thus write

$$\begin{aligned}
p(W) &= \sum_{w\in W} \alpha_w \cdot x_w - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (x_1^{g_\ell} \cdot x_2^{g_\ell}) \\
&= \sum_{w\in W} \alpha_w \cdot (\hat{x}_w + r_w) - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot ((\hat{x}_1^{g_\ell} + r_1^{g_\ell}) \cdot (\hat{x}_2^{g_\ell} + r_2^{g_\ell})) \\
&= \sum_{w\in W} \alpha_w \cdot \hat{x}_w + \sum_{w\in W} \alpha_w \cdot r_w - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot \hat{x}_2^{g_\ell}) \\
&\quad - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot r_2^{g_\ell} + \hat{x}_2^{g_\ell} \cdot r_1^{g_\ell}) + \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (r_1^{g_\ell} \cdot r_2^{g_\ell})
\end{aligned}$$

Now, letting

$$\begin{aligned}
\Lambda &= \sum_{w\in W} \alpha_w \cdot \hat{x}_w - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot \hat{x}_2^{g_\ell}), \\
\Gamma_i &= \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot r_{2,i}^{g_\ell} + \hat{x}_2^{g_\ell} \cdot r_{1,i}^{g_\ell}) \qquad (1)
\end{aligned}$$

and

$$\Omega = \sum_{w\in W} \alpha_w \cdot r_w + \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (r_1^{g_\ell} \cdot r_2^{g_\ell})$$

we have that checking that $p(W) = 0$ is equivalent to checking that

$$\Lambda - \sum_{i=1}^{n} \Gamma_i + \Omega = 0.$$

Observe that the parties can locally compute Λ, each party can locally compute Γ_i and the dealer can locally compute Ω. In our protocol, we will ask each P_i to compute Γ_i and share it to the other parties via our robust star-sharing scheme. This can be done by having the trusted dealer hand a random string s_i to P_i , which then broadcasts $\hat{\Gamma}_i = \Gamma_i - s_i$ to the parties. Similarly, the trusted dealer can compute Ω and share it to the parties. Since now Γ_i for each $i \in [n]$ and Ω are shared in a robust way across the parties, and Λ is known, the parties can locally compute a robust secret sharing of $p(W)$, open it by unmasking the secret with the help of the dealer, and check equality to 0. The only remaining problem is that a corrupt P_i may have cheated and share an incorrect Γ_i. Here is where the zk-FLIOP machinery becomes useful. Define the vector of inputs $\boldsymbol{y} \in \mathbb{F}^{|W|+4|\mathsf{mult}|+2}$ as:

$$\begin{aligned}
\boldsymbol{y} &= (y_1, \ldots, y_{4|\mathsf{mult}|+2}) \\
&= \left(\hat{\Gamma}_i, s_i, \left\{(\gamma_\ell \cdot \hat{x}_1^{g_\ell}), r_{2,i}^{g_\ell}, (\gamma_\ell \cdot \hat{x}_2^{g_\ell}), r_{1,i}^{g_\ell}\right\}_{g_\ell \in \mathsf{mult}}\right) \qquad (2)
\end{aligned}$$

and consider the 2-degree polynomial c defined by

$$c(\boldsymbol{y}) = y_1 + y_2 + \sum_{k=1}^{|\mathsf{mult}|} \left(y_{[4(k-1)+|W|+3]} \cdot y_{[4(k-1)+|W|+4]} + y_{[4(k-1)+|W|+5]} \cdot y_{[4(k-1)+|W|+6]}\right).$$

This polynomial checks that each party star-shared Γ_i correctly, by verifying that Eq. (1) holds. By Theorem 2.1, there exists a zk-FLIOP for proving the satisfiability of this polynomial with sublinear proof size. We thus let each party P_i prove that it shared the correct value, by proving that the output of the polynomial is 0. In particular, party P_i emulates the role of the prover in the zk-FLIOP protocol, whereas the other parties emulate together the role of the verifier. A crucial point that we rely upon in the protocol, is that each input to the circuit is known by either all parties *or* by P_i and the dealer. In addition, in the zk-FLIOP protocol, we ask the prover to star-share the proof that it generates in each step. This implies that each piece of information (inputs or the proof) is known by an honest participant (i.e., an honest party or the trusted dealer). This fact is what helps us to prevent a cheating prover from convincing the other parties that a false statement is correct. From the side of the verifiers, holding their star-shares of both the proof and the input, they can make the zk-FLIOP queries over their *shares*. Observe that here we crucially rely on the fact that in zk-FLIOP, all the queries are *linear*, and so querying the star-shares of the proof or the input, will yield a star-sharing of the answer. Then, the answers are revealed by having the trusted dealer send its star-share of the answers (these shares are eventually a random mask of the answer). Privacy is maintained in this process, since the parties see in each round, a masked proof which looks random, and answers to the linear queries, which by the zero-knowledge property of the zk-FLIOP, leak no information on the inputs and can be simulated. Formally, our protocol works as follows (we describe the protocol for *finite fields* and explain how to extend it to rings later):

Π_{vrfy}*:* Let $(\mathsf{P}_{\mathsf{FLIOP}}, \mathsf{V}_{\mathsf{FLIOP}})$ be a zk-FLIOP protocol with ρ rounds, ℓ-queries per round and message length $u_1, \ldots, u_\rho \in \mathbb{N}$.

1. The trusted dealer $\mathcal{D}$:
   (a) For each $i \in [n]$, it chooses a random $s_i \in \mathbb{F}$ and hands it to P_i.
   (b) chooses a random seed $\alpha \in \mathbb{F}$ and hands in to the parties.
   (c) For each $j \in [\rho]$ and $i \in [n]$, it chooses a random $t_j^i \in \mathbb{F}^{u_j}$ and hands it to P_i.
   (d) computes Ω (after expanding all α_w from α), chooses a random $\mu \in \mathbb{F}$ and then hand $\hat{\Omega} = \Omega - \mu$ to the parties.
2. The parties set for each $w \in W$: $\alpha_w = \alpha^w$ (or use α as a seed to a PRG).
3. Each party P_i locally compute Λ and Γ_i. Then, each P_i broadcasts $\hat{\Gamma}_i = \Gamma_i - s_i$ to the other parties.
4. For each $i \in [n]$, party P_i proves that Γ_i was computed correctly:
   Let $\boldsymbol{y}_i$ be the vector of inputs for the proof of P_i (as defined in Eq. (2)). Let $\boldsymbol{y}_i^{\mathcal{P}}$ a vector of elements generated by replacing all elements in $\boldsymbol{y}_i$ which are *not* known to all parties by 0, and let $\boldsymbol{y}_i^{\mathcal{D}}$ be a vector of elements generated by replacing all elements in $\boldsymbol{y}_i$ *not* known to $\mathcal{D}$ by 0. Note that $\boldsymbol{y}_i = \boldsymbol{y}_i^{\mathcal{P}} + \boldsymbol{y}_i^{\mathcal{D}}$.
   (a) For each round j of the zk-FLIOP:
      i. If $j = 1$, party P_i lets $\pi_j^i = \mathsf{P}_{\mathsf{FLIOP}}(\boldsymbol{y}_i, \perp)$. Otherwise, it lets $\pi_j^i = \mathsf{P}_{\mathsf{FLIOP}}(\boldsymbol{y}_i, \pi_{j-1}^i, r_{j-1}^i)$.

ii. P_i broadcasts $\hat{\pi}^i_j = \pi^i_j - t^i_j$ to the other parties.
iii. The dealer $\mathcal{D}$ chooses a random challenge r^i_j and hands it to the parties.
iv. The parties and the dealer let $q^i_{j,1}, \ldots, q^i_{j,\ell}$ be the query vector determined by $\mathsf{V}_{\mathsf{FLIOP}}$ based on r^i_j. Then, the parties compute the answers

$$\hat{a}^i_{j,1}, \ldots, \hat{a}^i_{j,\ell} \leftarrow \langle q^i_{j,1}, \boldsymbol{y}^{\mathcal{P}}_i || \hat{\pi}^i_j \rangle, \ldots, \langle q^i_{j,\ell}, \boldsymbol{y}^{\mathcal{P}}_i || \hat{\pi}^i_j \rangle.$$

Similarly, $\mathcal{D}$ computes his answers

$$\widetilde{a}^i_{j,1}, \ldots, \widetilde{a}^i_{j,\ell} \leftarrow \langle q^i_{j,1}, \boldsymbol{y}^{\mathcal{P}}_i || t^i_j \rangle, \ldots, \langle q^i_{j,\ell}, \boldsymbol{y}^{\mathcal{P}}_i || t^i_j \rangle.$$

v. The Dealer $\mathcal{D}$ sends $\widetilde{a}^i_{j,1}, \ldots, \widetilde{a}^i_{j,\ell}$ to the parties, who then compute

$$a^i_{j,1}, \ldots, a^i_{j,\ell} \leftarrow \hat{a}^i_{j,1} + \widetilde{a}^i_{j,1}, \ldots, \hat{a}^i_{j,\ell} + \widetilde{a}^i_{j,\ell}.$$

(b) The parties run the decision predicate of $\mathsf{V}_{\mathsf{FLIOP}}$ on all the queries' answers they received. If any party received reject, then it outputs reject. Otherwise, the parties proceed to the next step.

5. The parties locally compute $\hat{p}(W) = \Lambda - \sum_{i=1}^{n} \hat{\Gamma}_i + \hat{\Omega}$. Then, the dealer $\mathcal{D}$ hands $s = -\sum_{i=1}^{n} s_i + \mu$ to the parties.
6. The parties locally compute $p(W) = \hat{p}(W) + s$. If $p(W) = 0$, then the parties output accept. Otherwise, they output reject.

Proposition 3.1. *Let ϵ_w be additive error on each wire $w \in W$ (where W is the set of all output wires and inputs to multiplication gates), and let $(\mathsf{P}_{\mathsf{FLIOP}}, \mathsf{V}_{\mathsf{FLIOP}})$ be a ρ-rounds, ℓ-queries and ε-soundness error zk-FLIOP protocol. Then, Π_{vrfy} satisfies the following properties:*

1. Correctness: *If $\forall w \in W : \epsilon_w = 0$ and all parties follow the protocol's instructions, then the honest parties always output* accept.
2. Soundness: *If $\exists w \in W : \epsilon_w \neq 0$, then the honest parties output* accept *with probability of at most $\frac{|W|}{|\mathbb{F}|} + \varepsilon$.*
3. Privacy: *For every adversary $\mathcal{A}$ controlling a subset T of size $\leq n-1$, there exists a simulator $\mathcal{S}$, who receives $\{\epsilon_w, \hat{x}_w, \{r_{w,i}\}_{i \in T}\}_{w \in W}$ as an input, and outputs a transcript $\mathsf{view}_{\mathcal{S}}$, such that $\mathsf{view}_{\mathcal{S}} \equiv \mathsf{view}^{\pi_{\mathsf{vrfy}}}_{\mathcal{A}}$.*

Proof: Correctness. It is easy to see from the description of the protocol, that if no additive errors were introduced and all parties acted honestly in the protocol, then $p(W) = 0$. It remains to show that the parties will output accept in the zk-FLIOP protocol. Given a proof π^i_j, it holds that $\pi^i_j = \hat{\pi}^i_j + t^i_j$. Then, when the parties compute the answers to the linear queries, we have $\forall l \in [\ell]$:

$$a^i_{j,l} = \hat{a}^i_{j,l} + \widetilde{a}^i_{j,l} = \langle q^i_{j,l}, \boldsymbol{y}^{\mathcal{P}}_i || (\pi^i_j - t^i_j) \rangle + \langle q^i_{j,l}, \boldsymbol{y}^{\mathcal{D}}_i || t^i_j \rangle = \langle q^i_{j,l}, \boldsymbol{y}_i || \pi^i_j \rangle$$

and so by the completeness of the zk-FLIOP protocol, they will hold the correct answer and output accept.

SOUNDNESS. If $\exists w \in W : \epsilon_w \neq 0$, then the parties will output accept if $p(W) = 0$. This can happen if one of two events occur: (i) the random linear combination yield 0. since $\alpha_w = \alpha^w$ for a random α, we have that $p(W) = \sum_{w \in W} \alpha_w \cdot \epsilon_w = \sum_{w \in W} \alpha^w \cdot \epsilon_w$ and so, fixing all ϵ_w, this is a polynomial of degree $|W|$ evaluated on a random point α. Thus, by the Schwartz-Zippel lemma, $p(W) = 0$ with probability $\frac{|W|}{|\mathbb{F}|}$. (ii) the parties output accept in the zk-FLIOP, even though a corrupted party P_i shared an incorrect Γ_i. By the soundness property of the zk-FLIOP protocol, this can happen with probability of at most ε. Hence, by the union bound, the overall cheating probability is $\frac{|W|}{|\mathbb{F}|} + \varepsilon$.

PRIVACY. We construct a simulator $\mathcal{S}$ for our protocol and show that the view it generates is distributed identically to the adversary $\mathcal{A}$'s view in a real execution. The simulator $\mathcal{S}$ receives $\{\epsilon_w, \hat{x}_w, \{r_{w,i}\}_{i \in T}\}_{w \in W}$ as an input, and then interacts with $\mathcal{A}$ playing the role of the honest parties and the trusted dealer $\mathcal{D}$. In particular, $\mathcal{S}$ works as follows:

1. Playing the role of $\mathcal{D}$, it hands $\mathcal{A}$ a random s_i for each $i \in T$, a random seed α and a random t^i_j for each $i \in T$ and $j \in [\rho]$. In addition, $\mathcal{S}$ chooses a random $\hat{\Omega}$ and hands it to $\mathcal{A}$.
2. For each honest party P_i, it chooses a random $\hat{\Gamma}_i$ and hands it to $\mathcal{A}$.
3. $\mathcal{S}$ computes all α_w and then, knowing all the corrupted parties' inputs, it computes Γ_i for each corrupted party P_i. In addition, knowing all $\hat{x}_w$, it computes Λ.
4. Upon receiving from $\mathcal{A}$ all $\{\hat{\Gamma}_i\}_{i \in T}$, the simulator $\mathcal{S}$ computes for each $i \in T$, $\Gamma'_i = \hat{\Gamma}_i + s_i$.
5. Simulating the zk-FLIOP execution:
   - The prover P_i is honest: In each round $j \in [\rho]$, $\mathcal{S}$ chooses a random $\hat{\pi}^i_j$ and sends it to $\mathcal{A}$. Then, playing the role of $\mathcal{D}$, it hands a random challenge r^i_j to $\mathcal{A}$. To simulate the opening of the query answers, $\mathcal{S}$ run $\mathcal{S}_{FLIOP}$ to receive $a^i_{j,1}, \ldots, a^i_{j,\ell}$. Then, for each $l \in [\ell]$, it computes $\hat{a}^i_{j,l}$ (since it knows all the corrupted parties' inputs and so all the values in $\boldsymbol{y}^{\mathcal{P}}_i$) and then sets $\widetilde{a}^i_{j,l} = a^i_{j,l} - \hat{a}^i_{j,l}$ and hands the answers to $\mathcal{A}$.
   - The prover P_i is corrupted: In this case, $\mathcal{S}$ simply plays the role of the honest parties acting as verifiers in this proof, and the role of $\mathcal{D}$. Since it knows the corrupted parties' inputs, it knows the verifiers' inputs to this proof, and so it can perfectly simulate this execution.
6. $\mathcal{S}$ computes $p(W) = \sum_{w \in W} \alpha_w \cdot \epsilon_w + \sum_{i \in T} (\Gamma'_i - \Gamma_i)$ and $\hat{p}(W) = \Lambda - \sum_{i=1}^{n} \hat{\Gamma}_i + \hat{\Omega}$. Then it sets $s = p(W) - \hat{p}(W)$ and hands it to $\mathcal{A}$.

Observe that the view of $\mathcal{A}$ in a real execution consists of three types of values:(i) masked data which is distributed uniformly over $\mathbb{F}$; (ii) the answers to the zk-FLIOP linear queries; (iii) and the value of $p(W)$ which is determined by $\mathcal{A}$ (since it chooses the additive errors). In the simulation, values of type (i) are chosen uniformly from $\mathbb{F}$ and so are distributed the same as in the real execution.

Type (ii) of data is distributed the same by the ZK property of the zk-FLIOP. Finally, since $\mathcal{S}$ knows all the inputs held by $\mathcal{A}$ and the additive errors, it can compute the actual value of $p(W)$ and so perfectly simulate the opening of this value. We conclude that the view generated by the simulation is identically distributed to the view in the real execution. This concludes the proof. ■

Working over small fields. The soundness error of our protocol depends on the size of the field $\mathbb{F}$. When we compute the circuit over small fields, it is possible to run Π_{vrfy} over an extension field to reduce the error. This is carried-out by lifting each input to the verification protocol into the extension field. Suppose that we want the error to be $2^{-\varepsilon}$. Then, one can choose an extension field $\tilde{\mathbb{F}}$ such that $\frac{|W|}{|\tilde{\mathbb{F}}|} + \varepsilon_1 \leq 2^{-\varepsilon}$, where ε_1 is the soundness error of the zk-FLOIP protocol over $\tilde{\mathbb{F}}$.

Working over the ring $\mathbb{Z}_{2^k}$. When the circuit is computed over the ring $\mathbb{Z}_{2^k}$, then by Theorem 2.1, we still have a zk-FLIOP with sublinear cost. However, the probability that $p(W) = 0$ when the random coefficients taken as $r, r^2, \ldots, r^{|W|}$ and so p is a polynomial of degree $|W|$ evaluated on a random point r, is constant regardless of the size of the ring. Nevertheless, since the cost of our verification protocol is small, we can afford an "expensive" solution here, and run Π_{vrfy} over the extension ring $\mathbb{Z}_{2^k}[x]/f(x)$, i.e., the ring of polynomials with coefficients from $\mathbb{Z}_{2^k}$ modulo a polynomial $f(x)$ which is of the right degree and is irreducible over $\mathbb{Z}_2$. As shown in [5, 10], taking f of degree d, the number of roots of a polynomial of degree δ over $\mathbb{Z}_{2^k}[x]/f(x)$ is at most $2^{(k-1)d}\delta + 1$. Thus, the probability that $p(W) = 0$ when r is chosen at random, is at most $\frac{2^{(k-1)d}|W|+1}{2^{kd}} \approx \frac{|W|}{2^d}$. Hence, by choosing d appropriately, we can achieve a desired soundness error.

From an active dealer to an offline dealer. In the above description we treated the dealer as an active participant in the computation. Note however, that all the operations carried-out by the dealer in our protocol, can be done offline before the start of the computation, because they depend only on random data. These include operations over randomness it chooses for the execution of Π_{vrfy}, and operations over the prover's random shares of the masks, which were chosen by the dealer.

Now, there are two types of randomness that the dealer provides in the execution:

Type I: *randomness given to a single party.* This type of randomness can be handed to the intended party before the beginning of the execution. This includes: (i) random masks $s_i \in R$ and $\{t_j^i\}_{j\in[\rho]}$ where $t_j^i \in R^{u_j}$, given to each party P_i.

Type II: *randomness given to all parties during the protocol.* For each randomness of this type, the dealer can precompute it and send it to $F_{\mathsf{com}}^{\mathsf{dealer}}$ before the beginning of the computation. Then, whenever the parties reach the point where the randomness needs to be revealed, they can send a reveal command to $F_{\mathsf{com}}^{\mathsf{dealer}}$. This includes: (ii) a random seed $\alpha \in R$ given to all parties; (iii)

$\hat{\Omega} = \sum_{w \in W} \alpha_w \cdot r_w + \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (r_1^{g_\ell} \cdot r_2^{g_\ell}) - \nu$ given to all parties, where each α_w and γ_ℓ is expanded from α and $\nu \in R$ is random; (iv) a challenge $r_j^i \in R$ for each $i \in [n]$ and $j \in [\rho]$; (v) the queries' answers $\widetilde{a}_{j,1}^i, \ldots, \widetilde{a}_{j,\ell}^i$, for each $j \in [\rho]$ and $i \in [n]$ (which are computed over the random challenges and prover's inputs which are known to the dealer); and (vi) the random mask s.

Summing the above and given that the extension degree used in the verification protocol is d, then the amount of correlated randomness is

$$\left(3 + n \cdot \left(\sum_{j=1}^{\rho} u_j + \rho(1+\ell)\right)\right) \cdot d \quad \mathsf{ring\ elements}.$$

The main observation is that the amount of correlated randomness is *logarithmic* in the size of the input to the verification subprotocol, i.e., logarithmic in $|W|$. This holds since by Theorem 2.1, there exists a zk-FLIOP protocol, where the proof, $\sum_{j=1}^{\rho} u_j$, the number of rounds ρ and the number of queries $\ell \cdot \rho$ are all of size $\log(M)$, with M being the number of distinct monomials in the polynomial for which the proof takes place. As can be seen from Eq. (1), in our case, M equals to $2|\mathsf{mult}|$. It follows that the amount of required correlated randomness is $O(n \cdot \log|\mathsf{mult}| \cdot \mathsf{d})$.

Communication cost. The interaction in Π_{vrfy} consists of having each party sending the proof to the other parties in each round, and interaction with $F_{\mathsf{com}}^{\mathsf{dealer}}$ to reveal the public randomness. Thus, the overall cost is $(n \cdot \sum_{j=1}^{\rho} u_j) \cdot d + (3 + n \cdot (\rho + \rho \cdot \ell)) \cdot d \cdot F_{\mathsf{com}}^{\mathsf{dealer}}$ ring elements, which by Theorem 2.1, for the same reasoning explained above for the correlated randomness, is of size $O(n \cdot \log|\mathsf{mult}| \cdot \mathsf{d})$

Computation cost. In Π_{vrfy}, each party P_i first computes $\alpha_w = \alpha^w$ for each $w \in W$ and Λ and Γ_i. Each of these computations consists of $O(|W|)$ local multiplication operation. Then, the parties run the zk-FLIOP protocol to prove the correctness of Γ_i for each $i \in [n]$, where by Theorem 2.1, the computational complexity is $O(M)$, which means, as explained above, that the computation complexity is $O(n \cdot |W|)$.

Summing the above, we obtain:

Proposition 3.2. *Let ε be a statistical error bound. Then, Protocol Π_{vrfy} has communication cost $O(\log|\mathsf{mult}| \cdot \kappa)$ per party, computational cost $O(n \cdot |W|)$ per party and the amount of correlated randomness required from the dealer is $O(n \cdot \log|\mathsf{mult}| \cdot \kappa)$ per party, where $\kappa = \log_{|\mathbb{F}|}\left(\frac{|W|}{\varepsilon}\right)$ when R is finite field, and $\kappa = \log_2\left(\frac{|W|}{\varepsilon}\right)$ when $R = \mathbb{Z}_{2^k}$ (where W is the set of output wires and input wires to multiplication gate in the verified circuit).*

Concrete instantiation for the zk-FLIOP. Based on the general constructions from [5], we describe a concrete protocol in the full version for implementing the zk-FLIOP protocol in our setting with the following parameters:

- $\rho = \log(2|\mathsf{mult}|) - 1$
- $u_j = 3$ for $j \in [\rho - 1]$ and $u_\rho = 8$
- $\ell = 1$

Furthermore, we show how to optimize the protocol such that the number of queries becomes constant instead of logarithmic. The concrete costs of the realization we obtain are:

- *communication cost:* $3(\log(2|\mathsf{mult}|)-1)+8$ elements broadcasted by the prover.
- *Correlated randomness:* the dealer needs to provide $5(\log(2|\mathsf{mult}|) - 1) + 9$ elements.
- *Computation:* each party performs approximately $2|\mathsf{mult}|$ local operations.

3.2 The Main Protocol

We are now ready to present the main protocol to compute any arithmetic circuits with malicious security. Informally, Our protocol takes any secure-up-to-additive attack and star-sharing compliant protocol, and compile it into malicious security, by adding a verification step, where the parties run the protocol Π_{vrfy} from Section 3.1. Formally:

$\underline{\Pi_{\mathrm{MPC}}}$*:* Let C be the circuit to compute, defined over a ring R, let W be the set of C's output wires and input to multiplication gates and let ε be a desired statistical security bound. Let $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$ be a protocol to compute C which is secure-up-to-additive-attack with star-sharing compliance. Let $\tilde{R}$ be an extension ring of R defined as:

- If R is a finite field $\mathbb{F}$, then set $\tilde{R} = \mathbb{F}^\kappa$, such that κ is the smallest number for which $\frac{|W|}{|\mathbb{F}^\kappa|} \leq \varepsilon/2$.
- If $R = \mathbb{Z}_{2^k}$, then set $\tilde{R} = \mathbb{Z}_{2^k}[x]/f(x)$ where f is a polynomial of degree κ which is irreducible over $\mathbb{Z}_2$, such that κ is the smallest number for which $\frac{|W|}{|2^\kappa|} \leq \varepsilon/2$.

- **Preprocessing:** The dealer $\mathcal{D}$ hands the parties the following correlated randomness:
  - For input wire k held by party P_i, it hands a random mask $s^k_i \in R$ to P_i and a random $s^k_{i,j}$ to P_j such that $s^k_i = \sum_{j=1}^n s^k_{i,j}$.
  - It hands the parties the correlated randomness required by $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$. This includes a random $r_{w,i}$ for each party P_i and wire w.
  - It hands the parties the correlated randomness required by Π_{vrfy} as defined is Section 3.1 over $\tilde{R}$.
  - For each output wire w, it sends the random mask r_w of this wire to $F^{\mathsf{dealer}}_{\mathsf{com}}$.
- **The online protocol:**
  - SHARING THE INPUTS: For each wire k, with input v^k_i held by P_i, it broadcasts $\hat{v}^k_i = v^k_i - s^k_i$ to the other parties.

- Circuit emulation: The parties compute the circuit C gate-by-gate in some predetermined topological order, by running $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$, using the correlated randomness received from the dealer, up to and not including the output reconstruction step.
- Verification step: Let $(\hat{x}_w, r_{w,i})$ be the pair held by each party P_i on each wire $w \in W$. The parties lift $\left(\hat{x}_w, \{r_{w,i}\}_{i\in[n]}\right)_{w\in W}$ into $\tilde{R}$. Then, they run Π_{vrfy} with a zk-FLIOP protocol with soundness error $\varepsilon/2$ on the lifted values and on the correlated randomness received from the dealer.
  If any party outputs reject, then it sends abort to the other parties and aborts the protocol. Otherwise, the parties proceed to the next step.
- Output reconstruction: For each output wire w, with output intended to party P_i, let $\hat{x}_w$ be the value held by the parties on this wire. Then, the parties send (w, i) to $F^{\mathsf{dealer}}_{\mathsf{com}}$, who sends r_w to P_i. Finally, party P_i sets $x_w = \hat{x}_w + r_w$ as its output.

We thus obtain the following proposition:

Proposition 3.3. *Let f be a n-party functionality represented by an arithmetic circuit C over a ring R and let ε be a statistical security bound. Then, if $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$ is star-sharing compliant and securely computes f with additive security as defined in Definition 2.3, and $(\mathsf{P}_{\mathsf{FLIOP}}, \mathsf{V}_{\mathsf{FLIOP}})$ is public-coin zk-FLIOP as defined in Definition 2.4, then Π_{MPC} (ε)-securely computes f in the $F^{\mathsf{dealer}}_{\mathsf{com}}$-hybrid model with abort in the preprocessing model.*

Proof: We describe a simulator $\mathcal{S}$ for our protocol. In the simulation, $\mathcal{S}$ plays the role of the honest parties and the dealer $\mathcal{D}$ when interacting with the real-world adversary $\mathcal{A}$, who controls a set of parties T with $|T| \leq n-1$. The simulator $\mathcal{S}$ invokes $\mathcal{A}$ by handing it the correlated randomness for the honest parties as would $\mathcal{D}$ do. Then, in the online protocol it works as follows:

- Input sharing step: The simulator $\mathcal{S}$ sends random elements to $\mathcal{A}$ as the masked inputs of the honest parties. Upon receiving the masked inputs $\hat{x}_k$ of the corrupted parties for each input wire k from $\mathcal{A}$, it extracts the corrupted parties' inputs by computing $x_k = \hat{x}_k + r_k$.
- Circuit emulation: Let $\mathcal{S}_{add}$ be the simulator for $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$. The simulator $\mathcal{S}$ follows the instructions of $\mathcal{S}_{add}$ while interacting with $\mathcal{A}$. Playing the role of $\mathcal{S}_{add}$, it extracts the additive attack ϵ_w for each wire $w \in W$.
- Verification: Let $\mathcal{S}_{vrfy}$ be the simulator for Π_{vrfy} from Theorem 3.1. The simulator $\mathcal{S}$ invokes $\mathcal{S}_{vrfy}$ on $\{\epsilon_w, \hat{x}_w, \{r_{w,i}\}_{i\in T}\}_{w\in W}$, and follows its instructions. Let out be the output held by the honest parties, played by $\mathcal{S}$, at the end of the execution. If out = reject, then $\mathcal{S}$ sends abort to the trusted party computing f and outputs whatever $\mathcal{A}$ outputs. Else, out = accept. If $\forall w \in W : \epsilon_w = 0$, then $\mathcal{S}$ proceeds to the next step. Otherwise, $\exists w \in W : \epsilon_w \neq 0$ and the output is accept. In this case, $\mathcal{S}$ outputs fail and halts.
- Output reconstruction: The simulator $\mathcal{S}$ sends the corrupted parties' inputs to the trusted party computing f, to receive back their outputs. For each output wire w with output x_w on it, $\mathcal{S}$ sends to $\mathcal{A}$ the random mask $r_w = x_w - \hat{x}_w$.

For each output intended to an honest party P_j, it waits for $\mathcal{A}$'s command to $F_{\mathsf{com}}^{\mathsf{dealer}}$. If $\mathcal{A}$ sends abort to $F_{\mathsf{com}}^{\mathsf{dealer}}$, then $\mathcal{S}$ sends abort_j to the trusted party. Otherwise, it sends $\mathsf{continue}_j$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We show that $\mathcal{A}$'s view in the simulation is statistically close to its view in the real execution. First, observe that in the input sharing step, $\mathcal{A}$ sees random masked values in both executions. In the circuit emulation step, by the definition of $\Pi_{\mathsf{mpc}}^{\mathsf{add}}$, the simulation has has at most statistical distance from the real execution. In the verification step, by the privacy property of Π_{vrfy}, the views are distributed identically, except for the case $\mathcal{S}$ outputs fail. Note however that this event occurs when the honest parties output accept even though $\exists \epsilon_w \neq 0$. From the soundness property of Π_{vrfy}, it thus follows that $\Pr[\mathsf{fail}] = \varepsilon/2 + \varepsilon/2 = \varepsilon$. To see why this holds, recall that $\tilde{R}$ was chosen such that $\frac{|W|}{|\mathbb{F}^\kappa|} \leq \varepsilon/2$ when $R = \mathbb{F}$ and $\frac{|W|}{|2^\kappa|} \leq \varepsilon/2$ when $R = \mathbb{Z}_{2^k}$, and that the parties called the zk-FLIOP protocol with parameter $\varepsilon/2$. By the soundness property of Π_{vrfy} (Proposition 3.1), the cheating probability is $\frac{|W|}{|\mathbb{F}^\kappa|} + \frac{\varepsilon}{2}$ when $R = \mathbb{F}$, and $\frac{|W|}{2^\kappa} + \frac{\varepsilon}{2}$ when $R = \mathbb{Z}_{2^k}$, implying that it is bounded by ε. Finally, given that the view until the reconstruction step are distributed similarly in both executions, then the same applies for this step as well, since $\mathcal{A}$ sees only random values. Overall, by a standard hybrid argument, we have that $\mathcal{A}$'s view is distributed the same with statistical error ε as allowed by the theorem. This concludes the proof. ■

Combining Proposition 3.2 and Proposition 3.3, we obtain the following theorem, which summarize our main result in this work:

Theorem 3.1. *Let f be a n-party functionality represented by an arithmetic circuit C of size $|C|$ (number of multiplication gates and output wires) over a ring R which is either a finite field or the ring $\mathbb{Z}_{2^k}$ and let ε be a statistical security bound. Then, every protocol in the preprocessing model which securely computes f with additive security and is star-compliant, can be compiled into a ε-secure protocol, with additional $O(n \cdot \log|C| \cdot \kappa)$ correlated randomness and $O(\log|C| \cdot \kappa)$ communication per party, where $\kappa = \log_{|\mathbb{F}|}\left(\frac{|C|}{\varepsilon}\right)$ when R is finite field, and $\kappa = \log_2\left(\frac{|C|}{\varepsilon}\right)$ when $R = \mathbb{Z}_{2^k}$.*

From our main theorem we derive the following corollaries. We apply our construction on the well-known semi-honest protocol based on Beaver triples [1]. First, we obtain a protocol in the circuit-dependent preporocessing, where both the amortized communication cost and the amount of correlated randomness match the cost of the underlying semi-honest protocol, for rings of *any* size:

Corollary 3.1 (Circuit-dependent preprocessing). *Let C be a circuit with size $|C|$ (which is the number of multiplication gates, input and output wires in C) defined over a ring R which is either a finite field $\mathbb{F}$ or the ring $\mathbb{Z}_{2^k}$ and let ε be a statistical error bound. Then, there exists a protocol to ε-securely compute C with abort, with the following properties:*

– Communication: *each party sends* $(2-\frac{2}{n})\cdot|C|+O(\log|C|\cdot\kappa)$ *ring elements.*
– Correlated randomness: *the circuit-dependent preprocessing outputs* $4\cdot|C|+O(n\cdot\log|C|\cdot\kappa)$ *ring elements to each party.*
With PRG-based compression, this can be reduced to $|C|+O(n\cdot\log|C|\cdot\kappa)$ *elements to one party, and* $O(n\cdot\log|C|\cdot\kappa)$ *elements to the other parties.*

where κ *is defined as in Theorem 3.1.*

Proof: Consider the semi-honest protocol described in Appendix A.1, which is the circuit-dependent version of the well-known Beaver's [1] protocol, as described in [9]. In this protocol, the parties hold $\hat{x}_w = x_w - r_w$ for each wire w, which is a circuit's output wire or input wire to a multiplication gate. In addition, they hold for each multiplication gate g with input wires $w^g_{i_1}$ and $w^g_{i_2}$ and output wire w^g_o, an additive sharings of $r^g_{i_1}$, $r^g_{i_2}$, $r^g_{i_1}\cdot r^g_{i_2}$ and r^g_o. Then, they use these to locally compute an additive sharing of masked output (masked with r^g_o) and interact to reveal the masked output, by having each party sending $2-\frac{2}{n}$ ring elements. The amount of correlated randomness in this protocol is 4 ring elements per multiplication gate without compression. Alternatively, the dealer can hand each party a PRG seed from which its shares of $r^g_{i_1}$, $r^g_{i_2}$ and r^g_o are derived, thereby removing completely $3\cdot|C|$ elements of correlated randomness. For $r^g_{i_1}\cdot r^g_{i_2}$, the dealer can hand $n-1$ parties a PRG seed from which their shares are expanded, and give the remaining party one share for each gate. We remark that for each input, each party needs to send one element (masked input) to all parties, while for each output wire, the dealer sends the mask to one party. Thus, *per party*, the communication cost for an input/output wire is bounded by the cost per multiplication.

The protocol is thus star-sharing compliant. In addition, as shown in Appendix A.1, the protocol satisfies the property of additive security. Hence, by applying Theorem 3.1 on this protocol the corollary follows. ■

In the circuit-independent model, we have a similar result. Here the communication is slightly higher because the cost of the underlying semi-honest protocol is higher.

Corollary 3.2 (Circuit-independent preprocessing). *Let* C *be a circuit with size* $|C|$ *(number of multiplication gates, input and output wires in* C*) defined over a ring* R *which is either a finite field* $\mathbb{F}$ *or the ring* $\mathbb{Z}_{2^k}$ *and let* ε *be a statistical error bound. Then, there exists a protocol to* ε*-securely compute* C *with abort, with the following properties:*

– Communication: *each party sends* $(4-\frac{4}{n})\cdot|C|+O(\log|C|\cdot\kappa)$ *ring elements.*
– Correlated randomness: *the circuit-independent preprocessing outputs* $3\cdot|C|$ *ring elements to each party, and there is an additional circuit-dependent preprocessing which outputs* $O(n\cdot\log|C|\cdot\kappa)$ *elements to each party.*
With PRG-based compression, this can be reduced to $|C|+O(n\cdot\log|C|\cdot\kappa)$ *elements to one party, and* $O(n\cdot\log|C|\cdot\kappa)$ *elements to the other parties.*

where κ *is defined as in Theorem 3.1.*

Proof: The proof is identical to the proof of Corollary 3.1, with the only difference being the underlying protocol with additive security. Here we use the standard multplication with Beaver triples shown in Appendix A.2. The parties interact for each multiplication's input wire and thus communication is doubled. The correlated randomness consists of additive sharings of the input masks and their multiplication, and so the per gate each party stores 3 random ring elements. ■

Remark 3.1 (multicast Vs. private channels). The communication cost presented in Corollaries 3.1 and 3.2 is achieved when only private channels between the parties exist. In case the parties have access to a multicast channel, where sending one message to n parties has the same cost as sending n private messages, then the communication cost is 1 ring element per multiplication gate per party in the circuit-dependent preprocesssing model, and 2 ring elements with circuit-independent preprocessing.

4 Distributing the Dealer

In this section, we show how the role of the trusted dealer can be emulated by the parties in a secure way. Our focus here is only on the correlated randomness required by our compiler, ignoring the correlated randomness for the underlying additively-secure protocol, which is usually easier to generate. To this end, we need to present a MPC protocol which outputs to each party the correlated randomness required by our verification protocol. Our approach to this task is to view the dealer's work as computing an arithmetic circuit, and then one can use any general MPC protocol to compute this circuit by the parties. This is motivated by the fact that, as shown in Section 3.1, the computational work of the dealer in the verification protocol, is $O(n \cdot |C|)$. This implies that the computational work is asymptotically proportional to the size of the circuit (times the number of parties). We now show that the hidden constants are actually very small, which means that the circuit computed by the dealer has almost the same size as the original circuit. We remind the reader that general MPC protocols require interaction only for multiplication operations and not for linear operations. Thus, we are only interested here in counting the number of multiplication operations carried-out by the dealer.

When looking into our verification protocol Π_{vrfy}, we identify three computations which require multiplications:

- Computing the random coefficients α_w for each output wire or multiplication gate's input wire w. This computation is done by taking $\alpha_w = \alpha^w$ for a random $\alpha \in R$. Thus, for $|W|$ wires, this requires $|W|$ multiplications. Assuming that the number of outputs is considerably smaller compared to the number of multiplication gates, this amount to $2|C|$ multiplications.
- Computing $\Omega = \sum_{w \in W} \alpha_w \cdot r_w + \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (r_1^{g_\ell} \cdot r_2^{g_\ell})$. Recall that the random coefficients γ_ℓ are computed as a summation of several α_w coefficients, and so

are computed without interaction. Thus, the cost here is 2 multiplications for each multiplication gate g_ℓ, and so $2 \cdot |C|$.
– Computing the queries answers $\widetilde{a}^i_{j,1}, \ldots, \widetilde{a}^i_{j,\ell} \leftarrow \langle q^i_{j,1}, \boldsymbol{y}^{\mathcal{P}}_i || t^i_j \rangle, \ldots, \langle q^i_{j,\ell}, \boldsymbol{y}^{\mathcal{P}}_i || t^i_j \rangle$ in each round of the zk-FLIOP. The cost here depends of course on the way the zk-FLIOP is realized. When using the logarithmic construction described in the full version, the parties need to compute approximately $2|\mathsf{mult}|$ multiplications overall, and so $2|C|$ multiplications for each of the n calls to the zk-FLIOP.

Summing the above, we conclude that the size of the dealer's circuit, measured by the number of multiplications, is $4|C| + n \cdot 2|C|$. For the popular setting of 2-party secure computation, for instance, this amount so $8 \cdot |C|$.

Thus, to securely compute this circuit, the parties can use any state-of-art general MPC protocols for computing arithmetic circuits, such as the recent results of [15, 27, 24, 8], depending on the type of underlying ring/field. Together with our light online protocol, this yields a protocol for computing arithmetic circuits with practical potential.

Remark 4.1 (Distributing the dealer for PRG-based protocols.). The approach above works also when the semi-honest correlated randomness is compressed using a PRG. In particular, distributing the dealer does not require securely evaluating the PRG. To illustrate this, consider PRG compression in protocols based on multiplication triples (as in Corollary 3.1 and 3.2). When the n parties emulate the dealer, each party chooses a PRG seed from which it derives its shares of vectors a and b. In addition, all but one party derive their share of $c = a \cdot b$ from their seed. Then, the parties run an MPC protocol to compute the share of c of the remaining party from the $3n - 1$ vectors, and finally the correlated randomness for sublinear ZK verification. The crucial point is that feeding the MPC with an incorrect PRG output does not hurt the security of the online protocol since the latter is secure even with the "corruptible" version of the multiplication triples correlation (this was also observed in the context of SPDZ-style protocols and pseudorandom correlation generators, see [7]).

Acknowledgements

E. Boyle supported by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069 FA9550-21-1-0046, and ERC Project HSS (852952).
N. Gilboa supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center.
Y. Ishai supported by ERC Project NTSC (742754), NSF-BSF grant 2015782, BSF grant 2018393, and ISF grant 2774/20.
A. Nof supported by ERC Project NTSC (742754).

References

1. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO (1991)

2. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: Double your online spdz! improving SPDZ using function dependent preprocessing. In: ACNS (2019)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC (1988)
4. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT (2011)
5. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: CRYPTO (2019)
6. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCD (2019)
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO 2019, Part III. pp. 489–518 (2019)
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: CRYPTO (2020)
9. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: TCC (2019)
10. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: ACM CCS (2019)
11. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs. In: ASIACRYPT (2020)
12. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology 13(1), 143–202 (2000)
13. Cascudo, I., Gundersen, J.S.: A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In: TCC (2020)
14. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC (1988)
15. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: Spdz2k: Efficient MPC mod 2k for dishonest majority. In: CRYPTO (2018)
16. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: ESORICS (2013)
17. Damgård, I., Lauritsen, R., Toft, T.: An empirical study and some improvements of the minimac protocol for secure computation. In: SCN (2014)
18. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In: CRYPTO (2017)
19. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO (2012)
20. Damgård, I., Zakarias, S.: Constant-overhead secure computation of boolean circuits using preprocessing. In: TCC (2013)
21. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: STOC (2014)
22. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
23. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC (1987)
24. Hazay, C., Ishai, Y., Marcedone, A., Venkitasubramaniam, M.: Leviosa: Lightweight secure arithmetic computation. In: ACM CCS (2019)

25. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC (2013)
26. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: ACM CCS (2016)
27. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: EUROCRYPT (2018)
28. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: CRYPTO (2012)
29. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: CCS '20 (2020)
30. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS (1986)

A Protocols which are Secure-up-to-Additive-Attack

In this section, we present two instatiations for a protocol to compute an arithmetic circuit, which is secure up to additive attack, as defined in Definition 2.3 and star-sharing compliant as defined in Defintion 3.2. Recall that the requirement is that for each multiplication gate or output wire of the circuit, the parties will hold a masked value on this wire, plus an error that the adversary added, which can be extracted by a simulator.

A.1 Multiplication in the Circuit-Dependent Preprocessing Model [9]

In this model, the structure of the circuit is known in advance. At the beginning of the protocol, the parties hold two masked inputs $\hat{x} = x - r_1$ and $\hat{y} = y - r_2$. The parties wish to obtain $\hat{z} = x \cdot y - r_3$. Observe that

$$\begin{aligned} \hat{z} = x \cdot y - r_3 &= (\hat{x} + r_1)(\hat{y} + r_2) - r_3 \\ &= \hat{x} \cdot \hat{y} + r_1 \cdot \hat{y} + r_2 \cdot \hat{x} + r_1 \cdot r_2 - r_3 \end{aligned} \tag{3}$$

and so if the parties are given an additive sharing of $r_1, r_2, r_1 \cdot r_2$ and r_3, they can locally compute an additive sharing of $\hat{z}$. Note that in this approach, if a multiplication's output wire is entering multiple gates in the next layer, then we need to make sure that the same mask is used for the input wires of the following gates. This is why the correlated randomness for this protocol is circuit-dependent, i.e., depends on the structure of the circuit. The multiplication protocol thus works as follows:

- **Inputs**: Each party P_i holds: $\hat{x}$, $\hat{y}$, r_1^i, r_2^i, $(r_1 \cdot r_2)^i$ and r_3^i.
- **The protocol**:
  1. Each party P_i locally computes $z^i = r_1^i \cdot \hat{y} + r_2^i \cdot \hat{x} + (r_1 \cdot r_2)^i - r_3^i$ and sends z^i to P_1.
  2. Party P_1 computes $z' = \sum_{i=1}^{n} z^i$ and broadcasts z' to all the other parties.
  3. The parties compute $\hat{z} = \hat{x} \cdot \hat{y} + z'$ and store the result as the output.

Recall that when P_1 broadcasting z', this amounts to sending z' to all parties and then at the end run a batch check with constant cost for the entire circuit, to assert that the same z' was sent to all parties in each gate (see Section 2.3). Thus, the overall communication cost in this protocol is $2(n-1)$ elements, and so each party sends $2-\frac{2}{n}$ elements per multiplication gate. Note that for 2-party computation, this comes down to sending just a *single element* per party per multiplication.

Security up to an additive error. The above protocol does not guarantee correctness; a corrupted party can send incorrect values and cause the output to be incorrect. However, the only attack that corrupted parties can carry-out is to add an error to the output. To see this, consider a simulator that holds $\hat{x}, \hat{y}$ and the randomness of the corrupted parties. Such a simulator can predict the messages sent by the corrupted parties. Thus, it can interact with the adversary, by sending him random values as the messages from the honest parties. Once it receives the messages from the corrutped parties, it can compute the error by comparing the received messages and the messages that should have been sent.

A.2 Multiplication in the Circuit-Independent Preprocessing Model [1]

When the structure of the circuit to be computed is yet to be known, we view the preprocessing as a service which produces random multiplication triples (i.e., Beaver triples). These triples are later consumed by the online computation. In this model, the parties interact to compute the masked input for each multiplication gate or a circuit's output wire. Then, they locally compute an additive sharing of the multiplication's output value. Addition gates which are between two multiplication gates are locally computed over the additive sharing of wire values. The protocol works as follows:

- **Inputs**: Each party P_i holds: x^i, y^i, r_1^i, r_2^i and $(r_1 \cdot r_2)^i$.
- **The protocol**:
  1. Each party computes $x^i - r_1^i$ and $y^i - r_2^i$ and sends it to P_1.
  2. Party P_1 computes $\hat{x} = x - r_1 = \sum_{i=1}^{n}(x^i - r_1^i)$ and $\hat{y} = y - r_2 = \sum_{i=1}^{n}(y^i - r_2^i)$. Then, it broadcasts $\hat{x}$ and $\hat{y}$ to all the other parties.
  3. Each party P_i computes $z^i = r_1^i \cdot \hat{y} + r_2^i \cdot \hat{x} + (r_1 \cdot r_2)^i$. Then, party P_1 defines $\hat{x} \cdot \hat{t} + z^1$ as its output share, where each P_i, with $i \neq 1$ defines z^i as its output share.

Observe that the communication cost here is doubled compared to the multiplication protocol in the circuit-dependent preprocessing model.

By the same reasoning which was used to compute the additive error for each multiplication gate *separately* in the circuit-dependent model presented above, we can compute the additive error on each *multiplication's input wire* or *circuit's output wire*, given the masked inputs to multiplication gates which feed these wires and the corrupted parties' randomness.