Sublinear GMW-Style Compiler for MPC with Preprocessing[*]

Elette Boyle
IDC Herzliya, Israel
eboyle@alum.mit.edu

Niv Gilboa
Ben-Gurion University, Israel
gilboan@bgu.ac.il

Yuval Ishai
Technion, Israel
yuvali@cs.technion.ac.il

Ariel Nof
Technion, Israel
ariel.nof@cs.technion.ac.il

February 3, 2022

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 multiparty computation (MPC) [Yao86, GMW87, BGW88, CCD88] enable a set of parties with private inputs to compute a joint function of their inputs while revealing nothing but the output. MPC 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 large body of research.

[*]A preliminary version of this work appeared in [BGIN21].

A popular approach for the design of efficient MPC 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 [Bea91], who showed how to use "multiplication triples" for secure arithmetic computation with no honest majority, many MPC protocols make extensive use of correlated randomness [BDOZ11, DPSZ12, IKM$^+$13, NNOB12, DKL$^+$13, DZ13, DNNR17, CDE$^+$18, BGI19, BLN$^+$21]. In particular, a powerful technique for securing such protocols against malicious parties uses *homomorphic MACs* to authenticate the values produced by the online protocol [BDOZ11, DPSZ12].

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.

Current MPC with preprocessing protocols exhibit a trade-off between these two efficiency goals that leaves much to be desired. For the case of evaluating an arithmetic circuit C with $|C|$ multiplication gates, some protocols [BDOZ11, NNOB12, DPSZ12, CDE$^+$18] succeed to achieve malicious security with minimal online communication overhead (sublinear in $|C|$), but with a large correlated randomness overhead of $O(|C|)$ field elements over large fields (roughly 2x compared to the semi-honest baseline), or $O(|C| \cdot \kappa)$ for Boolean circuits or circuits over the rings $\mathbb{Z}_{2^k}$, where κ is a statistical security parameter (roughly κ times the semi-honest baseline). Other protocols [DZ13, CG20] 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 compiler from any MPC protocol with preprocessing 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, the compiler introduces no additional assumptions. Our compiler can be viewed as an information-theoretic variant of the "GMW compiler" [GMW87] that applies to the setting of MPC with preprocessing, and only incurs a sublinear additive cost in online communication and correlated randomness. This can be contrasted with a similar compiler from [IOZ14], in which the extra costs are (at least) linear in the circuit size.

Our compiler requires two properties from the underlying semi-honest secure protocol. First, in the presence of malicious parties, the protocol must be secure *up to additive attacks.* This strengthens the usual notion of semi-honest security by further requiring 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 [GIP$^+$14], who showed that many semi-honest protocols that are based on linear 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 $(x_w - r_w)$.[1] 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 the public masked value $(x_w - r_w)$ their shares of the mask r_w, or by any individual party together with the dealer who knows r_w—thus forming a "star" structure. Other than its robustness feature, another useful feature of this form of star-sharing is being *multiplicative* [CDM00], in the sense that shares of two secrets $x_w, x_{w'}$ can be locally converted to additive sharing of the product $x_w \cdot x_{w'}$.

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 completed before the start of (and thus independently 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.

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 has the following efficiency features.*

[1]Note that x_w may not be the *correct* wire value following an additive attack by the adversary. This is not an issue in the context of our compiler.

- 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, the security of Π' *relies on the same computational assumptions (if any) as that of* Π*.*

When R is a big field (of size $> 2^{\kappa}$), the κ term in the asymptotic complexity bounds of the theorem can be eliminated.

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 [Bea91].

Using circuit-dependent preprocessing (where the correlated randomness in the underlying semi-honest protocol 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 Pseudo-Random Generator (PRG)).

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 twice the communication, namely, $(4 - \frac{4}{n})$ elements per multiplication gate per party.

All of the above upper bounds match the best previously known bounds for the semi-honest model. 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 dominant cost that can be generated more efficiently in the circuit-independent case. In particular, this applies to generating oblivious transfer and multiplication triple correlations via pseudorandom correlation generators (PCGs) [BCG+19b, BCG+20].[2]

Corollary 1.2 (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 computing* C *with security against malicious parties, with 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 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 (multiplication triples preprocessing).*

[2]PCG constructions also exist for more complex correlations, including circuit-dependent multiplication triples, as well as authenticated multiplication triples [BCG+19b, BCG+20, OSY21, RS21]; however, these constructions perform more poorly in terms of concrete efficiency, and most are restricted to the two-party case.

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 a, b and all but one share of $a \cdot b$ are directly compressible via black-box use of a pseudorandom generator), together with additional $O(n \cdot \log |C| \cdot \kappa)$ (circuit dependent) 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 reduce the total storage cost, which may be a practical bottleneck, by a factor of κ (from $O(|C| \cdot \kappa)$ to $O(|C|)$), without substantially increasing the online communication as in previous works.

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 (semi-honest) 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 [BCG+19b, BCG+19a, YWL+20].

For concretely efficient PCG-based protocols with $n \geq 3$ parties, one can use a PCG for OLE [BCG+20] 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 (see Remark 3.9).

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 reduces to securely evaluating an arithmetic circuit with roughly $(4 + 2n)|C|$ multiplications.

1.2 Our Techniques

The main technical building block in our compiler is a *fully linear proof system* [BBC+19], 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). It was shown in [BBC+19] that for low-degree languages (i.e., languages for which membership can be checked using degree-d multivariate polynomials, for some constant d), there exist zero-knowledge 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 for proofs over inputs which are distributed or linearly secret-shared between two or more verifiers. As shown in [BBC+19], they provide a means for the verifiers to be convinced that their shares combine to an input in the language, while learning nothing else—including additional information about the input itself. This was used by [BBC+19, BGIN19, BGIN20] as a tool to compile semi-honest protocols to malicious security with sublinear communication cost in the honest-majority setting, leveraging the fact that the statement to be proven in certain MPC protocols can be represented by low-degree polynomial constraints on the values held secret shared or distributed across the parties.

However, there is one crucial property that these works all relied on: in the honest-majority setting, the secret sharing of values across parties is *robust*, meaning that the shares held by the honest parties determine all the other shares. For example, in protocols using Shamir secret sharing, the honest parties' shares can be interpolated to the entire secret sharing polynomial, yielding also the corrupted parties' share values. This robustness property 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. Indeed, without a form of robustness of the input, even the definition of soundness in the proof is not fully clear.

In the *dishonest*-majority setting, in contrast, simple secret sharing schemes cannot provide this kind of robustness. After all, if the honest parties can reconstruct the value, then so can the dishonest parties. Thus, at first glance, it seems that fully linear proof systems inherently 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.

Overcoming this challenge is one of the main conceptual ideas in this work. To summarize: In the preprocessing model, one can view the *dealer* as an additional party in the protocol, but who is *guaranteed to be honest*. Indeed, recall that the preprocessing model provides the parties with an honestly generated sample from the dictated preprocessing correlation.[3] From this view, the star-sharing scheme discussed above actually forms a robust secret sharing! Recall that in a star-sharing scheme, the dealer holds additive shares $(r_i)_{i\in[n]}$ of a random mask r, and each online (real) party P_i holds its respective share r_i of the mask along with the public masked value $(x - r)$. Robustness holds since any honest party together with the trusted dealer determine the shares held by the corrupted parties. Further, as noted above, the star-sharing scheme satisfies the same multiplicative property as Shamir secret sharing satisfies in the honest-majority setting, wherein parties can locally convert their shares of two secrets to additive shares of the product of the two secrets. Combining these properties, in a sense, our new protocol is an analog of the honest-majority protocols, but with the role of Shamir secret sharing replaced with star-sharing.

However, this new conceptual view does not follow directly. Recall that the actions of the dealer must be completely determined in the preprocessing phase, *before* any of the online inputs are known. This means the dealer's abilities as a party are limited. In particular, to execute this strategy, some parts of the protocol must be rearranged. We ensure 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.

Ultimately, we show how to mimic the proof structure from [BGIN20], where each party locally computes a share of a verification polynomial using its secret shares throughout the first part of the protocol, and then proves to all other parties that it computed this value correctly. This takes advantage of the fact that in a robust secret sharing scheme such as star-sharing, not only is the secret payload robustly shared, but also each party's share of the payload.

More concretely, for every input wire to an input gate, multiplication gate, or output gate in the circuit, the parties will hold a star-sharing of the corresponding wire value as part of the underlying semi-honest secure protocol. The verification polynomial is a (pseudo-)random linear combination of roughly circuit-size-many degree-2 constraints on these values, ensuring that each input wire to a multiplication or output gate, is computed as a correct degree-2 function of wires from the previous level (i.e., input wires to previous multiplication or input gates). Each party

[3]As discussed, this in turn can be emulated via a targeted secure protocol, or directly realized by an external party or trusted hardware.

can locally compute an *additive share* of this verification polynomial, as a function of their star shares of the given wire values, leveraging the multiplicative property of the star-sharing scheme. This process itself is a degree-2 function of the shares. Then, each party robustly star-shares its (short) share of the verification polynomial, and proves to the other parties that they performed the degree-2 computation correctly. If all parties performed their computation correctly, then the final verification polynomial can be revealed and checked whether it is equal to 0 as required.

Combined, our central technical contribution is a novel protocol with sublinear communication to verify the correctness of a semi-honest computation, which builds upon any (zero-knowledge, sublinear) 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. 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 [BBC$^+$19]. 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 multiplication triple based protocol [Bea91] to achieve malicious security [BDOZ11, DPSZ12, DKL$^+$13, CDE$^+$18, KOS16, KPR18], 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 ab, which are required for the semi-honest protocol). These are used to authenticate the opening of the actual values. For authentication over a field $\Delta \in \mathbb{F}$, the cheating probability is $\frac{1}{|\mathbb{F}|}$. Thus, for computations over a large field (for which $a, b, ab \in \mathbb{F}$), this method doubles the amount of correlated randomness compared to that of the semi-honest protocol. When computing over a small field, the authenticator value Δ, and thus the authenticated triples, should be produced over a larger field to obtain negligible cheating probability, 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 [NNOB12, BLN$^+$21] for Boolean circuits.

However, some improvements were suggested over the years. The MiniMac protocol [DZ13] (optimized and implemented in [DLT14]) focuses on reducing overall computation costs for circuits over small fields (including the size of correlated randomness) at the expense of greater online communication. Their idea is to batch the authentication via linear error-correcting codes (ECC). However, the ECC used must have good minimal distance in order to provide security within multiplications of batched vectors; achieving this requires lower rate of the ECC encoding, translating to greater overhead in communication. A recent work by [CG20] suggested an alternative to the linear ECC of MiniMAC, via "reverse multiplication friendly embeddings" for embedding $(\mathbb{F}_q)^k$-vector multiplications into a single $F_{q^{k'}}$ field multiplication [CCXY18, BMN18]. However, the gap between k and k' achievable within these embeddings again yields overheads in communication. In addition, 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 from the correlated randomness, 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.

Over a large ring $\mathbb{Z}_{2^k}$, the SPDZ-2k protocol [CDE$^+$18] 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.

A different approach for 2-party computation was suggested in the TinyTable protocol [DNNR17], 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.

Finally, there are several other general compilation techniques from semi-honest to malicious security in the dishonest majority setting, which can also be applied to (certain) protocols in the preprocessing model [IPS08, IOZ14, HIMV19, HVW20]. All of these compilers incur at least a constant multiplicative overhead in the online communication and correlated randomness, and typically have a bigger overhead for Boolean circuits or circuits over rings.

Collectively, prior solutions all require $\Omega(|C|)$ additional cost in either online communication, correlated randomness, or both.

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 secure 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 a functionality in the preprocessing model.

Definition 2.1 (MPC with preprocessing) *Let $\mathcal{F}$ be a family of 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* $\mathsf{NextMsg}$*, which given $(1^\kappa, f, j, i, x_i, r_i, \vec{m})$ outputs a vector of messages sent by P_i in round j, based on its input x_i, randomness r_i and vector $\vec{m}$ of messages sent to P_i in previous rounds. If the output of* $\mathsf{NextMsg}$ *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* $\mathsf{NextMsg}$*, 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* $r_1, \dots, r_n$*. 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 [Gol04, Can00]. 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 $\textsc{real}_{\Pi,\mathcal{A},T}(1^\kappa, f, \vec{v})$ 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 $\vec{v}$ 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.[4] We denote by $\textsc{ideal}_{\mathcal{F},\mathcal{S},T}(1^\kappa, f, \vec{v})$, the random variable that consists of the output of $\mathcal{S}$ and the honest parties in an ideal execution to compute f, over a vector of inputs $\vec{v}$, 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* $|T| \le n-1$*, there exists an ideal-world adversary* $\mathcal{S}$*, such that for every* $f \in \mathcal{F}$*, every* κ *and every vector of inputs* $\vec{v}$ *it holds that*

$$SD\left(\textsc{real}_{\Pi,\mathcal{A},T}(1^\kappa, f, \vec{v}), \textsc{ideal}_{\mathcal{F},\mathcal{S},T}(1^\kappa, f, \vec{v})\right) \le \varepsilon$$

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

[4]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.

2.1.1 Secure computation of circuits up to additive attacks [GIP$^+$14]

In this work, our protocol computes arithmetic circuits, which are defined in a natural way, using addition and multiplication gates. We next define a notion of security for computing arithmetic circuits that lies between semi-honest and malicious security, called "security up to additive attack" (for brevity, sometimes "additive security"), which was introduced by Genkin et al [GIP$^+$14]. In this model the real-world adversary is malicious, but the ideal-world adversary $\mathcal{S}$ is given extra power when interacting with the trusted party, to add errors to the values on 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 $\textsc{ideal}^{add}_{\mathcal{F},\mathcal{S},T}(1^\kappa, C, \vec{v})$ the random variable that 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 (Additive-secure MPC with preprocessing) *Let $\mathcal{F}$ be the family of n-party functionalities represented by arithmetic circuits. We say that a protocol $\Pi = (\mathsf{NextMsg}, \mathcal{D})$ 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 (additive-attack) adversary $\mathcal{S}$, such that for every circuit $C \in \mathcal{F}$, every κ, and every vector of inputs $\vec{v}$ it holds that $\textsc{real}_{\Pi,\mathcal{A},T}(1^\kappa, C, \vec{v}) \equiv \textsc{ideal}^{add}_{\mathcal{F},\mathcal{S},T}(1^\kappa, C, \vec{v})$.*

Instantiations. Many standard semi-honest protocols in the preprocessing model used in the literature are in fact also additively secure (or can easily be converted into being so). Most notably, the semi-honest protocol which uses the well-known Beaver's method [Bea91] to compute multiplication gates via random multiplication triples satisfies this definition. It is known that MPC with preprocessing (including Beaver's protocol) can benefit from circuit-dependent correlations [DNNR17, Cou19, BGI19]. For completeness, in Appendix A.1 we present a variant of Beaver's protocol relying on circuit-dependent preprocessing (as in e.g. [KKW18, BBC$^+$19, BNO19]), and in Appendix A.2, the standard circuit-independent version.

2.1.2 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 [Can00] 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.

2.2 Fully Linear Proof Systems

A main technical building block in our protocols is a *fully linear* proof system [BBC$^+$19], which was shown to enable information-theoretic sublinear-communication zero-knowledge proofs on secret-shared input statements [BBC$^+$19].

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 [BBC+19]) *A public-coin fully linear interactive proof system over R with ρ-rounds, ℓ-queries and message lengths $(u_1, \ldots, u_\rho) \in \mathbb{N}^t$, 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 $r_0 = \bot$. In each round $i \in [\rho]$:*

1. *P outputs a proof $\pi_i \in R^{u_i}$, computed as a function of $x, r_1, \ldots, r_{i-1}$ and its private randomness.*
2. *A random public challenge r_i is chosen uniformly from a finite set S_i.*
3. *ℓ linear oracle queries $q_1^i, \ldots, q_\ell^i \in R^{m+u_i}$ are determined based on $r_1, \ldots, r_i$. Then, V receives ℓ answers $(\langle q_1^i, x||\pi_i\rangle, \ldots, \langle q_\ell^i, x||\pi_i\rangle)$.*

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 a language. We say that ρ-round ℓ-query interactive fully linear protocol $(\mathsf{P_{FLIOP}}, \mathsf{V_{FLIOP}})$ 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 $\mathsf{V_{FLIOP}}$ always outputs* accept
- Soundness: *If $x \notin \mathcal{L}$, then for all P^*, the probability that $\mathsf{V_{FLIOP}}$ 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 $\mathcal{S}_{\mathsf{FLIOP}} \equiv \mathsf{view}_{[\mathsf{P_{FLIOP}}(x),\mathsf{V_{FLIOP}}]}(\mathsf{V_{FLIOP}})$ (where the verifier's view $\mathsf{view}_{[\mathsf{P_{FLIOP}}(x),\mathsf{V_{FLIOP}}]}(\mathsf{V_{FLIOP}})$ 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 communication and round complexity that are sublinear in the size of the input and number of monomials.

Theorem 2.5 ([BBC+19]) *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 [RB89, BDOZ11, DPSZ12]. 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 and change the opened value, 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 dealer $\mathcal{D}$ with shares $\hat{x}, (r_1, \ldots, r_n)$, if each party P_i holds the pair $(\hat{x}, r_i)$, $\mathcal{D}$ holds $\{r_i\}_{i=1}^n$, and moreover $\hat{x} = x - r$ for $r = \sum_{i=1}^n r_i$.*

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, this scheme is multiplicative in the sense that it allows a local conversion from star-sharing of x and y 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 of 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.

The parties prove correctness of the values on *all* wires by checking only the input wires to each multiplication gate and the output wires. Any input to a multiplication gate is a sum of the outputs of other multiplication gates and of input wires. Therefore, an input to a multiplication gate is a degree two polynomial in the inputs of other gates and in the circuit's input wires. The precise polynomial is determined by the circuit. Similarly, each output wire is a degree two polynomial in inputs to the circuit and in inputs to multiplication gates. Verifying the consistency of all degree two polynomials is made possible by the star sharing of all wire values, enabling the honest parties to obtain a sharing of the correct output.

In more detail, 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 x_w, the correct value on w, given the shares 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 thus wish to verify for each $w \in W$ that

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

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.

In the protocol, instead of checking equality to 0 for each wire 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

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

Next, let W^{g_ℓ} denote the set of wires w whose value is the output of a multiplication gate g_ℓ and let $\gamma_\ell = \sum_{w\in W^{g_\ell}} \alpha_w$. Put differently, W^{g_ℓ} includes exactly all the wires w such that $g_\ell \in G_w$. Letting mult be the set of all multiplication gates, we can thus write

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

Now, letting

$$\Lambda = \sum_{w\in W} \alpha_w \cdot \hat{x}_w - \sum_{g_\ell\in\mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot \hat{x}_2^{g_\ell}),$$

$$\Gamma_i = \sum_{g_\ell\in\mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot r_{2,i}^{g_\ell} + \hat{x}_2^{g_\ell} \cdot r_{1,i}^{g_\ell}) \tag{1}$$

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 $p = 0$ 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 sends $\hat{\Gamma}_i = \Gamma_i - s_i$ 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, 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 cheat and share an incorrect Γ_i. Here is where the zk-FLIOP machinery becomes useful. Define the vector of inputs $\vec{y} \in \mathbb{F}^{4|\mathsf{mult}|+2}$ as:

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

and consider the degree-2 polynomial c defined by

$$\begin{aligned} c(\vec{y}) &= \hat{\Gamma}_i + s_i - \Gamma_i \\ &= y_1 + y_2 - \sum_{k=1}^{|\mathsf{mult}|} \left(y_{[4(k-1)+3]} \cdot y_{[4(k-1)+4]} + y_{[4(k-1)+5]} \cdot y_{[4(k-1)+6]} \right). \end{aligned} \tag{3}$$

If P_i star-shares the correct Γ_i then $c(\vec{y})$ is identically zero. To verify that this is the case is sufficient to show that Eq. (1) holds. By Theorem 2.5, there exists a zk-FLIOP for proving the satisfiability of this relation, i.e. that $c(\vec{y})$ is identically zero, with sublinear proof size and negligible soundness error. 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 polynomial is known by either all parties *or* by P_i and the dealer. In detail, for all $1 \leq k \leq |\mathsf{mult}|$, all the parties know $y_1, y_{4(k-1)+3}$ and $y_{4(k-1)+5}$, while the dealer and P_i know $y_2, y_{4(k-1)+4}$ and $y_{4(k-1)+6}$.

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 and proof) is known by an honest participant (i.e., an honest party or the trusted dealer). This fact prevents a cheating prover from convincing the other parties that a false statement is correct.

As to the verifiers, who hold 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 reveal its star-share of each answer (this share is essentially 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):

$\underline{\Pi_{\mathsf{vrfy}}}$**:** Let $(\mathsf{P}_{\mathsf{FLIOP}}, \mathsf{V}_{\mathsf{FLIOP}})$ be a zk-FLIOP protocol with ρ rounds, ℓ-queries per round and message length $u_1, \ldots, u_\rho \in \mathbb{N}$ for the polynomial in Eq. (3).

1. The trusted dealer $\mathcal{D}$:
   (a) Chooses a random $s_i \in \mathbb{F}$ for each $i \in [n]$, and hands it to P_i.
   (b) Chooses a random field element $\alpha \in \mathbb{F}$ and hands it to the parties. Alternatively, α can be chosen to be a random seed for a PRG.
   (c) Chooses a random $t^i_j \in \mathbb{F}^{u_j}$ for each $j \in [\rho]$ and $i \in [n]$ and hands it to P_i.
   (d) Computes α_w from α for every wire w. If $\alpha \in \mathbb{F}$ then regard w as a unique integer in the range $1, \ldots, W$ and set $\alpha_w = \alpha^w$. Alternatively, if α is a seed for a PRG then expand it using the PRG to $|W|$ field elements deriving all the α_w values. The dealer then computes Ω, chooses a random $\mu \in \mathbb{F}$ and hands $\hat{\Omega} = \Omega - \mu$ 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 computes Λ and Γ_i. P_i then proceeds to send $\hat{\Gamma}_i = \Gamma_i - s_i$ to the other parties.

4. For each $i \in [n]$, party P_i proves that Γ_i was computed correctly:
   Let $\vec{y}_i$ be the vector of inputs for the proof of P_i (as defined in Eq. (2)). Let $\vec{y}_i^{\mathcal{P}}$ be a vector of elements generated by replacing all elements in $\vec{y}_i$ which are *not* known to all parties by 0, and let $\vec{y}_i^{\mathcal{D}}$ be a vector of elements generated by replacing all elements in $\vec{y}_i$ *not* known to $\mathcal{D}$ by 0. Note that $\vec{y}_i = \vec{y}_i^{\mathcal{P}} + \vec{y}_i^{\mathcal{D}}$.

   (a) For each round j of the zk-FLIOP:

      i. If $j = 1$, party P_i lets $\pi_j^i = \mathsf{P_{FLIOP}}(\vec{y}_i, \perp)$. Otherwise, it lets $\pi_j^i = \mathsf{P_{FLIOP}}(\vec{y}_i, \pi_{j-1}^i, r_{j-1}^i)$.

      ii. P_i broadcasts $\hat{\pi}_j^i = \pi_j^i - t_j^i$ to the other parties.

      iii. The dealer $\mathcal{D}$ chooses a random challenge r_j^i and hands it to the parties.

      iv. The parties and the dealer let $q_{j,1}^i, \ldots, q_{j,\ell}^i$ be the query vector determined by $\mathsf{V_{FLIOP}}$ based on r_j^i. Then, the parties compute the answers

$$\hat{a}_{j,1}^i, \ldots, \hat{a}_{j,\ell}^i \leftarrow \langle q_{j,1}^i, \vec{y}_i^{\mathcal{P}} || \hat{\pi}_j^i \rangle, \ldots, \langle q_{j,\ell}^i, \vec{y}_i^{\mathcal{P}} || \hat{\pi}_j^i \rangle.$$

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

$$\widetilde{a}_{j,1}^i, \ldots, \widetilde{a}_{j,\ell}^i \leftarrow \langle q_{j,1}^i, \vec{y}_i^{\mathcal{D}} || t_j^i \rangle, \ldots, \langle q_{j,\ell}^i, \vec{y}_i^{\mathcal{D}} || t_j^i \rangle.$$

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

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

   (b) The parties run the decision predicate of $\mathsf{V_{FLIOP}}$ 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 $\hat{p} = \Lambda - \sum_{i=1}^n \hat{\Gamma}_i + \hat{\Omega}$. Then, the dealer $\mathcal{D}$ hands $s = -\sum_{i=1}^n s_i + \mu$ to the parties.

6. The parties locally compute $p = \hat{p} + s$. If $p = 0$, then the parties output accept. Otherwise, they output reject.

Proposition 3.3 *Let Δ_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 $(\mathsf{P_{FLIOP}}, \mathsf{V_{FLIOP}})$ be a ρ-rounds, ℓ-queries and ε-soundness error zk-FLIOP protocol. Then, Π_{vrfy} satisfies the following properties:*

1. Correctness: *If $\forall w \in W : \Delta_w = 0$ and all parties follow the protocol's instructions, then the honest parties always output* accept.

2. Soundness: *If $\exists w \in W : \Delta_w \neq 0$, then the honest parties output* accept *with probability of at most $\frac{|W|}{|\mathbb{F}|} + \varepsilon$* [5].

[5]This is true when α_w is set by taking α^w. When α_w is generated via a PRG, it can be shown by the pseudorandom assumption that soundness holds.

3. Privacy: *For every adversary $\mathcal{A}$ controlling a subset T of size $\leq n-1$, there exists a simulator $\mathcal{S}$, who receives $\{\Delta_w, \hat{x}_w, \{r_{w,i}\}_{i\in T}\}_{w\in W}$ as an input, and outputs a transcript $\mathsf{view}_{\mathcal{S}}$, such that $\mathsf{view}_{\mathcal{S}} \equiv \mathsf{view}_{\mathcal{A}}^{\pi_{\mathsf{vrfy}}}$.*

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 $p = 0$. It remains to show that the parties will output accept in the zk-FLIOP protocol. Given a proof π_j^i, it holds that $\pi_j^i = \hat{\pi}_j^i + t_j^i$. Then, when the parties compute the answers to the linear queries, we have $\forall l \in [\ell]$:

$$a_{j,l}^i = \hat{a}_{j,l}^i + \widetilde{a}_{j,l}^i = \langle q_{j,l}^i, \vec{y}_i^{\mathcal{P}} || (\pi_j^i - t_j^i)\rangle + \langle q_{j,l}^i, \vec{y}_i^{\mathcal{D}} || t_j^i\rangle = \langle q_{j,l}^i, \vec{y}_i || \pi_j^i\rangle$$

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

Soundness. If $\exists w \in W : \Delta_w \neq 0$, then the parties will output accept if $p = 0$. 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 $p = \sum_{w\in W} \alpha_w \cdot \Delta_w = \sum_{w\in W} \alpha^w \cdot \Delta_w$ and so, fixing all Δ_w, this is a polynomial of degree $|W|$ evaluated on a random point α. Thus, by the Schwartz-Zippel lemma, $p = 0$ 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 $\{\Delta_w, \hat{x}_w, \{r_{w,i}\}_{i\in T}\}_{w\in W}$ 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_j^i 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 computs Λ.

4. Upon receiving from $\mathcal{A}$ all $\{\hat{\Gamma}_i\}_{\imath\in T}$, 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 $\hat{\pi}_j^i$ and sends it to $\mathcal{A}$. Then, playing the role of $\mathcal{D}$, it hands a random challenge r_j^i to $\mathcal{A}$. To simulate the opening of the query answers, $\mathcal{S}$ run $\mathcal{S}_{FLIOP}$ to receive $a_{j,1}^i, \ldots, a_{j,\ell}^i$. Then, for each $l \in [\ell]$, it computes $\hat{a}_{j,l}^i$ (since it knows all the corrupted parties' inputs and so all the values in $\vec{y}_i^{\mathcal{P}}$) and then sets $\widetilde{a}_{j,l}^i = a_{j,l}^i - \hat{a}_{j,l}^i$ 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 $p = \sum_{w \in W} \alpha_w \cdot \Delta_w + \sum_{i \in T} (\Gamma'_i - \Gamma_i)$ and $\hat{p} = \Lambda - \sum_{i=1}^{n} \hat{\Gamma}_i + \hat{\Omega}$. Then it sets $s = p - \hat{p}$ 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 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 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 2.5, we still have a zk-FLIOP with sublinear cost. However, the probability that $p = 0$ 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 [BBC$^+$19, BGIN19], 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 $p = 0$ 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 $\{t^i_j\}_{j \in [\rho]}$ where $t^i_j \in R^{u_j}$, 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 $F^{\mathsf{dealer}}_{\mathsf{com}}$ 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 $F^{\mathsf{dealer}}_{\mathsf{com}}$. 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

$$\left(3 + n \cdot \left(\sum_{j=1}^{\rho} u_j + \rho(1+\ell)\right)\right) \cdot d \ \ \textsf{ring elements} \tag{4}$$

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 2.5, 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

$$\left(n \cdot \sum_{j=1}^{\rho} u_j\right) \cdot d + (3 + n \cdot (\rho + \rho \cdot \ell)) \cdot d \cdot F_{\mathsf{com}}^{\mathsf{dealer}} \ \ \textsf{ring elements} \tag{5}$$

which by Theorem 2.5, 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 2.5, 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 3.4 *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).*

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 $\Pi_{\mathsf{mpc}}^{\mathsf{add}}$ 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 $s_i^k \in R$ to P_i and a random $s_{i,j}^k$ to P_j such that $s_i^k = \sum_{j=1}^{n} s_{i,j}^k$.
  - It hands the parties the correlated randomness required by $\Pi_{\mathsf{mpc}}^{\mathsf{add}}$. 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 $F_{\mathsf{com}}^{\mathsf{dealer}}$.
- **The online protocol:**
  - Sharing the inputs: For each wire k, with input v_i^k held by P_i, it broadcasts $\hat{v}_i^k = v_i^k - s_i^k$ to the other parties.
  - Circuit emulation: The parties compute the circuit C gate-by-gate in some predetermined topological order, by running $\Pi_{\mathsf{mpc}}^{\mathsf{add}}$, 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 $F_{\mathsf{com}}^{\mathsf{dealer}}$, 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 3.5 *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 $\Pi_{\mathsf{mpc}}^{\mathsf{add}}$ 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 with soundness error $\frac{\varepsilon}{2}$ as defined in Definition 2.4, then Π_{MPC} ε-securely computes f in the $F_{\mathsf{com}}^{\mathsf{dealer}}$-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 Δ_w for each wire $w \in W$.
- Verification: Let $\mathcal{S}_{vrfy}$ be the simulator for Π_{vrfy} from Theorem 3.3. The simulator $\mathcal{S}$ invokes $\mathcal{S}_{vrfy}$ on $\{\Delta_w, \hat{x}_w, \{r_{w,i}\}_{i\in T}\}_{w\in W}$, 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 $\mathsf{out} = \mathsf{reject}$, then $\mathcal{S}$ sends abort to the trusted party computing f and outputs whatever $\mathcal{A}$ outputs. Else, $\mathsf{out} = \mathsf{accept}$. If $\forall w \in W : \Delta_w = 0$, then $\mathcal{S}$ proceeds to the next step. Otherwise, $\exists w \in W : \Delta_w \neq 0$ 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 $F^{\mathsf{dealer}}_{\mathsf{com}}$. If $\mathcal{A}$ sends abort to $F^{\mathsf{dealer}}_{\mathsf{com}}$, 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 $\Pi^{\mathsf{add}}_{\mathsf{mpc}}$, 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 $\exists\Delta_w \neq 0$. 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 3.3), 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 3.4 and Proposition 3.5, we obtain the following theorem, which summarize our main result in this work:

Theorem 3.6 *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 $R = \mathbb{Z}_{2^k}$.*

Concrete cost. To obtain the concrete cost of the entire protocol, one can simply take the cost of the underlying semi-honest protocol and the cost of Π_{vrfy} given at the end of Section 3.1 and add them together.

Protocol instantiations. From our main theorem we derive the following corollaries. We apply our construction on the well-known semi-honest protocol based on Beaver triples [Bea91]. First, we obtain a protocol in the circuit-dependent preprocessing, 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 3.7 (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, where $\kappa = \kappa(\varepsilon)$ is defined as in Theorem 3.6.*
- 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 (in addition to the PRG seed size).*

Proof: Consider the semi-honest protocol described in Appendix A.1, which is the circuit-dependent version of the well-known Beaver's [Bea91] protocol, as described in [BGI19]. 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 3.6 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.

Recall that the (logarithmic size) extra correlated randomness introduced by our compiler *does* depend on the structure of the circuit, and thus the resulting protocol is in the circuit-dependent preprocessing model. However, as the semi-honest portion of the correlated randomness is a dominant cost that can be generated more efficiently in the circuit-independent case (e.g., via the use of highly efficient pseudorandom correlation generators (PCGs) [BCG$^+$19b] in the 2-party setting), we address this case as well.

Corollary 3.8 (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, , where $\kappa=\kappa(\varepsilon)$ is defined as in Theorem 3.6.*
- 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 (in addition to the PRG seed size). With PCG-based compression of (n-wise) multiplication triples over R, the entire correlated randomness can be reduced to $O(n\cdot\log|C|\cdot\kappa)$ elements to each party (in addition to the PCG seed size).*

Proof: The proof is identical to the proof of Corollary 3.7, 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. ■

As a final application of our main theorem, we consider an underlying semi-honest n-party protocol making use of *pairwise*-correlation preprocessing. Although this protocol requires even greater online communication—on the order of $O(n^2|C|)$ compared to $O(n|C|)$ total communication—once again the benefit is that the preprocessing correlation is of a simpler form that admits highly efficient generation procedures. More specifically, the correlation consists of several *pairwise* multiplication triples (easily generated from pairwise OT/OLE correlations), as opposed to fewer but somewhat more complex n-wise multiplication triples. Given these tradeoffs, this protocol will indeed be competitive in certain settings with $n\geq 3$. We refer the reader to Appendix A.3 for further details.

Remark 3.9 (Pairwise-correlation preprocessing) *There exists an n-party protocol for ε-securely computing an arithmetic circuit C as above, with the following properties:*

- Communication: *each party sends $2(n-1)+(4-\frac{4}{n})|C|$ ring elements.*
- Correlated randomness: *the circuit-independent preprocessing outputs $3(n-1)\cdot|C|$ R-elements per party (in the form of $n(n-1)\cdot|C|$ pairwise multiplication triples), and an additional circuit-dependent preprocessing which outputs $O(n\cdot\log|C|\cdot\kappa)$ elements to each party.*

With efficient PCG-based compression of pairwise random OLE correlations over R (equivalent to OT correlations when $R = \mathbb{Z}_2$), the entire correlated randomness can be reduced to $O(n \cdot \log |C| \cdot \kappa)$ elements to each party.

where $\kappa = \kappa(\varepsilon)$ is defined as in Theorem 3.6.

Remark 3.10 (Multicast vs. private channels) *The communication cost presented in Corollaries 3.7 and 3.9 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.*

3.3 Concrete costs with an instantiation of the zk-FLIOP

Based on the general constructions from [BBC$^+$19], we describe in Appendix B, an implementation of the zk-FLIOP protocol in our setting has the following parameters:

- Number of rounds: $\rho = \log(2|\mathsf{mult}|) - 1$
- Message length: $u_j = 3$ for $j \in [\rho - 1]$ and $u_\rho = 8$
- Number of linear queries per round: $\ell = 1$

The concrete features of the realization we obtain are:

- *Proof size:* $3(\log(2|\mathsf{mult}|) - 1) + 8$ elements broadcast by the prover.
- *Computation:* each verifier performs approximately $4|\mathsf{mult}|$ local operations for prime fields. When the original circuit was a Boolean circuit, this can be reduced to $3|\mathsf{mult}|$. In contrast, the prover performs $8|\mathsf{mult}$ operations over prime fields and $7|\mathsf{mult}|$ operations if the computed circuit is Boolean.
- *Soundness error:* When working over a finite field $\mathbb{F}$, the soundness error is at most $\frac{4\log|\mathsf{mult}|+1}{|\mathbb{F}|}$. When working over the ring Z_{2^k}, and the verification protocol is carried-out over the extension ring $\mathbb{Z}_{2^k}/f(x)$, where f is irreducible modulo 2 is of degree d, the soundness error is bounded by $\frac{4\log|\mathsf{mult}|+1}{2^d}$.

Plugging-in the above numbers into Equation (4) and Equation (5), we obtain that Π_{vrfy} has the following concrete costs:

- *Correlated randomness:* $(3 + 5n \cdot \log(2|\mathsf{mult}|)) \cdot d$ ring elements
- *Communication cost:* $n \cdot (\log(2|\mathsf{mult}|) + 2) \cdot d + (3 + 2n \cdot (\log(2|\mathsf{mult}|) - 1)) \cdot d \cdot F_{\mathsf{com}}^{\mathsf{dealer}}$ ring elements

where d is the extension degree.

Concrete online computation cost. The concrete computational cost of the protocol is dominated by ring multiplications. We give a bound on the number of these multiplications in the important case that the ring is a field, thereby providing an estimate of the protocol's computational cost.

The work of each party is divided into computing its share of Equation (1), proving in zk-FLIOP that its share is correct, and verifying in zk-FLIOP that the share of every other party is correct. For each of these stages, we bound the number of field multiplications as a function of the number of multiplication gates $|\mathsf{mult}|$.

To compute its share of Equation (1), each party first computes public random elements α_w for every wire $w \in W$. While it is possible to compute α_w as powers of a random value α, it is computationally cheaper in concrete terms to generate α_w as an expansion of a random seed to a long pseudo-random string, e.g. by using standard processors which incorporate AES instructions. Then, each party computes $\Lambda = \sum_{w \in W} \alpha_w \cdot \hat{x}_w - \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot \hat{x}_2^{g_\ell})$, and $\Gamma_i = \sum_{g_\ell \in \mathsf{mult}} \gamma_\ell \cdot (\hat{x}_1^{g_\ell} \cdot r_{2,i}^{g_\ell} + \hat{x}_2^{g_\ell} \cdot r_{1,i}^{g_\ell})$. If the field is a large prime field $\mathbb{F}_p$ then computing Λ requires $4|\mathsf{mult}|$ field multiplications and computing Γ_i requires $3|\mathsf{mult}|$ field multiplications. However, if the original circuit is a Boolean circuit and the field is $\mathbb{F}_{2^k}$, i.e. an extension field, then computing Λ and Γ_i does not require *any* field multiplications. The reason is that in each term of Λ and Γ_i there is at most one actual field element (either α_w or γ_ℓ) while all other elements are bits derived from the circuit. Therefore, the total number of field multiplications per party to compute its share of Equation 1 is $7|\mathsf{mult}|$ if the field is $\mathbb{F}_p$ and zero if the field is $\mathbb{F}_{2^k}$.

Based on the analysis in Appendix B, if the field is $\mathbb{F}_p$ then each party performs $8|\mathsf{mult}|$ field multiplications when it runs as a prover in the zk-FLIOP and $4|\mathsf{mult}|$ multiplications when it runs as a verifier, and if the field is $\mathbb{F}_{2^k}$ then the numbers are $7|\mathsf{mult}|$ field multiplications for a prover and $3|\mathsf{mult}|$ multiplications for a verifier. Since each party runs $n-1$ times as a verifier, the total number of field multiplications per party in an n-party protocol over a field $\mathbb{F}_p$ is $(15 + 4(n-1))|\mathsf{mult}|$, and over $\mathbb{F}_{2^k}$ the number of field multiplications is $(7 + 3(n-1))|\mathsf{mult}|$.

Cost comparison for online computation and communication. An important measure of the efficiency of an MPC protocol is total online time, which depends on both communication and computation in the online phase. The maliciously secure protocol we propose has negligible communication overhead compared to the base semi-honest protocol, but non-negligible computational overhead. This maliciously secure protocol is especially appealing when its computation requires less time than the semi-honest protocol, in which the bottleneck is typically its online communication. We give concrete examples of parameter ranges for 2PC in which the dominant cost of the whole protocol is the communication of the semi-honest portion of the protocol.

Consider 2PC for a Boolean circuit with $|\mathsf{mult}|$ multiplication gates. The zk-FLIOP for such a circuit is executed over an extension field $\mathbb{F}_{2^k}$, which is chosen to provide a desired soundness error. We give examples of concrete computational cost for $k = 64$ and $k = 128$, which are chosen to take advantage of hardware-based polynomial multiplication in modern processors. By the analysis in the beginning of this section, the soundness error for $k = 64$ and almost any reasonable circuit size, e.g. $|\mathsf{mult}| \leq 2^{30}$, is at most 2^{-57}, and the soundness error for $k = 128$ and $|\mathsf{mult}| \leq 2^{30}$ is at most 2^{-121}.

A field multiplication in $\mathbb{F}_{2^k}$ can be carried out in two stages, first a polynomial multiplication, doubling the input size, and then a reduction of the result back to the base field. Polynomial multiplication can be efficiently implemented in modern x86 architectures using the PCLMULQDQ instruction. The computational cost of reducing to the base field depends on the irreducible polynomial defining the field, and is lowest for specific choices of sparse polynomials.

With an appropriate choice of $\mathbb{F}_{2^{64}}$ and for sufficiently many multiplications, the polynomial multiplication requires 0.13 cycles per byte [Gue15], which translates to roughly one cycle per multiplication. In $\mathbb{F}_{2^{128}}$ the cost of polynomial multiplication tends to 0.3 cycles per byte, as the number of multiplications grows [GLL17], which is 4.8 cycles per multiplication. A good algorithm for reduction modulo a sparse polynomial on an x86 architecture [BG15] requires at most 13 cycles for $\mathbb{F}_{2^{64}}$ and 14 cycles for $\mathbb{F}_{2^{128}}$. Taken together, a large number of field multiplications requires 14 cycles per multiplication in $\mathbb{F}_{2^{64}}$ and 19 cycles per multiplication in $\mathbb{F}_{2^{128}}$.

To compare online communication and computation for 2PC of Boolean circuits, recall that in the semi-honest protocol using multiplication triples each party sends two bits for every multiplication (AND) gate or $2|\mathsf{mult}|$ bits for the whole circuit. Based on the previous analysis, the online computational cost is dominated by $10|\mathsf{mult}|$ field multiplications which require 140 cycles per multiplication gate for $\mathbb{F}_{2^{64}}$ and 190 cycles for $\mathbb{F}_{2^{128}}$.

Running on a single core at $3.6 \cdot 10^9$ cycles per second, the communication in the semi-honest protocol becomes the bottleneck for the whole protocol if the proof is in $\mathbb{F}_{2^{64}}$ (soundness error 2^{-57}) and network speed is at most 51 Mbps, or if the field is $\mathbb{F}_{2^{128}}$ (soundness error 2^{-121}) and network speed is at most 37 Mbps. Assuming that the online computation can be easily parallelized, these figures determine the number of cores required to ensure that online computation of the maliciously secure protocol does not take more time than communication for the semi-honest protocol. For example, two cores are sufficient if the field is $\mathbb{F}_{2^{64}}$ and network speed is 100 Mbps.

A different approach, which might be even more attractive, is to offload field multiplications to a GPU, taking advantage of the inherent parallelism of the work of a prover and a verifier in our protocol. One data point is the work of Ben-Sasson et al. [BSHST16] reporting a peak throughput of 2.09 billion multiplications in $\mathbb{F}_{2^{64}}$, implying that the communication of the semi-honest protocol would be the bottleneck for such implementations with network speeds at most 4 Gbps.

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 $\widetilde{a}^i_{j,1}, \ldots, \widetilde{a}^i_{j,\ell} \leftarrow \langle q^i_{j,1}, \vec{y}^{\mathcal{P}}_i || t^i_j \rangle, \ldots, \langle q^i_{j,\ell}, \vec{y}^{\mathcal{P}}_i || t^i_j \rangle$ 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 Appendix B, 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 [CDE$^+$18, KPR18, HIMV19, BCG$^+$20], 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.

Distributing the dealer for PRG/PCG-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 3.7 and 3.9). 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. This implies overall storage complexity which is sublinear in $|C$ for all parties but one party.

To achieve sublinear storage complexity for all parties, we can use PCGs (pseudorandom correlation generators) [BCG$^+$19a, BCG$^+$19b] to generate the semi-honest correlated randomness. Here each party receives a seed from which it locally derives its share of the desired correlation, which in our case is a multiplication triple. In particular, for secure 2-party computation of Boolean circuits, each triple can be locally generated using 2 random OT correlations, where the latter can be efficiently compressed using fast PCGs for OT [BCG$^+$19b, BCG$^+$19a, YWL$^+$20]. For concretely efficient PCG-based protocols for $n \geq 3$ parties, one can use a PCG for OLE [BCG$^+$20] 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.

We remark that unlike the first method of producing the entire correlated randomness using a circuit, when using PRG/PCGs, proving security is a bit more tricky, since a corrupted party may feed the MPC protocol (to compute the correlated randomness for the malicious execution) with incorrect PRG/PCG output. The crucial point here is that such an attack does not hurt the security of the online protocol. In particular, for the concrete instantiation we describe in Appendix B, one can observe that in order to not being caught, the adversary needs to produce a non-zero polynomial which will evaluate to 0 on a random point r, without knowing r. Thus, if the adversary cheats, it will be caught in the online verification protocol.

Acknowledgements

The authors thank Shay Gueron for helpful pointers regarding concrete efficiency estimations.

E. Boyle supported by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069 FA9550-21-1-0046, ERC Project HSS (852952), and ERC Project NTSC (742754). 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

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO*, 2019.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS*, 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO*, 2020.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[BG15] Manuel Bluhm and Shay Gueron. Fast software implementation of binary elliptic curve cryptography. *Journal of Cryptographic Engineering*, 5(3):215–226, 2015.

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *TCC*, 2019.

[BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In *ACM CCS*, 2019.

[BGIN20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In *ASIACRYPT*, 2020.

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. In *Crypto*, 2021.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.

[BLN$^+$21] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High-performance multi-party computation for binary circuits based on oblivious transfer. *J. Cryptol.*, 34(3):34, 2021.

[BMN18] Alexander R. Block, Hemanta K. Maji, and Hai H. Nguyen. Secure computation with constant communication overhead using multiplication embeddings. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT*, volume 11356 of *Lecture Notes in Computer Science*, pages 375–398, 2018.

[BNO19] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online spdz! improving SPDZ using function dependent preprocessing. In *ACNS*, 2019.

[BSHST16] Eli Ben-Sasson, Matan Hamilis, Mark Silberstein, and Eran Tromer. Fast multiplication in binary fields on gpus via register cache. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12, 2016.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, 1988.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO*, pages 395–426, 2018.

[CDE$^+$18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spdz2k: Efficient MPC mod 2k for dishonest majority. In *CRYPTO*, 2018.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT 2000*, pages 316–334, 2000.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In *TCC*, 2020.

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT*, volume 11477 of *Lecture Notes in Computer Science*, pages 473–503. Springer, 2019.

[DKL$^+$13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, 2013.

[DLT14] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the minimac protocol for secure computation. In *SCN*, 2014.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO*, 2017.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, 2013.

[GIP$^+$14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, 2014.

[GLL17] Shay Gueron, Adam Langley, and Yehuda Lindell. Aes-gcm-siv: Specification and analysis. *IACR Cryptol. ePrint Arch.*, 2017:168, 2017.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

[Gue15] Shay Gueron. Software methods for fast hashing. *Transactions on Networks and Communications*, 3(1):85, 2015.

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *ACM CCS*, 2019.

[HVW20] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. In *EUROCRYPT 2020, Proceedings, Part II*, pages 184–215, 2020.

[IKM$^+$13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *TCC*, 2013.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014, Part II*, pages 369–386, 2014.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO 2008*, pages 572–591, 2008.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 525–537. ACM, 2018.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS*, 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, 2018.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT*, 2021.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, 1989.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. *IACR Cryptol. ePrint Arch.*, 2021:274, 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS*, 2020.

A Protocols With Security up to Additive Attacks

In this section, we present two instantiations 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 latter 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 [BGI19]

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

$$\begin{aligned} \hat{z} = x \cdot y - r_3 &= (\hat{x} + r_1)(\hat{y} + r_2) - r_3 \\ &= \hat{x} \cdot \hat{y} + r_1 \cdot \hat{y} + r_2 \cdot \hat{x} + r_1 \cdot r_2 - r_3 \end{aligned} \tag{6}$$

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.[6]
  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 attack. 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 it random values as the messages from the honest parties. Once it receives the messages from the corrupted 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 [Bea91]

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.[7]

[6]Explicitly, each party sends z' to all other parties, effectively emulating the first round of Broadcast with Selective Abort (see $\mathcal{F}_{\mathsf{bc}}$ in Section 2.3). The second (echo) round of this broadcast procedure will be efficiently batch-checked together with *all* Broadcast with Selective Abort executions, by echoing a random linear combination of the collective list of received values.

[7]Explicitly, each party sends $\hat{x}, \hat{y}$ to all other parties, effectively emulating the first round of Broadcast with Selective Abort (see $\mathcal{F}_{\mathsf{bc}}$ in Section 2.3). The second (echo) round of this broadcast procedure will be efficiently batch-checked together with *all* Broadcast with Selective Abort executions, by echoing a random linear combination of the collective list of received values.

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 $\hat{x} \cdot \hat{y} + z^1$ 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.

A.3 Computation in the Pairwise-Correlation Preprocessing Model

We describe an additional underlying semi-honest secure protocol based on *pairwise*-correlation preprocessing, "GMW style" [GMW87]. Although the resulting protocol has higher online communication complexity than our previous two instantiations (which we do not attempt to concretely optimize), we include this instance as an additional alternative, because the corresponding preprocessing correlation admits particularly cheap generation, especially over $R = \mathbb{Z}_2$. More specifically, the preprocessing correlation of this protocol consists of pairwise, two-party multiplication triples, as opposed to the n-party multiplication triples from the previous sections. Although significantly more such multiplication triples are required—$n(n-1)$ per multiplication gate, as opposed to one n-wise triple—this means the protocol can take advantage of cheap and simple pseudorandom correlation generators that exist in the two-party setting. For example, for $R = \mathbb{Z}_2$, highly efficient "Silent OT" [BCG+19b] protocols enable generating a large number of pairwise-OT correlations, which can be non-interactively converted into pairwise multiplication triples over $\mathbb{Z}_2$ (at the rate of two OT instances per one multiplication triple).

The base version of the pairwise-correlation preprocessing protocol, à la [GMW87], computes additive secret shares of each circuit wire gate by gate, using $n(n-1)$ pairwise multiplication triples to convert from shares of wire values x, y to shares of their product xy, via pairwise products $x_i y_j$ between the shares of x and the shares of y. This version of the protocol does not directly fit into our compiler framework, since the values of the circuit are not *star-shared*; however, a simple modification will suffice. Namely, the online protocol can be augmented with a one-round procedure in which the collection of pairwise multiplication triples is converted into n-wise multiplication triples, for every gate of the circuit, in parallel, in which case the protocol can proceed precisely as the n-wise triple "Circuit-Independent Preprocessing Model" protocol as in Appendix A.2. The resulting online communication is thus the sum of the (quadratic in n) communication of the original pairwise-correlation semi-honest protocol for the multiplication triple conversion, plus the (linear in n) communication of the protocol as described in the previous section, which makes use of n-wise multiplication triples. It is likely this step can be improved; we leave optimization as a future goal.

Note that one can alternatively choose to view this multiplication triple conversion step as part of the preprocessing itself, i.e. simply as a means for effectively generating the n-wise multiplication triples for the Appendix A.2 protocol. However, the present view in combination with existing constructions pseudorandom generators, provides a benefit in that *entire size of the preprocessing correlation* (including the semi-honest correlation), and thus storage requirements from preprocessing to online phase, can be compressed to sublinear—*logarithmic*!—in the circuit size $|C|$.

For completeness, we describe the pairwise-triple to n-wise triple conversion protocol. This procedure is executed in parallel for every multiplication gate of the circuit C to be evaluated, on random inputs x_i^g, y_i^g for each party P_i, for each gate, in conjunction with $n(n-1)$ sets of pairwise multiplication triples. The values x_i^g, y_i^g are implicitly determined via expansion of a pseudorandom generator, allowing their description in the preprocessing to be compressed to small size.

Overall Preprocessing:

- For every multiplication gate g in C, for every pair $i \neq j \in [n]$ (order sensitive), parties P_i and P_j are given shares of a pairwise multiplication triple: $\left(a_{ij}^g, b_{ij}^g, (a_{ij}^g \cdot b_{ij}^g)\right) \in R^3$.
(Compressible to size logarithmic in $|C|$ using pseudorandom correlation generators.)

- Each party P_i is given a seed s_i to a pseudorandom generator. (Used to generate pseudorandom values $(x^g)^i, (y^g)^i \in R$ for each gate multiplication gate g, held by respective party P_i.)

Pairwise-triple to n-wise triple conversion protocol (executed by all n parties):
//Convert $n(n-1)$ pairwise triples to one n-wise multiplication triple
//Makes use of Pairwise-Triple Multiplication Subroutine, specified below

- **Inputs:** Each P_i holds (pseudorandom) x^i, y^i, and $(n-1)$ sets of shares $\left((a_{ij})^i, (b_{ij})^i, (a_{ij} \cdot b_{ij})^i\right)_{j \in [n] \setminus \{i\}}$.

- **The protocol:** Each party P_i performs the following steps.

  1. For $j \neq i$: P_i executes (in parallel) an instance of the Pairwise-Triple Multiplication subroutine together with party P_j, using input $\left(x^i, y^i, \left((a_{ij})^i, (b_{ij})^i, (a_{ij} \cdot b_{ij})^i\right)\right)$. Denote the corresponding output share as $(z_{ij})^i$.
  2. P_i defines its share of the n-wise multiplication triple to be $\left(x^i, y^i, z^i\right)$, where z^i is computed as $z^i = \sum_{j \neq i} (z_{ij})^i$.
  //The n-wise triple is for values $x := \sum_i x^i$ and $y := \sum_i y^i$, and their product xy.

Subroutine: Pairwise-triple multiplication (called by 2 parties P_i, P_j):

- **Inputs**: Indices $i, j \in [n]$ of the executing parties. 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 P_i computes $x^i - r_1^i$ and $y^i - r_2^i$ and sends to the other party P_j.
  2. Each party P_i computes $\hat{x} = x - r_1 = (x^i - r_1^i) + (x^j - r_1^j)$ and $\hat{y} = y - r_2 = (y^i - r_2^i) + (y^j - r_2^j)$. Then, each party P_i computes $z^i = r_1^i \cdot \hat{y} + r_2^i \cdot \hat{x} + (r_1 \cdot r_2)^i$. If $i < j$ then Party P_i defines $\hat{x} \cdot \hat{y} + z^i$ as its output share; otherwise, if $i > j$ then P_i defines z^i as its output share.

B Proving Correctness via ZK-FLIOP – A Concrete Protocol

In this section, we present a concrete instantiation for a ZK-FLIOP protocol (see Definition 2.4), with logarithmic number of rounds and logarithmic amount of correlated data, based on the general construction in Section 5.1 in [BBC+19]. This instantiation was also used in [BGIN19, BGIN20] in the honest majority setting. Then, we give a detailed description of how it is used in our verification protocol by having each party prove that it behaved honestly.

B.1 A ZK-FLIOP Concrete Instantiation

Let P be the prover and V be the verifier. Assume that P holds a vector $\vec{a} \in R^m$ and wishes to prove that the following statement is correct:

$$b - \sum_{k=1}^{m/2} a_{[2(k-1)+1]} \cdot a_{[2(k-1)+2]} = 0.$$

The idea behind the protocol is to split the above expression into two parts. Let $I_1, I_2 \in R^{m/2}$ be defined such that $I_1 = (\vec{a}_1, \ldots, \vec{a}_{m/2})$ and $I_2 = (\vec{a}_{m/2+1}, \ldots, \vec{a}_m)$. In addition, let g be a sub-circuit that takes a vector $\vec{a} \in R^M$ as an input and is defined as $g(\vec{a}) = \sum_{k=1}^{M/2} \left(a_{[2(k-1)+1]} \cdot a_{[2(k-1)+2]}\right)$. This implies that P wishes to prove that $b - (g(I_1) + g(I_2)) = 0$ (setting $M = m/2$). To this end, both the prover and the verifier define 1-degree polynomials $f_1, \ldots, f_{m/2}$ such that $\forall e \in [m/2]$: $f_e(1)$ is the eth input in I_1 and $f_e(2)$ is the eth input in I_2. In addition, the parties define another polynomial q defined as $q(x) = g(f_1(x), \ldots, f_{m/2}(x))$. By the definition of q, it holds that q is of degree-2 (since g is a degree-2 circuit and each f_e is a 1-degree polynomial), and that $q(1) = g(I_1)$ and $q(2) = g(I_2)$. We thus ask the prover to compute $q(1)$, $q(2)$ and $q(3)$ to hold enough points on q (note that to compute $q(3)$, the prover first compute $f_e(3)$ for each e using Lagrange interpolation).

In round 1 of the protocol, the proof π_1 will simply be $q(1)||q(2)||q(3)$. Given a random challenge $r \in R$, the verifier V can now make linear queries to the input and the proof. Specifically, V will want to test that q is defined correctly by the prover, by checking that $q(r) = g(f_1(r), \ldots, f_{m/2}(r))$. Hence, the linear queries are the Lagrange coefficients that correspond to this check. In addition, V will want to check that $b-(q(1)+q(2)) = 0$ (which is equivalent to checking that $b-(g(I_1)+g(I_2)) = 0$). This check defines another linear query to the input and proof. Note that if we reveal the queries' answers of the first test to V at this point, then this will result with linear amount of communication in the protocol below (since the verifier is emulated by multiple parties which need to communicate to reveal the answers). To solve this problem, the observation is that we can ask the prover P to prove that $q(r) - g(f_1(r), \ldots, f_{m/2}(r)) = 0$! This statement has exactly the same structure as the initial statement and the size of the input is reduced by half. We can thus continue recursively with the above process, until the prover is left with small constant amount of inputs, where the check can be done with constant cost. We remark that in the final step, there is an additional subtle issue to handle: since the verifier sees in the clear $q(r)$ and $f_1(r), f_2(r)$ (at the final step we have $m = 4$), we need to make sure that nothing is leaked. To prevent this, we add another random point to each f polynomial, which makes $f_1(r), f_2(r)$ be distributed uniformly over R. This means that the degree of each f is 2, and so the degree of q is now 4, implying that the proof in the last step consists of masked version of $f_1(0), f_2(0)$ and masked version of 6 points on q: $q(0), \ldots, q(5)$. Another optimization that we use is to perform a batch check for the second test. This is done by taking a random linear combination of all checks from all rounds and check that the result equals to 0.

Soundness error. If the prover P cheats and sends an invalid proof, then the polynomial $p(x) = q(x) - g(f_1(x), \ldots, f_{m/2})$ is not the zero-polynomial, and so the probability that $q(r) = 0$ for a random point r is $\frac{2}{|\mathbb{F}|}$ when $R = \mathbb{F}$. This follows since the degree of p is at most 2, and so by the Schwartz-Zippel Lemma, it has 2 roots when defined over a finite field. Given that there are $\log m$ rounds and that in the last round p is of degree-4, it follows that the success cheating probability is at most $\frac{4\log m}{|\mathbb{F}|}$. Observe that even if a cheating prover produces a valid proof to a false statement, the verifier might still output accept in case the random linear combination of the second test yield 0. This event happens with probability $\frac{1}{|\mathbb{F}|}$. Overall, the soundness error is therefore bounded by $\frac{4\log m+1}{|\mathbb{F}|}$.

When the working over the ring $\mathbb{Z}_{2^k}$, the verification protocol is carried-out over the extension ring $\mathbb{Z}_{2^k}/f(x)$, where f is irreducible modulo 2. As shown in [BBC$^+$19, BGIN19], if f is of degree d, then the number of roots of a polynomial of degree δ over the extension ring is $2^{(k-1)\cdot d}\delta+1$. Thus, the soundness error is bounded by $\frac{4\log m+1}{2^d}$.

B.2 Proving Correctness via zk-FLIOP in Π_{vrfy}

The above construction can be plugged as is in our verification protocol Π_{vrfy}. For completeness, we now present an explicit description of how it is used by each party to prove correctness in Step 4 of Π_{vrfy}, where the role of the verifier is jointly emulated by the parties and the dealer. Recall that in our protocol, a party P_i who holds a vector $\vec{y}$ of size $4|\mathsf{mult}|+2$ wishes to prove that $c(\vec{y})=0$, where

$$c(\vec{y}) = y_1 + y_2 - \sum_{k=1}^{2|\mathsf{mult}|} \left(y_{[2(k-1)+|W|+3]} \cdot y_{[2(k-1)+|W|+4]} \right).$$

The elements $y_1, \{y_{[2(k-1)+|W|+3]}\}_{k=1}^{2|\mathsf{mult}|}$ are known to all parties, whereas the elements $y_2, \{y_{[2(k-1)+|W|+4]}\}_{k=1}^{2|\mathsf{mult}|}$ are known to the dealer $\mathcal{D}$.

To emulate the verifier in the zk-FLIOP, we ask the prover to secret share the proof between the other parties and the dealer. This is done by having the dealer hand the prover a mask for each element in the proof, and then the prover sends the masked proof to the other parties. This is indeed a star-sharing of the proof. As star-sharing is linear, it allows both the dealer and each of the parties to query their own share of the proof, and then reconstruct the proof. The fact that each piece of information is known by all parties or by the dealer, is what guarantees that the parties will obtain the correct answers and will be able to carry-out the tests, thereby detecting any cheating (except for the soundness error computed above).

Formally, the protocol works as follows:

Initialization. The parties define:

$$M = |\mathsf{mult}| \ \text{ and } \ \rho = \log(2|\mathsf{mult}|) - 1.$$

The prover initializes a vector $\vec{z}_0 \in \mathbb{F}^{4|\mathsf{mult}|+1}$ such that: $z_0[1] = y_1 + y_2$ and $\forall k \in [4|\mathsf{mult}|] : z_0[k] = y_{k+2}$.
The parties initialize a vector $\vec{\hat{z}}_0 \in \mathbb{F}^{4|\mathsf{mult}|+1}$ such that: $\hat{z}_0[1] = y_1$ and $\forall k \in [4|\mathsf{mult}|] : \hat{z}_0[k] = y_{k+2}$ if y_{k+2} is known to the parties and 0 otherwise.
The dealer initializes a vector $\vec{\tilde{z}}_0 \in \mathbb{F}^{4|\mathsf{mult}|+1}$ such that: $\tilde{z}_0[1] = y_2$ and $\forall k \in [4|\mathsf{mult}|] : \tilde{z}_0[k] = y_{k+2}$ if y_{k+2} is known to the dealer and 0 otherwise.

Note that for each k it holds that $\vec{z}_0[k] = \vec{\hat{z}}_0[k] + \vec{\tilde{z}}_0[k]$

Finally, the dealer $\mathcal{D}$ chooses random $\tilde{\pi}_j \in \mathbb{F}^3$ for each $j \in [\rho-1]$, $\tilde{\pi}_\rho \in \mathbb{F}^5$ and $\tilde{s}_1, \tilde{s}_2 \in \mathbb{F}$, and hands it to the prover P_i.

For each round $j = 1, \dots, \rho - 1$:

- **Local Computation:**

1. The prover P_i:
(1) splits the input into two symmetric vectors:
$$I_1 = \left(\{ z_{j-1}[2(k-1)+1], z_{j-1}[2(k-1)+2] \}_{k=1}^{M} \right)$$
and
$$I_2 = \left(\{ z_{j-1}[M+2(k-1)+1], z_{j-1}[M+2(k-1)+2] \}_{k=1}^{M} \right).$$
(2) defines $2M$ degree-1 polynomials $f_1, \dots, f_{2M}$ such that $f_e(1)$ is the eth input in I_1 and $f_e(2)$ is the eth input in I_2.
(3) defines the polynomial q defined as $q(x) = g(f_1(x), \dots, f_{2M}(x))$. Note that q is a degree-2 polynomial.
2. The prover P_i locally computes $f_e(3)$ for each $e \in [2M]$, and then computes $q(1), q(2), q(3)$. Note that by definition $q(1) = g(I_1)$ and $q(2) = g(I_2)$
3. Then, the prover sets $\pi_j = (q(1), q(2), q(3))$.
4. The other parties define $\hat{I}_1$, $\hat{I}_2$ and $\hat{f}_1, \dots, \hat{f}_{2M}$ analogously to the above (i.e., using $\hat{z}_{j-1}$ instead of z_{j-1}).
The dealer defines $\widetilde{I}_1$, $\widetilde{I}_2$ and $\widetilde{f}_1, \dots, \widetilde{f}_{2M}$ analogously to the above (i.e., using $\widetilde{z}_{j-1}$ instead of z_{j-1}).

- **Communication:**

1. The prover P_i broadcasts $\hat{\pi}_j = \pi_j - \widetilde{\pi}_j$ to the other parties
2. The Dealer $\mathcal{D}$ chooses a random challenge $r_j \in \mathbb{F}$ and hands it the parties.

- **Query:**

1. For each $e \in [2M]$, the prover computes $f_e(r_j)$ via Lagrange interpolation.
2. The parties:
(1) compute $\hat{f}_e(r_j)$ for each $e \in [2M]$ via Lagrange interpolation.
(2) parse $\hat{\pi}_j$ as $\hat{q}(1), \dots, \hat{q}(3)$. Then, they locally compute $\hat{q}(r_j)$ via Lagrange interpolation.
(3) set $\hat{b}_j = \hat{z}_{j-1}[1] - (\hat{q}(1) + \hat{q}(2))$.
3. The Dealer $\mathcal{D}$:
(1) computes $\widetilde{f}_e(r_j)$ for each $e \in [2M]$ via Lagrange interpolation.
(2) sets $\widetilde{q}(1) = \widetilde{\pi}_j[1], \dots, \widetilde{q}(3) = \widetilde{\pi}_j[3]$ and then computes $\widetilde{q}(r_j)$ via Lagrange interpolation.
(3) sets $\widetilde{b}_j = \widetilde{z}_{j-1}[1] - (\widetilde{q}(1) + \widetilde{q}(2))$.

The prover defines $\vec{z}_j \in \mathbb{F}^{2M+1}$ such that $z_j[k] = f_k(r_j)$ for each $k \in [2M]$ and $z_j[0] = q(r_j)$.
The parties define $\vec{\hat{z}}_j \in \mathbb{F}^{2M+1}$ such that $\hat{z}_j[k] = \hat{f}_k(r_j)$ for each $k \in [2M]$ and $\hat{z}_j[0] = \hat{q}(r_j)$.
The dealer defines $\vec{\widetilde{z}}_j \in \mathbb{F}^{2M+1}$ such that $\widetilde{z}_j[k] = \widetilde{f}_k(r_j)$ for each $k \in [2M]$ and $\widetilde{z}_j[0] = \widetilde{q}(r_j)$.
Finally, the parties and dealer set: $L \leftarrow L/2$ and $M \leftarrow M/2$.

The last round ($j = \rho$). At the beginning of this step, $M = 2$.
The prover P_i holds $\vec{z}_{\rho-1} \in \mathbb{F}^5$, the parties hold $\vec{\hat{z}}_{\rho-1} \in \mathbb{F}^5$ and the dealer $\mathcal{D}$ holds $\vec{\tilde{z}}_{\rho-1} \in \mathbb{F}^5$.
Then:

- **Local computation:**
  1. The prover P_i:
     (1) chooses random $s_1, s_2 \in \mathbb{F}$. Then, it defines two degree-2 polynomials f_1, f_2 as: $f_1(0) = s_1,\ f_2(0) = s_2,\ f_1(1) = z_{\rho-1}[1],\ f_2(1) = z_{\rho-1}[2]$ and $f_1(2) = z_{\rho-1}[3],\ f_2(2) = z_{\rho-1}[4]$.
     (2) locally computes $f_1(3), f_2(3)$ and $f_1(4), f_2(4)$ via Lagrange interpolation.
     (3) Let q be a 4-degree polynomial defined as $q(x) = g(f_1(x), f_2(x))$.
     Then, it locally computes $q(0), \ldots, q(4)$ and sets $\pi_\rho = (f_1(0), f_2(0), q(0), \ldots, q(4))$

- **Communication:**
  1. The prover P_i broadcasts
  $$\hat{\pi}_\rho = (f_1(0) - \tilde{s}_1, f_2(0) - \tilde{s}_2, q(0) - t^i_\rho[1], \ldots, q(4) - t^i_\rho[4])$$
  to the other parties.
  2. The dealer $\mathcal{D}$ chooses random $r \in \mathbb{F}$ and hands it to the parties.

- **Query:**
  1. The parties:
     (a) parse $\hat{\pi}_\rho$ as $(\hat{f}_1(0), \hat{f}_2(0), \hat{q}(0), \ldots, \hat{q}(4))$. In addition, they define $\hat{f}_1(1) = \hat{z}_{\rho-1}[1],\ \hat{f}_2(1) = \hat{z}_{\rho-1}[2]$ and $\hat{f}_1(2) = \hat{z}_{\rho-1}[3],\ \hat{f}_2(2) = \hat{z}_{\rho-1}[4]$.
     (b) compute $\hat{f}_1(r), \hat{f}_2(r)$ and $\hat{q}(r)$ via Lagrange interpolation.
     (c) set $\hat{b}_\rho = \hat{z}_{\rho-1}[0] - (\hat{q}(1) + \hat{q}(2))$
  2. The Dealer $\mathcal{D}$:
     (a) sets: for each $k \in \{0, \ldots, 4\}$: $\tilde{q}(k) = t^i_\rho[k]$ and $\tilde{f}_0(1) = \tilde{s}_1,\ \tilde{f}_2(0) = \tilde{s}_2,\ \tilde{f}_1(1) = \tilde{z}_{\rho-1}[1],\ \tilde{f}_2(1) = \tilde{z}_{\rho-1}[2]$ and $\tilde{f}_1(2) = \tilde{z}_{\rho-1}[3],\ \tilde{f}_2(2) = \tilde{z}_{\rho-1}[4]$.
     (b) computes $\tilde{f}_1(r), \tilde{f}_2(r)$ and $\tilde{q}(r)$ via Lagrange interpolation.
     (c) sets $\tilde{b}_\rho = \tilde{z}_{\rho-1}[0] - (\tilde{q}(1) + \tilde{q}(2))$

- **Decision:**
  1. The dealer $\mathcal{D}$:
     (a) chooses random $\beta_1, \ldots, \beta_\rho \in \mathbb{F}$ and hands it to the parties.
     (b) hands the parties $\tilde{B} = \sum_{j=1}^{\rho} \beta_j \cdot \tilde{b}_j$, $\tilde{f}_1(r), \tilde{f}_2(r)$ and $\tilde{q}(r)$.
  2. The parties:
     (a) compute $\hat{B} = \sum_{j=1}^{\rho} \beta_j \cdot \hat{b}_j$ and then $B = \hat{B} + \tilde{B}$.
     (b) compute $\forall e \in [1]$: $f_e(r) = \hat{f}_e(r) + \tilde{f}_e(r)$ and $q(r) = \hat{q}(r) + \tilde{q}(r)$.
     (c) Check that:
     (1) B=0
     (2) $q(r) = f_1(r) \cdot f_2(r)$.
     If both equations hold, they output accept and otherwise, they output reject.

Communication cost. The prover broadcasts to the parties 3 elements in the first $\rho-1$ rounds and 8 elements in the last round. Overall, the communication is $3(\log(2|\mathsf{mult}|)-1)+8$ elements broadcast by the prover.

Prover's optimized computational cost. We describe two optimizations of the prover's work and assuming that the circuit is defined over a field, we provide a bound on the number of field multiplications that the prover performs. Since the prover's computational work is dominated by field multiplications, a bound on their number is a good estimate of the total computational cost of the prover.

For the first optimization, note that every f_e polynomial in the first $\rho-1$ rounds is defined by circuit values on two fixed points. These points can be chosen arbitrarily, and by choosing them to be 0 and 1, the computational cost is minimized. A second optimization is to compute the coefficient representation of each f polynomial and then evaluate it on additional points rather than working directly with Lagrange interpolation. If a polynomial $f_e(x)=ax+b$ then $b=f(0)$ and $a=f(1)-f(0)$ implying that the coefficients a,b can be computed without any field multiplication.

The field multiplications that the prover's performs in each round are part of computing the following: evaluating each f_e polynomial on one additional fixed point in step 2 of the prover's local computation, evaluating each f_e on one random point r_j in step 1 of the query, and evaluating q on three fixed points in step 2 of the prover's local computation. We estimate the number of multiplications separately for the first round and for rounds $2,\ldots,\rho-1$. The number of multiplications in the last round is a small constant and is not taken into account here.

If the proof is over a large prime field $\mathbb{F}_p$ then the additional fixed point is 2 and if it is an extension field $\mathbb{F}_{2^k}$ regarded as all polynomials of a variable z with binary coefficients, degree less than k and operations modulo an irreducible polynomial $h(z)$ then the additional point is z. In both cases, evaluating f on the extra point requires shifting a by a single bit, adding b and subtracting the modulus conditioned on the most significant bit being 1. This computation is very fast and does not require field multiplications.

Evaluating f_e on a random field element r_j requires computing ar_j+b with one multiplication per polynomial. Since there are $2|\mathsf{mult}|$ polynomials f_e in the first round and their number is halved in each round, the total number of multiplications for evaluating all polynomials on random elements is $4|\mathsf{mult}|$. If both the circuit and the proof are defined over a large prime field then each multiplication ar_j is a field multiplication. If the circuit is Boolean and the proof is over an extension field $\mathbb{F}_{2^k}$ then there are only $3|\mathsf{mult}|$ field multiplications. The reason is that in the first round the input values that define every f_e are

$$\left\{(\gamma_\ell\cdot\hat{x}_1^{g_\ell}),r_{2,i}^{g_\ell},(\gamma_\ell\cdot\hat{x}_2^{g_\ell}),r_{1,i}^{g_\ell}\right\}_{g_\ell\in\mathsf{mult}}.$$

Half of the polynomials f_e are defined by the values $r_{1,i}^{g_\ell},r_{2,i}^{g_\ell}$, which in the first round are given by wire values of the Boolean circuit and are thus either 0 or 1. It follows that in these polynomials $a\in\{0,1\}$, and computing ar_j does not require a field multiplication.

Evaluating q on three fixed points requires computing the sub-circuit g on each of these points. In the first round, the sub-circuit has half the multiplication gates of the original circuit, i.e. $|\mathsf{mult}|$ gates. However, the value of the sub-circuit on the first two fixed points 0 and 1 is already computed as part of the circuit evaluation prior to generating the proof. Therefore, the number of field multiplications required to evaluate q in the first round is $|\mathsf{mult}|$. The sub-circuit's size is halved in each following round thus requiring $3/2^i|\mathsf{mult}|$ field multiplications in the i-th round for $i = 2, \ldots, \rho - 1$ to evaluate three fixed points on q in each round, for a total of at most $4|\mathsf{mult}|$ field multiplications in all rounds.

The total number of field multiplications for the prover is therefore $8|\mathsf{mult}|$ for prime fields $\mathbb{F}_p$ and $7|\mathsf{mult}|$ for extension fields $\mathbb{F}_{2^k}$.

Verifier's (=Dealer's) computational cost. The verifier's work in each of the first $\rho - 1$ rounds is exactly to evaluate the f_e polynomials on a random field element r_j. It follows from the previous discussion on the prover's work that the cost of the verifier's work is dominated by $4|\mathsf{mult}|$ field multiplications for prime fields $\mathbb{F}_p$ and by $3|\mathsf{mult}|$ field multiplications for extension fields $\mathbb{F}_{2^k}$.