Linear Communication in Malicious Majority MPC

S. Dov Gordon*, Phi Hung Le†, and Daniel McVicker‡

Department of Computer Science, George Mason University

June 17, 2022

Abstract

The SPDZ multiparty computation protocol [17] allows n parties to securely compute arithmetic circuits over a finite field, while tolerating up to $n-1$ active corruptions. A line of work building upon SPDZ have made considerable improvements to the protocol's performance, typically focusing on concrete efficiency. However, the communication complexity of each of these protocols is $\Omega(n^2|C|)$.

In this paper, we present a protocol that achieves $\mathcal{O}(n|C|)$ communication. Our construction is very similar to those in the SPDZ family of protocols, but for one modular sub-routine for computing a verified sum. There are a handful of times in the SPDZ protocols in which the n parties wish to sum n public values. Rather than requiring each party to broadcast their input to all other parties, clearly it is cheaper to use some designated "dealer" to compute and broadcast the sum. In prior work, it was assumed that the cost of verifying the correctness of these sums is $O(n^2)$, erasing the benefit of using a dealer. We show how to amortize this cost over the computation of multiple sums, resulting in linear communication complexity whenever the circuit size is $|C| > n$.

1 Introduction

In their foundational result, Goldreich, Micali and Wigderson [20] present a secure multiparty computation (MPC) protocol for n parties to perform an arbitrary computation over their inputs, while guaranteeing privacy and correctness, even in the presence of an adversary that controls $n-1$ of the participants. In addition to the first construction tolerating $n-1$ corruptions by a semi-honest adversary, just as famously, the authors built a general compiler, using zero knowledge proofs, to provide security against an active adversary. For many years, the results were mainly of theoretical interest: the semi-honest construction requires an $O(n^2|C|)$ oblivious transfers for a circuit of size $|C|$, each (then) requiring expensive public key operations, while the zero knowledge proofs in the compiler relied upon an NP-reduction from circuit SAT to graph 3-coloring. Twenty years later, Gentry presented the first construction of fully homomorphic encryption (FHE) [19]. While it was also (then) of purely theoretical interest, the result implied, for the first time, the existence of MPC protocols with communication that grows only with the input size, remaining

*gordon@gmu.edu

†ple13@gmu.edu

‡dmcvicke@gmu.edu

independent of $|C|$. This exciting result offers and appealing tradeoff between communication and computation in MPC.[1]

In the decade since Gentry's result, cryptographers have been tremendously successful in making MPC, zero knowledge, and FHE more practical. Today, the line of work with the best concrete performance, measured in runtime, for the setting of an active adversary corrupting $t < n$ parties, uses a mix of the two approaches by relying on somewhat homomorphic encryption to preform a pre-processing step, and then leveraging the output in a more classical solution that resembles GMW (and its derivatives). Starting from Bendlin et al. [5] (BDOZ) and Damgård et al. (SPDZ) [17, 23, 2], this line of work employs somewhat homomorphic encryption for constructing multiplication triples. The reduction of MPC to the secure pre-processing of multiplication triples was first presented by Beaver [3]. These triples are secret shares of random values, (a, b, c), subject to to the constraint that $a \cdot b = c$. In the semi-honest setting, somewhat homorphic encryption provides a simple method for constructing such triples. Each party i locally samples and encrypts random a_i and b_i, and sends the ciphertexts to a desginated "dealer". This central party sums (homomorphically) the shares of a and b, and broadcasts the resulting ciphertexts. All parties homomorphically multiply, and now each holds identical encryptions of (a, b, c). Using a threshold sharing of the FHE secret key, they can locally recover a secret sharing of the triple. Thus, in the semi-honest setting, by leveraging somewhat homomorphic encryption (SHE) and relying on a designated dealer, we can reduce the total communication of GMW from $O(n^2|C|)$, which is required when performing pair-wise OT, to $O(n|C|)$, while avoiding the computational cost of a fully homomorphic evaluation of the circuit. The online phase, which uses these triples to perform the circuit evaluation, requires only $O(n|C|)$ total communication, and is extremely fast.

In the malicious setting (i.e. with an active adversary), the communication cost of the most efficient protocols, even when using SHE, remains $O(n^2|C|)$. This gap is, perhaps, somewhat surprising, because the classic GMW compiler preserves the asymptotic cost of all point-to-point communication. However, the semi-honest protocol just outlined above relies on a dealer to perform the homomorphic summation, so the communication pattern is no longer point-to-point. If we were to apply the GMW compiler, with the dealer proving correctness of its summation, we would require all parties to learn all n ciphertexts that were used in the summation. Put another way, each of the n parties must receive an $O(n)$ sized NP statement for verification. One possible way of closing this gap is to rely on *succinct proofs*: rather than proving correct summation to all parties, if we first establish (or assume) a public key infrastructure, the dealer can tailor each proof to each party, proving that the claimed summation contains a ciphertext signed by that party. This statement is independent of n, and while the circuit defining the relation – correct summation of n ciphertexts – still has size $O(n)$, a succinct proof prevents the proof size from growing and preserves the communication of the semi-honest protocol. However, there are two undesirable features of such a solution: in general these succinct proofs rely on strong, non-falsifiable assumptions, and, while a proof of correct signature verification does not change the asymptotic cost, the concrete cost of executing $n - 1$ such proofs will be quite high.

Amortizing Verified Sum: Our main technical result is a protocol for realizing a "verified sum" functionality, with low amortized communication cost. This functionality, which we repeatedly rely on in our full protocol, allows n users to securely sum n vectors, each of length $m \gg n$, using $O(n^2 + nm)$ communication. Setting $m = |C|$ allows us to perform a secure computation of circuit

[1] Asymptotically, using FHE for MPC requires no more computation than GMW. In concrete terms, however, it still introduces a hefty computational cost.

C with $O(n^2 + n|C|)$ communication.[2] This closes the gap in asymptotic performance between the semi-honest and malicious settings for $t < n$ corruptions, without relying on strong, non-falsifiable assumptions, or adding high constant overhead.

In our protocol for verified sum, each user i has input vector $(x_1^{(i)}, \ldots, x_m^{(i)})$. They begin by broadcasting a homomorphic hash of their full vector. The succinctness of the hash output ensures that they each send only a constant-sized hash to every other party, for a total communication of $O(n^2)$, independent of the vector length, m. They then send their full vectors to a central dealer to perform the summation. The dealer sends the aggregated vector to all parties, which requires $O(nm)$ communication. Finally, they verify the summation against the aggregated hash values, relying on the homomorphic property of the hash function, and abort if they find an inconsistency. This captures the main intuition of the protocol; technically, to prevent rushing attacks, we require a non-malleable commitment to the homomorphic hash. The precise details appear in Section 3. To prove that the construction is secure, we require the hash function to be collision resistant, and we require the non-malleable commitment scheme to be equivocal. To get the claimed asymptotic result, we can instantiate the hash function using the classic result of Chaum et al. [15], based on the discrete log assumption. We discuss in Section 2.8 how to set the group size so that we can support the hashing of somewhat homomorphic ciphertexts. We can instantiate the commitment scheme using any UC-secure commitment scheme, such as [14], based on linear codes.

Proving Correct Noise Bounds: The costliest portion of the SPDZ protocol is not the homomorphic operations, but rather the cost for each party to prove to every other that their ciphertexts are well formed. That is, each party must prove to all other parties that the random noise used in encrypting a_i and b_i comes from the appropriate range. Recently, Keller et al. [23] modified this zero-knowledge proof to reduce the computational complexity of verification: instead of each party i providing a proof of correctness for a_i and b_i to each of the other parties, which results in both $O(n^2)$ communication and verification time, they instead provide a single "global" proof, in which each party plays the role of prover and verifier simultaneously, in order to prove a good noise bound on the summed ciphertexts: $a = \sum a_i$ and $b = \sum b_i$. At a high level, the witnesses that each party holds for the validity of their own ciphertexts, and the challenge responses that each ought to provide can be combined, homomorphically, into a single witness for the aggregated value. This reduces computational cost of the proof to $O(n)$. However, the authors remark that $O(n^2)$ communication is still required in order to "commit" these challenges to each of the other $n - 1$ parties. Keller et al. are correct that $O(n^2)$ communication is required in order to commit each user to their ciphertexts, but, as with our proof of correct summation just described, we can amortize this cost away using the identical protocol for verified summation.

Putting the Pieces Together: Our pre-processing phase proceeds similarly to the HighGear protocol of Keller et al. [23] and the TopGear protocol of Baum et al. [2], except that we replace all instances of broadcast with our verified sum functionality; we use verified sum when we sum encrypted shares of Δ, a, b, and c, as well as when summing the commitments in the global zero knowledge proof. (Δ is an authentication tag that is used in an identical manner to that of prior work.) Due to the simplicity of this modification, our construction incurs only minimal computational overhead when compared with existing protocols, and current implementations can be easily upgraded to benefit from our asymptotic improvement in communication. We stress, however, that *recognizing* that this change would suffice for an $O(n)$ factor improvement in communication is one

[2]Here we are ignoring the security parameter for simplicity.

of our main contributions.

Other related work. Prior to the introduction of the global ZKPoK by Keller et al [23], the high cost of the ZKPoKs led to a temporary switch to an OT-based offline phase called MASCOT [22].

Another approach to prove that the ciphertext is well-formed and that the prover knows the plaintext was proposed by Pino et al [18]. The proof relies on bulletproofs [11] and it is very short. However, this technique is not compatible with the "global ZKPoK" approach used in Overdrive and TopGear. Using [18] directly would require pairwise proofs which add heavy overhead. Each party would need to verify O(n) proofs instead of one single joint proof. [18] would have the same communication complexity as Overdrive and TopGear, but the computation complexity is worse by a factor of n.

A recent line of work based on PCGs [9][6][7][8] achieves communication complexity sublinear in m (specifically, $n^2 \cdot o(m)$). Compared to our result, the PCG construction is optimized towards a small number of parties executing a large circuit, while we focus on scalability for large numbers of parties.

In the honest majority setting, many protocols achieve linear (or better) communication through a similar technique to our amortized addition verification. The batchwise multiplication verification technique of [4] efficiently computes many multiplication gates first through an semi-honest protocol, then obtains malicious security by checking the correctness of all multiplications in parallel afterwards.

2 Background

2.1 Notation.

In this paper, we denote the n parties as taking part in the computation as $P_1, \ldots, P_n$. Additionally, one party $D \in [n]$ is designated as the "dealer" and will have a unique role in several protocols. In protocol descriptions P_i will be shorthand to denote steps taken by every party $i \in [n]$. We denote the security parameter by λ and the number of triples to be generated in the offline phase by m. In our zero-knowledge protocol, we denote by λ_{snd} and λ_{zk} the statistical security parameters for soundness and zero-knowledge, respectively.

2.2 Security Model

We prove the security of our protocols in the Universal Composability (UC) model [12]. Briefly, the model describes a PPT adversary $\mathcal{A}$ controlling the corrupt parties in the protocol and a PPT environment $\mathcal{Z}$ which accesses the inputs and outputs of the honest parties and freely communicates with $\mathcal{A}$ throughout the protocol's execution.

We assume a *static malicious majority*, meaning the adversary is free to corrupt up to $n-1$ parties, chosen at the start of the computation. The protocol is shown to achieve *active security*, i.e. allows the corrupt parties to deviate from the honest protocol arbitrarily. We assume access to a functionality $\mathcal{F}_{Rand}$ which when run samples an element uniformly from a specified set and outputs said element to all parties. Our construction is designed to implement arithmetic circuits modulo a prime p.

2.3 Statistical Distance

Let A, B be discrete random variables with common range R. We define the *statistical distance* between A and B as

$$\frac{1}{2}\sum_{x\in R}|Pr[A=x]-Pr[B=x]|$$

We denote the statistical distance between A and B by $\Delta(A,B)$. We let the reader verify the following standard claim on their own.

Claim 1. *For discrete random variables A, B, if $\Delta(A,B)=\varepsilon$, then $Pr[A\neq B]\geq\varepsilon$.*

2.4 $(\varepsilon,\delta)-$independence

For random variables X, Y, let $W=\{x : \Delta(Y|X=x,Y) > \varepsilon)\}$. We say X and Y are $(\varepsilon,\delta)-$independent if $Pr[X\in W]\leq\delta$. We note that $(0,0)-$independence is the standard notion of stochastically independent random variables.

Claim 2. *Let X, Y be $(\varepsilon,\delta)-$independent random variables and let X' be a random variable that is independent of X and Y, and identically distributed to X. We claim $\Delta(X+Y,X'+Y)\leq\delta+\varepsilon(1-\delta)$*

Proof. Let R be the range of $X+Y$ and $X'+Y$:

$$\begin{aligned}
&\frac{1}{2}\sum_z|Pr[X+Y=z]-Pr[X'+Y=z]|\\
&=\frac{1}{2}\sum_z\left|\left(\sum_x Pr[X=x]\cdot Pr[Y=z-x|X=x]\right)-\left(\sum_x Pr[X'=x]\cdot Pr[Y=z-x]\right)\right|\\
&=\sum_x Pr[X=x]\cdot\frac{1}{2}\sum_z|Pr[Y=z-x|X=x]-Pr[Y=z-x]|\\
&=\sum_x Pr[X=x]\Delta(Y|X=x,Y)\\
&=\sum_{x\in W}Pr[X=x]\Delta(Y|X=x,Y)+\sum_{x\notin W}Pr[X=x]\Delta(Y|X=x,Y)
\end{aligned}$$

Since the maximum possible statistical distance between two variables is 1, and for all $x\notin W$ we have $\Delta(Y|X=x)\leq\varepsilon$ (by assumption), the final expression above becomes

$$\begin{aligned}
&\leq\sum_{x\in W}Pr[X=x]\cdot 1+\sum_{x\notin W}Pr[X=x]\cdot\varepsilon\\
&=Pr[X\in W]+Pr[X\notin W]\cdot\varepsilon\\
&=\delta+(1-\delta)\cdot\varepsilon
\end{aligned}$$

□

2.5 Commitment schemes

We define commitments using the ideal $\mathcal{F}_{\text{Com}}$ functionality shown in Figure 1. In addition to capturing the "binding" and "hiding" properties of standard commitments schemes, protocols implementing $\mathcal{F}_{\text{Com}}$ are *non-malleable* due to the UC model. Informally, this ensures that one cannot create commitments to values that are correlated with the values of another party's unopened commitments. This functionality is known to be impossible to securely implement in the plain model [13]. However, in the CRS model it can be realized, requiring $|x| + o(|x|)$ bits of communication in order to commit to a string x [14]. In practice, we may instantiate the functionality using a random oracle in order to reduce overhead.

$\mathcal{F}_{\text{Com}}$

$\mathcal{F}_{\text{Com}}$ proceeds as follows, running n parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$.

1. Upon receiving a message $(commit, sid, ssid, i, j, m)$ from P_i where $m \in 0, 1^\lambda$, record $(ssid, i, j, m)$ and send the message $(receipt, sid, ssid, i, j)$ to P_j and $\mathcal{A}$. Ignore any future commit messages.
2. Upon receiving a message $(open, sid, ssid, i, j)$ from P_i where $(ssid, i, j, m)$ was previously recorded, send the message $(open, sid, ssid, i, j, m)$ to P_j and $\mathcal{A}$. Otherwise abort.

Figure 1: Commitment ideal functionality

2.6 Random sampling

The $\mathcal{F}_{\text{Rand}}$ functionality (Figure 2) allows the parties to agree on a freshly-sampled random element from an arbitrary set. This can be implemented in $\mathcal{O}(n^2)$ through a simple commit-and-reveal protocol.

$\mathcal{F}_{\text{Rand}}$

Public parameters: an input set $\mathbb{G}$.
Functionality: Upon receiving a message $(sample)$ from all parties, sample a uniformly random element $g \xleftarrow{\$} \mathbb{G}$ and output g to all parties.

Figure 2: Random sample functionality

2.7 Ring-LWE

We use the leveled homomorphic encryption scheme proposed by Brakerski et al. [10] in our preprocessing phase. The BGV encryption scheme is built around the arithmetic of the cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/\Phi_k(X)$, where $\Phi_k(X) = \Pi_{i \in Z_k^*}(X - \omega_k^i)$ is the k^{th} cyclotomic polynomial, $\omega_k = exp(2\pi\sqrt{-1}/k) \in \mathbb{C}$ is the principal k^{th} complex root of unity. For the special case $k = 2^{x+1}$, we have $\Phi_k(X) = X^{2^x} + 1 = X^{k/2} + 1$. Let p and q be the plaintext and ciphertext modulus used in the BGV scheme. We denote $\mathcal{R}_p \equiv \mathcal{R}/p\mathcal{R}$ and $\mathcal{R}_q \equiv \mathcal{R}/q\mathcal{R}$ the plaintext and ciphertext ring respectively[3]. The security of the BGV homomorphic encryption schemes is based on the hardness of the ring learning with errors problem.

[3] p, q are not necessary prime numbers.

Definition 1. *[24] Let χ be a gaussian distribution over $\mathcal{R}_q$. Define the distribution (a, b) by sampling a, s uniformly from $\mathcal{R}_q$ and e according to χ, then set $b \leftarrow a \cdot s + e$. Define the distribution (a', b') by sampling a', b' uniformly from $\mathcal{R}_q$. The R-LWE assumption states that distinguishing between these two distributions is computationally infeasible.*

SIMD operations on plaintext slots. Let ℓ be the smallest integer such that $p^\ell \equiv 1 \mod k$, then $\Phi_k(X)$ can be split into ℓ irreducible polynomials such that $\Phi_k(X) = \Pi_{i=1}^{\ell} F_i(X)$ where all the polynomials $F_i(X)$ have degree $d = \phi(k)/\ell$ ($\phi(\cdot)$ is the Euler's totient function). This property allows us to pack ℓ plaintext messages $a_i \in F_p^d$ into a single message $a \in F_p^{\phi(k)}$ where $a \equiv a_i \mod F_i(X)$. Choosing p such that $p \equiv 1 \mod k$ enables SIMD operations on $\phi(k)$ plaintext slots.

Distributions for BGV. Let $N = \phi(k)$, p be the plaintext modulus, and $q = p_0 \cdot p_1$ be the ciphertext modulus. The following example distributions come from the SCALE-MAMBA [1] implementation of the BGV scheme.

- HWT(h, N): samples a vector of length N in $\mathcal{R}_q$ with elements chosen at random from $\{-1, 0, 1\}$ such that the number of non-zero elements is equal to h.
- ZO$(0.5, N)$: samples a vector $\mathbf{e}$ of length N in $\mathcal{R}_q$ with elements chosen from $\{-1, 0, 1\}$ such that $\Pr[e_i = -1] = \Pr[e_i = 1] = 1/4$ and $\Pr[e_i = 0] = 1/2$.
- dN(σ^2, N): samples a vector of length N in $\mathcal{R}_q$ with elements chosen according to an approximation to the discrete Gaussian distribution with variance σ^2.
- U(q, N): samples a vector of length N with elements chosen uniformly at random over the range $[-q/2, q/2)$.

Key Generation. The secret key is sampled from the HWT distribution $sk \equiv s \leftarrow HWT(h, N)$ where, typically, $h = 64$. The public key is defined as $pk \equiv (a, b)$ where $a \leftarrow U(q, N)$, and $b \leftarrow a \cdot s + p \cdot e$ for $e \leftarrow dN(\sigma^2, N)$ and p is the plaintext modulus.

Encryption/Decryption.

- $\text{Enc}_{pk}(x)$: Let $x \in \mathcal{R}_p$ be a plaintext message. The encryption of x is denoted as $\text{Enc}_{pk}(x) \equiv (c_0, c_1) \in \mathcal{R}_q^2$ where $c_0 \leftarrow b \cdot v + p \cdot e_0 + m$, $c_1 \leftarrow a \cdot v + p \cdot e_1$, $e_0, e_1 \leftarrow dN(\sigma^2, N)$.
- $\text{Dec}_{sk}(c)$: Given a ciphertext $c = (c_0, c_1) \in \mathcal{R}_q^2 \equiv Enc_{pk}(x)$, the underlying message x can be recovered by computing $x' \leftarrow c_0 - c_1 \cdot s$, then $x = x' \mod p$.

2.8 Homomorphic CRHF

Definition 2. *A* homomorphic collision-resistant hash function *over groups $(\mathbb{G}_1, +)$,$(\mathbb{G}_2, \cdot)$ is a pair of PPT algorithms (Gen, H) with the following properties:*

- Keyed-deterministic*: Gen outputs a key k, which along with H specifies a deterministic function $H_k : \mathbb{G}_1 \to \mathbb{G}_2$.*
- Homomorphic: $\forall a, b \in \mathbb{G}_1$, $H_k(a) \cdot H_k(b) = H_k(a + b)$

- Collision-resistant: *For all PPT adversaries* $\mathcal{A}$

$$Pr[k \leftarrow \mathsf{Gen} \wedge (a,b) \leftarrow \mathcal{A}^{H_k(\cdot)} \wedge a \neq b \wedge H_k(a) = H_k(b)] \leq negl(\lambda)$$

- Compressing: $|\mathbb{G}_1| > |\mathbb{G}_2|$

For simplicity we will omit the key k and write e.g. $H(a)$ when the key generation is clear from context.

in Figure 3 we provide a concrete instantiation of a homomorphic CRHF based on the discrete log assumption.

$\mathcal{H} - CRHF$

Public Parameters:
q: modulus of input group $\mathbb{Z}_q^m$
m: dimension of input group
q': dimension of output group $\mathbb{Z}_{q'}^*$. We require that $\mathbb{Z}_{q'}^*$ is a group such that the discrete log assumption holds.

Gen:
Sample m generators $g_1, \ldots, g_m$ of a subgroup of $\mathbb{Z}_{q'}^*$ of order q.
Output $g_1, \ldots, g_m$ as the key.

Hash algorithm:
Input:
$a_1, \ldots, a_m \in \mathbb{Z}_q$.
Output:
$H(a) := g_1^{a_1} \cdot \ldots \cdot g_m^{a_m}$

Homomorphism:
$H(a+b) := H(a) \cdot H(b) := g_1^{a_1+b_1} \cdot \ldots \cdot g_m^{a_m+b_m}$

Figure 3: Homomorphic collision-resistant hash function over $\mathbb{G}_1 = \mathbb{Z}_q^m$, $\mathbb{G}_2 < \mathbb{Z}_{q'}^* : |\mathbb{G}_2| = q$ [15]

H-CRHF over BGV ciphertexts

The hash function from Figure 3 is defined over an input group $\mathbb{Z}_q^m$. We use the following encoding to compute $H(\cdot)$ over a length-m vector of BGV ciphertexts $(a_1, \ldots, a_m) \in \left(R_q^2\right)^m$.

We write our vector of a's as $(a_{1,1}, a_{1,2}), \ldots, (a_{m,1}, a_{m,2})$. Since BGV addition is performed componentwise, we hash each component vector separately and concatenate the output $H(a) = H(a_{1,1}, a_{2,1}, \ldots, a_{m,1}) || H(a_{1,2}, a_{2,2}, \ldots, a_{m,2})$.

Each $a_{i,b} \in R_q$ is a polynomial in $\mathbb{Z}_q/\Phi_k(X)$, i.e. addition is componentwise over $k/2$ coefficients in $\mathbb{Z}_q$. Thus by expanding each polynomial into its vector of coefficients, we obtain for each instance of H an input of $k/2 \cdot m$ elements in $\mathbb{Z}_q$. Notably, this expansion on the input size of H has no impact on the output size.

Finally, H from Figure 3 requires that the inputs have a prime modulus. However, the BGV ciphertext modulus must be composite $q = p_0 \cdot p_1$[4]. We resolve these conflicting requirements

[4]We will perform a single multiplication on these ciphertexts, and therefore wish to support modulus switching.

by appealing to the Chinese Remainder Theorem: we generate two keys K_0 and K_1 with parameters $(p_0, m \cdot k/2, p_0')$ and $(p_1, m \cdot k/2, p_1')$ respectively (where p_0', p_1' are moduli for which the discrete log assumption is hard, and $p_0 | p_0' - 1$ and $p_1 | p_1' - 1$). The combined hash is then computed $H(a_{1,b}, \ldots, a_{mk/2,b}) = H_{K_0}(a_{1,b} \bmod p_0, \ldots, a_{mk/2,b} \bmod p_0) || H_{K_1}(a_{1,b} \bmod p_1, \ldots, a_{mk/2,b} \bmod p_1)$. By sampling p_0, p_1 from the Sophie Germain primes we guarantee $p_0' = 2p_0 + 1$ is a valid output group modulus (similar for p_1) and thus $|H(a_1, \ldots, a_m)| \approx 2|a_i|$.

2.9 Multiplication Triples

In the SPDZ online phase, the inputs of each party are shared among all parties using a simple additive secret sharing scheme. As a result, we can compute the addition gates of the circuit entirely with local operations by all parties. In order to compute multiplication gates, we use the circuit randomization technique of Beaver [3]: given sharings of inputs x, y and sharings of random $a, b, c = a \cdot b$ we locally compute shares of $(x - a)$ and $(y - b)$. Then, we open $(x - a)$ and $(y - b)$, from which we are able to locally compute shares of:

$$(x - a) \cdot (y - b) + a \cdot (y - b) + b \cdot (x - a) + c = x \cdot y$$

The bulk of the computation and communication occurs during the "preprocessing phase" in which we create these multiplication triples (a, b, c) for each multiplication gate in the circuit.

Since we are in the active security setting, corrupt parties can open their shares to arbitrary values and cause the computation to be incorrect. To circumvent this, the shares and all input values are authenticated using a global MAC key Δ, of which the parties also hold shares. To authenticate a shared value x, we compute shares of $\Delta \cdot x$, hence an *authenticated triple* is a share of the values $(a, \Delta a = \Delta \cdot a, b, \Delta b = \Delta \cdot b, c = a \cdot b, \Delta c = \Delta \cdot c)$.

Given shares of Δ and Δx we can verify that an opening of x is correct by locally computing shares of $\Delta x - x \cdot (\Delta)$. Rather than opening normally (i.e. sending all shares to P_D, who then broadcasts the reconstructed sum), we commit-then-broadcast these shares, and reject x if the result is nonzero. We generalize this technique to verify multiple openings with a single check by applying a random linear combination to these shares before opening.

$\Pi_{\text{MAC-Check}}$

Input: P_i has MAC key share $\Delta^{(i)}$ and authentication shares $(\Delta x_1)^{(i)}, \ldots, (\Delta x_k)^{(i)}$ for public opened values $x_1, \ldots, x_k$

Protocol:

1. All parties invoke $\mathcal{F}_{\text{Rand}}$ to generate k-1 linear functional $L(\cdot)$.
2. P_i invokes $\sigma^{(i)} \leftarrow L\left((\Delta x_1)^{(i)} - x_1 \cdot \Delta^{(i)}, \ldots, (\Delta x_k)^{(i)} - x_k \cdot \Delta^{(i)}\right)$.
3. P_i invokes $\mathcal{F}_{\text{Com}}$ to commit $\sigma^{(i)}$ to all other parties.
4. P_i opens previous commitment; receive $\sigma^{(j)}$ for $j \neq i$ from $\mathcal{F}_{\text{Com}}$.
5. Abort if $\sum_{i \in [n]} \sigma^{(i)} \neq 0$.

Figure 4: $\Pi_{\text{MAC-Check}}$ Protocol [16]

Even with the MAC-Check, it is still possible for the corrupt parties to produce correctly authenticated triples for which $c \neq a \cdot b$. To fix this, SPDZ uses the $\Pi_{\text{Sacrifice}}$ protocol. The protocol

works by creating two multiplication triples $(a, b, c), (a', b, c')$, sampling a random challenge t, and testing the value of $t \cdot c - c' - (t \cdot a - a') \cdot b$ for equality with zero. We see the resulting $t(c - ab) - (c' - a'b)$ is only zero if either both triples are correct or if t, which is chosen randomly *after* the triples were computed, is the specific solution to this equation.

$\Pi_{\text{Sacrifice}}$

Inputs: Authenticated triples $\left(a_j^{(i)}, a_j'^{(i)}, b_j^{(i)}, c_j^{(i)}, c_j'^{(i)}\right)_{j \in [m]}$

Use $\mathcal{F}_{\text{Rand}}$ to generate $t_1, \ldots, t_m \leftarrow \mathbb{F}_p$
For j in $[m]$ do:

1. Open $d_j \leftarrow t_j \cdot a_j^{(i)} - a_j'^{(i)}$
2. Open $e_j \leftarrow t_j \cdot c_j^{(i)} - c_j'^{(i)} - d_j \cdot b_j^{(i)}$
3. Abort if $e_j \neq 0$

Figure 5: Sacrifice Protocol [17][21]

3 Dealer-assisted Linear Summation

Our main tool is a protocol that allows parties to verifiably compute multiple sums in amortized linear communication. The protocol is described in Figure 6. We note that the protocol only guarantees correctness of the output sums, and we do not make any claims with respect to input privacy. Most importantly, its communication cost to compute m sums of n elements improves on the naive protocol used in prior work, which has cost $O(n^2 m)$ to broadcast and locally sum the values, element-by-element.

Each party begins by compressing its input vector using a homomorphic CRHF and commits its compressed input to all other parties. Next, everyone decommits their compressed inputs, and all parties locally combine them into a single hash. By the homomorphic property of the CRHF, the sum of the hashes is the same as the hash of the output vector when the sums are computed correctly. To prevent scenarios in which dishonest parties commit different inputs to different parties (and possibly collude with P_D to cause different outputs), the parties compare their locally computed sum of hashes with the sum computed by every other party. This phase in which the hashes are committed requires pairwise communication between the parties (as in prior works), but the data communicated has size independent of m. Hence, this phase incurs an $O(n^2)$ communication cost.

Now, to compute the sums, the parties send their input vectors to a designated dealer. The dealer aggregates the input vectors into a single vector of length m, and forwards these to all other parties, incurring $O(nm)$ communication cost. Using the hash computed in the previous phase, the parties can verify that the sum was computed correctly by applying the same function to the dealer's output.

Challenges in achieving modularity: While we present the protocol in this section, we do not provide a functionality or a proof of security here. Instead, our pre-processing protocol will make multiple "in-line" calls to the sub-routine described in this section, and we will prove later that the resulting protocol securely realizes the pre-processing functionality. Below, in Claim 3,

we provide a formal proof that the adversary cannot correlate the output of the summation with the honest parties' input values. We will then use this claim in later Sections when we prove that the end-to-end pre-processing protocol is secure. There are several reasons for this choice of presentation, but most importantly, we cannot claim that our protocol realizes the "natural" functionality that we would like to define, where each party provides an arbitrary input value, and all parties receive the sum. Our protocol securely realizes this functionality *only when the inputs are drawn from an appropriate distribution.* For example, when the inputs are ciphertexts, as they are in our pre-processing protocol.

$\Pi_{\text{Verified}-\text{Sum}}$

Public parameters: An input dimension m and a homomorphic collision-resistant hash function $H(\cdot)$.

Inputs: P_i has inputs $\left(x_1^{(i)}, \ldots, x_m^{(i)}\right) \in \mathbb{G}^m$

Commit phase:

1. P_i computes $y^{(i)} \leftarrow H\left(x_1^{(i)}, \ldots, x_m^{(i)}\right)$
2. P_i invokes $\mathcal{F}_{\text{Com}}$ with every other party to commit $y^{(i)}$

Open phase:

1. P_i opens its commitments from Commit phase; P_i receives $y^{(j)}$ from $\mathcal{F}_{\text{Com}}$ for every $j \neq i$
2. P_i sends $y = \sum_{i=1}^{n} y^{(i)}$ to every other party; if P_i receives $y' \neq y$ from any party, abort.
3. P_i sends $\left(x_1^{(i)}, \ldots, x_m^{(i)}\right)$ to P_D
4. P_D sends $(x_1, \ldots, x_m) := \left(\sum_{i=1}^{n} x_1^{(i)}, \ldots, \sum_{i=1}^{n} x_m^{(i)}\right)$ to all parties
5. P_i verifies that $H(x_1, \ldots, x_m) = y$, otherwise abort.

Output: $(x_1, \ldots, x_m)$

Figure 6: $\Pi_{\text{Verified}-\text{Sum}}$ Protocol.

Claim 3. *Let X be a random variable representing the sum of the honest parties' input vectors, let Z be a random variable representing the output of an execution of $\Pi_{\text{Verified}-\text{Sum}}$ where the honest input is X, and define the random variable $Y = Z - X$. For any distribution on X and any static, active PPT adversary corrupting up to $(n-1)$ parties in the $\mathcal{F}_{\text{Com}}$-hybrid model, X and Y are $(\varepsilon(\lambda), \delta(\lambda))-$independent, where ε and δ are negligible functions.*

Proof. We show this by a reduction to the collision-resistance property of H. We prove the case where there is only one honest party, but one can easily generalize the proof to the case of adversaries that corrupt fewer than $(n-1)$ parties. Assume there exists an adversary $\mathcal{A}$ and an input distribution X for which $Pr_x[\Delta(Y|X = x, Y) > \varepsilon] > \delta$. We construct the following adversary $\mathcal{B}$ with black-box access to $\mathcal{A}$:

- $\mathcal{B}$ samples x, x' according to the provided distribution X.
- $\mathcal{B}$ runs $\mathcal{A}$ on input x, receiving output z ; $\mathcal{B}$ sets $y \leftarrow z - x$.

- $\mathcal{B}$ rewinds $\mathcal{A}$ to the start of the Open phase.
- Simulating $\mathcal{F}_{\text{Com}}$, $\mathcal{B}$ equivocates its commitment to $H(x')$ and continues the Open phase normally with input x'.
- $\mathcal{B}$ receives output z'; $\mathcal{B}$ sets $y' \leftarrow z' - x'$

We claim (y, y') is a collision in H with non-negligible probability.

First, we observe that if $\mathcal{A}$ does not cause an abort, then $H(y) = H(y')$. It suffices to show that $y \neq y'$ with non-negligible probability.

We argue the distribution of (x, y) follows the distribution defined by $(X, Y|X = x)$. $x \sim X$ and $y \sim Y$ follow from $\mathcal{B}$'s sampling procedure, and the joint distribution $(X, Y|X)$ simply describes the event $X = x \wedge Y = y$ without assuming independence between X and Y. Now we argue that $y' \sim Y$ and is independent from (x, y). Because $\mathcal{F}_{\text{Com}}$ hides its input until the Open phase, y' is only dependent on x'. Since $\mathcal{B}$ samples x' independently from x, y' is also independent from x (and consequently independent from y).

Let Y' be denote the distribution of y', i.e. the same distribution as Y but independent from X. By assumption, $\Pr[\Delta(Y|X = x, Y') > \varepsilon] \geq \delta$, where the probability is taken over the choice of x. By Claim 1, the overall probability that $y \neq y'$ is greater than $\varepsilon \cdot \delta > negl(\lambda)$.

□

A Note on rewinding and UC-security: Though proofs in the UC-model do not allow for rewinding the adversary, we stress here that this is not a proof of a protocol implementing an ideal functionality. As explained above, our proof that the full pre-processing protocol achieves UC security is deferred until later sections, and the simulator there will not rewind the adversary. When we appeal to Claim 3 there, it will be to argue that our straight-line simulation is indistinguishable from a real execution.

Complexity. The communication of $\Pi_{\text{Verified}-\text{Sum}}$ consists of:

- Pairwise executions of $\mathcal{F}_{\text{Com}}$ on a single group element $\Rightarrow O(n^2)$
- Broadcast of a single group element from all parties $\Rightarrow O(n^2)$
- m group elements from each non-dealer party sent to $P_D \Rightarrow O(nm)$
- m group elements broadcast by $P_D \Rightarrow O(nm)$

In total, we obtain a communication complexity of $O(n^2 + nm)$ for $\Pi_{\text{Verified}-\text{Sum}}$ on m inputs.

Comparison with SPDZ broadcast. The original SPDZ paper also includes an optimized broadcast protocol claiming amortized $O(n)$ communication [17]. For clarity, we briefly discuss the differences with our own results.

$\Pi_{\text{Verified}-\text{Sum}}$ computes m sums with total communication $O(n^2 + nm)$. SPDZ's broadcast distributes n elements with $O(n^2)$ total communication. With the same amount of communication, our protocol outputs m *sums*, while SPDZ's broadcast outputs n *summands*. Hence, computing an equivalent output using SPDZ's technique will incur $O(mn^2)$ cost, since each of the m individual sums of n values requires $O(n^2)$ communication.

In addition, SPDZ broadcast is built assuming access to authenticated secret shares, and thus, as written, can only be used in the online phase. In contrast, $\Pi_{\text{Verified}-\text{Sum}}$ is designed as an optimization to the offline phase and requires no setup assumptions beyond the public parameters.

4 n-party ZKPoKs

As used in our construction, BGV admits certain selective failure attacks for corrupt ciphertexts which are not encrypted correctly. For example, if the noise parameter of one ciphertext is significantly outside of the expected distribution, multiplication with another ciphertext could create a product where decryption failure depends on the plaintext values. In SPDZ, this problem is solved by requiring each party to submit a zero-knowledge proof-of-knowledge that their own ciphertext is well-formed and that the they have knowledge of the plaintext.

In our setting, each party receives a ciphertext from the dealer who has no knowledge of the plaintext or encryption randomness, and thus is unable to provide a proof of knowledge. Instead, we turn to the n-party ZKPoK technique of [23, 2]. An n-party ZKPoK protocol allows multiple parties to create a single zero-knowledge proof for some relation that is defined by a function over the individual inputs — in our case, the summation. This is accomplished by having each party independently create a proof for their own input based on a single, global challenge, which can then be combined to form a proof for the summed ciphertexts. Much like the ciphertexts themselves, only the summed proof is necessary for each party to verify that the ciphertext is well-formed. Hence, we use $\Pi_{\text{Verified}-\text{Sum}}$ to optimize the zero knowledge protocols of prior works.

Intuitively, since the output of $\Pi_{\text{Verified}-\text{Sum}}$ is the same as the broadcast-then-sum steps computed in Baum et al.'s TopGear, the proofs of security are largely the same. In order to accomodate verifiers no longer knowing the individual proof statements, our security requirements are slightly relaxed from prior works. We provide our definition of n-party ZKPoKs below, with the changes between our definition and the definition of [2] highlighted.

Definition 3. *Let $\mathbb{L} \subseteq \mathbb{L}'$ be NP-languages for relations $\boldsymbol{P}$ and $\boldsymbol{P'}$, respectively. An n-party ZKPoK for $(\mathbb{L}, \mathbb{L}')$ with challenge set $\mathcal{C}$ is a tuple of algorithms* $(\text{Samp}, \text{Comm}, \text{Resp}, \text{Vrfy})$ *where* Samp *and* Comm *are PPT and* Resp *and* Vrfy *are deterministic, such that the following properties hold:*
Completeness: *The following protocol between n provers $P_1, \ldots, P_n$ and verifier V outputs 1 with probability 1:*

1. *All P_i independently execute*
$$(x_i, w_i) \leftarrow \text{Samp}$$
Verifier V receives $x_* \leftarrow \sum_i x_i$ as public input.[5]
2. *All P_i independently execute*
$$(\mathsf{comm}_i, \mathsf{state}_i) \leftarrow \mathsf{Comm}(x_i, w_i)$$
V receives $\mathsf{comm}_* \leftarrow \sum_i \mathsf{comm}_i$.
3. *V selects a random challenge $c \in \mathcal{C}$ and sends it to all P_i.*
4. *All P_i independently execute*
$$\mathsf{resp}_i \leftarrow \mathsf{Resp}(\mathsf{state}_i, c)$$
V receives $\mathsf{resp}_* \leftarrow \sum_i \mathsf{resp}_i$.

[5] Note that we have intentionally left the method by which V receives x_* unspecified. If all P_i send x_i to V directly and V computes x_* herself, then we obtain the definition of [2]; if x_* is the output of $\Pi_{\text{Verified}-\text{Sum}}$, we obtain our concrete instantiation.

5. *V executes* $b \leftarrow \mathsf{Vrfy}(x_*, \mathsf{comm}_*, c, \mathsf{resp}_*)$

6. *Protocol outputs* b.

Computational Knowledge Soundness: *Let* $(\mathcal{A}_1, \mathcal{A}_2)$ *be a pair of PPT algorithms and* $\varepsilon \in [0, 1)$. *Consider the following game:*

1. $\mathcal{A}_1$ *is run and outputs* $I \subseteq [n], x_I$ *and* $\mathsf{state}_1^{\mathcal{A}}$.

2. *Sample* $(x_j, w_j) \leftarrow \mathsf{Samp}$ *for each* P_j, $j \notin I$ *and compute* $x_* \leftarrow x_I + \sum_{j \notin I} x_j$.

3. *Execute* $(\mathsf{comm}_j, \mathsf{state}_j) \leftarrow Comm(x_j, w_j)$ *for* P_j, $j \notin I$.

4. $\mathcal{A}_2$ *on input of* $\mathsf{state}_1^{\mathcal{A}}, x_*, \{\mathsf{comm}_j\}_{j \notin I}$ *outputs* $\mathsf{state}_2^{\mathcal{A}}$ *and* comm_*.

5. *Choose* $c \in \mathcal{C}$ *uniformly at random and compute* $\mathsf{resp}_j \leftarrow Resp(\mathsf{state}_j, c)$ *for* $k \notin I$.

6. $\mathcal{A}_2$ *on input* $\mathsf{state}_2^{\mathcal{A}}, c, \{\mathsf{resp}_j\}_{j \notin I}$ *outputs* resp_*.

We say that $\mathcal{A}_1, \mathcal{A}_2$ *wins the above game with probability* $\delta > \varepsilon$ *if* $\mathsf{Vrfy}(x_*, \mathsf{comm}_*, c, \mathsf{resp}_*) = 1$. *We say that* $(\mathsf{Samp}, \mathsf{Comm}, \mathsf{Resp}, \mathsf{Vrfy})$ *is a* computational proof of knowledge *if there exists a PPT algorithm* $\mathsf{Extract}$ *which, for any fixed* I, x_I *generated by* $\mathcal{A}_1$, *with* $\{(x_j, w_j, \mathsf{state}_j, \mathsf{comm}_j)\}_{j \notin I} \leftarrow \mathsf{Samp}$ *as input and black-box access to* $\mathcal{A}_2(\mathsf{state}_1^{\mathcal{A}}, x_*, \{\mathsf{comm}_j\}_{j \notin I})$ *outputs* w_* *such that* $(x_*, w_*) \in \mathbb{L}'$ *in expected time* $poly(\lambda_{\text{snd}}/(\delta - \varepsilon))$.

Honest Verifier Zero Knowledge *There exists a PPT algorithm* $\mathcal{S}_{\text{ZK}}$ *indexed by a set* $I \subset [n]$, *which takes as input an element in the language* $\mathbb{L}$ *and a challenge* $c \in \mathcal{C}$, *and outputs tuples* $(\mathsf{comm}_i, \mathsf{resp}_i)_{i \in I}$. *We require that for all such* I *the output of* $\mathcal{S}_{\text{ZK}}$ *is statistically indistinguishable from a valid execution of the protocol.*

We now give a concrete instantiation of an n-prover ZKPoK for the language of well-formed ciphertexts. In addition to requiring knowledge of plaintext, we require that the ciphertexts are summations of no more than n individual freshly-generated ciphertexts, i.e. the norms of the plaintext and the randomness vectors are no more than n times the maximum of the sampling algorithm. In the case of a plaintext over R_p, this is $n \cdot p$; for $ZO(0.5, N)$ this is n; for $dN(\sigma^2, N)$ it depends on the choice of σ — the standard approach of setting $\sigma = 3.17$ and sampling using a discrete gaussian gives us a bound for $dN(3.17^2, N)$ summations of $20 \cdot n$ [2].

Theorem 1. *Let the languages* $\mathbb{L}, \mathbb{L}'$ *be defined as*

$$\mathbb{L} = \{(x_*, w_*) : w_* = (x_k, r_k, r'_k, r''_k)_{k \in [m]}, x_* = (\mathcal{E}nc(x_k, R_k))_{k \in [m]},$$

$$||x_k||_\infty \leq n \cdot p/2, ||r_k||_\infty, ||r'_k||_\infty \leq 20n, ||r''_k||_\infty \leq n\}$$

$$\mathbb{L}' = \{(x_*, w_*) : w_* = (x_k, r_k, r'_k, r''_k)_{k \in [m]}, x_* = (\mathcal{E}nc(x_k, R_k))_{k \in [m]},$$

$$||2 \cdot x_k||_\infty \leq 2^{\lambda_{\text{zk}}+2} \cdot n \cdot p/2, ||2 \cdot r_k||_\infty, ||2 \cdot r'_k||_\infty \leq 2^{\lambda_{\text{zk}}+2} \cdot 20n, ||2 \cdot r''_k||_\infty \leq 2^{\lambda_{\text{zk}}+2} \cdot n\}$$

Then the protocol Π_{ZK} *described in figure 7 is an* n*-prover ZKPoK.*

Π_{ZK}

Define:
λ_{snd}: statistical soundness security parameter
λ_{zk}: statistical zero-knowledge security parameter
U: number of inputs
V:

- $\lambda_{snd} + 2$ if proving a MAC key
- $(\lambda_{snd} + 2)/log_2(2N + 1)$ otherwise

$\mathcal{C}$:

- $\{0,1\}^{V \times U}$ if proving a MAC key
- $(\{X^j\} \cup \{0\})^{V \times U}$ otherwise

Input:
Private input: $\left(x_k^{(i)}, r_k^{(i)}\right)_{k \in [U]}$
Public input: $(E_k)_{k \in [U]} = \sum_{i \in [n]} \mathcal{E}nc\left(x_k^{(i)}, r_k^{(i)}\right)$

Comm:
Sample $\left(y_k^{(i)}, s_k^{(i)}\right)$ uniformly subject to $(||y_k^{(i)}||_\infty \leq 2^{\lambda_{\text{zk}}} \cdot p/2,\ ||s_{k,1}^{(i)}||_\infty, ||s_{k,2}^{(i)}||_\infty \leq 20 \cdot 2^{\lambda_{\text{zk}}}$, $||s_{k,3}^{(i)}||_\infty \leq 2^{\lambda_{\text{zk}}}$ for $k \in [V]$. If proving a MAC key, also add the constraint that $y_k^{(i)}$ must be a constant polynomial.

Compute $D_k^{(i)} = \mathcal{E}nc\left(y_k^{(i)}, s_k^{(i)}\right)$ for $k \in [V]$
$(D_k)_{k \in [V]} \leftarrow \Pi_{\text{Verified-Sum}}\left(D_k^{(i)}\right)_{i \in [n], k \in [V]}$

Chal:
Call $\mathcal{F}_{\text{Rand}}$ to generate challenge $c \leftarrow \mathcal{C}$

Resp:
Compute $\left(z_k^{(i)}\right)_{k \in [V]} = \left(y_k^{(i)}\right) + c\left(x_k^{(i)}\right)$ and $\left(t_k^{(i)}\right)_{k \in [V]} = \left(s_k^{(i)}\right) + c\left(r_k^{(i)}\right)$
$(z_k, t_k)_{k \in [V]} \leftarrow \Pi_{\text{Verified-Sum}}\left(\left(z_k^{(i)}, t_k^{(i)}\right)_{k \in [V]}\right)$

Vrfy:
Parties accept iff, for all $k \in [V]$,

- $\mathcal{E}nc(z_k, t_k) = (D + cE)_k$.
- $||z_k||_\infty \leq n \cdot 2^{\lambda+1} p/2$.
- $t_k := t_{k,1}, t_{k,2}, t_{k,3}$:
  1. $||t_{k,1}||_\infty \leq 20 \cdot n \cdot 2^{\lambda_{\text{zk}}+1}$.
  2. $||t_{k,2}||_\infty \leq 20 \cdot n \cdot 2^{\lambda_{\text{zk}}+1}$.
  3. $||t_{k,3}||_\infty \leq n \cdot 2^{\lambda_{\text{zk}}+1}$.
- (if proving a MAC key) z_k is a constant polynomial.

Figure 7: Π_{ZK} Protocol.

$\mathcal{S}_{\text{HVZK}}$

Input: Corrupt party set I, $E = \left(\mathcal{E}nc\left(x_k^{(j)}, r_k^{(j)}\right)\right)_{k\in[U]}$ for all $j \notin I$

Setup:

1. Sample a matrix c from $\mathcal{C}$
2. Sample $\left(z_k^{(j)}, t_k^{(j)}\right)_{k\in[V]}$ subject to $||z_k^{(j)}||_\infty \leq 2^{\lambda_{\text{zk}}} \cdot p/2$, $||t_{k,1}^{(j)}||_\infty, ||t_{k,2}^{(j)}||_\infty \leq 20 \cdot 2^{\lambda_{\text{zk}}}$, $||t_{k,3}^{(j)}||_\infty \leq 2^{\lambda_{\text{zk}}}$.
   If proving a MAC key, also constrain $z_k^{(j)}$ to be a constant polynomial.
3. Set $\left(D_k^{(j)}\right)_{k\in[V]} \leftarrow \left(\mathcal{E}nc\left(z_k^{(j)}, t_k^{(j)}\right)\right)_{k\in[V]} - c \cdot E$

Comm:
Send $\left(D_k^{(j)}\right)_{k\in[V]}$ to $\Pi_{\text{Verified-Sum}}$ and receive $(D_k)_{k\in[V]}$

Chal:
Emulating $\mathcal{F}_{\text{Rand}}$, send c to all $P_j \in I$

Resp:
Send $\left(z_k^{(j)}, t_k^{(j)}\right)_{k\in[V]}$ to $\Pi_{\text{Verified-Sum}}$ and receive $(z_k, t_k)_{k\in[V]}$

Vrfy:
Run the honest protocol Vrfy on the current transcript and output the result

Figure 8: $\mathcal{S}_{\text{HVZK}}$ simulation honest verifier zero knowledge (dishonest dealer).

Proof. (sketch)
(Completeness) Follows directly from the homomorphic property of the encryption scheme and the correctness of $\Pi_{\text{Verified-Sum}}$.

(Computational Knowledge Soundness) Given adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ which win the Soundness game with probability δ, an extractor $\mathcal{E}$ is constructed as follows:

1. Run $\mathcal{A}_1$ as described in the definition to generate $I, x_*, \{(x_j, w_j, \mathsf{state}_j, \mathsf{comm}_j)\}_{j\notin I}, \mathsf{state}_1^{\mathcal{A}}$.
2. Until $\mathcal{A}_2$ wins the soundness game, sample $c \xleftarrow{\$} \mathcal{C}$, compute $\mathsf{resp}_j \leftarrow \text{Resp}(\mathsf{state}_j, c)$ for all $j \notin I$, and run $\mathcal{A}_2(\mathsf{state}_2^{\mathcal{A}}, c, \{\mathsf{resp}_j\}_{j\notin I})$. Denote the winning transcript $(c, \mathsf{resp}_* = (z_k, t_k)_{k\in[V]})$.
3. For i = 1 to U :
   (a) Until $\mathcal{A}_2$ wins the soundness game, sample $c_i' \xleftarrow{\$} \mathcal{C}$ *subject to the constraint* $c_i' = c$ *in every column except the i-th and* $c_i' \neq c$, compute $\mathsf{resp}_j \leftarrow \text{Resp}(\mathsf{state}_j, c)$ for all $j \notin I$, and run $\mathcal{A}_2(\mathsf{state}_2^{\mathcal{A}}, c, \{\mathsf{resp}_j\}_{j\notin I})$. Denote the winning transcript $(c_k', \mathsf{resp}_*' = (z_{k,i}', t_{k,i}')_{k\in[V]})$
4. Solve the system of linear equations defined by $z_i - z_{i,i}' = (c - c_i') \cdot w_*^T$ for all $i \in [U]$, outputting w_*. Perform a similar process for each randomness vector.

By claim 3 $z_{k,i}, t'_{k,i} = 0$ for all $k \neq i$, hence the above process gives exactly U independent equations.

(Honest Verifier Zero Knowledge) The honest verifier zero knowledge simulator is identical to prior works, depicted in figure 8. We argue that $\Pi_{\text{Verified}-\text{Sum}}$ does not effect the simulation.

In the worst-case scenario, we assume a corrupt dealer, hence the adversary obtains the individual $(\mathsf{comm}_j, \mathsf{resp}_j)$ from each party, i.e. the same view as in prior works. The only additional messages sent by $\Pi_{\text{Verified}-\text{Sum}}$ are the CRHF of comm_j, a deterministic poly-time function of other messages in the view. Thus, the simulated honest parties' messages are indistinguishable from the honest messages in a real interaction. Finally, we claim that the indistinguishability of the corrupt parties' own messages follows from claim 3.

□

Complexity. The communication performed in Π_{Zk} consists of:

- $\Pi_{\text{Verified}-\text{Sum}}$ run on $2V$ ciphertexts.
- 1 call to $\mathcal{F}_{\text{Rand}}$

$\Pi_{\text{Verified}-\text{Sum}}$ on $2V$ elements costs $O(n^2 + nV)$. $\mathcal{F}_{\text{Rand}}$ can be implemented in $O(n^2)$ communication. Since V is bounded by the security parameter, we may ignore it as a constant term. Thus the total communication of an execution of Π_{ZK} (proving correctness of U inputs) is $O(n^2)$.

5 Distributed decryption

Distributed decryption allows the parties, each holding an additive share of the secret key, to collaboratively decrypt a ciphertext. We use the improved direct-to-sharing protocol of [17], which outputs an additive share of the plaintext to each party rather than revealing the decryption. We refer the reader to their work to find a proof of security.

Π_{DistDec}

Public input: Ciphertext (c_0, c_1)
Private input: Partial secret key $sk^{(i)}$

1. All P_i sample f_i from $\mathcal{R}_q$ with the constraint that all coefficients of f_i are $\leq 2^\lambda$ greater than the ciphertext noise bound.
2. P_D computes $x^{(D)} \leftarrow c_0 - sk^{(D)} \cdot c_1$,
   all other P_i compute $x^{(i)} \leftarrow -sk^{(i)} \cdot c_1 - f_i$
3. All P_i send $x^{(i)}$ to P_D
4. P_D computes $f_D \leftarrow \sum_{i \in [n]} x^{(i)}$

Output: Each P_i outputs f_i

Figure 9: Distributed Decryption Protocol

To facilitate running distributed decryption, we assume a trusted setup functionality $\mathcal{F}_{\text{KeyGen}}$ which creates a public/secret keypair and distributes additive shares of the secret key to all parties. This protocol can be implemented by the protocol of e.g. [25].

$\mathcal{F}_{\text{KeyGen}}$

1. Sample (pk, sk) according to BGV's Key Generation procedure.
2. Sample $sk^{(1)}, \ldots, sk^{(n-1)} \leftarrow \mathcal{R}_q$ uniformly
3. Set $sk^{(n)} \leftarrow sk - \sum_{i=1}^{n-1} sk^{(i)}$

Output: Each P_i receives $\left(pk, sk^{(i)}\right)$

Figure 10: Distributed Key Generation

Complexity

Π_{DistDec} consists of a single message in $\mathcal{R}_q$ from each party to P_D. Thus the total complexity is $O(n)$.

6 The complete preprocessing stage

Using $\Pi_{\text{Verified}-\text{Sum}}$, we construct a complete implementation of the SPDZ triple functionality. The full construction appears in Figure 11.

Theorem 2. Π_{Triple} *implements* $\mathcal{F}_{\text{Triple}}$ *in the* $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{KeyGen}}, \mathcal{F}_{\text{Com}})$*-hybrid model with UC-security against active, static adversaries corrupting up to* $(n-1)$ *parties.*

Proof. We construct a simulator $\mathcal{S}_{\text{Triple}}$ for all adversaries $\mathcal{A}$, described in figure 13, and argue that all PPT environments have negligible advantage in distinguishing between $\mathcal{S}_{\text{Triple}}$ running $\mathcal{F}_{\text{Triple}}$ and $\mathcal{A}$ running Π_{Triple}.

First, we argue that the final output is indistinguishable between the real and ideal worlds. In the ideal world, if the corrupt parties do not cause an abort then the output is a uniformly sampled triple consistent with the corrupt parties' chosen shares. In the real world, the protocol only produces output after running $\Pi_{\text{Sacrifice}}$ and $\Pi_{\text{MAC}-\text{Check}}$ on the intended output. This pair of protocols will cause an abort if the intended output is not a valid triple or not correctly authenticated. Thus to complete this claim we only need to show that the real-world protocol output has the correct distribution. From claim 3 and 2, it follows that $\forall j$ Δ, a_j, and b_j are statistically indistinguishable between the real and ideal world; $(\Delta a)_j$ and $(\Delta b)_j$ follows from the security of Π_{DDec}; the values of c_j and $(\Delta c)_j$ are fully determined by the preceding values, up to the choice of sharing among the honest parties.

Furthermore, we note that the adversary's views in the real and ideal world have identical distributions. Indeed, analyzing the simulation in Figure 13, the reader can verify that the simulator runs the protocol honestly with the adversary. (We have written out the steps explicitly only to make clear how the malicious values are extracted.)

Π_{Triple}

Key

1. $(pk, sk^{(i)}) \leftarrow \mathcal{F}_{\text{KeyGen}}$
2. Sample constant polynomial $\Delta^{(i)}$
3. $E_{\Delta}^{(i)} \leftarrow \mathcal{E}nc\left(\Delta^{(i)}\right)$
4. $E_{\Delta} \leftarrow \Pi_{\text{Verified}-\text{Sum}}\left(E_{\Delta}^{(i)}\right)$
5. Run Π_{ZK} on $(x, w) = \left(E_{\Delta}^{(i)}, \Delta^{(i)}\right)$

Factors

1. Sample $a_1^{(i)}, a_1'^{(i)}, b_1^{(i)}, \ldots, a_m^{(i)}, a_m'^{(i)}, b_m^{(i)} \leftarrow \mathcal{P}$
2. $E_{a_1}^{(i)} \leftarrow \mathcal{E}nc\left(a_1^{(i)}\right), \ldots, E_{b_m}^{(i)} \leftarrow \mathcal{E}nc\left(b_m^{(i)}\right)$
3. $\left(E_{a_1}, E_{a_1'}, E_{b_1}, \ldots, E_{a_m}, E_{a_m'}, E_{b_m}\right) \leftarrow \Pi_{\text{Verified}-\text{Sum}}\left(E_{a_1}^{(i)}, \ldots, E_{b_m}^{(i)}\right)$
4. Run Π_{ZK} on $\left(x = \left(E_{a_1}, \ldots, E_{b_m}\right), w = (a_1, \ldots, b_m)\right)$
5. $\forall j$: Compute $E_{\Delta a_j} = E_{\Delta} \cdot E_{a_j}, E_{\Delta a_j'} = E_{\Delta} \cdot E_{a_j'}, E_{\Delta b_j} = E_{\Delta} \cdot E_{b_j}$, $E_{c_j} = E_{a_j} \cdot E_{b_j}, E_{c_j'} = E_{a_j'} \cdot E_{b_j}$

Product

1. $\forall j : \widetilde{c}_j^{(i)} \leftarrow \Pi_{\text{DistDec}}\left(E_{c_j}\right)$ and $\widetilde{c}_j'^{(i)} \leftarrow \Pi_{\text{DistDec}}\left(E_{c_j'}\right)$
2. $\forall j : \widetilde{E}_{c_j}^{(i)} \leftarrow \mathcal{E}nc\left(\widetilde{c}_j^{(i)}\right)$ and $\widetilde{E}_{c_j'}^{(i)} \leftarrow \mathcal{E}nc\left(\widetilde{c}_j'^{(i)}\right)$
3. $\forall j : \left(\widetilde{E}_{c_j}, \widetilde{E}_{c_j'}\right)_{j\in[m]} \leftarrow \Pi_{\text{Verified}-\text{Sum}}\left(\widetilde{E}_{c_j}^{(i)}, \widetilde{E}_{c_j'}^{(i)}\right)_{j\in[m]}$
4. $\forall j$: Compute $E_{\Delta c_j} = E_{\Delta} \cdot \widetilde{E}_{c_j}, E_{\Delta c_j'} = E_{\Delta} \cdot \widetilde{E}_{c_j'}$

Finalize

1. $\forall j : (\Delta a_j) \leftarrow \Pi_{\text{DistDec}}\left(E_{\Delta a_j}\right)$, $(\Delta a_j') \leftarrow \Pi_{\text{DistDec}}\left(E_{\Delta a_j'}\right)$, $(\Delta b_j) \leftarrow \Pi_{\text{DistDec}}\left(E_{\Delta b_j}\right)$, $(\Delta c_j) \leftarrow \Pi_{\text{DistDec}}\left(E_{\Delta c_j}\right)$, $(\Delta c_j') \leftarrow \Pi_{\text{DistDec}}\left(E_{\Delta c_j'}\right)$
2. Run $\Pi_{\text{Sacrifice}}$ on $\left((a_j, \Delta a_j), \left(a_j', \Delta a_j'\right), (b_j, \Delta b_j), (c_j, \Delta c_j), \left(c_j', \Delta c_j'\right)\right)_{j\in[m]}$.
3. Run $\Pi_{\text{MAC}-\text{Check}}$

Output: $\left(\Delta^{(i)}, a_j^{(i)}, (\Delta a)_j^{(i)}, b_j^{(i)}, (\Delta b)_j^{(i)}, c_j^{(i)}, (\Delta c)_j^{(i)}\right)_{j\in[m]}$

Figure 11: Π_{Triple} Protocol.

$\mathcal{F}_{\text{Triple}}$

1. Receive $\Delta^{(d)}$ or $\perp$ from the Adversary
2. Sample random $\Delta \leftarrow \mathcal{R}_p$
3. Set $\Delta^{(h)} \leftarrow \Delta - \Delta^{(d)}$
4. For $j = 1$ to m do:
   (a) Receive $\left(a_j^{(d)}, (\Delta a)_j^{(d)}, b_j^{(d)}, (\Delta b)_j^{(d)}, c_j^{(d)}, (\Delta c)_j^{(d)}\right)$ or $\perp$ from the Adversary
   (b) Sample random a_j, b_j, and set $c_j := a_j \cdot b_j, (\Delta a)_j := \Delta \cdot a_j, (\Delta b)_j := \Delta \cdot b_j, (\Delta c)_j := \Delta \cdot c_j$
   (c) Set $a_j^{(h)} = a_j - a_j^{(d)}$, and define $b_j^{(h)}, c_j^{(h)}, (\Delta a)_j^{(h)}, (\Delta b)_j^{(h)}, (\Delta c)_j^{(h)}$ similarly

Output: If the Adversary inputs $\perp$, abort the functionality. Otherwise, honest parties receive shares of $\left(\Delta, a_j^{(h)}, (\Delta a)_j^{(h)}, b_j^{(h)}, (\Delta b)_j^{(h)}, c_j^{(h)}, (\Delta c)_j^{(h)}\right)_{j \in [m]}$.

Figure 12: Authenticate Multiplication Triple Generation Functionality

Next, we argue that the joint distribution over the views of the adversary-controlled parties and the output of the honest parties are indistinguishable between the real and ideal worlds. Our proof of this claim is adapted from the proof of SPDZ [17]. It follows from a reduction to the Ring-LWE problem. Intuitively, the reduction follows from the observation that the only difference between the real and ideal worlds is whether shares of the output triples are consistent with the ciphertexts sent during the protocol: in the ideal world, the functionality chooses a fresh set of output shares, which are independent of those encrypted during the simulation.

Formally, as a challenge we are given pk which is either generated according to BGV's key generation algorithm or drawn uniformly from R_q^2. Suppose we have access to an adversary $\mathcal{A}$ which can distinguish between the real and ideal worlds with non-negligible advantage ε. We then construct a new adversary $\mathcal{A}_r$ which by interacting with $\mathcal{A}$ correctly determines whether pk was generated via BGV's $KeyGen(\cdot)$ functionality or generated uniformly at random with non-negligible advantage ε, violating the R-LWE assumption. We describe the behavior of $\mathcal{A}_r$ in Figure 14

Our goal is to show that when $pk \leftarrow KeyGen(\cdot)$ the sample provided to $\mathcal{A}$ comes form the correct distribution: it is sampled from the real world distribution when $\mathcal{H} = \mathsf{REAL}$, and from the ideal world distribution when $\mathcal{H} = \mathsf{IDEAL}$. Hence $\mathcal{A}$ and $\mathcal{A}_r$ will have the same distinguishing advantage. On the other hand, if $pk \leftarrow Uniform(\cdot)$ then $\mathcal{A}_r$ provides $\mathcal{A}$ a sample from the same distribution regardless of $\mathcal{H}$. We define the distribution generated by $\mathcal{A}_r$ when $\mathcal{H} = \mathsf{REAL}$ as $\mathsf{REAL}^{\mathcal{A}_r}$, and define $\mathsf{IDEAL}^{\mathcal{A}_r}$ as the distribution when $\mathcal{H} = \mathsf{IDEAL}$.

We argue that $\mathcal{A}$'s view of $\mathsf{REAL}^{\mathcal{A}_r}$ is indistinguishable from $\mathcal{A}$'s view of Π_{Triple}. The only differences between the two protocols are those stated in step 2 of figure 14. The $\mathcal{F}_{\text{KeyGen}}$ emulation is statistically identical because the adversary never learns all shares of the additively shared fake secret key. The remaining two bullet points are workarounds for the places that Π_{Triple} uses the shares of sk, i.e. for distributed decryption. When $\mathcal{A}_r$ runs the extractor before Π_{ZK}, the protocol is rewound and only the subsequent call to Π_{ZK} ends up in the adversary's final view. By claim 3, the corrupt party's value used after the rewind is the same as the extracted value. Finally, the only remaining difference is the reliance of $\mathcal{A}_r$ on a simulated proof rather than a real proof. Therefore, when $\mathcal{H} = \mathsf{REAL}$, the distribution of the view and outputs provided to $\mathcal{A}$ is indistinguishable to that of the real world. We omit an identical argument demonstrating that $\mathsf{IDEAL}^{\mathcal{A}_r} \approx \mathcal{S}_{\text{Triple}}$.

$\mathcal{S}_{\text{Triple}}$

Let I be the set of parties corrupted by the adversary

Key

1. Emulating $\mathcal{F}_{\text{KeyGen}}$, generate a keypair (pk, sk) and secret share sk into shares $sk^{(i)}$.
2. For $i \in I$ send $sk^{(i)}$ to P_i, and store the complete sk in memory.
3. For $i \notin I$ Sample MAC keyshares $\Delta^{(i)}$. $E_{\Delta}^{(i)} \leftarrow \mathcal{E}nc\left(\Delta^{(i)}\right)$.
4. Run $\Pi_{\text{Verified}-\text{Sum}}$ with the corrupt parties, receiving $\mathcal{E}nc\left(\Delta\right)$ as output;
   Set $\widehat{\Delta} \leftarrow \mathcal{D}ec_{sk}\left(\mathcal{E}nc\left(\Delta\right)\right) - \sum_{i \notin I} \Delta^{(i)}$
5. Run Π_{ZK} as in the honest protocol.

Factors/Product

1. For $i \notin I$ sample $a_1^{(i)}, a_1'^{(i)}, b_1^{(i)}, \ldots, a_m^{(i)}, a_m'^{(i)}, b_m^{(i)}$
2. Run the **Factors** and **Product** phase from Π_{Triple} with the corrupt parties, using $a_1^{(i)}, a_1'^{(i)}, b_1^{(i)}, \ldots, a_m^{(i)}, a_m'^{(i)}, b_m^{(i)}$ as the honest inputs;
   Define $\widehat{a_1}, \widehat{b_1}, \widehat{c_1}, \ldots \widehat{a_m}, \widehat{b_m}, \widehat{c_m}$ analogously to $\widehat{\Delta}$ from step 4 of **Key**.

Finalize

1. $\forall j \in [m]$, define $\left(\Delta a\right)_j \leftarrow \mathcal{D}ec_{sk}\left(\mathcal{E}nc\left(\Delta\right) \cdot \mathcal{E}nc\left(a_j\right)\right)$ and define $\left(\Delta b\right)_j$ *and* $\left(\Delta c\right)_j$ similarly.
2. Run the Π_{DistDec} honestly with the corrupt parties. The simulator obtains:
   $(\Delta a)_j^{(i)}, (\Delta b)_j^{(i)}, (\Delta c)_j^{(i)}$, $i \notin I, j \in [m]$.
   Define $(\widehat{\Delta a})_j \leftarrow (\Delta a)_j - \sum_{i \notin I} (\Delta a)_j^{(i)}$ for all $j \in [m]$, and similar for $(\widehat{\Delta b})_j$ and $(\widehat{\Delta c})_j$.
3. Run $\Pi_{\text{Sacrifice}}$ and $\Pi_{\text{MAC}-\text{Check}}$ honestly with the corrupt parties.
4. If any of the previous steps caused an Abort, submit $\perp$ to $\mathcal{F}_{\text{Triple}}$ as the adversary's input. Otherwise submit:
   $\Big(\widehat{\Delta}, \widehat{a_1}, (\widehat{\Delta a})_1, \widehat{b_1}, (\widehat{\Delta b})_1, \widehat{c_1}, (\widehat{\Delta c})_1, \ldots, \widehat{a_m}, (\widehat{\Delta a})_m, \widehat{b_m}, (\widehat{\Delta b})_m, \widehat{c_m}, (\widehat{\Delta c})_m\Big)$.

Figure 13: $\mathcal{S}_{\text{Triple}}$ simulation.

Note that $\mathsf{IDEAL}^{\mathcal{A}_r}$ and $\mathsf{REAL}^{\mathcal{A}_r}$ are identical, aside from the output shares. Now, consider the case where $pk \leftarrow KeyGen(\cdot)$: since $\mathsf{REAL}^{\mathcal{A}_r} \approx \Pi_{\text{Triple}}$ and $\mathsf{IDEAL}^{\mathcal{A}_r} \approx \mathcal{S}_{\text{Triple}}$, $\mathcal{A}$'s probability of distinguishing between $\mathsf{REAL}^{\mathcal{A}_r}$ and $\mathsf{IDEAL}^{\mathcal{A}_r}$ is negligibly different from its probability of correctly distinguishing between Π_{Triple} and $\mathcal{S}_{\text{Triple}}$. In the other case where $pk \leftarrow Uniform(\cdot)$, we have $\mathsf{REAL}^{\mathcal{A}_r} \approx \mathsf{IDEAL}^{\mathcal{A}_r}$. Thus $\mathcal{A}_r$'s probability of correctly distinguishing the challenge "$pk \leftarrow KeyGen(\cdot)$" from "$pk \leftarrow Uniform(\cdot)$" is then

$$\frac{1}{2}Pr[\mathcal{H}' = \mathcal{H} | pk \leftarrow KeyGen(\cdot)] + \frac{1}{2}Pr[\mathcal{H}' \neq \mathcal{H} | pk \leftarrow Uniform(\cdot)]$$

$$= \frac{1}{2}(\varepsilon - negl()) + \frac{1}{2}negl()$$

which is non-negligible advantage, contradicting the assumption that BGV public keys are computationally indistinguishable from uniformly sampled elements. □

Communication complexity

$\mathcal{A}_r$

Input: $\mathcal{A}_r$ receives pk from the challenger

Main:

1. Sample $\mathcal{H} \leftarrow \{\mathsf{REAL}, \mathsf{IDEAL}\}$
2. Simulate $\mathcal{H}$ for $\mathcal{A}$, but emulate the functionalities/protocols below in the following manner
   - $\mathcal{F}_{\text{KeyGen}}$: sample random $sk^{(i)}$ for each P_i and output $(pk, sk^{(i)})$ to P_i.
   - Π_{ZK}: First run the Soundness Extractor $\mathcal{E}$ to learn the corrupt plaintext sum, then rewind and run the Π_{ZK} simulator normally.
   - $\Pi_{\text{DistDec}}(\mathcal{E}nc(x))$: Using knowledge of all $sk^{(i)}$ from $\mathcal{F}_{\text{KeyGen}}$ and x from $\mathcal{E}$:
     (a) Run Π_{DistDec} honestly, receive outputs $x^{(i)}$ on behalf of the honest parties.
     (b) From $sk^{(i)}$s and transcript of Π_{DistDec} compute corrupt parties' outputs $x^{(j)}$.
     (c) Compute $x' \leftarrow x - \sum_i x^{(i)} - \sum_j x^{(j)}$
     (d) If $\mathcal{H} = \mathsf{REAL}$: Choose an honest i; set $x^{(i)} \leftarrow x^{(i)} + x'$
3. $\mathcal{A}$ outputs $\mathcal{H}' \in \{\mathsf{REAL}, \mathsf{IDEAL}\}$, corresponding with which hybrid it believes was running.

Output: If $\mathcal{H}' == \mathcal{H}$ output "$pk \leftarrow KeyGen(\cdot)$", otherwise output "$pk \leftarrow Uniform(\cdot)$"

Figure 14: $\mathcal{A}_r$ reduction adversary for R-LWE

The communication of Π_{Triple} consists of:

- A constant number of executions of $\Pi_{\text{Verified-Sum}}$ on $O(m)$ inputs $\Rightarrow O(n^2 + nm)$
- A constant number of executions of Π_{ZK} on m inputs $\Rightarrow O(n^2)$
- $O(m)$ calls of $\Pi_{\text{DistDec}} \Rightarrow O(m) \cdot O(n)$
- m executions of $\Pi_{\text{Sacrifice}} \Rightarrow O(n^2 + nm)$
- One execution of $\Pi_{\text{MAC-Check}} \Rightarrow O(n^2)$

Thus our dominant term is $O(n^2 + nm)$, as claimed in the introduction.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1955264.

References

[1] Abdelrahaman Aly and Nigel P. Smart. Benchmarking privacy preserving scientific operations. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 509–529. Springer, Heidelberg, June 2019.

[2] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[4] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Heidelberg, August 2012.

[5] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[6] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[7] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

[8] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 244–276. Springer, Heidelberg, December 2020.

[9] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear gmw-style compiler for mpc with preprocessing. *IACR Cryptol. ePrint Arch.*, 2022:261, 2021.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[12] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[13] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[14] Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016.

[15] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 470–484. Springer, Heidelberg, August 1992.

[16] 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 Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

[17] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[18] Rafa'el del Pino, Vadim Lyubashevsky, and Gregor Seiler. Short discrete log proofs for FHE and ring-LWE ciphertexts. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 344–373. Springer, Heidelberg, April 2019.

[19] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987.

[21] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[22] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

[23] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[24] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[25] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300, 2019. `https://eprint.iacr.org/2019/1300`.