Using TopGear in Overdrive: A more efficient ZKPoK for SPDZ

Carsten Baum[1] Daniele Cozzo[2] and Nigel P. Smart[2,3]

[1] Bar Ilan University, Israel
[2] imec-COSIC, KU Leuven, Leuven, Belgium.
[3] University of Bristol, Bristol, UK.
carsten.baum@biu.ac.il, daniele.cozzo@kuleuven.be, nigel.smart@kuleuven.be

Abstract. The HighGear protocol (Eurocrypt 2018) is the fastest currently known approach to preprocessing for the SPDZ Multi-Party Computation scheme. Its backbone is formed by an Ideal Lattice-based Somewhat Homomorphic Encryption Scheme and accompanying Zero-Knowledge proofs. Unfortunately, due to certain characteristics of HighGear such current implementations use far too low security parameters in a number of places. This is mainly due to memory and bandwidth consumption constraints.

In this work we present a new approach to the ZKPoKs as introduced in the HighGear work. We rigorously formalize their approach and show how to improve upon it using a different proof strategy. This allows us to increase the security of the underlying protocols, all while maintaining roughly the same performance in terms of memory and bandwidth consumption.

1 Introduction

Multi-party computation (MPC) has turned in the last fifteen years from a mainly theoretical endeavor to one which is now practical, with a number of companies fielding products based on it. MPC comes in a number of flavours, depending on the underlying primitives (garbled circuits, secret sharing), number of parties (two or many parties), security model (passive, covert, active), and so on. In this work we will focus on n-party secret sharing-based MPC secure against a dishonest majority of parties. In this setting the most efficient protocol known is the SPDZ protocol [15] from 2012, along with its many improvements such as [14, 24, 25].

The SPDZ protocol family uses a form of authenticated secret sharing over a finite field $\mathbb{F}_p$ to perform the secure computation. The protocol is divided into two phases, an offline phase (which produces, among other things, multiplication triples) and an online phase (which consumes the multiplication triples to enable multiplication to be performed). In this work we will be focusing on the offline phase, specifically how to increase its security level while keeping or increasing the performance. The SPDZ protocol, among others, has been implemented in the SCALE-MAMBA system [2], which we refer to as a reference implementation allowing to measure and compare our contribution in practice.

To understand the difference of our approach in terms of efficiency and security we first consider the genesis of the problem we solve. The SPDZ protocol is itself based on an earlier work called BDOZ [7]. The BDOZ protocol used a form of pairwise MACs to authenticate a secret sharing amongst n-parties. At the heart of the offline phase for BDOZ is a pairwise multiplication protocol, using linearly homomorphic encryption. To ensure that active adversaries do not cheat in this phase pairwise zero-knowledge proofs are utilized to ensure active adversaries cannot deviate from the protocol without detection. In total $O(n^2)$ ZKPoKs need to be carried out per multiplication triple of BDOZ.

The main contribution of the SPDZ paper [15] over BDOZ was to replace the pairwise MACs with a global MAC. This was made possible by upgrading the linearly homomorphic encryption used in BDOZ to a limited form of Somewhat Homomorphic Encryption (SHE) based on the BGV encryption scheme [10]. This new scheme permits to reduce the number of ZK proofs per multiplication triple by a factor of n. At the same time, due to the Smart-Vercauteren SIMD packing underlying BGV plaintext spaces the overhead due to the ZKPoKs per multiplication triple was drastically reduced. This is because a single ZKPoK could be used to simultaneously prove statements for many thousands of multiplication triples.

However, BGV is a lattice based SHE scheme and ZKPoKs for these are relatively costly – due to the necessity of proving bounds on the size of plaintexts and randomness. For a single ciphertext, the basic Σ-protocol has challenge space $\{0, 1\}$ and must thus be repeated many times to achieve reasonable security levels. Furthermore, to provide zero-knowledge one needs to "blow-up" the proven bounds, thus an adversary can only be shown to prove a statement which is strictly weaker than what an honest party proves[4]. This introduces what is called the *soundness slack* between the honest proven language $\mathbb{L}$ and the adversarially proven language $\mathbb{L}'$.

To get around the problem of having to perform multiple proofs for the same ciphertext, SPDZ uses a standard amortization technique [12] to prove U statements at once, where U is the number of ciphertexts used in the protocol. This boosts the soundness security from $1/2$ to 2^{-U}, at the expense of introducing even more soundness slack, namely an additional factor of $2^{U/2}$. At the same time, [12] crucially needs to send $V = 2 \cdot U - 1$ auxiliary ciphertexts. In the implementation of this ZKPoK in the original, and subsequent works, the authors set U to be the same security level as the statistical zero-knowledge parameter $\mathsf{ZK\_sec}$.

Despite this use of amortization techniques, the ZKPoKs were for a long time considered too slow. This resulted in two new techniques based on cut-and-choose (being introduced in [14]). The first of these produced only covert security, but was highly efficient, and thus for a number of years all implementations of the SPDZ offline phase only provided covert security. The second approach of [14] provided actively secure ZKPoK which seemed asymptotically more efficient than those provided in [15], but which due to large memory requirements were impossible to implement.

This inability to provide efficient actively secure offline phased based on SHE led to a temporary switch to an Oblivious Transfer-based offline phase, called MASCOT [24]. However, in 2018 Keller et al. [25] introduced the Overdrive suite of offline protocols. There, they revisited the original SPDZ offline phase and showed two interesting ways how to optimize it. Firstly, in so-called Low Gear, for a small number of parties the original pairwise ZKPoKs of the BDOZ methodology could be more efficient than that of SPDZ. They observe that an $O(n^2)$ algorithm can beat an $O(n)$ algorithm for small values of n and in addition improve the parameters of the SHE. This was partially also enabled by using the same SIMD packing in BDOZ as was being used for SPDZ.

The second variant of Overdrive, called High Gear, works for larger values of n. Here, the original SPDZ ZKPoK was revisited and tweaked. Instead of at each iteration each party proving a statement to each other party, the parties proved a single joint statement. Thus the n parties act as a single proving entity for a joint statement of their secret inputs and an accumulated ciphertext. This did *not* provide an improvement in *communication efficiency*, but it did make the *computational* costs a factor of n smaller.

In the SCALE-MAMBA system (as of v1.2 in Nov 2018) only the High-Gear variant of Overdrive is implemented for the case of dishonest majority MPC, even when $n = 2$. However, like all prior work the system adopts $U = \mathsf{ZK\_sec}$, and thus achieves the same soundness security $\mathsf{Snd\_sec}$ as zero-knowledge indistinguishability. This is neither as efficient, nor as secure, as one would want for two combined reasons.

1. The zero-knowledge security $\mathsf{ZK\_sec}$ is related to a statistical distance, whereas the soundness security $\mathsf{Snd\_sec}$ is related to the probability that an adversary can cheat in an interactive protocol. A low value for $\mathsf{Snd\_sec}$ is rather acceptable than a low value for $\mathsf{ZK\_sec}$.
2. The practical complexity of the protocol, in particular the memory and computational consumption, is dominated by $\mathsf{Snd\_sec}$. It turns out that $\mathsf{ZK\_sec}$ has very virtually no effect on the overall execution time of the offline phase, for large values of p.

It is for this reason that [25] gives performance metrics for 40, 64 and 128-bit active security, and why v1.2 of SCALE-MAMBA utilizes only 40-bits of security for $\mathsf{Snd\_sec}$ and $\mathsf{ZK\_sec}$, since the execution time is highly dependent on $\mathsf{Snd\_sec}$.

Our Contribution. We first formalize the type of statement which the Overdrive ZKPoK tries to prove. The original treatment in [25] is relatively intuitive. We formalize the statement by presenting a generalization of

[4] There do exist ZKPoKs for lattice-based primitives which prove exact bounds. Unfortunately, their computation and communication overhead makes them no match in practice for protocols having soundness slack.

standard Zero Knowledge-protocols to what we call an n-party Zero Knowledge-protocol. The formalization is tailored at the use of such proofs in preprocessing.

We then present a modified ZKPoK for the HighGear variant of Overdrive, which we denote TopGear. It treats the soundness security $\mathsf{Snd\_sec}$ and the zero-knowledge security $\mathsf{ZK\_sec}$ separately. This ZKPoK in its unamortized variant (i.e. only proving one statement at a time) uses a non-binary challenge space in a similar way as was done in [8]. Hence we obtain a challenge space of $2 \cdot N + 1$ in the "base" ZKPoK (where N is the ring dimension). We then amortize this base ZKPoK by proving U statements in parallel using a technique from [3]. This enables us to achieve an arbitrary knowledge soundness of $\mathsf{Snd\_sec}$ by selecting the number of auxiliary ciphertexts V such that $V \geq (\mathsf{Snd\_sec} + 2)/\log_2(2 \cdot N + 1)$.

Since N is often $32768 = 2^{15}$, we are able to achieve a high soundness security, with a low value of V, e.g. we can obtain 128 bits of soundness security by setting V to be 8. Thus we can use a smaller amount of amortization than [25], and thus a smaller memory footprint and bandwidth for the same level of $\mathsf{ZK\_sec}$. Alternatively, we can select higher values of $\mathsf{ZK\_sec}$, if so desired, as this has little impact on the overall performance.

However, this leads to a change in the soundness slack. This slack mainly results from needing $\mathsf{ZK\_sec}$ to be larger to ensure zero-knowledge. In signature schemes based on lattices, such as BLISS, this issue is usually dealt with using rejection sampling, e.g. [18, 19, 26]. As the slack will be removed due to the processing that happens after the ZKPoK is executed (during a modulus switch operation in the SHE scheme), we start with a simpler yet more efficient technique to achieve zero-knowledge called "noise drowning". At the same time, we still generally obtain a smaller slack and thus better SHE parameters due to the change of the ZKPoK from [12] to [3]. An additional, different slack in our work is due to the use of the larger challenge space of [8]. Namely, their technique (and therefore our ZKPoK) does not allow to extract an "exact" preimage, but only that of a related ciphertext. We show that this can be corrected easily in the case of SPDZ preprocessing.

Other Related Work. Multiple techniques have been introduced to cope with the problem of amortized ZKPoK for lattice-based primitives. Multiple subsequent works [4, 13, 16] have introduced more and more efficient proofs of knowledge which have small (down to linear in $\mathsf{Snd\_sec}$) slack. Unfortunately, all of these require U to be in the multiple of 1000s to be efficient, which is far from practically feasible. Later, Baum and Lyubashevsky [5] showed how to build small-slack proofs for realistic sizes of U, though their idea was limited to structured lattices. The problem was only recently resolved in [3] which we therefore use as a building block in our work.

Another recent methodology to perform such ZKPoKs as needed in this paper is given in [17] based on bullet proofs. These give very short proofs but are not competitive with the approach in this paper. Firstly, we use amortization over the number of parties to produce a joint proof whereas the direct application of [17] would require pairwise proofs which would not scale well with the number of parties. The paper [17] also concentrates on the case of "small moduli q" of the SHE plaintext space. In our case, we easily need q to have a size of say 500 bits. Apart from the problem of constructing a secure DLP group of a given order at this size (leading to probably needing to use finite fields, or a group size larger than q to deal with integer overflow), this also leads to very large computational expenses. To sign and verify a proof from [17] requires *at least* $12 \cdot \phi(m) \cdot \log_2 q$ exponentiations. In our case this would equate to around 2^{27} exponentiations for each proof.

2 Preliminaries

In this section we provide a recap of the the BGV encryption scheme [10] as well as those building blocks of SPDZ that are used in combination with it. Most of the details about BGV can be found in [10, 20–22], although we will employ a variant which supports circuits of multiplicative depth one only.

Notation. We generally let P_i denote a party, of which there are n in total. Those parties are modeled as probabilistic polynomial time (PPT) Turing machines. We let $[n]$ denote the interval $[1, \ldots, n]$. If M is a matrix then we write $M^{(r,c)}$ for the entry in the r-th row and c-th column. Vectors are (usually) written in bold, and their elements in non-bold with a subscript, thus $\boldsymbol{x} = (x_i)_{i\in[n]}$. We will write $\boldsymbol{x}[i]$ to denote the i-th element in the vector $\boldsymbol{x}$. All modular reduction operations $x \pmod q$ will be to the centered interval $(-q/2, q/2]$.

We let $a \leftarrow X$ denote randomly assigning a value a from a set X, where we assume a uniform distribution on A. If A is an algorithm, we let $a \leftarrow A$ denote assignment of the output, where the probability distribution is over the random tape of A; we also let $a \leftarrow b$ be a shorthand for $a \leftarrow \{b\}$, i.e. to denote normal variable assignment. If $\mathcal{D}$ is a probability distribution over a set X then we let $a \leftarrow \mathcal{D}$ denote sampling from X with respect to the distribution $\mathcal{D}$.

We will make use of the following standard lemma in a number of places

Lemma 2.1. *Let $\mathcal{D}$ be any distribution bounded whose values are bounded by B. Then the distributions $\mathcal{D} + (0, \cdots, B')$ (by which we mean the distribution obtained from sampling from the two distributions and adding the result) is statistically close to the uniform distribution $\mathcal{U}(0, \cdots, B')$, with statistical distance bounded by $\frac{B}{B'}$.*

SPDZ Secret Sharing. This SPDZ protocol [15] processes data using an authenticated secret sharing scheme defined over a finite field $\mathbb{F}_p$, where p is prime. The secret sharing scheme is defined as follows: Each party P_i holds a share $\alpha_i \in \mathbb{F}_p$ of a global MAC key $\alpha = \sum_{i\in[n]} \alpha_i$. A data element $x \in \mathbb{F}_p$ is held in secret shared form as a tuple $\{x_i, \gamma_i\}_{i\in[n]}$, such that $x = \sum_i x_i$ and $\sum \gamma_i = \alpha \cdot x$. We denote a value x held in such a secret shared form as $\langle x \rangle$. The main goal of the SPDZ offline phase is to produce random triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that $c = a \cdot b$. If we wish to denote the specific value on which γ_i is a MAC share then we write $\gamma_i[x]$.

The Rings. The BGV encryption scheme, as we will use it, is built around the arithmetic of the cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$, where $\Phi_m(X)$ is the m-th cyclotomic polynomial. For an integer $q > 0$, we denote by $\mathcal{R}_q$ the ring obtained as reduction of $\mathcal{R}$ modulo q. We take m to be a power of two, $m = 2^{n+1}$ and hence $\Phi_m(X) = X^N + 1$ where $N = 2^n$. Elements of $\mathcal{R}$ (resp. $\mathcal{R}_q$) can either be thought of as polynomials (of degree less than N) or as vectors of elements (of length N).

The canonical embedding of $\mathcal{R}$ is the mapping of $\mathcal{R}$ into $\mathbb{C}^{\phi(m)}$ given by $\sigma(x) = (x(\zeta_m^i))_{i\in[n]}$, where we think of x as a polynomial. We are interested in two norms of elements x in $\mathcal{R}$ (resp. $\mathcal{R}_q$). For the ∞-norm in the standard polynomial embedding we write $\|x\|_\infty$, whereas the ∞-norm in the canonical embedding we will write as $\|x\|_\infty^{\mathsf{can}} = \|\sigma(x)\|_\infty$. By standard inequalities we have $\|x \cdot y\|_\infty^{\mathsf{can}} \le \|x\|_\infty^{\mathsf{can}} \cdot \|y\|_\infty^{\mathsf{can}}$, $\|x\|_\infty^{\mathsf{can}} \le \|x\|_1$, $\|x\|_\infty^{\mathsf{can}} \le \phi(m) \cdot \|x\|_\infty$ and $\|x\|_\infty \le \|x\|_\infty^{\mathsf{can}}$; with the last two inequalities holding due to our specific choice of cyclotomic ring. Such norms can also be employed on elements of $\mathcal{R}_q$ by using the standard (centered) embedding of $\mathcal{R}_q$ into $\mathcal{R}$.

We will use the following two facts in a number of places.

Lemma 2.2. *Let m be a power of two. Thhen $\|2 \cdot (X^i - X^j)^{-1} \pmod{\phi_m(X)}\|_\infty \le 1$ for all $0 \le i, j < 2 \cdot N$.*

Proof. Given in [8].

Lemma 2.3. *In the ring $\mathcal{R}$ defined by $\Phi_m(X)$ with m a power of two we have that for all $a \in \mathcal{R}$ that $\|a \cdot X^i\|_\infty = \|a\|_\infty$.*

Proof. This follows as X^i acts as a shift operation, with the wrap-around modulo $\phi(m)$ simply negating the respective coordinate.

Distributions as used in BGV. Following [22, Full version, Appendix A.5] and [2] we use different distributions to define the BGV scheme, all of which produce vectors of length N which we consider as elements in $\mathcal{R}$.

- $\mathsf{HWT}(h, N)$: This generates a vector of length N with elements chosen at random from $\{-1, 0, 1\}$ subject to the condition that the number of non-zero elements is equal to h.

- $\mathsf{ZO}(0.5, N)$: This generates a vector of length N with elements chosen from $\{-1, 0, 1\}$ such that the probability of each coefficient is $p_{-1} = 1/4$, $p_0 = 1/2$ and $p_1 = 1/4$. Thus if $x \leftarrow \mathsf{ZO}(0.5, N)$ then $\|x\|_\infty \leq 1$.
- $\mathsf{dN}(\sigma^2, N)$: This generates a vector of length N with elements chosen according to an approximation to the discrete Gaussian distribution with variance σ^2.
- $\mathsf{RC}(0.5, \sigma^2, N)$: This generates a triple of elements (r_1, r_2, r_3) where r_3 is sampled from $\mathsf{ZO}_s(0.5, N)$ and r_1 and r_2 are sampled from $\mathsf{dN}_s(\sigma^2, N)$.
- $\mathsf{U}(q, N)$: This generates a vector of length N with elements generated uniformly modulo q in a centred range. Thus $x \leftarrow \mathsf{U}(q, N)$ implies $\|x\|_\infty \leq q/2$.

Following prior work on SPDZ we select $\sigma = 3.17$ and hence we can approximate the sampling from the discrete Gaussian distribution using a binomial distribution, as is done in NewHope [1]. In such a situation an element $x \leftarrow \mathsf{dN}(\sigma^2, N)$ is gauranteed to satisfy $\|x\|_\infty \leq 20$.

The Two-Level BGV Scheme. We consider a two-leveled homomorphic scheme, given by the algorithms {KeyGen, Enc, SwitchMod, Dec}. The plaintext space is the ring $\mathcal{R}_p$, for some prime modulus p, which is the same modulus used to define the SPDZ secret sharing scheme. The algorithms are parametrized by a *computational* security parameter κ and are defined as follows. First we fix two moduli q_0 and q_1 such that $q_1 = p_0 \cdot p_1$ and $q_0 = p_0$, where p_0, p_1 are prime numbers. Encryption generates level one ciphertexts, i.e. with respect to the largest modulo q_1, and level one ciphertexts can be moved to level zero ciphertexts via the modulus switching operation. We require $p_1 \equiv 1 \pmod{p}$ and $p_0 - 1 \equiv p_1 - 1 \equiv 0 \pmod{p}$. The first condition is to enable modulus switching to be performed efficiently, whereas the second is to enable fast arithmetic using Number Theoretic Fourier Transforms.

The algorithms of the BGV scheme are then as follows:

- $\underline{\mathsf{KeyGen}(1^\kappa)}$: The secret key $\mathfrak{sk}$ is randomly selected from a distribution with Hamming weight h, i.e. $\underline{\mathsf{HWT}(h, N)}$, much as in other systems, e.g. HELib [23] and SCALE [2]. The public key, $\mathfrak{pk}$, is of the form (a, b), such that $a \leftarrow \mathsf{U}(q_1, N)$ and $b = a \cdot \mathfrak{sk} + p \cdot \epsilon \pmod{q_1}$, where $\epsilon \leftarrow \mathsf{dN}(\sigma^2, N)$. This algorithm also outputs the relinearization data $(a_{\mathfrak{sk},\mathfrak{sk}^2}, b_{\mathfrak{sk},\mathfrak{sk}^2})$ [11], where

$$a_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{U}(q_1, N) \quad \text{and} \quad b_{\mathfrak{sk},\mathfrak{sk}^2} = a_{\mathfrak{sk},\mathfrak{sk}^2} \cdot \mathfrak{sk} + p \cdot r_{\mathfrak{sk},\mathfrak{sk}^2} - p_1 \cdot \mathfrak{sk}^2 \pmod{q_1},$$

  with $r_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{dN}(\sigma^2, N)$.
- $\underline{\mathsf{Enc}(m, \boldsymbol{r}; \mathfrak{pk})}$: Given a plaintext $m \in \mathcal{R}_p$, and randomness $\boldsymbol{r} = (r_1, r_2, r_3)$ chosen from $\mathsf{RC}(0.5, \sigma^2, n)$, i.e. $r_1, r_2 \leftarrow \mathsf{dN}(\sigma^2, N)$ and $r_3 \leftarrow \mathsf{ZO}(0.5, N)$, this algorithm sets

$$c_0 = b \cdot r_3 + p \cdot r_1 + m \pmod{q_1}, \quad c_1 = a \cdot r_3 + p \cdot r_2 \pmod{q_1}.$$

  Hence the initial ciphertext is $\mathfrak{ct} = (1, c_0, c_1)$, where the first index denotes the level (initially set to be equal to one). If the level ℓ is obvious we drop it in future discussions and refer to the ciphertext as an element in $\mathcal{R}_{q_\ell}^2$.
- $\underline{\mathsf{SwitchMod}((1, c_0, c_1))}$: We define a modulus switching operation which allows us to move from a level one to a level zero ciphertext, *without altering* the plaintext polynomial, that is

$$(0, c_0', c_1') \leftarrow \mathsf{SwitchMod}((1, c_0, c_1)), \quad c_0', c_1' \in \mathcal{R}_{q_0}.$$

  The effect of this operation is also to scale the noise term (see below) by a factor of $q_0/q_1 = 1/p_1$.
- $\underline{\mathsf{Dec}((c_0, c_1); \mathfrak{sk})}$: Decryption is obtained by switching the ciphertext to level zero (if it is not already at level zero) and then decrypting $(0, c_0, c_1)$ via the equation $(c_0 - \mathfrak{sk} \cdot c_1 \pmod{q_0}) \pmod{p}$, which results in an element of $\mathcal{R}_p$.

Homomorphic Operations. Ciphertexts at the same level ℓ can be added,

$$(\ell, c_0, c_1) \boxplus (\ell, c_0', c_1') = (\ell, (c_0 + c_0' \pmod{q_\ell}), (c_1 + c_1' \pmod{q_\ell})),$$

with the result being a ciphertext, which encodes a plaintext that is the sum of the two initial plaintexts. Ciphertexts at level one can be multiplied together to obtain a ciphertext at level zero, where the output ciphertext encodes a plaintext which is the product of the plaintexts encoded by the input plaintexts. We do not present the method here, although it is pretty standard consisting of a modulus-switch, tensor-operation, then relinearization (which we carry out in this order). We write the operation as

$$(1, c_0, c_1) \odot (1, c_0', c_1') = (0, c_0'', c_1''), \quad \text{with} \quad c_0'', c_1'' \in \mathcal{R}_{q_0},$$

or more simply as $(c_0, c_1) \odot (c_0', c_1') = (c_0'', c_1'')$ as the levels are implied.

Ciphertext Noise. The noise term associated with a ciphertext is the value $\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}}$. To derive parameters for the scheme we need to maintain a handle on this value. The term is additive under addition and is roughly divided by p_1 under a modulus switch. For the tensoring and relinearization in multiplication the terms roughly multiply. A ciphertext at level zero will decrypt correctly if we have $\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty \le q_0/2$, which we can enforce by requiring $\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}} \le q_0/2$.

We would like a ciphertext (adversarially chosen or not) to decrypt correctly with probability $1 - 2^{-\epsilon}$. In [22] ϵ is chosen to be around -55, but the effect of ϵ is only in producing the following constants: we define e_i such that $\mathsf{erfc}(e_i)^i \approx 2^{-\epsilon}$ and then we set $\mathfrak{c}_i = e_i^i$. This implies that $\mathfrak{c}_1 \cdot \sqrt{V}$, is a high probability bound on the canonical norm of a ring element whose coefficients are selected from a distribution with variance V, while $\mathfrak{c}_2 \cdot \sqrt{V_1 \cdot V_2}$ is a similar bound on a product of elements whose coefficient are chosen from distributions of variance V_1 and V_2 respectively.

With probability much greater than $1 - 2^{-\epsilon}$ the "noise" of an honestly generated ciphertext (given honestly generated keys) will be bounded by

$$\begin{aligned}
\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}} &= \|((a \cdot \mathfrak{sk} + p \cdot \epsilon) \cdot r_3 + p \cdot r_1 + m - (a \cdot r_3 + p \cdot r_2) \cdot \mathfrak{sk}\|_\infty^{\mathsf{can}} \\
&= \|m + p \cdot (\epsilon \cdot r_3 + r_1 - r_2 \cdot \mathfrak{sk})\|_\infty^{\mathsf{can}} \\
&\le \|m\|_\infty^{\mathsf{can}} + p \cdot \left(\|\epsilon \cdot r_3\|_\infty^{\mathsf{can}} + \|r_1\|_\infty^{\mathsf{can}} + \|r_2 \cdot \mathfrak{sk}\|_\infty^{\mathsf{can}}\right) \\
&\le \phi(m) \cdot p/2 \\
&\qquad + p \cdot \sigma \cdot \left(\mathfrak{c}_2 \cdot \phi(m)/\sqrt{2} + \mathfrak{c}_1 \cdot \sqrt{\phi(m)} + \mathfrak{c}_2 \cdot \sqrt{h \cdot \phi(m)}\right) \\
&= B_{\mathsf{clean}}.
\end{aligned}$$

Recall this is the average case bound on the noise of honestly generated ciphertexts. In the preprocessing ciphertexts can be adversarially generated, and determining (and ensuring) a worst case bound on the resulting ciphertexts is the main focus of the HighGear and TopGear protocols.

Distributed Decryption. The BGV encryption scheme supports a form of distributed decryption, which is utilized in the SPDZ offline phase. A secret key $\mathfrak{sk} \in \mathcal{R}_q$ can be additively shared amongst n parties by giving each party a value $\mathfrak{sk}_i \in \mathcal{R}_q$ such that $\mathfrak{sk} = \mathfrak{sk}_1 + \ldots + \mathfrak{sk}_n$. We assume, as is done in most other works on SPDZ, that the key generation phase, including the distribution of the shares of the secret key to the parties, is done in a trusted setup.

To perform a distributed decryption of a ciphertext $\mathfrak{ct} = (c_0, c_1)$ at level zero, each party computes $d_i \leftarrow c_0 - c_1 \cdot \mathfrak{sk}_i + p \cdot R_i \pmod{q_0}$ where R_i is a uniformly random value selected from $[0, \ldots, 2^{\mathsf{DD\_sec}} \cdot B/p]$ where B is an upper bound on the norm $\|c_0 - c_1 \cdot \mathfrak{sk}\|_\infty$. The values d_i are then exchanged between the players and the plaintext is obtained from $m \leftarrow (d_1 + \ldots + d_n \pmod{q_0}) \pmod{p}$. The statistical distance between the distribution of the *coefficients* of d_i and uniformly random elements of size $2^{\mathsf{DD\_sec}} \cdot B$ is bounded by $2^{-\mathsf{DD\_sec}}$ by Lemma 2.1.

To ensure valid decryption we need the value of q_0 to satisfy $q_0 > 2 \cdot (1 + n \cdot 2^{\mathsf{DD\_sec}}) \cdot B$ instead of $q_0 > 2 \cdot B$ for a scheme without distributed decryption, which implies parameter growth in the BGV scheme when using this distributed decryption procedure.

3 n-Prover Zero Knowledge-Protocols

The Overdrive ZKPoK is an $n+1$ party Σ-protocol between n provers and one verifier[5]. Unlike traditional proofs of knowledge, there is a difference between the language used for completeness and the language that the soundness guarantees; much like the protocols considered in [9, Definition 2.2]. As the Overdrive paper does not formalize such proofs, our first contribution is to do precisely this. We give a generalized treatment of such proofs beyond regular Σ-protocols.

Let Samp be a PPT algorithm which, on input $n, i \in \mathbb{N}, 0 < i \leq n$ outputs a pair of values x_i, w_i where we consider x_i as the public and w_i as the private value. We require that if for all $i \in [n]$ we sample $(x_i, w_i) \leftarrow \mathsf{Samp}_n(i)$, then a given predicate $\mathbf{P}$ always holds, i.e. we have that $\mathbf{P}(x_1, \ldots, x_n, w_1, \ldots, w_n) = 1$. The predicate $\mathbf{P}$ defines a language $\mathbb{L}$ via the binary relation on the pairs $(\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{w} = (w_1, \ldots, w_n))$.

Consider a set of n provers $\{P_1, \ldots, P_n\}$, each with private input w_i and public input x_i. The provers wish to convince a verifier V (that could be one or all of the provers) that, for the public values $x_1, \ldots, x_n$ they know $w_1, \ldots, w_n$ such that $\mathbf{P}$ holds. The guarantee provided by our proof is, however, only that $\mathbf{P}'(x_1, \ldots, x_n, w_1, \ldots, w_n) = 1$ for some second language $\mathbb{L}'$, defined by a predicate $\mathbf{P}'$, with $\mathbb{L} \subseteq \mathbb{L}'$. This is still sufficient for the preprocessing of SPDZ.

Definition 3.1. *An n-party ZKPoK-protocol with challenge set $\mathcal{C}$ for the languages $\mathbb{L}, \mathbb{L}'$ and sampler* Samp_n *is defined as a tuple of algorithms* $(\mathsf{Comm}, \mathsf{Resp}, \mathsf{Verify})$*. While* Comm *is a PPT algorithm, we assume that* $\mathsf{Resp}, \mathsf{Verify}$ *are not. The verifier V will have input $x_1, \ldots, x_n$. The protocol is executed in the following four phases:*

1. *Each prover P_i* independently *executes the algorithms*
$$(\mathsf{comm}_i, \mathsf{state}_i) \leftarrow \mathsf{Comm}(x_i, w_i)$$
and sends (comm_i) *to the verifier.*
2. *The verifier selects a challenge value $c \in \mathcal{C}$ and sends it to each prover.*
3. *Each prover P_i, again independently, runs the algorithm*
$$\mathsf{resp}_i \leftarrow \mathsf{Resp}(\mathsf{state}_i, c)$$
and sends resp_i *to the verifier.*
4. *The verifier accepts if* $\mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}_i, x_i\}_{i \in [n]}, c) = \mathsf{true}$.

Such a protocol should satisfy the following three properties

- **Correctness:**
If all P_i, each on input $(x_i, w_i) \leftarrow \mathsf{Samp}_n(i)$ honestly follow the protocol, then an honest verifier will accept with probability one.
- **Computational Knowledge Soundness:** *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a pair of PPT algorithms and $\epsilon \in [0, 1)$. Consider the following game:*
  1. *$\mathcal{A}_1$ outputs $I \subseteq [n]$ and* $\mathsf{state}_1^{\mathcal{A}}$.
  2. *Choose $(x_j, w_j) \leftarrow \mathsf{Samp}_n(j)$ honestly for each $P_j, j \notin I$.*
  3. *$\mathcal{A}_1$ on input $\mathsf{state}_1^{\mathcal{A}}, \{x_j\}_{j \notin I}$ outputs $\mathsf{state}_2^{\mathcal{A}}, \{x_i\}_{i \in I}$.*
  4. *Compute $(\mathsf{comm}_j, \mathsf{state}_j) \leftarrow \mathsf{Comm}(x_j, w_j)$ for $j \notin I$.*
  5. *$\mathcal{A}_2$ on input $\mathsf{state}_2^{\mathcal{A}}, \{\mathsf{comm}_j\}_{j \notin I}$ outputs $\mathsf{state}_3^{\mathcal{A}}, \{\mathsf{comm}_i\}_{i \in I}$.*
  6. *Choose $c \in \mathcal{C}$ uniformly at random and compute $\mathsf{resp}_j \leftarrow \mathsf{Resp}(\mathsf{state}_j, c)$.*
  7. *$\mathcal{A}_2$ on input $\mathsf{state}_3^{\mathcal{A}}, c, \{\mathsf{resp}_j\}_{j \notin I}$ outputs $\{\mathsf{resp}_i\}_{i \in I}$.*
  8. *We say that $\mathcal{A}_1, \mathcal{A}_2$ wins if* $\mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}_i, x_i\}_{i \in [n]}, c) = \mathsf{true}$.

[5] In the way it is used each prover also acts as an independent verifier.

Assume that $\mathcal{A}$ wins the above game with probability $\delta > \epsilon$ where the probability is taken over the randomness of $\mathcal{A}_2$ and the choice of c.
Then we say that the protocol is a computational proof of knowledge *if there exists a PPT algorithm* $\mathsf{Extract}$ *which, for any fixed set I and* $\mathsf{state}_2^{\mathcal{A}}, \{x_i\}_{i \in I}$ *generated by $\mathcal{A}_1$, with honestly generated* $\{x_j, w_j, \mathsf{state}_j, \mathsf{comm}_j\}_{j \notin I}$ *as input and black-box access to* $\mathcal{A}_2(\mathsf{state}_2^{\mathcal{A}}, \{\mathsf{comm}_j, \mathsf{state}_j\}_{j \notin I})$ *outputs* $\{w_i\}_{i \in I}$ *such that* $\mathbf{P}'(x_1, \ldots, x_n, w_1, \ldots, w_n) = 1$ *in expected* $q(\mathsf{Snd\_sec})/(\delta - \epsilon)$ *steps where $q(\cdot)$ is a positive polynomial.*

- **Honest Verifier Zero-Knowledge:** *There is a PPT algorithm* Sim_I *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 then outputs tuples* $\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i \notin I}$. *We require that for all such I the output of* Sim_I *is statistically indistinguishable from a valid execution of the protocol.*

Since the execution of the commitment and response phases are independent for each player, we only need to look at indistinguishability of the distribution of the values $(c, \{\mathsf{comm}_i, \mathsf{resp}_i\}_{i \notin I})$ produced in a valid and a simulated execution of the protocol. Whatever the adversary does cannot affect the zero-knowledge property, as the values sent by the honest provers are independent of those sent by adversarially-controlled parties. Our formalism for HV-ZK is therefore only to allow the simulator to be applied when some provers are adversarial.

The knowledge extraction follows the standard definition of a proof of knowledge, but also incorporates the slack-definition in [3] adapted to our situation of n-Prover ZKPoK-protocols: Note that we are not assuming a special soundness definition as is usual in Σ protocols, since our knowledge extractor will require multiple rewind queries to the dishonest provers with correlated challenges. Also note that the knowledge extractor above outputs a witness for a predicate $\mathbf{P}'$ which is potentially different from the predicate that honest parties were using. In traditional ZK proofs-protocols we have that $\mathbf{P} = \mathbf{P}'$ but in many lattice based protocols these two predicates are distinct. The gap between the two predicates $\#\mathbf{P}'/\#\mathbf{P}$ is what we earlier referred to as the "soundness slack".

The above definitions imply two security parameters $\mathsf{Snd\_sec}$ and $\mathsf{ZK\_sec}$. The soundness parameter $\mathsf{Snd\_sec}$ controls the value ϵ from Definition 3.1. Usually we set $\epsilon = 2^{-\mathsf{Snd\_sec}}$, which then (loosely speaking) is the probability that an adversary with control over a set of provers $I \subseteq [n]$ can make an honest verifier accept for values $\{x_i\}_{i \in I}$ without actually having valid witnesses w_i. The second parameter is the zero-knowledge parameter $\mathsf{ZK\_sec}$, which defines the statistical distance between the distributions of genuine and simulated transcripts. We let this distance be bounded by $2^{-\mathsf{ZK\_sec}}$.

Relations to Other Definitions. It might seem that the above definition is related to multi-prover interactive proofs [6], but this is not true. Firstly, the provers in our definition may arbitrarily collude during the above protocol – which differs from multi-prover proofs where they cannot coordinate. Moreover, our definition can be seen as each P_i individually trying to convince the verifier about the correctness of an individual statement, but the soundness takes into account a combination of the individual statements as expressed by the language $\mathbb{L}$. Therefore, in an n-party ZK proof of knowledge it might happen that any subset of $n-1$ successful provers cannot produce a correct witness for the overall statement $(x_1, \ldots, x_n)$ by pooling their w_i. In multi-prover interactive proofs, each P_i may itself have full knowledge of the complete witness.

Definition 3.1 also differs from just running n proofs in parallel, as the predicate $\mathbf{P}$ might introduce constraints over all x_i, w_i. This is exactly how [25] used it in their work, where they perform checks both on the individual ciphertexts and on all of them simultaneously, thus saving runtime.

4 A n-Prover ZKPoK for SPDZ

Our protocol, which we call TopGear, is given in Figure 1. The protocol is a n-Prover ZKPoK-Protocol in which, in the way we have described it, the n players act both as a set of provers *and* individually as verifiers; as this is how the proof will be used in the SPDZ protocol. Thus in our description in Figure 1 the challenge is produced via calling a random functionality $\mathcal{F}_{\mathsf{Rand}}$ which produces a single joint challenge between the players. Such a functionality is standard, see for example [15]. further.

Protocol Π_{ZKPoK}

The protocol is parametrized by integer parameters U, V and $\mathsf{flag} \in \{\mathsf{Diag}, \perp\}$ as well as $\mathfrak{pk}$ and further parameters of the encryption scheme.

Sampling Algorithm: Samp

1. If $\mathsf{flag} = \perp$ then generate the plaintext $\boldsymbol{m} \in \mathcal{R}_p^U$ (considered as an element of $\mathcal{R}_{q_1}^U$) uniformly at random in $\mathcal{R}_p^U$. If $\mathsf{flag} = \mathsf{Diag}$ then instead for each $k \in [U]$ let $\boldsymbol{m}^{(k)}$ be a random "diagonal" message in $\mathcal{R}_p$.
2. Generate a randomness triple as $R \in \mathcal{R}_{q_1}^{U \times 3}$, each of whose rows is generated from $\mathsf{RC}\left(\sigma^2, 0.5, N\right)$.
3. Compute the ciphertexts by encrypting each row separately, thus obtaining $C \leftarrow \mathsf{Enc}(\boldsymbol{m}, R; \mathfrak{pk}) \in \mathcal{R}_{q_1}^{U \times 2}$.
4. Output $(x = (C), w = (\boldsymbol{m}, R))$.

Commitment Phase: Comm

1. Each P_i samples V pseudo-plaintexts $\boldsymbol{y}_i \in \mathcal{R}_{q_1}^V$ and pseudo-randomness vectors $S_i = (s_i^{(l,\ell)}) \in \mathcal{R}_{q_1}^{V \times 3}$ such that, for all $l \in [V]$, $\|\boldsymbol{y}_i^{(l)}\|_\infty \leq 2^{\mathsf{ZK\_sec}-1} \cdot p$ and $\|s_i^{(l,\ell)}\|_\infty \leq 2^{\mathsf{ZK\_sec}} \cdot \rho_\ell$.
2. Party P_i computes $A_i \leftarrow \mathsf{Enc}(\boldsymbol{y}_i, S_i; \mathfrak{pk}) \in \mathcal{R}_{q_1}^{V \times 2}$.
3. The players broadcast $\mathsf{comm}_i \leftarrow A_i$.

Challenge Phase: Chall

1. Parties call $\mathcal{F}_{\mathsf{Rand}}$ to obtain a $V \times U$ challenge matrix W.
2. If $\mathsf{flag} = \perp$ this is a matrix with random entries in $\left\{X^i\right\}_{i=0\dots,2\cdot N-1} \cup \{0\}$. If $\mathsf{flag} = \mathsf{Diag}$ then W is a random matrix in $\{0,1\}^{V \times U}$.

Response Phase: Resp

1. Each P_i computes $\boldsymbol{z}_i \leftarrow \boldsymbol{y}_i + W \cdot \boldsymbol{m}_i$ and $T_i \leftarrow S_i + W \cdot R_i$.
2. Party P_i sets $\mathsf{resp}_i \leftarrow (\boldsymbol{z}_i, T_i)$, and broadcasts resp_i.

Verification Phase: Verify

1. Each party P_i computes $D_i \leftarrow \mathsf{Enc}(\boldsymbol{z}_i, T_i; \mathfrak{pk})$.
2. The parties compute $A \leftarrow \sum_{i=1}^n A_i$, $C \leftarrow \sum_{i=1}^n C_i$, $D \leftarrow \sum_{i=1}^n D_i$, $T \leftarrow \sum_{i=1}^n T_i$ and $\boldsymbol{z} \leftarrow \sum_{i=1}^n \boldsymbol{z}_i$.
3. The parties check whether $D = A + W \cdot C$, and then whether the following inequalities hold, for $l \in [V]$,

$$\|\boldsymbol{z}^{(l)}\|_\infty \leq n \cdot 2^{\mathsf{ZK\_sec}} \cdot p, \quad \|T^{(l,\ell)}\|_\infty \leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}} \cdot \rho_\ell \text{ for } \ell = 1, 2, 3.$$

4. If $\mathsf{flag} = \mathsf{Diag}$ then the proof is rejected if $\boldsymbol{z}^{(l)}$ is not a constant polynomial (i.e. a "diagonal" plaintext element).
5. If all checks pass, the parties accept, otherwise they reject.

Figure 1. Protocol for global proof of knowledge of a set of ciphertexts

To understand its workings and security, first we give the two languages and then a security proof for the standard case $\mathsf{flag} = \perp$. The reason for this flag is that in the SPDZ protocol [15], at one stage ciphertexts need to be proved to be "Diagonal", namely each plaintext slot component contains the same element. We will discuss this after giving the proof for the main protocol.

The Honest Language: In an honest execution of the preprocessing each party P_i first generates a set of U ciphertexts given by, for $k \in [U]$,

$$\mathfrak{ct}_i^{(k)} = \mathsf{Enc}\left(m_i^{(k)}, (r_i^{(k,1)}, r_i^{(k,2)}, r_i^{(k,3)}); \mathfrak{pk}\right).$$

Party P_i wishes to keep the values $m_i^{(k)}, r_i^{(k,1)}, r_i^{(k,2)}, r_i^{(k,3)}$ private, whereas the ciphertexts $\mathfrak{ct}_i^{(k)}$ are public. If we define $C_i = (\mathfrak{ct}_i^{(1)}, \dots, \mathfrak{ct}_i^{(U)})$ and $\boldsymbol{m}_i, R_i$ equivalently, then we can instead say that P_i wishes to prove knowledge of $w_i = (\boldsymbol{m}_i, R_i)$ for a given $x_i = (C_i)$. By abuse of notation we relate to $\boldsymbol{m}_i, R_i, C_i$ via the equation $C_i \leftarrow \mathsf{Enc}(\boldsymbol{m}_i, R_i; \mathfrak{pk})$.

The proof shows a statement about $\boldsymbol{m} = \sum_{i=1}^n \boldsymbol{m}_i$, $R = \sum_{i=1}^n R_i$ and $C = \sum_{i=1}^n C_i$, i.e. when summed over all parties. It works over $\mathcal{R}$ (and not $\mathcal{R}_{q_1}$) to make $||\cdot||_\infty$ meaningful.

We define the "honest" language $\mathbb{L}$ as

$$\mathbb{L} = \Big\{((x_1, \ldots, x_n),(w_1, \ldots, w_n)) : \\ x_i = C_i, \quad w_i = (\boldsymbol{m}_i, R_i), \\ C = \sum C_i, \quad \boldsymbol{m} = \sum \boldsymbol{m}_i, \quad R = \sum R_i, \\ C = \mathsf{Enc}(\boldsymbol{m}, R; \mathfrak{pk}) \text{ and for all } k \in [U] \\ \|\boldsymbol{m}^{(k)}\|_\infty \leq n \cdot p/2, \quad \|R^{(k,\ell)}\|_\infty \leq n \cdot \rho_\ell \Big\}.$$

where $\rho_1 = \rho_2 = 20$ and $\rho_3 = 1$. Note, the language says nothing about whether the initial witnesses encrypt to the initial public values C_i, it only considers a joint statement about all players inputs. In the above definition we abuse notation by using Enc as a procedure irrespective of the distributions of the input variables (as $\boldsymbol{m}^{(k)}$ and $R^{(k,\ell)}$ are now elements in $\mathcal{R}$ and not necessarily in the correct domain). Here we simply apply the equations

$$b \cdot R^{(k,3)} + p \cdot R^{(k,1)} + \boldsymbol{m}^{(k)} \pmod{q_1}, \quad a \cdot R^{(k,3)} + p \cdot R^{(k,2)} \pmod{q_1}.$$

For dishonest provers (where we assume the worst case of all provers being dishonest) we will only be able to show that the inputs are from the language

$$\mathbb{L}'_c = \Big\{((x_1, \ldots, x_n),(w_1, \ldots, w_n)) : \\ x_i = C_i, \quad w_i = (\boldsymbol{m}_i, R_i), \\ C = \sum C_i, \quad \boldsymbol{m} = \sum \boldsymbol{m}_i, \quad R = \sum R_i, \\ C = \mathsf{Enc}(\boldsymbol{m}, R; \mathfrak{pk}) \text{ and for all } k \in [U] \\ \|c \cdot \boldsymbol{m}^{(k)}\|_\infty \leq 2^{\mathsf{ZK\_sec}+1} \cdot n \cdot p, \\ \|c \cdot R^{(k,\ell)}\|_\infty \leq 2^{\mathsf{ZK\_sec}+2} \cdot n \cdot \rho_\ell \Big\}.$$

Notice that not only the bounds on $\boldsymbol{m}^{(k)}, R^{(k)}$ have increased, but the values within the norms have also been multiplied by a factor of two.

Thus we see that, at a high level, the bounds for the honest language are $|w_i| < B$, whilst the bounds for the proven language are $|c \cdot w_i| < 2^{\mathsf{ZK\_sec}+2} \cdot B$. This additional factor $2^{\mathsf{ZK\_sec}+2}$ is called the *soundness slack*. This soundness slack *could* be reduced by utilizing rejection sampling as in lattice signature schemes, but this would complicate the protocol (being an n-party proof) and (more importantly) it turns out that the soundness slack has no effect on the parameters needed in the protocol.

There is an added complication arising from the language $\mathbb{L}'_c$, corresponding to the factor of c in Lemma 2.2. However, this can be side-stepped by a minor modification to how the ZKPoKs are used with the SPDZ offline phase (which we describe in Section 5).

Theorem 4.1. *Let* $\mathsf{flag} = \perp$ *and* $V \geq (\mathsf{Snd\_sec}+2)/(2N+1)$, *then the algorithms in Figure 1 are an* n*-party ZKPoK protocol according to Definition 3.1 for the languages* $\mathbb{L}$ *and* $\mathbb{L}'_2$ *with soundness error* $2^{-\mathsf{Snd\_sec}}$ *and statistical distance* $2^{-\mathsf{ZK\_sec}}$ *in the simulation.*

Proof.
Correctness: In proving the correctness property all the parties are assumed to be honest. We therefore have that $(x_i, w_i) \leftarrow \mathsf{Samp}_n(i)$ are generated by the sampling algorithm. Thus for all $k \in [U]$, $\|\boldsymbol{m}_i^{(k)}\|_\infty \leq \frac{p}{2}$

and $\|r_i^{(k,\ell)}\|_\infty \leq \rho_\ell$, where $\rho_1 = \rho_2 = 20$ and $\rho_3 = 1$. As summing over the n values increases the overall norm by at most n, we have $((x_1, \ldots, x_n), (w_1, \ldots, w_n)) \in \mathbb{L}$.

We now prove that the protocol terminates by considering the checks in Step 3 of Verify. The equality $D = A + W \cdot C$ follows at once from the fact that everything is defined as a linear function of their arguments and the BGV encryption, in particular, is linear and works component-wise. By repeatedly applying Lemma 2.3 (and using the fact that $U \leq 2^{\mathsf{ZK\_sec}}$ in practice) we obtain

$$\begin{aligned}
\|\boldsymbol{z}^{(k)}\|_\infty &\leq \sum_{i=1}^{n} \|(\boldsymbol{y}_i + W \cdot \boldsymbol{m}_i)^{(k)}\|_\infty < n \cdot \left(2^{\mathsf{ZK_sec}} \cdot \frac{p}{2} + U \cdot \frac{p}{2}\right) \\
&\leq p \cdot n \cdot 2^{\mathsf{ZK_sec}}, \\
\|T^{(k,\ell)}\|_\infty &\leq \sum_{i=1}^{n} \|(S_i + W \cdot R_i)^{(k,\ell)}\|_\infty < n \cdot \left(2^{\mathsf{ZK_sec}} \cdot \rho_\ell + U \cdot \rho_\ell\right) \\
&\leq 2 \cdot n \cdot 2^{\mathsf{ZK_sec}} \cdot \rho_\ell.
\end{aligned}$$

Honest verifier zero-knowledge: Following our definition of n-prover ZKPoK-protocol we give in Figure 2 a simulator, parametrized by a set of corrupted parties I and a challenge W, which produces transcripts. It is clear that the statistical difference in the distribution of the *coefficients* of the ring elements in $\boldsymbol{z}_j$ and T_j for $j \notin I$ in a real execution and the simulated execution can be bounded by Lemma 2.1 as $2^{-\mathsf{ZK\_sec}}$.

The Simulator Sim_I

The input is a challenge $V \times U$ matrix W defined as in the main protocol.

1. For all $j \notin I$ do
   (a) Sample $\boldsymbol{z}_j \in \mathcal{R}_{q_1}^V$ such that, for $l \in [V]$, we have $\|\boldsymbol{z}_j^{(l)}\|_\infty \leq 2^{\mathsf{ZK\_sec}-1} \cdot p$.
   (b) Sample $T_j \in \mathcal{R}_{q_1}^{V \times 3}$ such that, for $l \in [V]$ and $\ell = 1, 2, 3$, we have $\|T_j^{(l,\ell)}\|_\infty \leq 2^{\mathsf{ZK\_sec}} \cdot \rho_\ell$.
   (c) Set $\mathsf{resp}_j \leftarrow (\boldsymbol{z}_j, T_j)$.
   (d) Compute $\mathsf{comm}_j = A_j \leftarrow \mathsf{Enc}(\boldsymbol{z}_j, T_j; \mathfrak{pk}) - W \cdot C_j$.
2. Output $(\mathsf{comm}_j, \mathsf{resp}_j)_{j \notin I}$.

Figure 2. Simulator Sim_I for our protocol

Knowledge soundness: The knowledge extractor follows the methodology of [3, Lemma 3 and 5]. We refer the reader there for more details. Here we outline the technique and the effect it has on our bounds for the language $\mathbb{L}_2'$. In the following we let δ denote the probability that, for fixed $I, \{x_i\}_{i \in I}$ of dishonest provers $\mathcal{A}_2$ produces a valid transcript (sampled over the randomness tape χ of $\mathcal{A}_2$, the possible challenges W and the values $\{A_j, \boldsymbol{z}_j, T_j\}_{j \notin I}$). By assumption we have $\delta > 2^{-\mathsf{Snd\_sec}}$.

We first need to ensure that we can apply the proof of [3] in our setting. Their construction only has a single prover to extract from and they do not have extra messages which honest participants in the protocol may send. To show applicability of their extractor, we construct a "hybrid" prover $\hat{\mathcal{P}}$ that can be combined with their technique.

Claim. Let $\mathcal{A}$ be an adversary as in the soundness definition of Definition 3.1. Then there exists a prover $\hat{\mathcal{P}}$ which as input only obtains the random tape $\hat{\chi}$ (as well as implicitly the parameters of the encryption scheme and $\{x_j, w_j\}_{j \notin I}$ generated by Samp) and a challenge c. The player $\hat{\mathcal{P}}$ has the same chance δ of outputting a correct transcript (when the probability is taken over $\hat{\chi}, W$) as $\mathcal{A}$.

Proof. We construct $\hat{\mathcal{P}}$ as follows:

1. Let $\hat{\chi} = (\chi_1 || \chi_2)$ where χ_1 is of the same length as the randomness of $\mathcal{A}_2$ and χ_2 is sufficiently long to run $n - |I|$ independent instances of Comm.
2. Compute $(\mathsf{comm}_j, \mathsf{state}_j) \leftarrow \mathsf{Comm}(x_j, w_j)$ for all $j \notin I$ using a fresh part of χ_2 for each instance.

3. Run $\mathcal{A}_2$ on $\{\mathsf{comm}_j\}_{j\notin I}$, obtain $\{\mathsf{comm}_i\}_{i\in I}$.
4. Output $\{\mathsf{comm}_i\}_{i\in[n]}$ and wait for the challenge c.
5. Upon input c compute $(\mathsf{resp}_j) \leftarrow \mathsf{Resp}(\mathsf{state}_j, c)$ for all $j \notin I$.
6. Continue to run $\mathcal{A}_2$ on new inputs $c, \{\mathsf{resp}_j\}_{j\notin I}$. Then output $\{\mathsf{resp}_j\}_{j\notin I}$ and whatever $\mathcal{A}_2$ outputs.

Observe that by assumption Resp is a deterministic algorithm. Therefore, there is only one value $\{\mathsf{resp}_j\}_{j\notin I}$ which we can give to $\mathcal{A}_2$ in Step 7 of the soundness game. Thus, $\mathcal{A}_2$'s actions are fully determined given $\chi, c, \{\mathsf{comm}_j\}_{j\notin I}$. As $\hat{\mathcal{P}}$ generates $\{\mathsf{comm}_j\}$ in the same way as it is done in the security experiment, it has the same chance δ of outputting a correct transcript (when probability is taken over $\hat{\chi}, c$) as $\mathcal{A}$. The only additional runtime comes from running $\mathsf{Comm}, \mathsf{Resp}$ for simulated honest parties, which is small. □

They then construct an ensemble of extractors $\{\hat{\mathcal{E}}_k\}_{k\in[U]}$, one for each of the inputs. This also works in our setting, as we can extract each $k-$th plaintext and randomness from each of the parties simultaneously. More in detail, each $\hat{\mathcal{E}}_k$, when adapted to our setting, works as follows:

1. Run $\hat{\mathcal{P}}$ on random challenges W and uniformly random choices of $\hat{\chi}$ until it outputs an accepting transcript. Save the accepting challenge W as well as response $\{\boldsymbol{z}_i, T_i\}_{i=1}^{n}$.
2. Select a new challenge matrix $W^{(k)}$ which is identical to W except in *column* k. This challenge is passed to $\hat{\mathcal{P}}(\hat{\chi})$ and the process is repeated, either until one of the $W^{(k)}$ succeeds or if $\mathsf{Snd\_sec}/\delta$ challenges were generated but none succeeded. In the latter case, the extractor aborts.
3. If the extractor succeeds, then it outputs $\{\boldsymbol{z}_i, \boldsymbol{z}_i^{(k)}, T_i, T_i^{(k)}\}_{i\in[n]}$ as well as $W, W^{(k)}$.

Assume that $\hat{\mathcal{E}}_k$ succeeds in outputting the two accepting transcripts W, $\{\boldsymbol{z}_i, T_i\}_{i=1}^{n}$, $W^{(k)}$, $\{\boldsymbol{z}_i^{(k)}, T_i^{(k)}\}_{i\in[n]}$ with the constraints as given above. Then, for this $k \in [U]$ we can compute the following: First, let $l \in [V]$ denote an index such that $w_{l,k} \neq w'_{l,k}$, where $w_{i,j}$ (resp. $w'_{i,j}$) are the entries of W (resp. $W^{(k)}$). Using Lemma 2.2 we can write $g = 1/(w_{l,k} - w'_{l,k}) \in \mathcal{R}$ with $\|2 \cdot g\|_\infty \leq 1$.

Setting $D_i \leftarrow \mathsf{Enc}(\boldsymbol{z}_i, T_i; \mathfrak{pk})$ and $D_i^{(k)} \leftarrow \mathsf{Enc}(\boldsymbol{z}_i^{(k)}, T_i^{(k)}; \mathfrak{pk})$ we obtain two matrix equations

$$D = A + W \cdot C \text{ and } D^{(k)} = A + W^{(k)} \cdot C.$$

Write $E = D - D^{(k)}$ and note that the (i,j)'th element $e_{i,j}$ is equal to

$$\begin{aligned} e_{i,j} &= \sum_{t=1}^{U} (w_{i,t} - w'_{i,t}) \cdot c_{t,j} \\ &= (w_{i,k} - w'_{i,k}) \cdot c_{k,j} && \text{by choice of } W^{(k)}. \end{aligned}$$

Hence $2 \cdot c_{k,j} = 2 \cdot g \cdot e_{l,j}$, which implies, by the linearity of encryption, that we can extract a message $m^{(k)}$ and randomness values corresponding to row $R^{(k,\cdot)}$, which satisfy the bounds for $\ell = 1,2,3$

$$\begin{aligned} \|2 \cdot m^{(k)}\|_\infty &\leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+1} \cdot \frac{p}{2}, \\ \|2 \cdot R^{(k,\ell)}\|_\infty &\leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+1} \cdot \rho_\ell. \end{aligned}$$

In a similar way as above we can moreover extract the individual $m_i^{(k)}, R_i^{(k)}$ which add up to $m^{(k)}, R^{(k)}$. By concatenation of the outputs which we obtain this way from all $\hat{\mathcal{E}}_k$, this then permits to output a witness for $\mathbb{L}'_2$ as required.

We now consider runtime and success probability of this process. Following the analysis of [3, Lemma 5], Step 1 of $\hat{\mathcal{E}}_k$ takes expected time $1/\delta$ to succeed. Afterwards in Step 2 of each $\hat{\mathcal{E}}_k$ we make at most $\mathsf{Snd\_sec}/\delta$ queries to $\hat{\mathcal{P}}$. Following the analysis of [3], by running $\hat{\mathcal{E}}_k$ at most $3 \cdot \mathsf{Snd\_sec}$ times it will output a pair of values, except with probability at most $2^{-\mathsf{Snd\_sec}}$. By a union bound, all the above extractors will output a witness for $\mathbb{L}'_2$ in expectancy[6] in time $(3U \cdot \mathsf{Snd\_sec}^2)/\delta$ with probability at least $1 - U \cdot 2^{\mathsf{Snd\_sec}}$. □

[6] One can improve the runtime by making a different analysis: as the success probability for each run of $\hat{\mathcal{E}}_k$ is constant, one can analyze the chance of $O(\lambda)$ consecutive calls to different extractors having $\geq U$ success events using a Hoeffding bound.

The value $q(\lambda, U) = 3 \cdot U \cdot \mathsf{Snd\_sec}^2$ in the above proof should, correctly, be taken into account in our security estimates later. However, the effect is marginal, and so to aid exposition we drop this consideration from now on.

For the proof of "diagonal" elements, the main difference is that soundness must ensure that the extracted $\boldsymbol{m}$ encodes a "diagonal" plaintext as well. Unfortunately, as we have to multiply with ring elements in the extraction process, this might not necessarily be true. Instead, in such a case we fall back to a binary challenge matrix W (as depicted in Figure 1), where the "diagonal" property follows as the extractor only performs additions and subtractions on the values $\boldsymbol{z}_i, T_i$ in the process, but no multiplications with inverses. As a side effect, the proof actually yields bounds on the exact $\boldsymbol{m}^{(k)}, R^{(k,\ell)}$ and not just their multiples. One can easily show the following

Corollary 4.1. *Let* $\mathsf{flag} = \mathsf{Diag}$ *and* $V \geq \mathsf{Snd\_sec}+2$*, then the algorithms in Figure 1 are an* n*-party ZKPoK protocol according to Definition 3.1 for the languages* $\mathbb{L}$ *and* $\mathbb{L}'_1$ *with soundness error* $2^{-\mathsf{Snd\_sec}}$ *and statistical distance* $2^{-\mathsf{ZK\_sec}}$ *in the simulation.*

5 SPDZ Offline Phase

We nowshow how to combine the ZKPoK from Section 4 with the offline phase of the SPDZ protocol. After a brief recap of it, we outline the necessary changes to the HighGear protocol of [25]. Recall that the offline phase of SPDZ primarily generates shared random triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that $c = a \cdot b$ where a, b are chosen uniformly at random from $\mathbb{F}_p$ (and no subset of parties either know a, b or c, or can affect their distribution). This is done, by each party P_i encoding $\phi(m)$ $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ values into two elements a_i and b_i in $\mathcal{R}_p$ These a_i and b_i are encrypted via the BGV scheme, and the parties obtain

$$\mathfrak{ct}_{a_i} = \mathsf{Enc}(a_i, \boldsymbol{r}_{a,i}; \mathfrak{pk}) \text{ and } \mathfrak{ct}_{b_i} = \mathsf{Enc}(b_i, \boldsymbol{r}_{b,i}; \mathfrak{pk}).$$

Using the homomorphic properties of the BGV encryption scheme the parties can then compute an encryption of the product $c \in \mathcal{R}_p$ via

$$\mathfrak{ct}_c = (\mathfrak{ct}_{a_1} \boxplus \cdots \boxplus \mathfrak{ct}_{a_n}) \odot (\mathfrak{ct}_{b_1} \boxplus \cdots \boxplus \mathfrak{ct}_{b_n}). \tag{1}$$

The product $\mathfrak{ct}_c$ is decrypted using a distributed decryption protocol which gives to each party a share $c_i \in \mathbb{F}_p$ of the plaintext of $\mathfrak{ct}_c$. To achieve security in the online phase, one furthermore needs to compute shares of the MACs $\gamma[a], \gamma[b]$ and $\gamma[c]$, which are obtained in a similar manner.

In the offline phase, the main attack vector is that dishonest parties could produce ciphertexts which contain maliciously chosen noise or a plaintext unbeknownst to the sending party. This would result in either selective failure attacks or information leakage during the distributed decryption procedure. Thus each ciphertext $\mathfrak{ct}_{a_i}$ needs to be accompanied by a ZKPoK showing that it is not too far from being an honestly generated ciphertext. As the ZKPoKs bound the noise term associated to every ciphertext, we use this bound to derive the parameters for the BGV encryption scheme. This in turn ensures that all ciphertexts will validly decrypt. Quite obviously we want to execute U such proofs in parallel such as to amortize.

As noticed in HighGear [25], the ciphertexts $\mathfrak{ct}_{a_i}$ are only ever used in a sum (as in Equation 1). Therefore it is possible to replace n individual ZKPoKs for $\mathfrak{ct}_{a_i}$ by a single ZKPoK for the sum $\mathfrak{ct}_a = \mathfrak{ct}_{a_1} \boxplus \cdots \boxplus \mathfrak{ct}_{a_n}$ – which is exactly the strategy we outlined in the previous two sections. However, our proof comes at the expense of not obtaining guarantees about the original ciphertext sum $\mathfrak{ct}_a$, but instead of $2 \cdot \mathfrak{ct}_a = \mathfrak{ct}_a \boxplus \mathfrak{ct}_a$. Luckily this of no concern in preprocessing for SPDZ - we can simply later adjust some of the shares by a factor of two and continue as before. The modifications are explained in Figure 3 for the case of triple production. The modifications to obtain other forms of preprocessed data such as those in [14] are immediate.

The overhead V for the ciphertexts encrypting α_i is quite big when compared to those of the triples and MAC shares. This is because our proof from Section 4 is not as efficient when $\mathsf{flag} = \mathsf{Diag}$ (see Corollary 4.1). However, this is not an issue as we only produce one such ciphertext during the offline phase. To mitigate this we actually run the ZKPoK for a fixed value ($V = 16$) and then repeat this $\mathsf{Snd\_sec}/V$ times. This makes no difference to the overall running time, but keeps the memory requirements low for this part of the protocol.

Protocol Π_{offline}

Init:

1. Each P_i runs $\mathsf{Samp}_n(i)$ with $\mathsf{flag} = \mathsf{Diag}$ and $U = 1$ to obtain $\alpha_i \in \mathbb{F}_p$ as well as the ciphertext $\mathfrak{ct}_{\alpha_i}$.
2. For $i = 1, \ldots, \mathsf{Snd\_sec}/16$
   (a) The parties perform the other phases of Π_{ZKPoK} with $\mathsf{flag} = \mathsf{Diag}, U = 1$ and $V = 16$. If any proof rejects they abort.
3. The parties set $\mathfrak{ct}_\alpha \leftarrow (\mathfrak{ct}_{\alpha_1} \boxplus \cdots \boxplus \mathfrak{ct}_{\alpha_n})$.

Triples:

1. We set $V = (\mathsf{Snd\_sec} + 2)/\log_2(2 \cdot N + 1)$ and $U = 2 \cdot V$.
2. Each P_i runs $\mathsf{Samp}_n(i)$ for this value of U with $\mathsf{flag} = \perp$. It thus obtains the plaintext vectors $\hat{\boldsymbol{a}}_i^{(k)}, \hat{\boldsymbol{b}}_i^{(k)}, \hat{\boldsymbol{f}}_i^{(k)} \in (\mathbb{F}_p)^{\phi(m)}$ as well as the ciphertexts $\mathfrak{ct}_{\hat{a}_i}^{(k)}$, $\mathfrak{ct}_{\hat{b}_i}^{(k)}$ and $\mathfrak{ct}_{\hat{f}_i}^{(k)}$ for $k \in [U]$.
3. The parties then run the remaining steps of the protocol Π_{ZKPoK} using U, V and $\mathsf{flag} = \perp$. If any of the proofs fail, then they abort.
4. The parties set $\mathfrak{ct}_a^{(k)} \leftarrow 2 \cdot (\mathfrak{ct}_{\hat{a}_1}^{(k)} \boxplus \cdots \boxplus \mathfrak{ct}_{\hat{a}_n}^{(k)})$, $\mathfrak{ct}_b^{(k)} \leftarrow 2 \cdot (\mathfrak{ct}_{\hat{b}_1}^{(k)} \boxplus \cdots \boxplus \mathfrak{ct}_{\hat{b}_n}^{(k)})$ and $\mathfrak{ct}_f^{(k)} \leftarrow 2 \cdot (\mathfrak{ct}_{\hat{f}_1}^{(k)} \boxplus \cdots \boxplus \mathfrak{ct}_{\hat{f}_n}^{(k)})$ for $k \in [U]$.
5. The parties compute $\mathfrak{ct}_c^{(k)} \leftarrow \mathfrak{ct}_a^{(k)} \odot \mathfrak{ct}_b^{(k)}$ as well as $\mathfrak{ct}_{c+f}^{(k)} \leftarrow \mathfrak{ct}_c^{(k)} \boxplus \mathfrak{ct}_f^{(k)}$ for $k \in [U]$.
6. Using the distributed decryption operation of the BGV scheme they then decrypt $\boldsymbol{\Delta}^{(k)}$ for $k \in [U]$.
7. P_1 sets $\hat{\boldsymbol{c}}_1^{(k)} \leftarrow \boldsymbol{\Delta}^{(k)} - \boldsymbol{f}_1^{(k)}$, while each remaining P_i sets $\hat{\boldsymbol{c}}_i^{(k)} \leftarrow -\boldsymbol{f}_i^{(k)}$ for $k \in [U]$.
8. The parties compute a fresh encryption of each $\hat{\boldsymbol{c}}^{(k)}$ via $\tilde{\mathfrak{ct}}_c^{(k)} \leftarrow \mathsf{Enc}(\boldsymbol{\Delta}^{(k)}, \mathbf{0}; \mathfrak{pk}) - \mathfrak{ct}_f^{(k)}$ with default random coins $\mathbf{0}$ for $k \in [U]$.
9. The parties compute $\mathfrak{ct}_{\alpha \cdot a}^{(k)} \leftarrow \mathfrak{ct}_\alpha \odot \mathfrak{ct}_a^{(k)}$, $\mathfrak{ct}_{\alpha \cdot b}^{(k)} \leftarrow \mathfrak{ct}_\alpha \odot \mathfrak{ct}_b^{(k)}$ and $\mathfrak{ct}_{\alpha \cdot c}^{(k)} \leftarrow \mathfrak{ct}_\alpha \odot \tilde{\mathfrak{ct}}_c^{(k)}$ for $k \in [U]$.
10. The MAC values $\gamma_i^{(k)}[\boldsymbol{a}]$, $\gamma_i^{(k)}[\boldsymbol{b}]$, $\gamma_i^{(k)}[\boldsymbol{c}]$ are obtained by applying the $\mathsf{DistDec}$ protocol in Figure 14 from [25] for $k \in [U]$.
11. Each party sets $\boldsymbol{a}_i^{(k)} \leftarrow 2 \cdot \hat{\boldsymbol{a}}_i^{(k)}$, $\boldsymbol{b}_i^{(k)} \leftarrow 2 \cdot \hat{\boldsymbol{b}}_i^{(k)}$ and $\boldsymbol{c}_i^{(k)} \leftarrow 2 \cdot \hat{\boldsymbol{c}}_i^{(k)}$. It then obtains the shares of the elements encoded in $\boldsymbol{a}_i^{(k)}, \boldsymbol{b}_i^{(k)}, \boldsymbol{c}_i^{(k)}$ as well as their MACs $\gamma_i^{(k)}[\boldsymbol{a}]$, $\gamma_i^{(k)}[\boldsymbol{b}]$, $\gamma_i^{(k)}[\boldsymbol{c}]$ by mapping the associated polynomials into the slot representation. They thus obtain $U \cdot \phi(m)$ shares.

Figure 3. TopGear version of the SPDZ Offline Phase

In the protocol in Figure 3 we have utilized the more efficient Distributed Decryption protocol from HighGear to obtain the MAC shares, and have merged in the $\mathsf{ReShare}$ protocol of [14, Figure 11]. This protocol is needed to obtain the shares of c and the fresh encryption of c. The latter has been done so as to demonstrate our modified ZKPoKs make no difference to the overall protocol.

The proof of security of this offline phase follows exactly as in the original SPDZ papers [14,15], all that changes is the bound on the noise of the resulting ciphertexts. Suppose (c_0, c_1) is a ciphertext corresponding to one of $\mathfrak{ct}_\alpha, \mathfrak{ct}_a, \mathfrak{ct}_b$ or $\mathfrak{ct}_f$ in our protocol. To prove security, it is necessary to obtain worst case bounds on the value $\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}}$. We know that

$$c_0 - \mathfrak{sk} \cdot c_1 = 2 \cdot \sum_{i=1}^{n} (m_i + p \cdot (\epsilon \cdot r_i^{(3)} + r_i^{(1)} - r_i^{(2)} \cdot \mathfrak{sk}))$$

where $(m_i, r_i^{(1)}, r_i^{(2)}, r_i^{(3)})$ are bounded due to the ZKPoK. From the soundness of it we can guarantee that the ciphertexts must satisfy

$$\Big\| 2 \cdot \sum_{i \in [n]} m_i \Big\|_\infty \le 2^{\mathsf{ZK\_sec}+1} \cdot n \cdot p,$$

$$\Big\| 2 \cdot \sum_{i \in [n]} r_i^{(\ell)} \Big\|_\infty \le 2^{\mathsf{ZK\_sec}+2} \cdot n \cdot \rho_\ell.$$

Due to our assumption of an honest key generation phase, we also know that with probability $1 - 2^{-\epsilon}$ we have $\|\epsilon\|_\infty^{\mathsf{can}} \leq \mathfrak{c}_1 \cdot \sigma \cdot \sqrt{\phi(m)}$ and $\|\mathfrak{sk}\|_\infty^{\mathsf{can}} \leq \mathfrak{c}_1 \cdot \sqrt{h}$. Using the inequality $\|x\|_\infty^{\mathsf{can}} \leq \phi(m) \cdot \|x\|_\infty$ we obtain

$$\begin{aligned}
\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}} &\leq \sum_{i=1}^{n} \|2 \cdot m_i\|_\infty^{\mathsf{can}} + p \cdot \Big(\|\epsilon\|_\infty^{\mathsf{can}} \cdot \|2 \cdot e_{2,i}\|_\infty^{\mathsf{can}} + \|2 \cdot e_{0,i}\|_\infty^{\mathsf{can}} \\
&\qquad\qquad + \|\mathfrak{sk}\|_\infty^{\mathsf{can}} \cdot \|2 \cdot e_{1,i}\|_\infty^{\mathsf{can}} \Big) \\
&\leq 2 \cdot \phi(m) \cdot 2^{\mathsf{ZK_sec}+1} \cdot n \cdot p/2 \\
&\qquad + p \cdot \Big(\mathfrak{c}_1 \cdot \sigma \cdot \phi(m)^{3/2} \cdot 2 \cdot 2^{\mathsf{ZK_sec}+1} \cdot n \\
&\qquad\qquad + \phi(m) \cdot 2 \cdot 2^{\mathsf{ZK_sec}+1} \cdot n \cdot 20 \\
&\qquad\qquad + \mathfrak{c}_1 \cdot \sqrt{h} \cdot \phi(m) \cdot 2 \cdot 2^{\mathsf{ZK_sec}+1} \cdot n \cdot 20 \Big) \\
&= \phi(m) \cdot 2^{\mathsf{ZK_sec}+2} \cdot n \cdot p \cdot \Big(\frac{41}{2} + \mathfrak{c}_1 \cdot \sigma \cdot \phi(m)^{1/2} + 20 \cdot \mathfrak{c}_1 \cdot \sqrt{h} \Big) \\
&= B_{\mathsf{clean}}^{\mathsf{dishonest}}.
\end{aligned}$$

Using this bound we can then derive the parameters for the BGV system using exactly the same methodology as can be found in [2].

6 Results

Recall we have three different security parameters in play, apart from the computational security parameter κ of the underlying BGV encryption scheme. The main benefit of TopGear over HighGear is that it potentially enables higher values of the parameter Snd_sec to be obtained. Recall $2^{-\mathsf{Snd\_sec}}$ is the probability that an adversary will be able to produce a convincing ZKPoK for an invalid input. The other two security parameters are ZK_sec and DD_sec, which measure the statistical distance of *coefficients* of ring elements generated in a protocol to the same coefficients being generated uniformly at random from a similar range.

In the context of the HighGear ZKPoK in the Overdrive paper [25] the two security parameters are set to be equal, i.e. ZK_sec = Snd_sec. In practice the value of Snd_sec needs to be very low for the HighGear ZKPoK as it has a direct effect on the memory consumption of the underlying protocol. Thus in SCALE v1.2 the default value for Snd_sec is 40. This is unfortunate as having a high probability of an adversary being able to get away with cheating in a ZKPoK is not desirable. The first goal of our work is to enable Snd_sec to be taken to be as large as is desired, whilst also obtaining an efficiency saving.

As a second goal we also aim to increase the values of ZK_sec and DD_sec. These measure statistical distances of *coefficients*. But recall the ring elements have many thousands of coefficients, and in the course of a protocol execution we generate many such ring elements. Picking ZK_sec and DD_sec at low values is also not desirable, is it potentially introduces leakage about the either the plaintexts or the secret keys. Hence, after demonstrating the effect of our new protocol with respect to the increase in Snd_sec, we then turn to examining the effect of increasing the other security parameters.

In Table 1 we give various parameter sizes for the degree N and moduli $q_0 = p_0, q_1 = p_0 \cdot p_1$ for different plaintext space sizes p, and different security levels ZK_sec, Snd_sec, DD_sec and computational security parameter κ, and two parties[7]. We selected parameters for which $\mathsf{ZK\_sec}, \mathsf{DD\_sec} \leq \mathsf{Snd\_sec}$ to keep the table managable. We use the methodology described in [2] to derive parameter sizes for both HighGear and TopGear; which maps the computational security parameter is mapped to lattice parameters using Albrecht's tool[8]. A row which with values of $\star$ in the ZK_sec columns means that the values do not change when one varies this parameter is 40 or 80.

[7] Similar values can be obtained for other values of n, we selected $n = 2$ purely for illustration here, the effect of n on the values is relatively minor.

[8] https://bitbucket.org/malb/lwe-estimator

					HighGear			TopGear				
$\log_2 p$	κ	DD_sec	Snd_sec	ZK_sec	N	$\log_2 p_0$	$\log_2 p_1$	N	U	V	$\log_2 p_0$	$\log_2 p_1$
64	80	40	40	40	8192	177	114	8192	6	3	176	115
64	80	40	80	40	8192	177	114	8192	12	6	176	115
64	80	40	128	40	8192	177	114	8192	18	9	176	115
64	80	40	80	80	8192	177	144	8192	12	6	176	115
64	80	40	128	80	16384	177	174	8192	18	9	167	115
64	80	80	40	40	16384	218	163	16384	6	3	217	164
64	80	80	80	40	16384	218	163	16384	12	6	217	164
64	80	80	128	40	16384	218	163	16384	18	9	217	164
64	80	80	80	80	16384	218	163	16384	12	6	217	164
64	80	80	128	80	16384	218	173	16384	18	9	217	164
64	80	128	40	40	16384	266	205	16384	6	3	265	206
64	80	128	80	$\star$	16384	266	205	16384	12	6	265	206
64	80	128	128	$\star$	16384	266	205	16384	18	9	265	206
64	128	40	40	40	16384	178	123	16384	6	3	177	124
64	128	40	80	40	16384	178	123	16384	12	6	177	124
64	128	40	128	40	16384	178	133	16384	18	9	177	124
64	128	40	80	80	16384	178	153	16384	12	6	177	124
64	128	40	128	80	16384	178	173	16384	18	9	177	124
64	128	80	40	40	16384	218	163	16384	6	3	217	164
64	128	80	80	40	16384	218	163	16384	12	6	217	164
64	128	80	128	40	16384	218	163	16384	18	9	217	164
64	128	80	80	80	16384	218	163	16384	12	6	217	164
64	128	80	128	80	16384	218	173	16384	18	9	217	164
64	128	128	40	$\star$	32768	266	205	32768	6	3	266	205
64	128	128	80	$\star$	32768	266	205	32768	10	5	266	205
64	128	128	128	$\star$	32768	266	205	32768	16	8	266	205
64	128	128	128	128	32768	266	225	32768	16	8	266	205
128	80	40	40	40	16384	305	186	16384	6	3	305	186
128	80	40	80	$\star$	16384	305	186	16384	12	6	305	186
128	80	40	128	$\star$	16384	305	186	16384	18	9	305	186
128	80	80	40	$\star$	16384	345	226	16384	6	3	345	226
128	80	80	80	$\star$	16384	345	226	16384	12	6	345	226
128	80	80	128	$\star$	16384	345	226	16384	18	9	345	226
128	80	128	40	$\star$	16384	393	268	16384	6	3	393	268
128	80	128	80	$\star$	16384	393	268	16384	12	6	393	268
128	80	128	128	$\star$	16384	393	268	16384	18	9	393	268
128	128	40	40	$\star$	32768	306	185	32768	6	3	306	185
128	128	40	80	$\star$	32768	306	185	32768	10	5	306	185
128	128	40	128	$\star$	32768	306	185	32768	16	8	306	185
128	128	80	40	$\star$	32768	346	225	32768	6	3	346	225
128	128	80	80	$\star$	32768	346	225	32768	10	5	346	225
128	128	80	128	$\star$	32768	346	225	32768	16	8	346	225
128	128	128	40	$\star$	32768	394	277	32768	6	3	394	277
128	128	128	80	$\star$	32768	394	277	32768	10	5	394	277
128	128	128	128	$\star$	32768	394	277	32768	16	8	394	277
128	128	128	128	128	32768	394	277	32768	16	8	394	277

Table 1. SHE parameters sizes for various security parameters in HighGear and TopGear (two parties). With DD_sec, ZK_sec $\leq$ Snd_sec and DD_sec, ZK_sec, Snd_sec $\in \{40, 80, 128\}$. The light grayed rows show the default parameters used in SCALE-MAMBA v1.2. The medium gray denote the parameters we use in the experiments in Sections 6.1 and 6.2. The dark grayed rows show the parameters we would recommend, i.e. the ones we use in Section 6.3.

From the table we see that the values of $\mathsf{ZK\_sec}$ and $\mathsf{Snd\_sec}$ produce relatively little effect on the overall parameter sizes, especially for large values of the plaintext modulus p. This is because the modulus switch, within the homomorphic evaluation, squashes the noise by a factor of at least p. The values $\mathsf{ZK\_sec}$ and $\mathsf{Snd\_sec}$ only blow up the noise by a factor of $2^{\mathsf{ZK\_sec}+\mathsf{Snd\_sec}/2+2}$ (for HighGear) and $2^{\mathsf{ZK\_sec}+2}$ (for TopGear), and hence a large p value cancels out this increase in noise due to the ZKPoK security parameters. In addition the parameter sizes are identical for both HighGear and TopGear, except in the case of some parameters for low values of $\log_2 p$.

We based our implementation and experiments on the SCALE-MAMBA system [2] which has an implementation of the HighGear protocol, which we modified to test against our TopGear protocol. We focused on the case of 128-bit plaintext moduli in our experiments, being the recommended size in SCALE-MAMBA v1.2 to support other MPC operations (such as fixed point operations). We first baselined the implementation in SCALE-MAMBA of HighGear against the implementation reported in [25]. The experiments in [25] were executed on i7-4790 and i7-3770S CPUs, compared to our experiments which utilized i7-7700K CPUs. From a pure CPU point of view our machines should be roughly 30% faster. The ping time between our machines was 0.47 milliseconds, whereas that for [25] was 0.3 milliseconds.

Keller et al [25], in the case of 128-bit plaintext moduli, and with the security settings equivalent to our setting of $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 64$, utilize a ciphertext modulus of 572 bits, whereas SCALE-MAMBA v1.2 utilizes a ciphertext modulus of 541 bits. In this setting Keller et al [25] achieve a maximum throughput of 5600 triples per second, whereas SCALE-MAMBA's implementation of HighGear obtains a maximum throughput of roughly 2900 triples per second. We suspect the reason for the difference in costs is that SCALE-MAMBA is performing other operations related to storing the triples for later consumption by online operations. This also means that memory utilization grows as more triples are produced, leading to larger numbers of non-local memory accesses. These effects decrease the measurable triple production rate in SCALE-MAMBA.

We now turn to examining the performance differences between HighGear and the new TopGear protocol within our modified version of SCALE-MAMBA. We first looked at two security settings so as to isolate the effect of increasing the $\mathsf{Snd\_sec}$ parameter alone. Our first setting was the standard SCALE-MAMBA setting of $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$, our second was the more secure setting of $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ and $\mathsf{Snd\_sec} = 128$. There are two main parameters in SCALE-MAMBA one can tweak which affect triple production; i) the number of threads devoted to executing the zero-knowledge proofs and ii) the number of threads devoted to taking the output of these proofs and producing triples. We call these two values t_{ZK} and t_{Tr}; we looked at values for which $t_{\mathsf{ZK}}, t_{\mathsf{Tr}} \in \{1, 2, 4, 8\}$. We focus here on triple production for simplicity, a similar situation to that described below occurs in the case of bit production. We examine memory consumption (Section 6.1) and triple production (Section 6.2) in these settings so as to see the effect of changing $\mathsf{Snd\_sec}$. After this we examine increasing all the security parameters in Section 6.3, and the effect this has on memory and triple production.

6.1 Memory Consumption

We see from Table 1 that the parameters in TopGear for the underlying FHE scheme are generally identical to the those in HighGear. The only difference being when the extra soundness slack in HighGear compared to TopGear is not counter balanced by size of the underlying plaintext modulus. However, the real affect of TopGear comes in the amount of data one has to simultaneously process. Running the implementation in SCALE-MAMBA for HighGear one sees immediately that memory usage is a main constraint of the system.

A rough (under) estimation of the memory requirements of the ZKPoKs in HighGear and TopGear can be given by the sizes of the input and auxillary ciphertexts to the ZKPoK. A single ciphertext takes (roughly) $\phi(m) \cdot \log_2(p_0 \cdot p_1)$ bits to represent it. There are U input ciphertexts and V auxillary ciphertexts per player (where V is set to $2 \cdot U - 1$ in HighGear). Hence, the total number of bits required to process a ZKPoK is at least

$$(U + V) \cdot n \cdot \phi(m) \cdot \log_2(p_0 \cdot p_1).$$

Now in HighGear we need to take $U = \mathsf{Snd\_sec}$, which is what limits the applicability of large soundness security parameters in the implementations of HighGear. Whereas in TopGear we can take $V = (\mathsf{Snd\_sec} +$

$2)/\log_2(2 \cdot N + 1)$, and have arbitrary choice on U, although in practice we select $U = 2 \cdot V$. For the proofs for the encryptions of α we have $U = 1$ and set $V = 16$, simply to reduce memory costs, and then repeat the TopGear proof $\mathsf{Snd\_sec}/16$ times. Thus, all other things being equal (which the above table gives evidence for) the memory requirements in TopGear are reduced by a factor of roughly $\log_2(2 \cdot N + 1)$. For the range of N under consideration (i.e. 8192 to 32768) this gives a memory saving of a factor of between 14 and 16. A similar saving occurs in the amount of data which needs to be transferred when executing the ZKPoK. Note, this is purely the saving for holding the zero-knowledge proofs, the overall effect on memory consumption will be much less.

To see this in practice we examined the memory consumption of running HighGear and TopGear with the above settings (of $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$, and $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$, $\mathsf{Snd\_sec} = 128$) the results being given in Tables 2 and 3. We give the percentage memory consumption (given in thems of the percentage maximum resident set size obtained from `/usr/bin/time -v`). This is the maximum percentage memory consumed by the whole system when producing two million multiplication triples only. This value can vary from run to run as the different threads allocate and de-allocate memory, thus figures will inevitably vary. However, they do give an indication of memory overall consumption in a given configuration.

	t_{ZK}			
t_{Tr}	1	2	4	8
1	25	41	68	98
2	25	38	68	98
4	28	49	75	98
8	32	52	81	98

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$

	t_{ZK}			
t_{Tr}	1	2	4	8
1	70	98	-	-
2	72	98	-	-
4	73	98	-	-
8	76	98	-	-

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$, $\mathsf{Snd\_sec} = 128$

Table 2. Percentage memory consumption for HighGear for two players and $\log_2 p = 128$.

	t_{ZK}			
t_{Tr}	1	2	4	8
1	7	9	15	27
2	8	10	15	26
4	10	12	17	28
8	14	17	21	33

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$

	t_{ZK}			
t_{Tr}	1	2	4	8
1	11	19	33	63
2	12	18	33	64
4	14	21	34	64
8	16	24	39	70

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$, $\mathsf{Snd\_sec} = 128$

Table 3. Percentage memory consumption for TopGear for two players and $\log_2 p = 128$.

We find that for HighGear with the higher security parameters we are unable to perform some experiments due to memory consumption producing an abort of the SCALE-MAMBA system. We see immediately that with TopGear we obtain much reduced memory consumption, and we are able to cope with a much larger value for the security parameter $\mathsf{Snd\_sec}$.

6.2 Triple Production Throughput

We now turn to looking at throughput of the overall triple production process. We have found the best metric to look at is the average time per triple. However due to the set up costs, (e.g. producing the zero-knowledge proofs for the ciphertext encrypting the MAC key α) this average time decreases as one runs the system. In the Appendix we provide graphs to show how this average time decreases as more triples are produced for various settings. In this section we summarize the average number of triples per second we could obtain for the various settings, after computing two million triples. See Tables 4 and 5 for a summary.

t_{Tr}	t_{ZK} 1	2	4	8
1	1503	1602	1562	1335
2	1488	2347	2212	1976
4	1272	1876	2150	1865
8	976	1307	1464	1533

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$

t_{Tr}	t_{ZK} 1	2	4	8
1	1240	1369	-	-
2	1426	1834	-	-
4	1231	1612	-	-
8	940	1129	-	-

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$, $\mathsf{Snd\_sec} = 128$

Table 4. Maximum Triples per Second for HighGear for two players and $\log_2 p = 128$, after computing two million triples.

t_{Tr}	t_{ZK} 1	2	4	8
1	1594	1589	1579	1569
2	2421	2493	2463	2444
4	2364	2493	2590	2481
8	2644	1650	1658	1642

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$

t_{Tr}	t_{ZK} 1	2	4	8
1	1501	1490	1490	1426
2	2267	2192	2222	2141
4	2293	2237	2272	2114
8	1526	1540	1494	1464

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$, $\mathsf{Snd\_sec} = 128$

Table 5. Maximum Triples per Second for TopGear for two players and $\log_2 p = 128$, after computing two million triples.

We immediately see that actual performance for larger values of $\mathsf{Snd\_sec}$ is better for TopGear, both due to reduced memory consumption (U and V are smaller), but also because the ratio of V to U in HighGear is larger than that in TopGear (2 vs 1/2 in general). Thus in practice the actual work needed per triple is less in TopGear than in HighGear. In both security settings the TopGear protocol works best, when we have $t_{\mathsf{Tr}} \in \{2, 4\}$.

6.3 Recommendations

TopGear allows one to utilize a higher security for the parameter $\mathsf{Snd\_sec}$ than is currently normally done in implementations of SPDZ. Given that $2^{-\mathsf{Snd\_sec}}$ represents the *probability* that an adversary can pass of an invalid ZKPoK as valid, the default SCALE-MAMBA v1.2 setting of $\mathsf{Snd\_sec} = 40$ is arguably too low. Thus increasing it to 128 seems definitely prudent.

As mentioned above we also recommend using higher values for $\mathsf{ZK\_sec}$ and $\mathsf{DD\_sec}$. Despite these measuring statistical distances, and hence can be arguably smaller than $\mathsf{Snd\_sec}$, in practice they measure the statistical distance of distributions of coefficients from uniformly random. Each ZKPoK/distributed decryption produces tens of thousands of such coefficients, and thus having $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ is also probably too low.

Thus we end by giving some experimental results using TopGear for settings of $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 80$, $\mathsf{Snd\_sec} = 128$. and $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 128$. Again we focus on the two party case with a plaintext prime of 128 bits in length, with results giving in Tables 6 and 7. We see that with $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 80$ we obtain a performance which is slightly less what SCALE-MAMBA v1.2 achieves using much weaker security levels (comparing the values in Table 6 with the left tables in Table 2 and 4). On the other hand with

$\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 128$ the performance drops of markedly. We thus believe that $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 80$ gives a suitable compromise.

Also notice in the tables here the high memory consumption in the case of $t_{\mathsf{Tr}} = 1$ and $t_{\mathsf{ZK}} = 8$ compared to (say) $t_{\mathsf{Tr}} = 2$ and $t_{\mathsf{ZK}} = 8$. This is because in this case memory is increasing as the validated ciphertexts are being produced by the eight zero-knowledge proof threads *faster* than the single triple production thread can process them.

	t_{ZK}			
t_{Tr}	1	2	4	8
1	12	19	35	75
2	13	19	33	65
4	15	21	36	67
8	18	26	40	76

Memory Consumption

	t_{ZK}			
t_{Tr}	1	2	4	8
1	1288 (85)	1303 (81)	1302 (83)	1237 (92)
2	1904 (127)	1865 (79)	1930 (87)	1824 (92)
4	1798 (141)	1779 (94)	1773 (82)	1686 (90)
8	1183 (121)	1189 (90)	1152 (78)	1172 (76)

Triples per Second

Table 6. Percentage memory consumption and triples per second for TopGear for two players with $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 80$ and $\log_2 p = \mathsf{Snd\_sec} = 128$. We also give (in brackets) the percentage throughput compared to the (low security) standard SCALE-MAMBA v1.2 settings using HighGear.

	t_{ZK}			
t_{Tr}	1	2	4	8
1	14	21	40	90
2	14	22	38	78
4	17	24	40	78
8	18	28	47	87

Memory Consumption

	t_{ZK}			
t_{Tr}	1	2	4	8
1	1066 (70)	1082 (72)	1070 (68)	1026 (76)
2	1485 (99)	1547 (65)	1466 (66)	1440 (72)
4	1358 (106)	1404 (74)	1445 (67)	1356 (72)
8	937 (96)	984 (75)	969 (66)	938 (61)

Triples per Second

Table 7. Percentage memory consumption and triples per second for TopGear for two players with $\log_2 p = \mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 128$. Again, we also give (in brackets) the percentage throughput compared to the (low security) standard SCALE-MAMBA v1.2 settings using HighGear.

Acknowledgments

This work has been supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, and by the FWO under an Odysseus project GOH9718N.

References

1. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A New Hope. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016.
2. A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N. P. Smart, and T. Wood. SCALE-MAMBA v1.2: Documentation, 2018.

3. C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.
4. C. Baum, I. Damgård, K. G. Larsen, and M. Nielsen. How to prove knowledge of small secrets. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 478–498, Santa Barbara, CA, USA, Aug. 14–18, 2016. Springer, Heidelberg, Germany.
5. C. Baum and V. Lyubashevsky. Simple amortized proofs of shortness for linear relations over polynomial rings. Cryptology ePrint Archive, Report 2017/759, 2017. http://eprint.iacr.org/2017/759.
6. M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *20th Annual ACM Symposium on Theory of Computing*, pages 113–131, Chicago, IL, USA, May 2–4, 1988. ACM Press.
7. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.
8. F. Benhamouda, J. Camenisch, S. Krenn, V. Lyubashevsky, and G. Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 551–572, Kaoshiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer, Heidelberg, Germany.
9. F. Benhamouda, S. Krenn, V. Lyubashevsky, and K. Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 305–325, Vienna, Austria, Sept. 21–25, 2015. Springer, Heidelberg, Germany.
10. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In S. Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, Jan. 8–10, 2012. Association for Computing Machinery.
11. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, Oct. 22–25, 2011. IEEE Computer Society Press.
12. R. Cramer and I. Damgård. On the amortized complexity of zero-knowledge protocols. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 177–191, Santa Barbara, CA, USA, Aug. 16–20, 2009. Springer, Heidelberg, Germany.
13. R. Cramer, I. Damgård, C. Xing, and C. Yuan. Amortized complexity of zero-knowledge proofs revisited: Achieving linear soundness slack. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 479–500, Paris, France, Apr. 30 – May 4, 2017. Springer, Heidelberg, Germany.
14. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, Sept. 9–13, 2013. Springer, Heidelberg, Germany.
15. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, Aug. 19–23, 2012. Springer, Heidelberg, Germany.
16. R. del Pino and V. Lyubashevsky. Amortization with fewer equations for proving knowledge of small secrets. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 365–394, Santa Barbara, CA, USA, Aug. 20–24, 2017. Springer, Heidelberg, Germany.
17. R. del Pino, V. Lyubashevsky, and G. Seiler. Short discrete log proofs for FHE and Ring-LWE ciphertexts. Cryptology ePrint Archive, Report 2019/057, 2019. http://eprint.iacr.org/2019/057.
18. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.
19. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839.

20. C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.
21. C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482, Cambridge, UK, Apr. 15–19, 2012. Springer, Heidelberg, Germany.
22. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867, Santa Barbara, CA, USA, Aug. 19–23, 2012. Springer, Heidelberg, Germany.
23. S. Halevi and V. Shoup. Algorithms in HElib. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, Aug. 17–21, 2014. Springer, Heidelberg, Germany.
24. M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, Oct. 24–28, 2016. ACM Press.
25. M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ great again. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, Apr. 29 – May 3, 2018. Springer, Heidelberg, Germany.
26. V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616, Tokyo, Japan, Dec. 6–10, 2009. Springer, Heidelberg, Germany.

A Run Time Graphs

In Figure 4 we provide graphs of the throughput for HighGear in our low security, $\mathsf{Snd\_sec} = 40$, setting, with the comparable graph for TopGear in Figure 5 for two players; given graphs up to the production of 2 million triples. The fact that the graphs are not straight, they have bumps in them, is because the triple production threads are producing triples faster than the ciphertexts can be supplied by the threads doing the ZKPoKs. Thus the triple production threads often need to wait until a ZKPoK has been completed before they can proceed. In Figure 6 and Figure 7 we provide similar graphs of the throughput for HighGear and TopGear in our high security setting $\mathsf{Snd\_sec} = 128$.

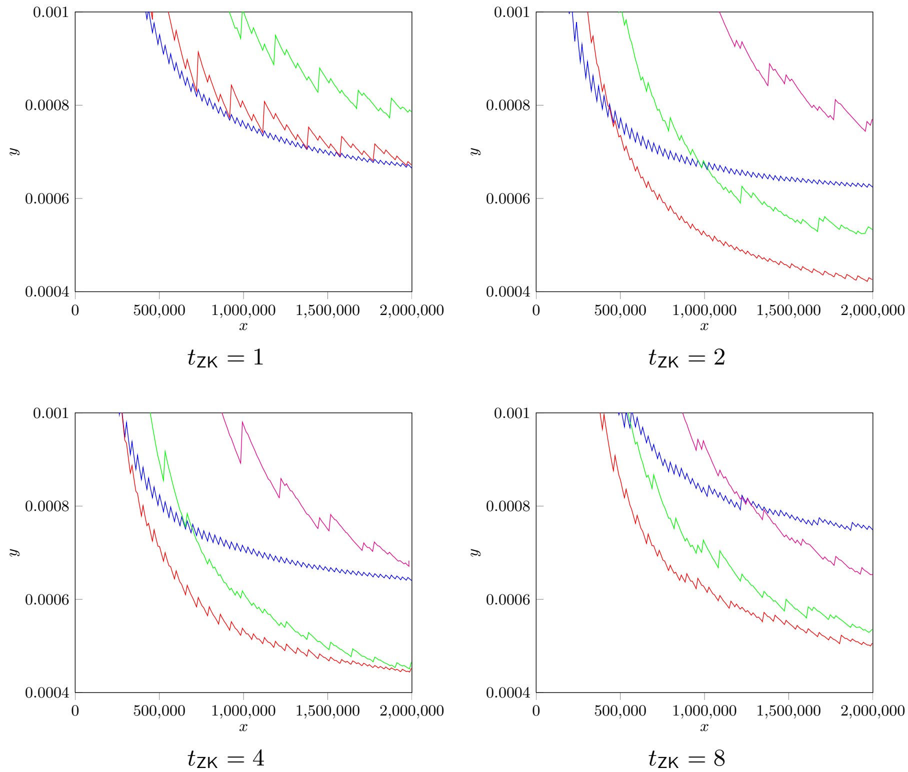

Fig. 4. Average time y to produce a triple given the number of triples that have been produced x for HighGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} = 8$

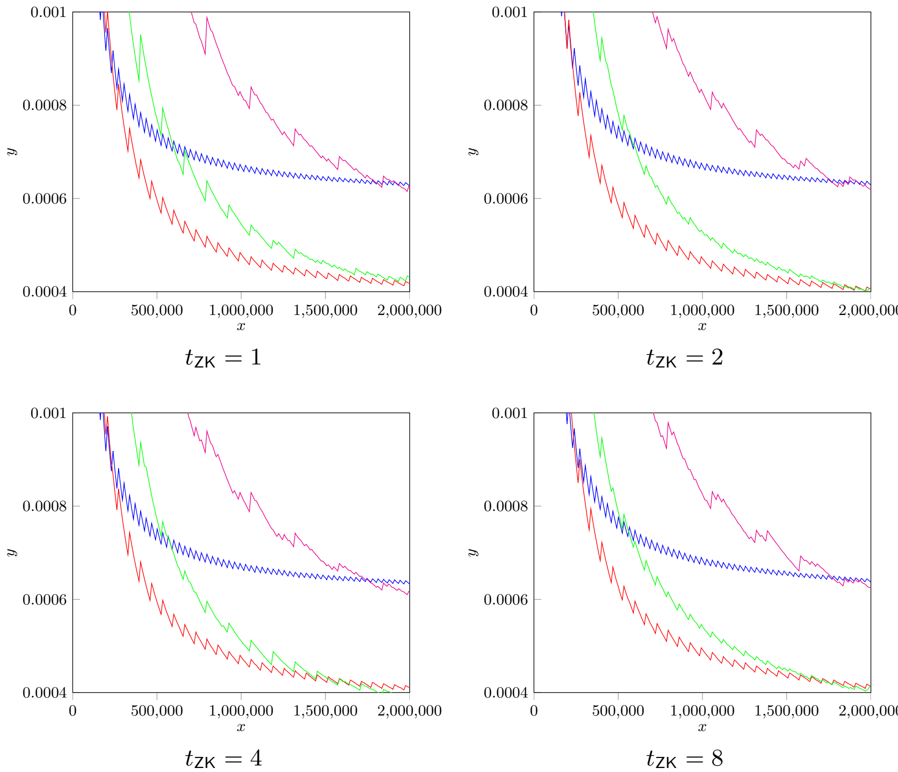

Fig. 5. Average time y to produce a triple given the number of triples that have been produced x for TopGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} = 8$

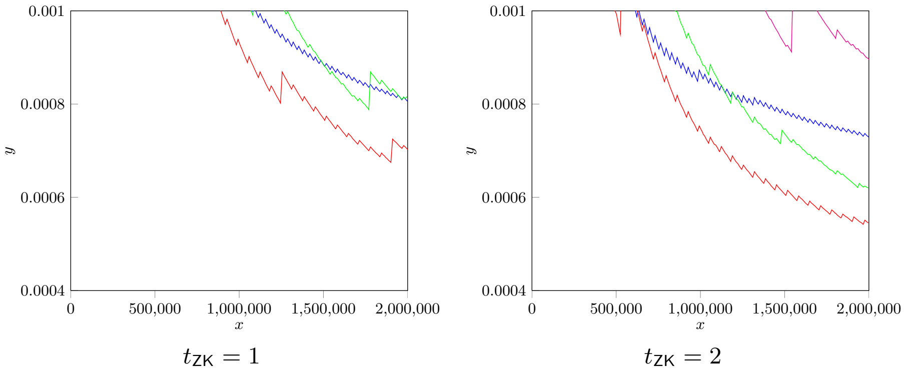

Fig. 6. Average time y to produce a triple given the number of triples that have been produced x for HighGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ and $\mathsf{Snd\_sec} = 128$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} = 8$

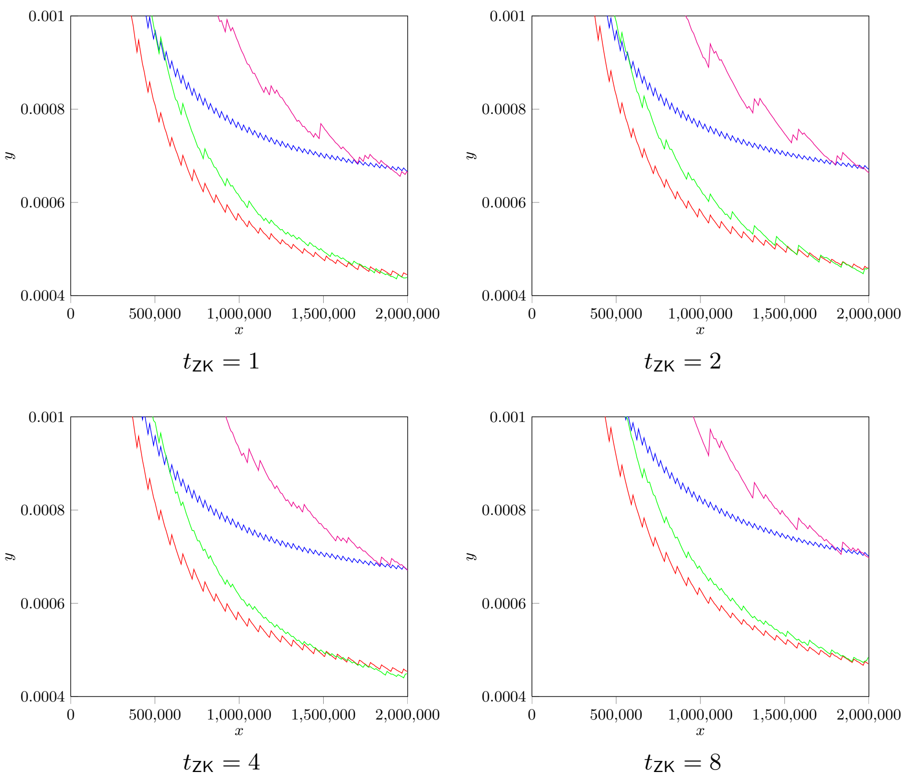

Fig. 7. Average time y to produce a triple given the number of triples that have been produced x for TopGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ and $\mathsf{Snd\_sec} = 128$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} = 8$