Using TopGear in Overdrive: A more efficient ZKPoK for SPDZ

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

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] University of Bristol, Bristol, UK.
daniele.cozzo@kuleuven.be, nigel.smart@kuleuven.be

Abstract. We present a modification to the ZKPoKs used in the HighGear offline protocol for the SPDZ Multi-Party Computation protocol. This modification allows us to both increase the security of the underlying protocols, whilst at the same time maintaining roughly the same performance in terms of memory and bandwidth consumption. The last two being major constraints of the original HighGear protocol. We argue the inefficiency of HighGear means that current implementations of SPDZ use far too low security parameters in a number of places. We show that using TopGear one can select high security parameters for all cases.

1 Introduction

Multi-party computation (MPC) has turned in the last fifteen years from a mainly theoretical endeavour 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 circuit, 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 still the SPDZ protocol [10] from 2012, along with its many improvements such as [9, 17, 18].

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 Beaver triples) and an online phase (which consumes the Beaver triples to enable multiplication to be performed). In this work we will be focusing on the increasing the underlying security offered by the offline phase, whilst maintaining the same performance. The SPDZ protocol, amongst others, has been implemented in the SCALE-MAMBA system [2], which we refer to as a reference implementation for judging our own contribution.

To understand our contribution 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 protocol called BDOZ [3]. The BDOZ protocol used a form of pairwise MAC 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 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. Thus in total $O(n^2)$ ZKPoKs need to be carried out per Beaver triple.

The main contribution of the SPDZ paper [10] over BDOZ was the pairwise MACs were replaced by a global MAC. This was enabled by replacing the linearly homomorphic encryption used in BDOZ by a limited form of Somewhat Homomorphic Encryption (SHE) based on the BGV encryption scheme [6]. Now the pairwise ZKPoKs could be replaced by $O(n)$ ZKPoKs per Beaver triple. Due to the Smart-Vercauteren SIMD packing underlying BGV the implied constant here was relatively low, as a single ZKPoK could be used to prove statements needed for many thousands of Beaver triples.

However, BGV is a lattice based SHE scheme and thus ZKPoKs are relatively costly. In particular the basic underlying Σ-protocol has challenge space $\{0, 1\}$, and thus soundness security of only $1/2$. Not only that, but to provide zero-knowledge, one needs to "blow-up" the parameters proven, thus an adversary can only be shown to prove a statement which is strictly weaker than what an honest party proves. This introduces what is called the *soundness slack* between the honest proven language $\mathbb{L}$ and the adversarially

proven language $\mathbb{L}'$. In particular to obtain statistical zero-knowledge of $2^{-\mathsf{ZK\_sec}}$ the parameters are (roughly) blown up by a factor of $2^{\mathsf{ZK\_sec}}$.

To get around the first of these problems (the low soundness security) SPDZ uses a standard amortization technique [8] to prove U statements at once. This boosts the soundness security from $1/2$ to 2^{-U}, at the expense of introducing slighly more soundness slack, namely $2^{U/2}$. 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 considered too slow. This resulted in two new techniques based on cut-and-choose being introduced in [9]. The first of these techniques 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 technique of [9] provided actively secure ZKPoK which seemed assymptotically more efficient than that provided in [10], but which due to large memory requirements was impossible to implement in practice.

This inability to provide efficient actively secure offline phased based on SHE led to a temporary switch to an OT-based offline phase, called MASCOT [17]. However, in 2018 Keller et al [18] introduced the Overdrive suit of offline protocols. The Overdrive paper revisited the previous work and came to some interesting conclusions. 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 the SPDZ style methodology. Thus an $O(n^2)$ algorithm can beat an $O(n)$ algorithm for small values of n, this was enabled by using in BDOZ the same SIMD packing as was being used for SPDZ. Interestingly this simple optimization had never been tried before on the BDOZ protocol.

The second variant of Overdrive, so called High Gear, was 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. 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 soundess security $\mathsf{Snd\_sec}$ as zero-knowledge indistinguishability. This is not efficient, or 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. Thus a low value for $\mathsf{Snd\_sec}$ is more of a concern 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 [18] 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 [18] is relatively intuitive. We formalize the statement by presenting a generalization of standard Σ-protocols to what we call an n-party Σ-protocol. This allows us to somewhat simplify the presentation of our protocol and also to elaborate on the effect of the various security parameters.

We then present a modified ZKPoK for the HighGear variant of Overdrive, which we denote TopGear. In this variant we treat the soundess security $\mathsf{Snd\_sec}$ and the zero-knowledge security $\mathsf{ZK\_sec}$ seperately. This ZKPoK in its unamortized variant (i.e. only proving one statement at once) uses a larger challenge set than $\{0, 1\}$. In particular, using a Lemma from [4] we are able to use a challenge set of size $2 \cdot N$, where N is the ring dimension. We then amortize this by proving U statements in parallel using a modification of the technique of [8]. This enables us to achieve a soundness security of $\mathsf{Snd\_sec} = U \cdot \log_2(2 \cdot N)$ for the

main zero-knowledge proofs in the offline phase of the SPDZ protocol. Since N is often $32768 = 2^{15}$ we are able to achieve a high soundness security, with a low value of U, i.e. using $U = 8$ we can obtain 128 bits of soundness security. Thus we can use a smaller amount of amortization than in [18], and thus a smaller memory footprint etc. This then allows us to also select higher values of $\mathsf{ZK\_sec}$, if so desired, as this has little affect on performance.

However, we still obtain a soundness slack. Namely a difference between the honestly proven language and the language which we can guarantee a dishonest prover can select their statements from. This distinction comes from two sources. The first source comes 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. [11,12,19]. However, in our case this slack will be removed due to the processing that happens after the ZKPoK is executed (during a modulus switch operation in the homomorphic processing). The second source comes from the use of the larger challenge set. To remove this we need to perform a minor tweak to how the outputs of the ZKPoK are used in the offline phase of the SPDZ protocol.

2 Preliminaries

In this section we outline the details of what we require of the BGV encryption scheme [6]. Most of the details can be found in [6,13–15], although we will only require a variant, which supports circuits of multiplicative depth one.

Notation: We assume that all the parties are probabilistic polynomial time Turing machines. We let $[n]$ denote the interval $[1, \ldots, n]$. If M is matrix then we write $M^{(r,c)}$ for the entry in the r-th row and c-th column. We shall write vectors (usually) in bold, and there elements in non-bold with a subscript, thus $\boldsymbol{x} = (x_i)_{i\in[n]}$, sometimes we will also 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 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 [10] 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 global MAC key $\alpha_i \in \mathbb{F}_p$ and 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 $\sum \gamma_i = \alpha \cdot x$ where $\alpha = \sum_i \alpha_i$. 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 $\mathbf{C}^{\phi(m)}$ 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 $\|x\|_\infty$, where as the ∞-norm in the canonical embedding we will write as $\|x\|_\infty^{\mathsf{can}} = \|\sigma(x)\|_\infty$. By standard inequalities we have $\|x \cdot y\|_\infty^{\mathsf{can}} \leq \|x\|_\infty^{\mathsf{can}} \cdot \|y\|_\infty^{\mathsf{can}}$, $\|x\|_\infty^{\mathsf{can}} \leq \|x\|_1$, $\|x\|_\infty^{\mathsf{can}} \leq \phi(m) \cdot \|x\|_\infty$ and $\|x\|_\infty \leq \|x\|_\infty^{\mathsf{can}}$; with the last two inequalities holding due to our specific choice of cyclotimic 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 then $\|2 \cdot (X^i - X^j)^{-1} \pmod{\phi_m(X)}\|_\infty \leq 1$ for all $0 \leq i, j < 2 \cdot N$ then*

Proof. Given in [4].

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.

Various Distributions: Following [15][Full version, Appendix A.5] and [2] we need 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 following algorithms $\{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SwitchMod}, \mathsf{Dec}\}$. 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.

- $\mathsf{KeyGen}(1^\kappa)$: The secret key $\mathfrak{sk}$ is randomly selected from a distribution with Hamming weight h, i.e. $\mathsf{HWT}(h, N)$, much as in other systems, e.g. HELib [16] and SCALE [2] etc. 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 relinearisation data $(a_{\mathfrak{sk},\mathfrak{sk}^2}, b_{\mathfrak{sk},\mathfrak{sk}^2})$ [7], 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 $\mathcal{R}^2_{q_\ell}$.
- $\underline{\mathsf{SwitchMod}((1, c_0, c_1))}$: 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 $\|c_0 - \mathfrak{sk} \cdot c_1\|^{\mathsf{can}}_\infty$. 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 $\|c_0 - \mathfrak{sk} \cdot c_1\|^{\mathsf{can}}_\infty \le q_0/2$.

We would like a ciphertext (adversarially chosen or not) to decrypt correctly with probability $1 - 2^{-\epsilon}$. In [15] 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. And $c_2 \cdot \sqrt{V_1 \cdot V_2}$ is a similar bound on a produce 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

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

$$= B_{\mathsf{clean}}.$$

Recall this is the average case bound on the noise of honestly generated ciphertexts. In our protocols 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. Thus the need for distributed decryption makes the parameters get a lot larger.

3 n-Prover Σ-Protocols

The Overdrive ZKPoK is an $n+1$ party Σ-protocol between n provers and one verifier[3]. The Overdrive ZKPoK also has a difference, unlike traditional Σ-protocols, between the language used for completeness and the language for soundness; much like the protocols considered in [5][Definition 2.2]. The Overdrive paper does not formalize such Σ-protocols, so our first contribution is to do precisely this.

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 (who could be one or all of the provers) that a given predicate holds between the input values, i.e. $\mathbf{P}(x_1, \ldots, x_n, w_1, \ldots, w_n) = 1$, and that the provers hold knowledge of the said witnesses w_i. The predicate $\mathbf{P}$ defines a language $\mathbb{L}$; namely a binary relation on the pairs $(\boldsymbol{x}, \boldsymbol{w})$.

In the definition below we also utilize another language $\mathbb{L}'$ such that $\mathbb{L} \subset \mathbb{L}'$ defined by a second predicate $\mathbf{P}'$. In the following definition if one sets $n = 1$, $\mathbb{L} = \mathbb{L}'$ and utilizes perfect zero-knowledge then we have the traditional definition of a Σ-protocol.

Definition 3.1. *An n-party Σ-protocol with challenge set $\mathcal{C}$ for the languages $\mathbb{L}$ and $\mathbb{L}'$ is defined as a tuple of PPT algorithms* $(\mathsf{Comm}, \mathsf{Resp}, \mathsf{Verify})$. *The protocol is then executed in the following four phases:*

1. *Each prover* independently *executes the algorithms*

$$(\mathsf{comm}_i, \mathsf{state}_i) \leftarrow \mathsf{Comm}(w_i)$$

   and sends (x_i, comm_i) to the verifier.
2. *The verifier selects a challenge value $c \in \mathcal{C}$ and sends it to each prover.*
3. *The provers execute, again independently, the algorithms*

$$\mathsf{resp}_i \leftarrow \mathsf{Resp}(\mathsf{state}_i, c)$$

   and send resp_i to the verifier.
4. *The verifier accepts if* $\mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i \in [n]}, c) = \mathsf{true}$.

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

Such a protocol should satisfy the following three properties

- **Correctness:** *A set of honest provers should make the verifier accept with probability one if the first predicate* $\mathbf{P}$ *is true.*
- **Special Soundness:** *Two accepting conversations with the same commitment but different challenges, can be passed into a PPT algorithm* $\mathsf{Extract}$ *(called the knowledge extractor) so as to obtain a witness* $(w'_1, \ldots, w'_n)$*, to the second predicate* $\mathbf{P}'$*. In other words given two tuples* $(\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i\in[n]}, c)$ *and* $(\{\mathsf{comm}_i, \mathsf{resp}'_i\}_{i\in[n]}, c')$ *such that*

$$\mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i\in[n]}, c) = \mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}'_i\}_{i\in[n]}, c') = \mathsf{true},$$

  we can obtain

$$(w'_1, \ldots, w'_n) \leftarrow \mathsf{Extract}(\{\mathsf{comm}_i, \mathsf{resp}_i, \mathsf{comm}'_i, \mathsf{resp}'_i\}_{i\in[n]}, c, c')$$

  such that $\mathbf{P}'(x_1, \ldots, x_n, w'_1, \ldots, w'_n) = \mathsf{true}$.
- **Honest Verifier Zero-Knowledge:** *There is a PPT algorithm* $\mathbf{Sim}_A$ *indexed by a set* $A \subset [n]$*, which takes as input an element in the language* $\mathbb{L}$ *and a challenge* $c \in \mathcal{C}$*, and then outputs tuples* $\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i\notin A}$*. We require that for all such* A *the output of* $\mathbf{Sim}_A$ *is statistically indistinguishable from a valid execution of the protocol.*

We first discuss the definition of zero-knowledge. Firstly, the adversary, controlling the players in A, can always learn something about the inputs to the predicate $\mathbf{P}$, given it was true, and his own input. For example if the predicate was $x_1 + x_2 = 2$ and the adversary has the witness $x_1 = 1$, then he knows the other players witness was $x_2 = 1$. We require that the adversary learns no more than he can already deduce from his own input and the predicate being true.

Secondly, since the execution of the commitment and response phases are independent for each player, we only need to look at indistguishability of the distribution of the values $\{c, \{\mathsf{comm}_i, \mathsf{resp}_i\}_{i\notin A}\}$ produced in a valid and a simulated execution of the protocol. What the adversary does cannot affect the zero-knowledge as the adversaries values are independent. Our formalism for HV-ZK is therefore only to allow the simulator to be applied when some provers are adversarial. Unlike in [5] we use a notion of statistical zero-knowledge, whereas [5] uses computational zero-knowledge. This is because our final protocol will be have statistical zero-knowledge. Modifying the definition for other forms of zero-knowledge is straight forward.

The above definitions implies two implied security parameters, which are important in what follows (and in prior treatments of the HighGear ZKPoK have been confusingly merged into one parameter). The first parameter is the soundness parameter $\mathsf{Snd\_sec} = \log_2 |\mathcal{C}|$; this gives the probability $2^{-\mathsf{Snd\_sec}}$ that a dishonest prover (by which we mean all provers are dishonest) can make an honest verifier accept. The second parameter is the zero-knowledge parameter $\mathsf{ZK\_sec}$, which defines the closeness of the distributions of genuine from simulated transcripts. We let the statistical distance between the distributed of valid transcripts and transcripts produced by the simulator be bounded by $2^{-\mathsf{ZK\_sec}}$.

The knowledge extractor above outputs a witness for a predicate $\mathbf{P}'$ which is potentially different from the predicate that honest parties are proving. In traditional Σ-protocols (such as Schnorr's protocol) we have that $\mathbf{P} = \mathbf{P}'$ but in many lattice based protocols these two predicates are distinct. The gap between the two predicates $\#\mathbf{P}'/\#\mathbf{P}$ what we earlier referred to as the "soundness slack".

4 ZKPoK

Our protocol, which we call TopGear, is given in Figure 1 and Figure 2. The protocol is a n-Prover Σ-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 Σ-protocol will be used in the SPDZ protocol. Thus in our description in Figure 1 and Figure 2 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 [10], so we do not discuss its implementation further.

Protocol Π_{ZKPoK}: Commitment, Challenge and Response Phases

The protocol is parametrized by an integer parameter U and a $\mathsf{flag} \in \{\mathsf{Diag}, \perp\}$.

INPUT: Each (honest) party P_i enters U plaintexts $\boldsymbol{m}_i \in \mathcal{R}_p^U$ (considered as elements of $\mathcal{R}_{q_1}^U$) and U randomness triples $R_i \in \mathcal{R}_{q_1}^{U\times 3}$, where each row of R_i $(r_i^{(j,1)}, r_i^{(j,2)}, r_i^{(j,3)})$ is generated from $\mathsf{RC}\left(\sigma^2, 0.5, N\right)$. For honest P_i we therefore have, for all $j \in [U]$, $\|\boldsymbol{m}_i^{(j)}\|_\infty \leq \frac{p}{2}$ and $\|r_i^{(j,k)}\|_\infty \leq \rho_k$, where $\rho_1 = \rho_2 = 20$ and $\rho_3 = 1$. We write $V = 2 \cdot U - 1$.

Commitment Phase: Comm

1. P_i computes the BGV encryptions by applying the BGV encryption algorithm to the j plaintext/randomness vectors in turn to obtain $C_i \leftarrow \mathsf{Enc}(\boldsymbol{m}_i, R_i; \mathfrak{pk}) \in \mathcal{R}_{q_1}^{U\times 2}$.
2. The players broadcast C_i.
3. Each P_i samples V pseudo-plaintexts $\boldsymbol{y}_i \in \mathcal{R}_{q_1}^V$ and pseudo-randomness vectors $S_i = (s_i^{(j,k)}) \in \mathcal{R}_{q_1}^{V\times 3}$ such that, for all $j \in [V]$, $\|\boldsymbol{y}_i^{(j)}\|_\infty \leq 2^{\mathsf{ZK\_sec}-1} \cdot p$ and $\|s_i^{(j,k)}\|_\infty \leq 2^{\mathsf{ZK\_sec}} \cdot \rho_k$.
4. Party P_i computes $A_i \leftarrow \mathsf{Enc}(\boldsymbol{y}_i, S_i; \mathfrak{pk}) \in \mathcal{R}_{q_1}^{V\times 2}$.
5. The players broadcast $\mathsf{comm}_i \leftarrow A_i$.

Challenge Phase: Chall

1. Parties call $\mathcal{F}_{\mathsf{Rand}}$ to obtain a random vector $\boldsymbol{e} = (e_i)$ such that $\boldsymbol{e} \in \left\{\left\{X^i\right\}_{i=0\ldots,2\cdot N-1}\right\}^U$ if $\mathsf{flag} = \perp$ and $\boldsymbol{e} \in \{0,1\}^U$ if $\mathsf{flag} = \mathsf{Diag}$.

Response Phase: Resp

1. Parties define the $V \times U$ matrix $M_{\boldsymbol{e}}$ where

$$M_{\boldsymbol{e}}^{(k,l)} = \begin{cases} e_{k-l+1} \text{ if } 1 \leq k-l+1 \leq U \\ 0 \qquad \text{otherwise} \end{cases}$$

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

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

Protocol Π_{ZKPoK}: Verification Phase

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 $D = A + M_{\boldsymbol{e}} \cdot C$, and then whether the following inequalities hold, for $j \in [V]$,

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

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

Figure 2. Protocol for global proof of knowledge of a set of ciphertexts: Part II

To understand its workings and security, first we give the two languages and then a security proof. We present our proofs in the simpler case where no additional predicate needs to be proved about the ciphertexts. In the SPDZ protocol [10], at one stage ciphertexts need to be proved to be "Diagonal", namely each plaintext slot component contains the same element. Adding this additional requirement into our protocols below can be done using the standard method given in [10]. However, this results in a reduced soundness security parameter for the underlying protocol. Later we shall see that this does not matter *in practice* in this *specific* case.

The Honest Language: In an honest execution of the SPDZ offline phase each party P_i has as input a set of U ciphertexts given by, for $j \in [U]$,

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

Party P_i wishes to keep the values $w_i = (m_i^{(j)}, r_i^{(j,1)}, r_i^{(j,2)}, r_i^{(j,3)})_{j=1}^U$ private, where as the ciphertext values $x_i = (\mathfrak{ct}_{i,j})_{j=1}^U$ are public. The proof aims to prove something about the sum of these values, when summed over the players. We write the vector of player P_i's plaintexts as $\boldsymbol{m}_i$, the $U \times 3$ matrix of its random values as R_i and the resulting $U \times 2$ elements of $\mathcal{R}_{q_1}$ giving the ciphertexts as C_i. By abuse of notation we relate these via the equation $C_i \leftarrow \mathsf{Enc}(\boldsymbol{m}_i, R_i; \mathfrak{pk})$. Thus the proof argues about the following values over $\mathcal{R}$ (and not $\mathcal{R}_{q_1}$), $\boldsymbol{m} = \sum_{i=1}^n \boldsymbol{m}_i$, $R = \sum_{i=1}^n R_i$ and $C = \sum_{i=1}^n C_i$. Given the above notation, for honest provers the language we define is

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

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 discusses a joint statement about all players inputs (i.e. there sum). In the notation for the language we abuse notation by using Enc as simply a procedure irrespective of the distributiosn of the input variables (as $\boldsymbol{m}^{(j)}$ and $R^{(j,k)}$ are now elements in $\mathcal{R}$ and not necessarily in the correct domain). Thus we mean simply apply the equations

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

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

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

Notice, how not only have the bounds increased on the right, but also the values within the norms have also increased by 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 $|2 \cdot w_i| < 2^{\mathsf{ZK\_sec}+U/2} \cdot B$, where U is our amortization parameter. This explosion in the proved bounds, $2^{\mathsf{ZK\_sec}+U/2}$, 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 $\mathbb{L}'$, corresponding to the factor of two in Lemma 2.2 arising in the inequality on the left. However, this complication is then 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. *The protocol in Figure 1 and Figure 2 is an n-party Σ protocol in the sense of Definition 3.1 for the languages $\mathbb{L}$ and $\mathbb{L}'$. The statistical distance between* the coefficients *of the ring elements in an honest and simulated transcripts is bounded by $2^{-\mathsf{ZK\_sec}}$.*

The probability of an adversary being able to produce an invalid proof is given by

- 2^{-U} *if* $\mathsf{flag} = \mathsf{Diag}$.
- $(2 \cdot N)^{-U}$ *if* $\mathsf{flag} = \perp$.

Proof. In the proof we concentrate on the case $\mathsf{flag} = \perp$, as the case $\mathsf{flag} = \mathsf{Diag}$ is (roughly) the same.

Correctness: In proving the correctness property all the parties are assumed to be honest. We show that if this is the case then the final output passes the check. The equality $D = A + M_{\boldsymbol{e}} \cdot C$ follows at once from the fact that everything is defined as a linear function of their arguments and the BGV encryption function, in particular, is linear and works componentwise.

As for the bounds, verification is carried out according to the honest-prover language $\mathbb{L}$, as the parties are honest and thus produce elements with the right bounds. By repeatedly applying Lemma 2.3 one has, and using the fact that $U \leq 2^{\mathsf{ZK\_sec}}$ in practice,

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

Honest verifier zero-knowledge: Following our definition of n-prover Σ-protocol we give in Figure 3 a simulator, parametrized by a set of adversaries A, which produces transcripts. It is clear that the statistical difference in the distribution of the *coefficients* of the ring elements in $\boldsymbol{z}_i$ and T_i for $i \notin A$ in a real execution and the simulated execution can be bounded by Lemma 2.1 by $2^{-\mathsf{ZK\_sec}}$.

The Simulator $\mathbf{Sim}_A$

If $\mathsf{flag} = \perp$ then the input is a challenge value $\boldsymbol{e} \in \left\{\{X^i\}_{i=0,\ldots,2\cdot N-1}\right\}^U$ otherwise the input is a vector $\boldsymbol{e} \in \{0,1\}^U$.

1. For all $i \notin A$ do
   (a) Sample $\boldsymbol{z}_i \in \mathcal{R}_{q_1}^V$ such that, for $j \in [V]$, we have $\|\boldsymbol{z}_i^{(j)}\|_\infty \leq 2^{\mathsf{ZK\_sec}} \cdot p$.
   (b) Sample $T_i \in \mathcal{R}_{q_1}^{V \times 3}$ such that, for $j \in [V]$ and $k = 1, 2, 3$, we have $\|T_i^{(j,k)}\|_\infty \leq 2 \cdot 2^{\mathsf{ZK\_sec}} \cdot \rho_k$.
   (c) Set $\mathsf{resp}_i \leftarrow (\boldsymbol{z}_i, T_i)$.
   (d) Compute $\mathsf{comm}_i = A_i \leftarrow \mathsf{Enc}(\boldsymbol{z}_i, T_i; \mathfrak{pk}) - M_{\boldsymbol{e}} \cdot C_i$.
2. Output $(\mathsf{comm}_i, \mathsf{resp}_i)_{i \notin A}$.

Figure 3. Simulator $\mathbf{Sim}_A$ for our protocol

Special soundness: Let the two accepting transcripts be $(\{A_i\}_{i\in[n]}, \boldsymbol{e}, \{\boldsymbol{z}_i, T_i\}_{i\in[n]})$ and $(\{A_i\}_{i\in[n]}, \boldsymbol{e}', \{\boldsymbol{z}_i', T_i'\}_{i\in[n]})$. We need to show we can recover a set of witnesses $(\boldsymbol{m}_i, R_i)_{i\in[n]}$ such that the sum $\boldsymbol{m} = \sum \boldsymbol{m}_i$ and $R = \sum R_i$ satisfies the bounds in the language $\mathbb{L}'$. The proof relies crucially on the shape of the matrix $M_{\boldsymbol{e}}$ generated

in the proof, which is

$$M_{\boldsymbol{e}} = \begin{pmatrix} e_1 & 0 & & & \dots & 0 \\ e_2 & e_1 & 0 & & \dots & 0 \\ e_3 & e_2 & e_1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \vdots \\ e_U & e_{U-1} & e_{U-2} & \dots & & e_1 \\ 0 & e_U & e_{U-1} & & \dots & e_2 \\ 0 & 0 & e_U & e_{U-1} & \dots & e_2 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & & 0 & e_U & e_{U-1} \\ 0 & \dots & & \dots & 0 & e_U \end{pmatrix}$$

Since both the transcripts accept, we obtain a system of equations

$$(M_{\boldsymbol{e}} - M_{\boldsymbol{e}'}) \cdot C = D - D'$$

where $D = \sum_{i\in[n]} \mathsf{Enc}(\boldsymbol{z}_i, T_i; \mathfrak{pk})$, $D' = \sum_{i\in[n]} \mathsf{Enc}(\boldsymbol{z}'_i, T'_i; \mathfrak{pk})$ and $C = \sum C_i$.

As the challenge vectors are distinct, there is an index j such that $e_j \neq e'_j$. Take the *largest* of these indices and consider the submatrix obtained from $M_{\boldsymbol{e}} - M_{\boldsymbol{e}'}$ by taking the rows from j to $j + U/2 - 1$, and columns 1 to $U/2$. This is an upper triangular $U/2 \times U/2$ matrix, whose diagonal elements are defined by $e_j - e'_j$, which we will denote by $\mathcal{U}$. This matrix gives rise to equations in the first $U/2$ ciphertexts $\mathfrak{ct}_1, \dots, \mathfrak{ct}_{U/2}$,

$$\mathcal{U} \cdot \begin{pmatrix} \mathfrak{ct}_1 \\ \dots \\ \mathfrak{ct}_{U/2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}_j \\ \dots \\ \boldsymbol{d}_{j+U/2-1} \end{pmatrix}.$$

This turns into $2 \cdot (U/2)$ equations in the unknowns (inherent in the $\mathfrak{ct}_1$) namely $\boldsymbol{m}^{(\ell)}$ and $R^{(\ell,k)}$ for $\ell = 1, \dots, U/2$ and $k = 1, 2, 3$, in terms of the known values (inherent in the $\boldsymbol{d}_j, \dots, \boldsymbol{d}_{j+U/2-1}$) namely $T^{(\ell',k)}$, $T'^{(\ell',k)}$, $\boldsymbol{z}^{(\ell')}$, $\boldsymbol{z}'^{(\ell')}$ for $\ell' = j, \dots, j + U/2 - 1$ and $k = 1, 2, 3$. Equating suitable coefficients in the encryption equation we can solve for the variables using back substitution. Using Lemma 2.2 and the bounds on the known values (from the successful acceptance of the two transcripts) we obtain the bounds, for $j = 1, \dots, U/2$ and $k = 1, 2, 3$,

$$\|2 \cdot \boldsymbol{m}^{(\ell)}\|_\infty \leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot \frac{p}{2},$$
$$\|2 \cdot R^{(\ell,k)}\|_\infty \leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot \rho_k.$$

To obtain similar bounds for the same variables with $\ell = U/2 + 1, \dots, U$ we return to the original matrix equation and now take the *smallest* index j such that $e_j \neq e'_j$. Then we take the $U/2 \times U/2$ submatrix consisting of rows $U/2 + j$ to $j + U/2 - 1$ and columns $U/2 + 1, \dots, U$. This is a lower triangular matrix $\mathcal{L}$, giving rise to an equation

$$\mathcal{L} \cdot \begin{pmatrix} \mathfrak{ct}_{U/2+1} \\ \dots \\ \mathfrak{ct}_U \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}_{U/2+j} \\ \dots \\ \boldsymbol{d}_{j+U-1} \end{pmatrix}.$$

We now solve the linear system in the same way, obtaining solutions such that for $j = U/2 + 1, \dots, U$ and $k = 1, 2, 3$,

$$\|2 \cdot \boldsymbol{m}^{(\ell)}\|_\infty \leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot \frac{p}{2},$$
$$\|2 \cdot R^{(\ell,k)}\|_\infty \leq 2 \cdot n \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot \rho_k.$$

From these solutions for the sums it is then easy to obtain witnesses for each individual players contribution which produces the given sum.

Given these extracted sums we can now produce witnesses for the specific players inputs in any way we choose (as long as they add up to the correct sum). Recall that the language does not require that the witness produce the encryptions C_i entered by the players, only that the sum C is correct.

5 SPDZ Offline Phase

The application of the previously considered ZKPoK is to the offline phase of the SPDZ protocol. We briefly recap on this here and give the necessary adaption from the HighGear protocol of [18]. Recall 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 selected 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, at a high level, 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 $\mathcal{R}_p$ The elements 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

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

This is then passed to a distributed decryption protocol which gives to each party a share $c_i \in \mathbb{F}_p$ of the underlying plaintext behind $\mathfrak{ct}_c$. The associated shared MAC values, $\gamma_i[a], \gamma_i[b]$ and $\gamma_i[c]$, to obtain the full sharing $\langle a\rangle, \langle b\rangle, \langle c\rangle$ are obtained in a similar manner.

The problem is that dishonest parties could produce invalid ciphertexts, which would result in either selective failure attacks or information leakage during the distributed decryption procedure. Thus in the SPDZ protocol [10] each ciphertext $\mathfrak{ct}_{a_i}$ needs to be accompanied by a ZKPoK that it is not too far from being a valid ciphertext. Essentially we use the ZKPoK to bound the noise term associated to even adversarial ciphertexts, and then we use this bound to ensure that all ciphertexts will validly encrypt (by using this bound to derive the parameters for the BGV encryption scheme). This is done for the ciphertexts encrypting a_i and the ciphertext encrypting the shares of the MAC key α_i. Where we execute U such proofs in parallel so as to make use of good amortization in the ZKPoK.

In HighGear from [18] it is noticed that the ciphertexts $\mathfrak{ct}_{a_i}$ are only ever used in a sum (as in Equation 1). Thus one can replace the n ZKPoKs for $\mathfrak{ct}_{a_i}$ by a single ZKPoK for the sum $\mathfrak{ct}_a = \mathfrak{ct}_{a_1} \boxplus \cdots \boxplus \mathfrak{ct}_{a_n}$. It is this strategy we adapted in the previous section, to enable smaller values of U to be used. However, our adaption comes at the expense of us having a bound on the noise of not the original ciphertext sum $\mathfrak{ct}_a$ but of the sum $2 \cdot \mathfrak{ct}_a = \mathfrak{ct}_a \boxplus \mathfrak{ct}_a$. Luckily this does not affect our final application as we can simply multiply the underlying plaintexts by two and continue as normal. The modifications are explained in Figure 4 for the case of triple production. The modifications to obtain other forms of offline data such as those in [9] are immediate.

Note we repeat the proofs for the ciphertexts encrypting α_i a total of $\mathsf{Snd\_sec}/U$ times. This is because we only have soundness parameter U for the proofs when $\mathsf{flag} = \mathsf{Diag}$. However, this is not an issue since the amortization does not produce any advantages here when we are trying to prove a single fixed ciphertext has been validly created. We select U so that the ZKPoKs for the a'_i, b'_i and f'_i need only be repeated once for the desired soundness security parameter.

In the protocol in Figure 4 we have utilized the more efficient Distributed Decryption protocol from HighGear to obtain the MAC shares, and have merged in the $\mathsf{ReShare}$ protocol, [9][Figure 11], 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 [9, 10], all that changes is the bounds 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. We need to obtain worst case bounds on the value of $\|c_0 - \mathfrak{sk} \cdot c_1\|_\infty^{\mathsf{can}}$ for such ciphertexts, comparable to the average case bound on honestly generated fresh ciphertexts given

Π_{offline}

Init:

- Each party generates $\alpha'_i \in \mathbb{F}_p$.
- Each party sets $\alpha_i \leftarrow 2 \cdot \alpha'_i$.
- Each party computes an encryption $\mathfrak{ct}_{\alpha'_i}$ of the plaintext consisting of α'_i in each slot position.
- The resulting ciphertexts, and associated random coins, are passed to Protocol Π_{ZKPoK} from Figure 1 and Figure 2 using $\mathsf{flag} = \mathsf{Diag}$, where we use the same input for all U inputs of the protocol Π_{ZKPoK}. The ZKPoK is repeated $\mathsf{Snd\_sec}/U$ times to achieve the desired soundness security parameter.
- The parties set $\mathfrak{ct}_\alpha \leftarrow 2 \cdot (\mathfrak{ct}_{\alpha'_1} \boxplus \cdots \boxplus \mathfrak{ct}_{\alpha'_n})$.

Triples:

- Each party generates $\boldsymbol{a}'_i, \boldsymbol{b}'_i, \boldsymbol{f}'_i \in (\mathbb{F}_p)^{\phi(m)}$ and the associated elements $a'_i, b'_i, f'_i \in \mathcal{R}_p$.
- Each party sets $\boldsymbol{a}_i \leftarrow 2 \cdot \boldsymbol{a}'_i$, $\boldsymbol{b}_i \leftarrow 2 \cdot \boldsymbol{b}'_i$. and $\boldsymbol{f}_i \leftarrow 2 \cdot \boldsymbol{f}'_i$.
- Each party computes an encryption $\mathfrak{ct}_{a'_i}$, $\mathfrak{ct}_{b'_i}$ and $\mathfrak{ct}_{f'_i}$ of a'_i, b'_i and f'_i.
- The resulting ciphertexts, and associated random coins, are passed to Protocol Π_{ZKPoK} from Figure 1 and Figure 2 using $\mathsf{flag} = \perp$.
- The parties set $\mathfrak{ct}_a \leftarrow 2 \cdot (\mathfrak{ct}_{a'_1} \boxplus \cdots \boxplus \mathfrak{ct}_{a'_n})$, $\mathfrak{ct}_b \leftarrow 2 \cdot (\mathfrak{ct}_{b'_1} \boxplus \cdots \boxplus \mathfrak{ct}_{b'_n})$ and $\mathfrak{ct}_f \leftarrow 2 \cdot (\mathfrak{ct}_{f'_1} \boxplus \cdots \boxplus \mathfrak{ct}_{f'_n})$.
- The parties compute $\mathfrak{ct}_c \leftarrow \mathfrak{ct}_a \odot \mathfrak{ct}_b$ and $\mathfrak{ct}_{c+f} \leftarrow \mathfrak{ct}_c \boxplus \mathfrak{ct}_f$.
- Using the distributed decryption operation of the BGV scheme the parties obtain in the clear $c + f$.
- Party one sets $c_1 \leftarrow (c + f) - f_1$ and party $i \neq 1$ sets $c_i \leftarrow -f_i$.
- The parties compute a fresh encryption of $\boldsymbol{c}$ via $\mathfrak{ct}'_c \leftarrow \mathsf{Enc}(c + f, \mathbf{0}; \mathfrak{pk}) - \mathfrak{ct}_f$ with some default random coins $\mathbf{0}$.
- The parties compute $\mathfrak{ct}_{\alpha \cdot a} \leftarrow \mathfrak{ct}_\alpha \odot \mathfrak{ct}_a$, $\mathfrak{ct}_{\alpha \cdot b} \leftarrow \mathfrak{ct}_\alpha \odot \mathfrak{ct}_b$ and $\mathfrak{ct}_{\alpha \cdot c} \leftarrow \mathfrak{ct}_\alpha \odot \mathfrak{ct}'_c$.
- The MAC values $\gamma_i[a]$, $\gamma_i[b]$, $\gamma_i[c]$ are obtained by applying the $\mathsf{DistDec}$ protocol in Figure 14 from [18] and mapping the associated polynomials into the slot representation.

Figure 4. TopGear Variant of the SPDZ Offline Phase

earlier. 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 $(m_i, r_i^{(1)}, r_i^{(2)}, r_i^{(3)})$ were enterred into our ZKPoK. From the soundness of our ZKPoK we can guarantee that even adversarially produced inputs must satisfy

$$\left\| 2 \cdot \sum_{i \in [n]} m_i \right\|_\infty \leq 2^{\mathsf{ZK\_sec}+U/2+1} \cdot n \cdot p,$$
$$\left\| 2 \cdot \sum_{i \in [n]} r_i^{(k)} \right\|_\infty \leq 2^{\mathsf{ZK\_sec}+U/2+2} \cdot n \cdot \rho_k.$$

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

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

$$
\begin{aligned}
&\quad + \phi(m) \cdot 2 \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot n \cdot 20 \\
&\quad + \mathfrak{c}_1 \cdot \sqrt{h} \cdot \phi(m) \cdot 2 \cdot 2^{\mathsf{ZK\_sec}+U/2+1} \cdot n \cdot 20\Big) \\
&= \phi(m) \cdot 2^{\mathsf{ZK\_sec}+U/2+2} \cdot n \cdot p \cdot \Big(\frac{41}{2} + \mathfrak{c}_1 \cdot \sigma \cdot \phi(m)^{1/2} + 20 \cdot \mathfrak{c}_1 \cdot \sqrt{h}\Big) \\
&= B_{\mathsf{clean}}^{\mathsf{dishonest}}.
\end{aligned}
$$

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 $\mathsf{Snd\_sec}$ 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 $\mathsf{ZK\_sec}$ and $\mathsf{DD\_sec}$, 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 [18] the two security parameters are set to be equal, i.e. $\mathsf{ZK\_sec} = \mathsf{Snd\_sec}$. In practice the value of $\mathsf{Snd\_sec}$ 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 $\mathsf{Snd\_sec}$ 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 $\mathsf{Snd\_sec}$ 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 $\mathsf{ZK\_sec}$ and $\mathsf{DD\_sec}$. 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. So whilst a high statistical distance is not as worrisome as a high probability of cheating, picking $\mathsf{ZK\_sec}$ and $\mathsf{DD\_sec}$ at low values we feel is also not desirable. Hence, after demonstrating the effect of our new protocol with respect to the increase in $\mathsf{Snd\_sec}$, 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 $\mathsf{ZK\_sec}$, $\mathsf{Snd\_sec}$, $\mathsf{DD\_sec}$ and computational security parameter κ, and two parties[4]. 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[5]. A row which with values of $\star$ in the $\mathsf{ZK\_sec}$ columns means that the values do not change when one varies this parameter is 40 or 80.

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 $2^{\mathsf{ZK\_sec}+U/2+2}$ (where $U = \mathsf{Snd\_sec}$ for HighGear and $U = \mathsf{Snd\_sec}/\log_2(2 \cdot N)$ 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 impementation 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 to support other MPC operations (such as fixed point operations). We first baselined the implementation

[4] 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.

[5] 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	$\log_2 p_0$	$\log_2 p_1$
64	80	40	40	40	8192	177	114	8192	3	177	114
64	80	40	80	40	8192	177	114	8192	6	177	114
64	80	40	128	40	8192	177	114	8192	10	177	114
64	80	40	80	80	8192	177	144	8192	6	177	114
64	80	40	128	80	16384	177	174	8192	10	177	165
64	80	80	40	40	16384	218	163	16384	3	218	163
64	80	80	80	40	16384	218	163	16384	6	218	163
64	80	80	128	40	16384	218	163	16384	9	218	163
64	80	80	80	80	16384	218	163	16384	6	218	163
64	80	80	128	80	16384	218	173	16384	9	218	163
64	80	128	40	$\star$	16384	266	205	16384	3	266	205
64	80	128	80	$\star$	16384	266	205	16384	6	266	205
64	80	128	128	$\star$	16384	266	205	16384	9	266	205
64	128	40	40	40	16384	178	123	16384	3	178	123
64	128	40	80	40	16384	178	123	16384	6	178	123
64	128	40	128	40	16384	178	133	16384	9	178	123
64	128	40	80	80	16384	178	153	16384	6	178	123
64	128	40	128	80	16384	178	173	16384	9	178	123
64	128	80	40	40	16384	218	163	16384	3	218	163
64	128	80	80	40	16384	218	163	16384	6	218	163
64	128	80	128	40	16384	218	163	16384	9	218	163
64	128	80	80	80	16384	218	163	16384	6	218	163
64	128	80	128	80	16384	218	173	16384	9	218	163
64	128	128	40	$\star$	32768	266	205	32768	3	266	205
64	128	128	80	$\star$	32768	266	205	32768	5	266	205
64	128	128	128	$\star$	32768	266	205	32768	8	266	205
64	128	128	128	128	32768	266	225	32768	8	266	205
128	80	40	40	$\star$	16384	305	186	16384	3	305	186
128	80	40	80	$\star$	16384	305	186	16384	6	305	186
128	80	40	128	$\star$	16384	305	186	16384	9	305	186
128	80	80	40	$\star$	16384	345	226	16384	3	345	226
128	80	80	80	$\star$	16384	345	226	16384	6	345	226
128	80	80	128	$\star$	16384	345	226	16384	9	345	226
128	80	128	40	$\star$	16384	393	268	16384	3	393	268
128	80	128	80	$\star$	16384	393	268	16384	6	393	268
128	80	128	128	$\star$	16384	393	268	16384	9	393	268
128	128	40	40	$\star$	32768	306	185	32768	3	306	185
128	128	40	80	$\star$	32768	306	185	32768	5	306	185
128	128	40	128	$\star$	32768	306	185	32768	8	306	185
128	128	80	40	$\star$	32768	346	225	32768	3	346	225
128	128	80	80	$\star$	32768	346	225	32768	5	346	225
128	128	80	128	$\star$	32768	346	225	32768	8	346	225
128	128	128	40	$\star$	32768	394	277	32768	3	394	277
128	128	128	80	$\star$	32768	394	277	32768	5	394	277
128	128	128	128	$\star$	32768	394	277	32768	8	394	277
128	128	128	128	128	32768	394	277	32768	8	394	277

Table 1. SHE parameters sizes for various security parameters in HighGear and TopGear (two parties). With $\mathsf{DD\_sec}, \mathsf{ZK\_sec} \leq \mathsf{Snd\_sec}$ and $\mathsf{DD\_sec}, \mathsf{ZK\_sec}, \mathsf{Snd\_sec} \in \{40, 80, 128\}$. The light grayed rows show the default parameters used in SCALE-MAMBA. 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.

in SCALE-MAMBA of HighGear against the implementation reported in [18]. The experiments in [18] were executed on i7-4790 and i7-3770S CPUs, compared to our experiments which utilzied 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 [18] was 0.3 milliseconds.

Keller et al [18], 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 utilizes a ciphertext modulus of 541 bits. In this setting Keller et al [18] 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 $V = 2 \cdot U - 1$ auxillary ciphertexts per player. Hence, the total nunber of bits required to process a ZKPoK is at least

$$3 \cdot U \cdot n \cdot \phi(m) \cdot \log_2(p_0 \cdot p_1).$$

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 $U = \mathsf{Snd\_sec}/\log_2(2 \cdot N)$. 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 $\log_2(2 \cdot N)$. 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 very 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

DD_sec = ZK_sec = Snd_sec = 40

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

DD_sec = ZK_sec = 40, 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	13	21
2	8	11	13	23
4	9	12	16	24
8	15	16	20	27

DD_sec = ZK_sec = Snd_sec = 40

	t_{ZK}			
t_{Tr}	1	2	4	8
1	11	16	26	47
2	12	16	29	53
4	12	17	32	54
8	16	20	33	54

DD_sec = ZK_sec = 40, 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 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 you run 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.

We immediately see that although, asymptotically in the security parameter, the computational difference between HighGear and TopGear should be the same (in terms of the amount of work needed to be done per triple), the actual performance is much better for TopGear. This is because of both the reduced memory foot and in addition the fact that the ration of $V = 2 \cdot U - 1$ to U in HighGear is larger than that in TopGear (79/40 vs 5/3 for this security setting). Thus in practice the actual work needed per triple is less in TopGear than in HighGear for (non-asymptotic) values of Snd_sec. This is born out in our experiments. In both security settings the TopGear protocol works best, when we have $t_{ZK}, t_{Tr} \in \{2, 4\}$. We find we achieve a better throughput with TopGear in these two security setting. Indeed at the higher security level we are only marginally less efficient using TopGear, than we would be using HighGear using the lower security setting.

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	1494	1515	1519	1492
2	1862	2487	2403	2309
4	2487	2439	2463	2481
8	2079	1848	1683	1675

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

t_{Tr} \ t_{ZK}	1	2	4	8
1	1464	1481	1468	1400
2	2132	2207	2150	2057
4	1605	2183	2184	2008
8	1406	1587	1612	1510

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

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

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 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.

t_{Tr} \ t_{ZK}	1	2	4	8
1	11	16	27	49
2	13	20	27	50
4	13	20	30	51
8	17	24	35	58

Memory Consumption

t_{Tr} \ t_{ZK}	1	2	4	8
1	1254 (83)	1275 (79)	1273 (81)	1216 (91)
2	1814 (121)	1757 (74)	1795 (81)	1748 (88)
4	1270 (99)	1798 (95)	1689 (78)	1620 (86)
8	1206 (123)	1283 (98)	1231 (84)	1197 (78)

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 settings using HighGear.

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 currently achieves using much weaker

security levels (comparing the tables 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.

	t_{ZK}			
t_{Tr}	1	2	4	8
1	13	19	31	58
2	13	21	32	60
4	14	22	34	66
8	18	26	44	66

Memory Consumption

	t_{ZK}			
t_{Tr}	1	2	4	8
1	994 (66)	1035 (64)	943 (60)	1001 (75)
2	1386 (93)	1377 (58)	1466 (66)	1314 (66)
4	1061 (83)	1371 (73)	1366 (63)	1281 (68)
8	977 (100)	1023 (78)	1028 (70)	980 (64)

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 settings using HighGear.

Acknowledgments

This work has been supported 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 and MAMBA documentation, 2018.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.

9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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 5 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 6 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 7 and Figure 8 we provide similar graphs of the throughput for HighGear and TopGear in our high security setting $\mathsf{Snd\_sec} = 128$.

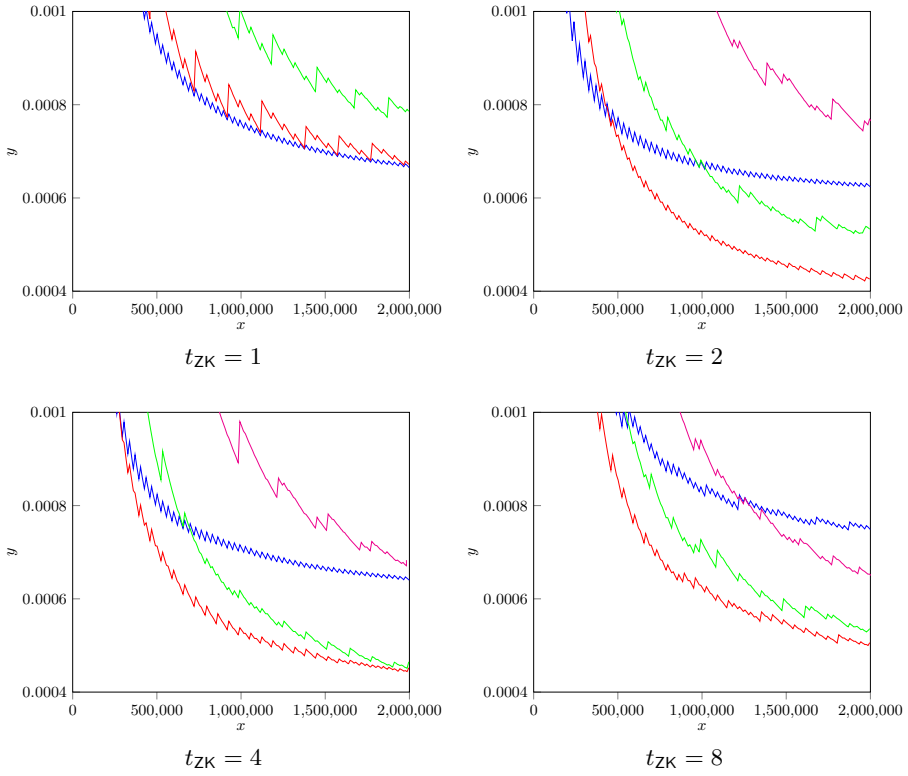

Fig. 5. Average time y to produce a triple given the number of triples that have been produced x for HighGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} =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

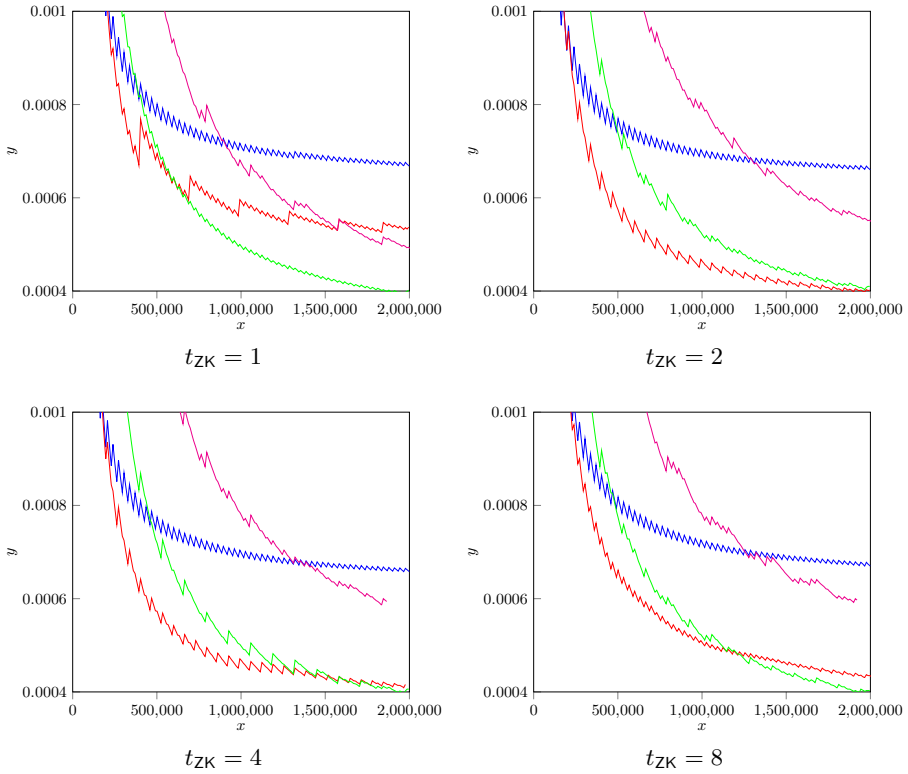

Fig. 6. Average time y to produce a triple given the number of triples that have been produced x for TopGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = \mathsf{Snd\_sec} = 40$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}} = 8$

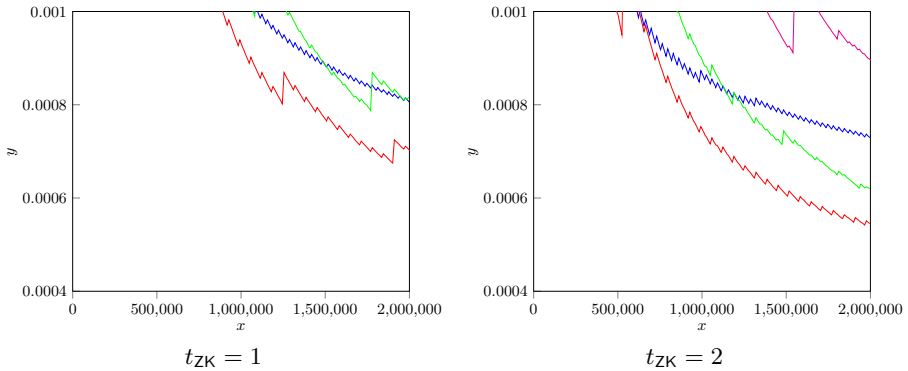

Fig. 7. Average time y to produce a triple given the number of triples that have been produced x for HighGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ and $\mathsf{Snd\_sec} = 128$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}}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

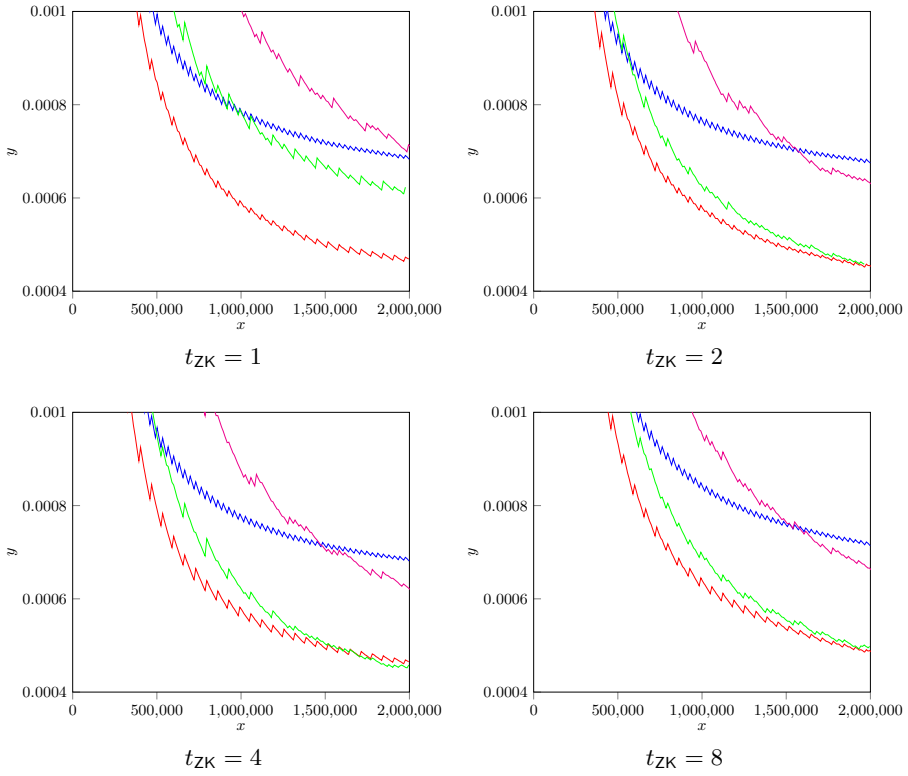

Fig. 8. Average time y to produce a triple given the number of triples that have been produced x for TopGear with parameters $\mathsf{DD\_sec} = \mathsf{ZK\_sec} = 40$ and $\mathsf{Snd\_sec} = 128$.
Blue $t_{\mathsf{Tr}} = 1$, Red $t_{\mathsf{Tr}} = 2$, Green $t_{\mathsf{Tr}} = 4$, Magenta $t_{\mathsf{Tr}}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