Overdrive: Making SPDZ Great Again

Marcel Keller, Valerio Pastro, and Dragos Rotaru*

University of Bristol, Yale University
m.keller@bristol.ac.uk, vpastro86@gmail.com, dragos.rotaru@bristol.ac.uk

Abstract. SPDZ denotes a multiparty computation scheme in the preprocessing model based on somewhat homomorphic encryption (SHE) in the form of BGV. At CCS '16, Keller et al. presented MASCOT, a replacement of the preprocessing phase using oblivious transfer instead of SHE, improving by two orders of magnitude on the SPDZ implementation by Damgård et al. (ESORICS '13). In this work, we show that using SHE is faster than MASCOT in many aspects:

1. We present a protocol that uses semi-homomorphic (addition-only) encryption. For two parties, our BGV-based implementation is 6 times faster than MASCOT on a LAN and 20 times faster in a WAN setting. The latter is roughly the reduction in communication.
2. We show that using the proof of knowledge in the original work by Damgård et al. (Crypto '12) is more efficient in practice than the one used in the implementation mentioned above by about one order of magnitude.
3. We present an improvement to the verification of the aforementioned proof of knowledge that increases the performance with a growing number of parties, doubling it for 16 parties.

Keywords: Multiparty computation, somewhat homomorphic encryption, BGV, zero-knowledge proofs of knowledge

1 Introduction

Multi-party computation (MPC) allows a set of parties to jointly compute a function over their inputs while keeping them private. In the last decade MPC has developed from a largely theoretical field to a practical one where many applications have been developed on top of it [?,?]. This is mostly due to the rise of compilers which translate high-level code to secure branching, additions and multiplications on secret data [?,?,?,?].

* The work in this paper was partially supported by EPSRC via grants EP/M012824 and EP/N021940/1, by the European Union's H2020 Programme under grant agreement number ICT-644209, and 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. The second author conducted this work while working as a post-doctoral researcher at Yale University, supported by NSF grants CNS-1562888, 1565208, and DARPA SafeWare W911NF-15-C-0236.

A high number of applications require to evaluate an arithmetic circuit (over the integers or modulo p) due to the easiness of expressing them rather than performing bitwise operations in a binary circuit. This is especially true for linear programming of satellite collisions where fixed and floating point numbers are intensively used [?,?]. A recent line of work even looked at how to decrease the amount of storage needed throughout sequential computations from one MPC engine to another with symmetric key primitives evaluated as arithmetic circuits [?,?].

To accomplish MPC one can select between two paradigms: garbled circuits [?,?,?] or secret sharing [?,?,?]. We will concentrate on the latter because it is the most suitable at the moment to evaluate arithmetic circuits although there have been some recent theoretical improvements on garbling modulo p made by Ball et al. [?]. Since our goal in this paper is to have secure computation within a system that scales with the number of parties as well as to provide a guarantee against malicious players we will focus on SPDZ [?,?].

It is no surprise that homomorphic encryption can help with multiparty computation. In the presence of malicious adversaries, however, there needs to be assurances that parties actually encrypt the information that they are supposed to. Zero-knowledge proofs are the essential tool to achieve this, and there exist compilers to make passive protocols secure against an active adversary. However, these proofs are relatively expensive, and it is the aim of SPDZ to reduce this cost by using them as little as possible.

The core idea of SPDZ is that, instead of encrypting the parties' inputs, it is easier to work with random data, conduct some checks at the end of the protocol, and abort if malicious behavior is detected. In order to evaluate a function with private inputs the computation is separated in two phases, a preprocessing or offline phase and an online phase. The latter uses information-theoretic algorithms to compute the results from the inputs and correlated randomness produced by the offline phase.

The correlated randomness consists of secret-shared random multiplication triples, that is (a, b, ab) for random a and b. In SPDZ, the parties encrypt random additive shares of a and b under a global public key, use the homomorphic properties to sum up and multiply the shares, and then run a distributed decryption protocol to learn their share of ab. With respect to malicious parties, there are two requirements on the encrypted shares of a and b. First, they need to be independent of other parties' shares, otherwise the sum would not be random, and second, the ciphertexts have to be valid. In the context of lattice-based cryptography, this means that the noise must be limited. Both requirements are achieved by using zero-knowledge proofs of knowledge and bounds of the cleartext and encryption randomness. It turns out that this is the most expensive part of the protocol.

The original SPDZ protocol [?] uses a relatively simple Schnorr-like protocol [?] to prove knowledge of cleartexts and correctness of ciphertexts, but the later implementation [?] uses more sophisticated cut-and-choose-style protocols for both covert and active security. We have found that the simpler Schnorr-like protocol which guarantees security against active malicious parties is actually more efficient than the cut-and-choose proof with covert security.

Intuitively, it suffices that the encryption of the sum of all shares has to be correct because only the sum is used in the protocol. We take advantage of this by replacing the per-party proof by a global proof in Section 4. This significantly reduces the computation because every party only has to check one proof instead of $n - 1$. However, the communication complexity stays the same because the independence requirement means that every party still has to commit to every other party in some sense. Otherwise, a rushing adversary could make its input dependent on others, resulting in a predictable triple.

Section 3 contains our largest theoretical contribution. We present a replacement for the offline phase of SPDZ based solely on the additive homomorphism of BGV. This allows to reduce the communication and computation compared to SPDZ because the ciphertext modulus can be smaller. At the core of our scheme is the two-party oblivious multiplication protocol by Bendlin et al. [**?**], which is based on the multiplication of ciphertexts and constants. Unlike their work, we restrict the underlying cryptosystem to BGV, which enables us to avoid the costliest part of their protocol, the proof of correct multiplication. Instead, we replace this check by the SPDZ sacrifice, and show that increasing the entropy in the secret key suffices to counter the leakage that stems from malicious behaviour in the multiplication protocol.

We do not consider the restriction to BGV to be a loss. Bendlin et al. suggest two flavors for the underlying cryptosystem, lattice-based and Paillier-like. For lattice-based cryptosystems, Costache and Smart [**?**] have shown that BGV is very competitive for large enough cleartext moduli such as needed by our protocol. On the other hand, Paillier only supports simple packing techniques and it makes difficult to manipulate individual slots [**?**]. Another advantage of BGV over Paillier is the heavy parallelization with CRT and FFT since in the lattice based cryptosystem the ciphertext modulus can be a product of many small primes.

2 Preliminaries

In the following section we define the basic notation and give an overview of the BGV encryption scheme and the SPDZ protocol.

2.1 Security Model

We use the UC (Universally Composable) framework of Canetti [**?**] to prove the security of our schemes against malicious, static adversaries, except for proofs of knowledge where we use rewinding to extract inputs from the adversary. Previous works [**?**,**?**] do this by having all inputs encrypted under a public key for which the secret key is known to the simulator in the registered key model. In our Low Gear protocol, this would involve sending extra ciphertexts not used in the protocol otherwise, which is why we opt for limited UC security.

Our protocols work with n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ where up to $n-1$ corruptions can take place before the protocol starts. We say that a protocol Π implements securely a functionality $\mathcal{F}$ if any probabilistic polynomial time adversary Adv cannot distinguish between interacting with $\mathcal{F}$ or a simulator $\mathcal{S}$ and Π with computational security k and statistical security sec.

2.2 BGV

We now give a short overview of the leveled encryption scheme developed by Brakerski et al. [**?**] required for our pre-processing phase. Since the protocols used for generating the triples need only multiplication by scalars or ciphertext addition the BGV scheme is instantiated with a single level. For completion we present the details required to understand our paper yet the reader can consult the following papers: [**?**,**?**,**?**,**?**].

Underlying Algebra. Let $R = \mathbb{Z}[X]/\langle f(x) \rangle$ be a polynomial ring with integer coefficients modulo $f(x)$. In our case $R = \mathbb{Z}[x]/\langle \Phi_m(X) \rangle$ where $\Phi_m(X) = \prod_{i \in Z_m^*}(X - \omega_m^i) \in \mathbb{Z}[X]$ and $\omega_m = \exp(2\pi/m) \in \mathbb{C}$ is a principal m'th complex root of unity and $\omega_m^i = \exp(2\pi\sqrt{-1}/m) \in \mathbb{C}$ iterates over all primitive complex mth roots of unity.

The ring R is also called the ring of algebraic integers of the m'th cyclotomic polynomial. For example when $m \geq 2$ and m is a power of two, the polynomial $\Phi_m(X) = X^{m/2} + 1$. Notice that the degree of $\Phi_m(X)$ is equal to $\phi(m)$ which makes R a field extension with degree $N = \phi(m)$. Next we define $R_q = R/qR \cong R/\langle(\Phi_m(X), q)\rangle$ where q is not necessarily a prime number. The latter will be used as the ciphertext modulus.

Plaintext Slots. Since triples are generated for arithmetic circuits modulo p, the plaintext space is the ring $R_p = R/pR$, for technical reasons p and q are co-prime. If $p \equiv 1 \mod m$ we have that $\Phi_m(X) = F_1(X) \ldots F_l(X) \mod p$ splits into l irreducible polynomials where each $F_i(X)$ has degree $d = \phi(m)/l$ and $F_i(X) \cong \mathbb{F}_p^d$. It is useful to think of an element $a \in R_p$ as a vector of size l where each element is $(a \mod F_i(X))_{i=1}^l$. This in turn allows to manipulate l plaintexts at once using SIMD (Single Instruction Multiple Data) operations.

Distributions. Throughout the definitions we will refer to a polynomial $a \in R$ as a vector of size $N = \phi(m)$. To realize the cryptosystem we need to sample at various times from different distributions to generate a vector of length N with coefficients mod p or q (which means an element from R_p or R_q). We will keep R_q throughout the following definitions:

- $\mathcal{U}(R_q)$ the uniform distribution where each unique polynomial $a \in R_q$ has an equal chance to be drawn. This can be done sampling each coefficient of a uniformly at random (from the integers modulo q).

– $\mathcal{DG}(\sigma^2, R_q)$ the discrete Gaussian with variance σ^2. Taking samples from it is performed in a similar way to the uniform one but calling for each coefficient of $a \in R_q$ the normal Gaussian $\mathcal{N}(\sigma^2)$ and then round it to the nearest integer.
– $\mathcal{ZO}(0.5)$ outputs a vector of length N where each entry has values in $\{-1, 0, 1\}$. Here zero has a $1/2$ probability to appear, whereas $\{-1, 1\}$ have both a probability of $1/4$.
– $\mathcal{HWT}(h)$ outputs a random vector of length N where at least h entries are non-zero and each entry is in the $\{-1, 0, 1\}$ set.

Ring-LWE. Hardness of the BGV scheme is based on the Ring version of Learning with Errors problem [?]. For a secret $s \in R_p$, recall that a Ring-LWE sample is produced by choosing $a \in R_q$ uniformly at random, an error $e \leftarrow \chi$ from a special Gaussian distribution and compute $b = a{\cdot}s{+}e$. Turns out that if an adversary manages to break the BGV encryption scheme in polynomial time one can also build a polynomial time distinguisher for Ring-LWE samples and the uniform distribution, namely $(a, b = a \cdot s + e) \cong (a', b')$ where $(a', b') \xleftarrow{\$} \mathcal{U}(R_q^2)$.

Key-Generation, Encryption and Decryption. The cryptosystem used in Section 3 is identical to Damgård et al. [?] bar the augmentation data needed for modulus switching. First the public, secret key pairs are generated as follows:

– $\mathsf{KeyGen}()$: Sample $s \leftarrow \mathcal{HWT}(h)$, $a \leftarrow \mathcal{U}(R_q)$, $e \leftarrow \mathcal{DG}(\sigma^2, R_q)$ and then $b \leftarrow a \cdot s + p \cdot e$. Now set the public key $pk \leftarrow (a, b)$. Notice that pk looks very similar with a Ring-LWE sample.
– $\mathsf{Enc}_{\mathsf{pk}}(m)$: To encrypt a message $m \in R_p$, sample a small polynomial with coefficients $v \leftarrow \mathcal{ZO}(0.5)$, and two Gaussian polynomials $e_1, e_2 \leftarrow \mathcal{DG}(\sigma^2, R_q)$. The ciphertext will be a pair $c = (c_0, c_1)$ where $c_0 = b \cdot v + p \cdot e_0 + m \in R_q$ and $c_1 = a \cdot v + p \cdot e_1 \in R_q$.
– $\mathsf{Dec}_{\mathsf{sk}}(c)$: To decrypt a ciphertext $c \in R_q^2$ one can simply compute $m' \leftarrow c_0 - s \cdot c_1 \in R_q$ and then set $m \leftarrow m' \bmod p$ to get the original plaintext. The decryption works only if noise $\nu = (m' \bmod p) - m$ associated with c is less than $q/2$ such that the ciphertext will not wrap around the modulus R_q.

2.3 Zero-Knowledge Proofs

In a typical scenario a zero-knowledge (ZK) proof allows a verifier to check the validity of statement claimed by a prover without revealing anything other than the claim is true. Previous implementations have used one of two approaches: a Schnorr-like protocol [?,?,?] and cut-and-choose [?]. We will call SPDZ using either of the two protocols SPDZ-1 and SPDZ-2, respectively. Analysing the communication complexity, we found that the Schnorr-like protocol is more efficient because it only involves sending two extra ciphertexts per ciphertext to be proven whereas Damgård et al. [?] suggest that, for malicious security, their protocol is most efficient with 32 extra ciphertexts. It is also worth noting that the Schnorr-like protocol seems to be easier to implement.

The Schnorr-like protocol is based on the following 3-move standard Σ-protocol. To prove knowledge of x in a field $\mathbb{F}$ such that $f(x) = y$ without revealing x:

1. The prover $\mathcal{P}$ sends a commitment $a = f(s)$ for a random s.
2. The verifier $\mathcal{V}$ then samples a random $e \overset{\$}{\leftarrow} \mathbb{F}$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ replies with $z = s + e \cdot x$. Finally $\mathcal{V}$ checks whether $f(z) = a + e \cdot y$.

If f is homomorphic with respect to the field operations, the protocol is clearly correct. Security of the prover (honest-verifier zero-knowledge) is achieved by simulating (a, e, z) from any e by sampling $z \in \mathbb{F}$ and computing $a = f(z) - e \cdot y$. Security for the verifier (special soundness) allows to extract the secret from two different transcripts (z, c), (z', c') with $c \neq c'$. This can be done by computing $x = (z - z') \cdot (c - c')^{-1}$ since the inversion of $c - c'$ is possible because both are elements from a field.

For our setting x is an integer (or a vector thereof), and we would like to prove that $\|x\|_\infty \leq B$ for some bound. For this case, Damgård and Cramer [**?**] have presented an amortized protocol (proving several pre-images at once) where s has to be chosen in a large enough interval (to statistically hide $E \cdot x$) and the challenge E is sampled from a set of matrices such that any $(E - E')$ is invertible over $\mathbb{Z}$ for any $E \neq E'$. The preimage is now extracted as $x = (E - E')^{-1}(z - z')$, thus a bound on $\|z\|_\infty$ also implies a bound on $\|x\|_\infty$.

However, it is not possible to make these bounds tight. Namely, an honest prover using $\|x\|_\infty \leq B$ will achieve that $\|z\|_\infty \leq B'$ for some $B' > B$. The quotient between the two bounds is called slack. Damgård et al. [**?**] also show that in the Fiat-Shamir setting (where the challenge is generated using a random oracle on a), a technique called rejection sampling can be used to reduce the slack. This involves sampling different s until the response z achieves the desired bound. In any case, we will see in Section 3.4 that the slack of this proof is too small to make it worthwhile using the cut-and-choose proof instead.

Figure 1 shows the functionality that the proofs above implement. For a simplified exposition we also assume that $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$ generates correct keys. In previous works this has been done by separate key registration [**?**] or key generation functionalities [**?**,**?**].

2.4 Overview of SPDZ

The SPDZ protocol [**?**,**?**] can be viewed as a two-phase protocol where inputs are shared via an additive secret sharing scheme. First part is the pre-processing phase where triples are generated. The classical way to produce these triples is either by oblivious transfer or HE (homomorphic encryption). Each has its own advantages and caveats but we will discuss in depth about the HE techniques where ciphertexts are passed around players. Since in our setting parties can deviate maliciously from the protocol by inserting too much noise in the encryption algorithm the solution to prevent this is to use ZK proofs.

$\mathcal{F}^{S}_{\mathsf{ZKPoK}}$

The functionality generates keys $(\mathsf{pk}, \mathsf{sk})$ and sends them to P_A and pk to P_B. If P_A is corrupted, the adversary chooses the keys. Then, the following can happen repeatedly:

1. P_A inputs either a vector $\mathbf{a}$ or a value a. In the latter case, $\mathbf{a}$ is defined to contain a in all slots.
2. If P_A is honest, P_B receives $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{a})$, otherwise $\mathsf{Enc}'_{\mathsf{pk}}(\mathbf{a})$, where Enc' has noise at most S times as much as regular encryption.
3. The adversary can abort any time.

Fig. 1. Proof of knowledge of ciphertext

These random triples are further used in the online phase where parties interact by broadcasting data whenever a value is revealed. Privacy and correctness is then guaranteed by authenticated shared values with information theoretic MAC's on top of them.

More formally, an authenticated secret value $x \in \mathbb{F}$ is defined as the following:

$$[\![x]\!] = (x^{(1)}, \dots, x^{(n)}, m^{(1)}, \dots, m^{(n)}, \Delta^{(1)}, \dots, \Delta^{(n)})$$

where each player P_i holds an additive sharing tuple $(x^{(i)}, m^{(i)}, \Delta^{(i)})$ such that:

$$x = \sum_{i=1}^{n} x^{(i)}, x \cdot \Delta = \sum_{i=1}^{n} m^{(i)}, \Delta = \sum_{i=1}^{n} \Delta^{(i)}.$$

For the pre-processing phase the goal is to model a Triple command which generates a tuple $([\![a]\!], [\![b]\!], [\![c]\!])$ where $c = a \cdot b$ and a, b are uniformly random from $\mathbb{F}$.

To open a value $[\![x]\!]$ all players P_i broadcast their shares $x^{(i)}$, commit and then open $m^{(i)} - x \cdot \Delta^{(i)}$, afterwards they check if the sum is equal to zero. One can check multiple values at once by taking a random linear combination of $m^{(i)} - x \cdot \Delta^{(i)}$ exactly as in the MAC Check protocol in Figure 5 in Section 3.

In the online phase the main task is to evaluate an arbitrary circuit with secret inputs. After the parties provided their inputs using the Input command next step is to perform addition and multiplication between authenticated shared values. Since the addition is linear it can be done via local computation while to multiply two values $[\![x]\!], [\![y]\!]$ requires some interaction between the parties. To compute $[\![x \cdot y]\!]$ a fresh random triple $[\![a]\!], [\![b]\!], [\![c]\!]$ is provided by the Input command and then Beaver's trick is applied [**?**]. Namely open $[\![x - a]\!]$ to get ϵ and $[\![x - b]\!]$ to get ρ. In the end obtain the authenticated product by setting $[\![x \cdot y]\!] \leftarrow [\![c]\!] + \epsilon[\![b]\!] + \rho[\![a]\!] + \epsilon \cdot \rho$.

Offline phase. We now outline the core ideas of the preprocessing phase of SPDZ. Assume that the parties have a global public key, a secret sharing of the secret key $\varDelta$, and that there is a distributed decryption protocol that allows the parties to decrypt an encryption such that they receive a secret sharing of the cleartext (see Damgård et al. [**?**] Reshare procedure for details).

For passive security only, the parties can simply broadcast encryptions of randomly sampled shares a_i, b_i and their share of the MAC key $\varDelta_i$. These encryptions can be added up and multiplied to produce encryptions of $(a \cdot b, a \cdot \varDelta, b \cdot \varDelta, a \cdot b \cdot \varDelta)$ if the encryption allows multiplicative depth two. Distributed decryption then allows the parties to receive an additive secret sharing of each of those values, which already is enough for a triple. Since achieving a higher multiplicative depth is relatively expensive, SPDZ only uses a scheme with multiplicative depth one and extends the distributed decryption to produce a fresh encryption of $a \cdot b$, which then can be multiplied with the encryption of $\varDelta$.

In the context of an active adversary there are two main issues: First, the ciphertexts input by corrupted parties have to be correct and independent of the honest parties ciphertexts. This is where zero-knowledge of knowledge of bounded values are needed. Second, the distributed decryption protocol actually allows the adversary to add an error, that is, the parties can end up with a triple $(a, b, ab+e)$ with e known to the adversary and where the MACs have additional errors as well. While an error on a MAC will make the MAC check fail in any case, the problem of an incorrect triple requires more attention. This is where the so-called SPDZ sacrifice comes in. Imagine two triples with potential errors $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket ab + e \rrbracket)$ and $(\llbracket a' \rrbracket, \llbracket b' \rrbracket, \llbracket a'b' + e' \rrbracket)$, and let t be a random field element. Then,

$$\begin{aligned} & t \cdot (ab + e) - (a'b' + e') - (ta - a') \cdot b - a' \cdot (b - b)' \\ & = tab + te - a'b' - e' - tab - a'b - a'b + a'b' \\ & = te - e', \end{aligned}$$

which is 0 with probability negligible in sec for a field of size at least 2^{sec} if either $e \neq 0$ or $e' \neq 0$. The use of MACs means that the adversary cannot forge the result of this computation, hence any error will be caught with overwhelming probability since with the additive secret sharing of our triples the parties have to reveal $\llbracket ta - a' \rrbracket$ and $\llbracket b - b' \rrbracket$. Therefore, one of the triples has to be discarded in order keep the other one "fresh" for use in the online phase. For MASCOT, Keller et al. [**?**] found that the sacrifice also works with two triples (a, b, ab) and $(a', b, a'b)$, which implies $b - b' = b - b = 0$. Such a combined triple is cheaper to produce (both in MASCOT and SPDZ), and requires less revealing for the check.

3 Low Gear: Triple Generation Using Semi-Homomorphic Encryption

The multiplication of secret numbers is at the heart of many secret sharing-based multiparty computation protocols because linear secret sharing scheme makes addition easy, and the two operations together are complete.[1] Both Bendlin et al. [?] and Keller et al. [?] have effectively reduced the problem of secure computation to computing an additive secret sharing of the product of two numbers known to two different parties. The former used semi-homomorphic encryption which allows to add two ciphertexts to get an encryption of the sum of cleartexts whereas the latter used oblivious transfer which is known to be complete for any protocol.

The semi-homomorphic solution works roughly as follows: One party sends an encryption $\mathsf{Enc}(a)$ of their input under their own public key to the other, which replies by $C := b \cdot \mathsf{Enc}(a) - \mathsf{Enc}(c_B)$, where b denotes the second party's input and c_B is chosen at random. Any semi-homomorphic encryption scheme allows the multiplication of a known value with a ciphertext, hence the decryption of the second message is $c_A := b \cdot a - c_B$, which makes (c_A, c_B) an additive secret sharing of $a \cdot b$. Here the noise of C might reveal information about b but this can mitigated by adding random noise from an interval that is sec larger than the maximum noise of C. This technique sometimes called "drowning" was also used in the distributed decryption of SPDZ.

In the context of a malicious adversary there are two concerns with the above protocol: $\mathsf{Enc}(a)$ might not be a correct encryption and C might not be computed correctly. In both cases, Bendlin et al. use a zero-knowledge proof of knowledge to make sure that both parties behave correctly.

To prove the correctness of $\mathsf{Enc}(a)$, there are relatively efficient proofs based on amortized Σ-protocols (reducing the overhead per ciphertext by processing several ciphertexts at once), but for the proof of correct multiplication amortization is not possible in our context because the underlying ciphertext $\mathsf{Enc}(a)$ is different in every instance. The main goal of our work in this chapter is therefore to avoid the proof of correct multiplication altogether of defer this to a later check in the protocol sometimes called "sacrificing".

Recall that the goal in this family of protocols is generate random multiplication triples (a, b, ab). The sacrifice will guarantee that the parties have shares of correct triples, but there is a possibility of a selective failure attack. If C was not computed correctly, just the fact that the check passed (otherwise the parties abort without using their private data) can reveal information meant to stay private in the protocol. We will show that, in case of BGV, this is information about the private key. Therefore in a first step we define in Section 3.1 an enhanced CPA notion that we will use later in our proofs, and we show that increasing the entropy in the private key suffices to achieve this notion.

In Section 3.2, we will then use our multiplication protocol a first time to compute SPDZ-style MACs, that is, additive secret sharings of the product of a value and a global MAC key, which itself is secret-shared additively. It is straightforward to compute such a global product from the two-party protocol. Consider that $\sum_i a_i \cdot \sum_i b_i = \sum_{i,j} a_i \cdot b_j$. Every summand in the right-hand side can be computed either locally or by the two-party protocol, and the additive operation

[1] This fact is mirrored in the world of garbled circuits, where the free-XOR technique only requires to garble AND gates, which compute the product of two bits.

is trivially commutative with the addition of shares. The resulting protocol can be seen as a full-threshold version verifiable secret sharing as observed by Keller et al. because it guarantees the soundness of inputs by parties and the correctness of linear operations.

Building on the authentication protocol, we present the multiplication triple generation in Section 3.3 using the two-party multiplication protocol once more. Note that the after-the-fact check of correct multiplication works differently in the two protocols. In the authentication protocol, we make use of the fact that changing values are always multiplied with the same share of the MAC key. In the triple generation, however, both values change from triple to triple, thus we rely on the SPDZ sacrifice there. For this, we use a trick used by Keller et al. that reduces the complexity by generating a pair of triples $((a, b, ab), (a', b, a'b))$ for the sacrifice instead of two independent triples.

Finally, we present our choice of BGV parameters in Section 3.4, following the considerations of Damgård et al. [?], which in turn are based on Gentry et al. [?]. We found that the ciphertext modulus is about 100 bits shorter compared to original SPDZ for fields of size 2^{64} to 2^{128}, which makes a significant contribution to the reduced complexity of our protocol because SPDZ requires a modulus of bit length more than 300 for 64-bit fields and 40-bit security.

3.1 Enhanced CPA Security

We want to reduce the security of our protocol to an enhanced version of the CPA game for the encryption scheme. In other words, if the encryption scheme in use is enhanced-CPA secure, then even a selective failure caused by the adversary does not reveal private information.

$\mathcal{G}_{\mathsf{cpa+}}$

1. The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(k)$, sends pk to the adversary.
2. The challenger sends $c = \mathsf{Enc}_{\mathsf{pk}}(m)$ for a random message m.
3. For $j \in \mathsf{poly}(k)$:
   (a) The adversary sends c_j to the challenger.
   (b) The challenger checks if $\mathsf{Dec}_{\mathsf{sk}}(c_j) = 0$; if this is the case the challenger sends OK to the adversary; else, the challenger sends FAIL to the adversary and aborts.
4. The challenger samples $b \xleftarrow{\$} \{0, 1\}$ and sends m to the adversary if $b = 0$ and a random m' otherwise.
5. The adversary sends $b' \in \{0, 1\}$ to the challenger and wins the game if $b = b'$.

Fig. 2. Enhanced CPA game

We say that an encryption scheme is *enhanced-CPA secure* if, for all PPT adversaries in the game from Figure 2, $\Pr[b = b'] - 1/2$ is negligible in k.

Achieving enhanced-CPA security. The game without zero checks in step 3 clearly can be reduced to the standard CPA game. We therefore only show how to achieve security with respect to the extra oracle queries.

Since decryption is deterministic, for every c, there exists a set S_c of potential secret keys such that the decryption is zero:

$$S_c = \{\mathsf{sk}' : \mathsf{Dec}_{\mathsf{sk}'}(c) = 0\}.$$

The adversary can only win if the actual secret key is in $S = \bigcap_j S_{c_j}$. Assume this happens with probability

$$p_S = \Pr_{\mathsf{sk}' \leftarrow \mathsf{KeyGen}}[\mathsf{sk}' \in S].$$

A successful adversary learns only $\log p_S$ bits of entropy of the secret key. It follows that an adversary passing this step with probability at least $2^{-\mathsf{sec}}$ can learn at most sec bits of entropy.

The key generation of BGV generates s of length N such that s has $h = 64$ non-zero entries at randomly chosen places, which are chosen uniformly from $\{-1, 1\}$. The entropy is therefore

$$\log \binom{N}{h} + h.$$

It is easy to see that choosing $h' = h + \mathsf{sec}$ non-zero entries increases the entropy by sec bits[2] for large enough N. Because $\binom{N}{k}$ monotonously increases for $k \leq N/2$,

$$\binom{N}{h + \mathsf{sec}} \geq \binom{N}{h}$$

for $N \geq 2 \cdot (h + \mathsf{sec})$. It follows that

$$\log \binom{N}{h + \mathsf{sec}} + h + \mathsf{sec} \geq \left(\log \binom{N}{h} + h\right) + \mathsf{sec},$$

which is the desired result. We will later see that N is much bigger than $2 \cdot (h + \mathsf{sec})$ for $h = 64$ and $\mathsf{sec} = 128$.

Finally, we have make sure that zero-test cannot be used to reveal information about m. Informally, the cryptosystem only allows affine linear operations, and thus, the adversary succeeds only with negligible probality due to the high entropy of m. However, if the cryptosystem would allow to generate an encryption of a bit of m from $\mathsf{Enc}_{\mathsf{pk}}(m)$, the adversary could test this bit for zero with success probability $1/2$. Therefore, we have to assume that non-linear operations on ciphertexts are not possible. To this end, Bitansky et al. [**?**] have introduced the notion of linear targeted malleability. A stronger notion thereof, linear-only encryption, has been conjectured by Boneh et al. [**?**] to apply to the cryptosystem by Peikert et al. [**?**], which is also based on learning with errors. The definition by Bitansky et al. is as follows.

[2] $\mathsf{sec}/2$ would suffice, but sec does not affect the efficiency and allows for a simpler analysis.

Definition 1. *An encryption scheme has the* linear targeted malleability *property if for any polynomial-size adversary A and plaintext generator $\mathcal{M}$ there is a polynomial-size simulator S such that, for any sufficiently large $\lambda \in \mathbb{N}$, and any auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, the following two distributions are computationally indistinguishable:*

$$\left\{ \begin{array}{l|r} & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{pk}, & (s, a_1, \ldots, a_m) \leftarrow \mathcal{M}(\mathsf{pk}) \\ a_1, \ldots, a_m, & (c_1, \ldots, c_m) \leftarrow (\mathsf{Enc}_{\mathsf{pk}}(a_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(a_m)) \\ s, & (c'_1, \ldots, c'_k) \leftarrow A(\mathsf{pk}, c_1, \ldots, c_m; z) \\ \mathsf{Dec}_{\mathsf{sk}}(c'_1), \ldots, \mathsf{Dec}_{\mathsf{sk}}(c'_k) & \textit{where} \\ & \mathsf{ImVer}_{\mathsf{sk}}(c'_1) = 1, \ldots, \mathsf{ImVer}_{\mathsf{sk}}(c'_k) = 1 \end{array} \right\}$$

and

$$\left\{ \begin{array}{l|r} \mathsf{pk}, & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ a_1, \ldots, a_m, & (s, a_1, \ldots, a_m) \leftarrow \mathcal{M}(\mathsf{pk}) \\ s, & (\Pi, \mathbf{b}) \leftarrow S(\mathsf{pk}; z) \\ a'_1, \ldots, a'_k & (a'_1, \ldots, a'_k)^\top \leftarrow \Pi \cdot (a_1, \ldots, a_m)^\top + \mathbf{b} \end{array} \right\}$$

where $\Pi \in \mathbb{F}^{k \times m}$, $\mathbf{b} \in \mathbb{F}^k$, and s is some arbitrary string (possibly correlated with the plaintexts).

In the context of BGV, verifying whether a ciphertext is the image of the encryption (ImVer) can be trivially done by checking membership in R_q, which is possible without the secret key.

It is straight-forward to see that linear targeted malleability allows to reduce enhanced-CPA game to a game without a zero-test oracle. We simply replace the decryption of the adversary's queries by the $a'_1, \ldots, a'_k$ computed using S according to the definition, which can be tested for zero without knowing the secret key. The two games are computationally indistinguishable by the definition, and the modified one can be reduced to the normal CPA game as argued above.

3.2 Input Authentication

As in Keller et al. [**?**], we want to implement a functionality (Figure 3) that commits the parties to secret sharings and that provides the secure computation of linear combinations of inputs. However, instead of using oblivious transfer for the pairwise multiplication of secret numbers we use our building block based on semi-homomorphic encryption. See Figure 4 for our protocol.

In case parties P_i and P_j are honest,

$$\begin{aligned} \Delta^{(i)} \cdot \rho - \sigma^{(i)} - \sum_{k=1}^{m} t_k \cdot \mathbf{d}_k^{(i)} &= \Delta^{(i)} \cdot (\sum_{k=1}^{m} t_k \cdot x_k) - \sum_{k=1}^{m} t_k \cdot e_k^{(i)} - \sum_{k=1}^{m} t_k \cdot \mathbf{d}_k^{(i)} \\ &= \sum_{k=1}^{m} t_k \cdot (\Delta^{(i)} \cdot x_k - e_k^{(i)} - \mathbf{d}_k^{(i)}) = 0 \end{aligned}$$

$\mathcal{F}_{[\![\cdot]\!]}$

Input: On input $(\mathsf{Input}, \mathsf{id}_1, \ldots, \mathsf{id}_l, x_1, \ldots, x_l, P_j)$ from party P_j and $(\mathsf{Input}, \mathsf{id}_1, \ldots, \mathsf{id}_l, P_j)$ from all players P_i where $i \neq j$ store $\mathsf{Val}[\mathsf{id}_k] \leftarrow x_k$ for all $k \in [1 \ldots l]$.

Linear Combination: On input $(\mathsf{LinComb}, \overline{\mathsf{id}}, \mathsf{id}_1, \ldots, \mathsf{id}_l, c_1, \ldots, c_l, c)$ from all parties where $\mathsf{id}_k \in \mathsf{Val.Keys}()$ store $\mathsf{Val}[\overline{\mathsf{id}}] = \sum_{k=1}^{l} \mathsf{Val}[\mathsf{id}_k] \cdot c_k + c$.

Open: On input $(\mathsf{Open}, \mathsf{id})$ from all parties, send $\mathsf{Val}[\mathsf{id}]$ to the adversary; wait for input x from the adversary and then send x to all parties.

Check: On input $(\mathsf{Check}, \mathsf{id}_1, \ldots, \mathsf{id}_l, x_1, \ldots, x_l)$ from all parties wait for adversary's input. If the input is OK and $\mathsf{Val}[\mathsf{id}_k] = x_k$ for all $k \in [1 \ldots l]$ then send OK to every party, otherwise send $\perp$ and terminate.

Abort: On input Abort from the adversary send $\perp$ to all parties and terminate.

Fig. 3. Functionality $\mathcal{F}_{[\![\cdot]\!]}$

for all $i \neq j$. This means that P_i's check succeeds if. The last equation follows from the homomorphism of the encryption scheme.

Furthermore, one can check similarly that

$$\sum_i m_k^{(i)} = x_k \cdot \sum_i \Delta^{(i)},$$

which is the desired equation underlying the MAC. If it does not hold because of P_j's behaviour, we would like the check to fail for some honest P_i. Informally, the fact that P_j cannot predict the coefficients t_k makes it impossible for P_j to provide correct $(\rho, \sigma^{(i)})$ to an honest party P_i after computing $C^{(i)}$ incorrectly. However, this opens the possibility for leakage by a selective failure attack, which is why we need the underlying cryptosystem to achieve enhanced-CPA security.

The most intricate part of the simulator $\mathcal{S}_{[\![\cdot]\!]}$ (Figure 6) is simulating the **Input** phase for a corrupted P_j while the same phase for honest P_j is straightforward given that Enc' statistically hides the noise of $\mathbf{x}^{(i)} \cdot \mathsf{Enc}(\boldsymbol{\Delta})$. Note that the $(x_m, e_m^{(i)}, d_m^{(i)})$ are only used for the check. This maintains P_j's privacy even after sending ρ and $\{\sigma^{(i)}\}_{i \neq j}$.

Theorem 1. *$\Pi_{[\![\cdot]\!]}$ implements $\mathcal{F}_{[\![\cdot]\!]}$ in the $\mathcal{F}_{\mathsf{Commit}}$-hybrid model with rewinding in a presence of a dishonest majority if the underlying cryptosystem achieves enhanced CPA-security.*

Proof (Sketch). We focus on the case of a corrupted P_j in the **Input** phase because the adversary has a larger degree of freedom with the encryptions $C^{(i)}$. However, with rewinding in step 6 we can extract the values used by the adversary. This extraction takes time dependent inverse to the success probability, similar the soundness argument for Σ-protocols. Too see this, consider that the space of all possible challenges $\{t_k\}_{k=1}^{m}$ has size $|\mathbb{F}|^m$. The extractor requires the

$\Pi_{[\![\cdot]\!]}$

Initialize: Each Party P_i does the following:

1. Sample a MAC key $\Delta^{(i)} \xleftarrow{\$} \mathbb{F}$.
2. Initialize two instances of $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$ with every other Party P_j (one as prover, one as verifier), receiving $(\mathsf{pk}_{ij}, \mathsf{sk}_{ij})$ and pk_{ji}.
3. Using $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$, send an encryption $\mathsf{Enc}_{\mathsf{pk}_{ij}}(\boldsymbol{\Delta}^{(i)})$ to every other party where $\boldsymbol{\Delta}^{(i)}$ denotes a cleartext with all slots set to $\Delta^{(i)}$.

Input: On input $(\mathsf{Input}, \mathsf{id}_1, \ldots, \mathsf{id}_l, x_1, \ldots, x_l, P_j)$ from P_j and $(\mathsf{Input}, \mathsf{id}_1, \ldots, \mathsf{id}_l, P_j)$ from all P_i where $i \neq j$:

1. For each input x_k where $k \in [1 \ldots l]$ P_j samples randomly $x_k^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends them to the designated party i. Then P_j sets its corresponding share $x_k^{(j)}$ accordingly such that $\sum_{l=1}^{n} x_k^{(l)} = x_k$.
2. We assume that $l < m$ where m is the number of ciphertext slots in the encryption scheme. Let $\mathbf{x}$ denote the vector containing x_k in the first l entries and a random number in the m-th one.
3. For every party P_i:
   (a) P_j computes $C^{(i)} = \mathbf{x} \cdot \mathsf{Enc}_{\mathsf{pk}_{ij}}(\boldsymbol{\Delta}^{(i)}) - \mathsf{Enc}'_{\mathsf{pk}_{ij}}(\mathbf{e}^{(i)})$ for random $\mathbf{e}^{(i)}$ and sends $C^{(i)}$ to P_i. Enc' denotes encryption with noise $p \cdot 2^{\mathsf{sec}}$ larger than in normal encryption.
   (b) P_i decrypts $\mathbf{d}^{(i)} = \mathsf{Dec}_{\mathsf{sk}_{ij}}(C^{(i)})$.
4. The parties use $\mathcal{F}_{\mathsf{Rand}}$ to generate random t_k for $k = 1, \ldots, m$.
5. P_j computes $\rho = \sum_{k=1}^{m} t_k \cdot x_k$ and $\sigma^{(i)} = \sum_{k=1}^{m} t_k \cdot e_k^{(i)}$, and sends $(\rho, \sigma^{(i)})$ to P_i.
6. P_i checks whether $\Delta^{(i)} \cdot \rho - \sigma^{(i)} - \sum_{k=1}^{m} t_k \cdot \mathbf{d}_k^{(i)} = 0$ and aborts if not.
7. P_j sets its MAC share associated to x_k as $m_k^{(j)} \leftarrow \sum_{i \neq j} e_k^{(i)} + x_k \cdot \Delta^{(j)}$ and each party P_i does so for $m_k^{(i)} \leftarrow d_k^{(i)}$.
8. All parties store their authenticated shares $x_k^{(i)}, m_k^{(i)}$ as $[\![x]\!]$ under the identifiers $\mathsf{id}_1, \ldots, \mathsf{id}_l$.

Linear Combination: On input $(\mathsf{LinComb}, \overline{\mathsf{id}}, \mathsf{id}_1, \ldots, \mathsf{id}_l, c_1, \ldots, c_l, c)$ from all parties, every P_i retrieves the share-MAC pairs $x_k^{(i)}, m(x_k)^{(i)}_{k \in [1 \ldots l]}$ and computes:

$$y^{(i)} = \sum_{k=1}^{l} c_k \cdot x_k^{(i)} + c \cdot s_1^{(i)}$$

$$m(y)^{(i)} = \sum_{k=1}^{l} c_k \cdot m(x_k)^{(i)} + c \cdot \Delta^{(i)}$$

$s_1^{(i)}$ denotes a fixed sharing of 1, for example, $(1, 0, \ldots, 0)$.

Open: On input $(\mathsf{Open}, \mathsf{id})$ from all parties each P_i looks-up for the share $x^{(i)}$ with identifier id and broadcasts it. Then each party reconstructs $x = \sum_{i=1}^{n} x^{(i)}$.

Check: On input $(\mathsf{Check}, \mathsf{id}_1, \ldots, \mathsf{id}_l, x_1, \ldots, x_l)$ from all parties:

1. Sample public vector $r \leftarrow \mathcal{F}_{\mathsf{Rand}}(\mathbb{F}^l)$.
2. Compute $y^{(i)} = \sum_{k=1}^{m} r_k^{(i)} x_k^{(i)}$ and $m(y)^{(i)} = \sum_{k=1}^{m} r_k^{(i)} m_k(x_k)^{(i)}$.
3. Run Π_{MACCheck} with $y^{(i)}, m(y)^{(i)}$.

Fig. 4. Protocol for two-party input authentication

Π_{MACCheck}

Each party P_i uses $y^{(i)}, m(y)^{(i)}, \Delta^{(i)}$ in the following way:

1. Compute $\sigma^{(i)} \leftarrow m(y)^{(i)} - \Delta^{(i)} y^{(i)}$.
2. Call $\mathcal{F}_{\mathsf{Commit}}$ with $(\mathsf{Commit}, \sigma^{(i)})$ to receive handle τ_i.
3. Broadcast $\sigma^{(i)}$ to all parties by calling $\mathcal{F}_{\mathsf{Commit}}$ with (Open, τ_i).
4. If $\sigma^{(1)} + \cdots + \sigma^{(n)} \neq 0$ then abort and output $\perp$; otherwise continue protocol.

Fig. 5. Protocol for MAC checking

$\mathcal{S}_{[\![\cdot]\!]}$

Let H denote the set of honest parties and A the complement thereof.

Init:

1. Emulating $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$, generate $(\mathsf{pk}_{ij}, \mathsf{sk}_{ij})$ for all $i \in H$ and $j \in A$ and send all pk_{ij} to the adversary.
2. Emulating $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$, send $\mathsf{Enc}_{\mathsf{pk}_{ij}}(x)$ for random x to the adversary for all $i \in H$.

Input: We assume that $j \notin H$.

1. Receive $x_k^{(i)}$ from the adversary for all $k = 1, \ldots, l$ and $i \in H$ and store them.
2. Receive the ciphertext $C^{(i)}$ from the adversary and decrypt it to get $\mathbf{d}^{(i)}$ for all $i \in H$.
3. Emulating $\mathcal{F}_{\mathsf{Rand}}$, sample random t_i for $i = 1, \ldots, m$.
4. Receive $(\rho, \sigma^{(i)})$ from the adversary for all $i \in H$.
5. Check whether $\sigma^{(i)} + \sum_{k=1}^{m} t_k \cdot d_k^{(i)} = 0$ for all $i \in H$ and abort if not.
6. Rewinding the adversary, collect enough answers $(\sigma^{(i)}, \rho)$ for random $\mathbf{t}$ in order to reconstruct $\mathbf{x}$ and $\mathbf{e}^{(i)}$. Choose $\mathbf{t}$ such that it is linearly independent from all previously used $\mathbf{t}$.
7. Compute $m_k^{(i)}$ for every $i \in A$ and store it.
8. Input $(x_1, \ldots, x_l)$ to $\mathcal{F}_{[\![\cdot]\!]}$.

Linear Combination: For every $i \in A$, compute shares and MACs as an honest party would.

Open:

1. Receive the value from the $\mathcal{F}_{[\![\cdot]\!]}$.
2. If the value is a linear combination of previously opened values, compute the honest parties' shares accordingly. Otherwise, sample new shares.
3. Send the honest parties's shares to adversary.
4. Receive the corrupted parties' shares from the adversary.
5. Input the sum to $\mathcal{F}_{[\![\cdot]\!]}$.

Check:

1. Emulating $\mathcal{F}_{\mathsf{Rand}}$, send r to all corrupted parties.
2. Emulating $\mathcal{F}_{\mathsf{Commit}}$ receive $\sigma^{(i)}$ for all $i \in A$.
3. If $\sum_{i \in A} \sigma^{(i)}$ does not match the result computed from stored shares, abort $\mathcal{F}_{[\![\cdot]\!]}$.

Fig. 6. Simulator for $\Pi_{[\![\cdot]\!]}$

responses to m linearly independent challenges $\{t_k\}_{k=1}^{m}$. The adversary can only prevent this by restricting the correct responses to an incomplete subspace of $S \subset \mathbb{F}^m$, that is $|S| \leq \mathbb{F}^{m-1}$. Such an adversary will succeed with probability at most $|\mathbb{F}|^{m-1}/|\mathbb{F}|^m = |\mathbb{F}|^{-1}$, which is negligible because we require the size of $\mathbb{F}$ to be exponential in the security parameter. It follows that the soundness extractor for Σ-protocols by Damgård [**?**] can be adapted to our case.

After the extraction, it is straight-forward to simulate the rest of the protocol because the Linear Combination phase does not involve communication, and producing a correct MAC in the **Check** phase for an incorrect output in the **Open** phase is equivalent to extracting Δ. This argument can also be extended to the random linear combination used in the **Check** phase similarly to Keller et al. [**?**]. It is easy to see that extracting Δ is in turn equivalent to breaking the security of the underlying cryptosystem.

We therefore construct a distinguisher in the enhanced-CPA security game from an environment distinguishing between the real and the ideal world. The difference between $\mathsf{Enc}_{\mathsf{pk}_{ij}}(\Delta^{(i)})$ in the real world and $E = \mathsf{Enc}_{\mathsf{pk}_{ij}}(x)$ for random x in the simulation can trivially be reduced to our CPA security game because the adversary never receives $\Delta^{(i)}$. Furthermore, $\mathbf{x}$ and $\mathbf{e}^{(i)}$ extracted from the adversary can be used to compute $C' = C^{(i)} - \mathbf{x} \cdot E - \mathbf{e}^{(i)}$. By the check conducted by the honest party P_i, the adversary learns whether C decrypts to the expected value, or equivalently, whether C' is an encryption of zero. We therefore forward C' to the zero test our enhanced CPA game.

3.3 Triple Generation

Recall that the goal is to produce random authenticated triples $([\![a]\!], [\![b]\!], [\![ab]\!])$ such that a, b are randomly sampled from $\mathbb{F}$ as described in Figure 8. Our protocol in Figure 7 is modeled closely after MASCOT [**?**], replacing oblivious transfer with semi-homomorphic encryption. The construction of "global" multiplication from a two-party works exactly the same way in both cases. The **Sacrifice** step is exactly the same as in SPDZ and MASCOT and essentially guarantees that corrupted parties have used the same inputs in the **Multiplication** and **Authentication** steps. This is the only freedom the adversary has because all other arithmetic is handled by $\mathcal{F}_{[\![\cdot]\!]}$ at this stage.

Theorem 2. *Π_{Triple} implements $\mathcal{F}_{\mathsf{Triple}}$ in the $(\mathcal{F}_{[\![\cdot]\!]}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model with a dishonest majority of parties.*

Proof (Sketch). For the proof we use $\mathcal{S}_{\mathsf{Triple}}$ in Figure 9. The simulator is based on important two facts: First, it can decrypt $C^{(ji)}$ for a corrupted party P_j because it generated the keys emulating $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$. Second, the adversary is committed to all shares of corrupted parties' by the input to $\mathcal{F}_{[\![\cdot]\!]}$ in the **Authenticate** step. This allows the simulator to determine exactly whether the **Sacrifice** step in $\Pi_{[\![\cdot]\!]}$ will fail. Furthermore, the adversary only learns encryptions of honest parties' shares, corrupted parties' shares, $\boldsymbol{\rho}$, and the result of the check. If the check fails, the protocol aborts, $\boldsymbol{\rho}$ is independent of any output information because $\hat{\mathbf{b}}$ and

Π_{Triple}

Multiply:

1. Each party P_i samples $\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \hat{\mathbf{b}}^{(i)} \xleftarrow{\$} \mathbb{F}$ (such that the length of every vector matches the number of slots in the encryption scheme).
2. Every unordered pair (P_i, P_j) execute the following:
   (a) P_i uses $\mathcal{F}^S_{\mathsf{ZKPoK}}$ to send P_j the encryption $\mathsf{Enc}_{\mathsf{pk}_{ij}}(\mathbf{a}^{(i)})$.
   (b) P_j computes $C^{(ij)} = \mathbf{b}^{(j)} \cdot \mathsf{Enc}_{\mathsf{pk}_{ij}}(\mathbf{a}^{(i)}) - \mathsf{Enc}'_{\mathsf{pk}_{ij}}(\mathbf{e}^{(ij)})$ for random $\mathbf{e}^{(ij)} \xleftarrow{\$} \mathbb{F}$ and sends it to P_i. $\mathsf{Enc}'_{\mathsf{pk}_{ij}}$ denotes encryption with noise $p \cdot 2^{\mathsf{sec}}$ larger than normal encryption times the slack in the zero-knowledge proof.
   (c) P_i decrypts $\mathbf{d}^{(ij)} = \mathsf{Dec}_{\mathsf{sk}_{ij}}(C^{(ij)})$.
   (d) Repeat the last two steps with $\hat{\mathbf{b}}^{(i)}$ to get $\hat{\mathbf{e}}^{(ij)}$ and $\hat{\mathbf{d}}^{(ij)}$.
3. Each party P_i computes $\mathbf{c}^{(i)} = \mathbf{a}^{(i)} \cdot \mathbf{b}^{(i)} + \sum_{j \neq i}(\mathbf{e}^{(ij)} + \mathbf{d}^{(ij)})$ and $\hat{\mathbf{c}}^{(i)}$ similarly.

Authenticate: Party P_i calls $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Input}$ with $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \hat{\mathbf{b}}^{(i)}, \mathbf{c}^{(i)}, \hat{\mathbf{c}}^{(i)})$ and then $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{LinComb}$ to get vectors of handles of the sum of shares. E.g., we denote by $[\![\mathbf{a}]\!]$ the vector of handles for the respective sums of elements $\{\mathbf{a}^{(i)}\}_{i=1\ldots n}$.

Sacrifice: The parties do the following:

1. Call $r \leftarrow \mathcal{F}_{\mathsf{Rand}}$.
2. Call $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{LinComb}$ for $r \cdot [\![\mathbf{b}]\!] - [\![\hat{\mathbf{b}}]\!]$ and store them as $[\![\boldsymbol{\rho}]\!]$.
3. Reveal $\boldsymbol{\rho} \leftarrow \mathcal{F}_{[\![\cdot]\!]}.\mathsf{Open}([\![\boldsymbol{\rho}]\!])$.
4. Call $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Open}(\cdot)$ on $\boldsymbol{\tau} \leftarrow r \cdot \mathbf{c} - \hat{\mathbf{c}} - \boldsymbol{\rho} \cdot \mathbf{a}$. If $\boldsymbol{\tau} \neq 0$ then abort; else continue.
5. Call $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Check}$ an all opened values. If any check fails then abort, otherwise continue the protocol.

Output: $([\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{c}]\!])$ as a vector of valid triples.

Fig. 7. Protocol for random triple generation

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

$\mathcal{F}_{\mathsf{Triple}}$ offers the same interface as $\mathcal{F}_{[\![\cdot]\!]}$ and the following function:

Triple: On input $(\mathsf{Triple}, \mathsf{id}_a, \mathsf{id}_b, \mathsf{id}_c)$ from all parties sample $a, b \xleftarrow{\$} \mathbb{F}$ and store $(\mathsf{Val}[\mathsf{id}_a], \mathsf{Val}[\mathsf{id}_b], \mathsf{Val}[\mathsf{id}_c]) = (a, b, c)$ where $c = a \cdot b$.

Fig. 8. Functionality for random triple generation.

$\hat{\mathbf{c}}$ are discarded at the end, and finally, an environment deducing information from the encryptions can be used to break the enhanced-CPA security of the underlying cryptosystem. In addition, the environment only learns handles to triples in the **Output** steps, from which no information can be deduced.

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

Let H denote the set of honest parties and A the complement thereof.

Initialize: Emulating $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$, for every $i \in A$ and $j \in H$, generate all key pairs $(\mathsf{pk}_{ij}, \mathsf{sk}_{ij})$ and the send relevant parts to the relevant party.

Multiply:

1. For every $i \in A$ and $j \in H$, emulate two instance of $\mathcal{F}^{S}_{\mathsf{ZKPoK}}$:
   (a) Send $\mathsf{Enc}_{\mathsf{pk}_{ji}}(0)$ to the adversary and receive $C^{(ji)}$.
   (b) Receive $\mathbf{a}^{(i)}$ from the adversary and reply with $\mathsf{Enc}'_{\mathsf{pk}ij}(\mathbf{e}^{(ij)})$ for random $\mathbf{e}^{(ji)}$.

Authenticate: Emulating $\mathcal{F}_{[\![\cdot]\!]}$, receive $\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \hat{\mathbf{b}}^{(i)}, \mathbf{c}^{(i)}, \hat{\mathbf{c}}^{(i)}$ for all $i \in A$ from the adversary and return the desired handles.

Sacrifice:

1. Emulating $\mathcal{F}_{\mathsf{Rand}}$, sample $r \xleftarrow{\$} \mathbb{F}_p$ and send it to the adversary.
2. Sample $\boldsymbol{\rho} \xleftarrow{\$} \mathbb{F}_p^m$ and send it to the adversary emulating $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Open}$. Set Fail if the adversary inputs a different value in response.
3. Given the adversary's inputs in **Authenticate** and $\mathsf{Dec}_{\mathsf{sk}_{ji}}(C^{(ji)})$, we can compute $\boldsymbol{\tau}$. Send it to the adversary emulating $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Open}$. If the response is different, or $\boldsymbol{\tau} \neq 0$, set Fail.
4. Emulating $\mathcal{F}_{[\![\cdot]\!]}.\mathsf{Check}$, abort if Fail is set.

Fig. 9. Simulator for Π_{Triple}

3.4 Parameter Choice

Since we do not need multiplication of ciphertexts, the list of moduli used in previous works [?,?] collapses to one q ($= q_1 = q_0 = p_0$ depending on context). The other main parameter is the number of ciphertext slots denoted by $N = \phi(m)$. Gentry et al. [?] give the following inequality for the largest modulus:

$$N \geq \frac{\log(q/\sigma)(k+110)}{7.2}$$

for a computational security k, which gives

$$N \geq \log q \cdot 33.1 \tag{1}$$

for 128-bit security. $\sigma = 3.2$ does not make difference in this inequality.

The second constraint on q and $\phi(m)$ depends on the noise of the ciphertext that will decrypted. Damgård et al. compute the bound B_{clean} on the noise of a freshly generated ciphertext:

$$B_{\mathsf{clean}} = N \cdot p/2 + p \cdot \sigma(16 \cdot N \cdot \sqrt{n/2} + 6 \cdot \sqrt{N} + 16 \cdot \sqrt{n \cdot h \cdot N})$$

p denotes the plaintext modulus, and n denotes the number of parties, which appears because of the distributed ciphertext generation (the secret is the sum of n secret keys). Setting $n = 1$ because we do not use distributed ciphertext generation, and $h = 64 + \mathsf{sec} \leq 192$, $\sigma = 3.2$ as in the previous works, we get

$$B_{\mathsf{clean}} \leq p \cdot (37N + 685\sqrt{N}).$$

In the multiplication protocol, one party multiplies the ciphertext with a number in $\mathbb{F}_p$, adds a number in $\mathbb{F}_p$, and then "drowns" the noise with statistical security sec (adding extra noise sampling from interval that is 2^{sec} larger than the current noise bound). Furthermore, depending on the proof of knowledge used, we can only assume that the noise of the ciphertext being sent is $S \cdot B_{\mathsf{clean}}$ for some soundness slack $S \geq 1$. Therefore, the noise before decryption is bounded by

$$p \cdot S \cdot B_{\mathsf{clean}} \cdot (1 + 2^{\mathsf{sec}}),$$

which must be smaller than $q/2$ for correct decryption. Hence,

$$2 \cdot p^2 \cdot S \cdot \left(37N + 685\sqrt{N}\right)(1 + 2^{\mathsf{sec}}) < q. \tag{2}$$

Putting things together, (2) implies that, loosely, $120 \leq \log q$ or $384 \leq \log q$ if $\mathsf{sec} = 40$ or $\mathsf{sec} = 128$ and $p \geq 2^{\mathsf{sec}}$ (the latter is a requirement of SPDZ-like sacrificing). Using this in (1) gives $N \geq 3972$ or $N \geq 12711$. For both values of N as well as a ten times larger N,

$$\log\left(37N + 685\sqrt{N}\right) \approx 20 \pm 2.$$

Hence,

$$\log q \gtrsim 21 + 2\log p + \log S + \mathsf{sec} \pm 2.$$

The proof of knowledge in the first version of SPDZ [?] has the worst soundness slack with

$$S = N \cdot \mathsf{sec}^2 \cdot 2^{\sec/2+8}.$$

Thus,

$$\log S \leq \log N + 2\log \mathsf{sec} + \sec/2 + 8$$

and

$$\log q \gtrsim 29 + 2\log p + 3\mathsf{sec}/2 + 2\log \mathsf{sec} + \log N \pm 2.$$

Note that, even though this estimate is now five years old, we found our parameters to uphold against more recent estimates [?] tested using the script that is available online [?]. The main reason is that our parameters have a considerable margin because we require N to be a power of two.

More recently, Damgård et al. [?] presented an improved version of the cut-and-choose proof used in a previous implementation of SPDZ [?], but the reduced slack does not justify the increased complexity caused by several additional ciphertexts being computed and sent in the proof. Consider that, even for $\mathsf{sec} = 128$ and $N = 2^{16}$ (the latter being typical for our parameters, $\log S$ is about 100, increasing the ciphertext modulus length by less than 25 percent.

We have calculated the ciphertext modulus q's bit length for various parameters and for our protocol with semi-homomorphic encryption and SPDZ (using somewhat homomorphic encryption), both with the Schnorr-like protocol [?,?] and the recent cut-and-choose proof [?]. Table 1 shows results of our calculation as well as the results given by Damgård et al. [?]. One can see that using cut-and-choose instead of the Schnorr-like protocol does not make any difference for SPDZ. This is because the scaling (also called modulus switching) involves the division by a number larger than the largest possible slack of the Schnorr-like protocol (roughly 2^{100}), hence the slack will be eliminated. For our Low Gear protocol, the slack has a slight impact, increasing the size of a ciphertext by up to 25 percent. However, this does not justify the use of a cut-and-choose proof because it involves sending seven instead of two extra ciphertexts per proof.

Table 1 also shows Low Gear ciphertexts are about 30 percent shorter than SPDZ ciphertexts. Consider that Table 3 in Section 5 shows a reduction in the communication from SPDZ to Low Gear of up to 50 percent. The main reason for the additional reduction is the fact that for one guaranteed triple, SPDZ involves producing two triples $((a,b,c),(d,e,f)$, of which (a,b,d,e) require a zero-knowledge proof. In Low Gear on the other hand, we produce $(a,b,c,\hat{b},\hat{c})$, of which only a requires a zero-knowledge proof.

Low Gear		SPDZ			sec	$\log(\lvert\mathbb{F}_p\rvert)$
[?]	[?]	1 [?]	2 [?]	2 [?]		
238	199	330	330	332	40	64
367	327	526	526	526	40	128
276	224	378	378	N/A	64	64
406	352	572	572	N/A	64	128
504	418	700	700	N/A	128	128

Table 1. Ciphertext modulus bit length ($\log(q)$) for two parties.

4 High Gear: SPDZ With Global ZKPoK Check

In terms of computation, the most expensive part of SPDZ is anything related to the encryption scheme, encryption, decryption, and homomorphic operations. The encryption algorithm is not only used for inputs but also both by the prover and the verifier in the zero-knowledge proof. While using a non-interactive zero-knowledge allows the parties to only generate one proof per input, independently of the number of parties, every party has to verify every other party's proof because every other party is assumed to be corrupted. With a growing number of parties, this is clearly the computational bottleneck of the protocol. In this section, we present a way to avoid this by summing all proofs and only checking the sum. This is similar to the threshold proofs presented by Keller et al. [?]. However, this neither reduces the communication nor the asymptotic computation because every party still has to send every proof to every party and then sum all the received proofs. Nevertheless, summing up the proofs is much cheaper than verifying them individually.

The High Gear protocol is meant to surpass Low Gear when executed with a high number of parties. To achieve this we design a new zero-knowledge proof which scales better when increasing the number of players. One can think of the High Gear proof of knowledge as customized interactive proof version from Damgård et al. [?] whereas Low Gear is a protocol ran with the non-interactive proof. The latter requires knowledge of the first message of the proof (sometimes called the commitment) to compute the challenge. In the context of combining the proof with many parties, the first message is the sum of an input from each party, which means that communication is required in any case. Therefore, there is less of an advantage in using the non-interactive proof.

Figure 10 shows our adapation of the zero-knowledge proof in Figure 9 from Damgård et al. [?]. The main conceptual difference is going from a two-party to a multi-party protocol. However, we have also simplified the bounds.

In the following we will prove that our protocol achieves the natural extension of the Σ-protocol properties in the multi-party setting.

Correctness. The equality in step 6 follows trivially from the linearity of the encryption. It remains to check the probability that an honest prover will fail the bounds check on $\|\mathbf{z}\|_\infty$ and $\|\mathbf{t}\|_\infty$ where the infinity norm $\|\cdot\|_\infty$ denotes the maximum of the absolute values of the components.

Remember that the honestly generated $E^{(i)}$ are (τ, ρ) ciphertexts. The bound check will succeed if the infinity norm of $\sum_{i=1}^{n}(\mathbf{y}^{(i)} + \sum_{k=1}^{\mathsf{sec}}(M_{e_{jk}} \cdot \mathbf{x}^{(i)}))$ is at most $2 \cdot n \cdot B_{\mathsf{plain}}$. This is always true because $\mathbf{y}^{(i)}$ is sampled such that $\|\mathbf{y}^{(i)}\|_\infty \leq B_{\mathsf{plain}}$ and $\|M_{\mathsf{e}} \cdot \mathbf{x}^{(i)}\|_\infty \leq \mathsf{sec} \cdot \tau \leq 2^{\mathsf{sec}} \cdot \tau = B_{\mathsf{plain}}$. A similar argument holds regarding ρ and B_{rand}.

Special soundness. To prove this property one must be able to extract the witness given response from two different challenges. In this case consider the transcripts $(\mathbf{x}, \mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$ and $(\mathbf{x}, \mathbf{a}, \mathbf{e}', (\mathbf{z}', T'))$ where $\mathbf{e} \neq \mathbf{e}'$. Recall that each

Π_{gZKPoK}

Denote $B_{\mathsf{plain}} = 2^{\mathsf{sec}} \cdot \tau$ and $B_{\mathsf{rand}} = 2^{\mathsf{sec}} \cdot \rho$ bounds for plaintext and randomness used for encryption where $\rho = 2 \cdot 3.2 \cdot \sqrt{N}$ and $\tau = p/2$.
Let $V = 2 \cdot \mathsf{sec} - 1$ and $M_{\mathbf{e}} \in \{0,1\}^{V \times \mathsf{sec}}$ the matrix associated with the challenge $\vee$ such that $M_{kl} = e_{k-l+1}$ for $1 \leq k - l + 1 \leq$ sec and 0 in all other entries. The randomness used for encryptions of $\mathbf{x}^{(i)}, \mathbf{y}^{(i)}$ is packed into matrices $\mathbf{r}^{(i)} \leftarrow (r_1^{(i)}, \ldots, r_{\mathsf{sec}}^{(i)})$ and $\mathbf{s}^{(i)} \leftarrow (s_1^{(i)}, \ldots, s_V^{(i)})$. Hence $\mathbf{r}^{(i)}, \in \mathbb{Z}^{\mathsf{sec} \times 3}$ and $\mathbf{s}^{(i)} \in \mathbb{Z}^{V \times 3}$ (each row has 3 entries accordingly to Enc defined in Section 2.2). Recall that here $\mathbf{x}^{(i)}$ is a vector with sec entries: $(\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{\mathsf{sec}}^{(i)})$ and $\mathbf{y}^{(i)}$ has V entries: $(\mathbf{y}_1^{(i)}, \ldots, \mathbf{y}_V^{(i)})$.

1. Each party P_i broadcasts $E^{(i)} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}^{(i)}, \mathbf{r}^{(i)})$.
2. Each party P_i samples each entry of $\mathbf{y}^{(i)}$ and $\mathbf{s}^{(i)}$ randomly w.r.t to the bounds $\|\mathbf{y}_j^{(i)}\|_\infty \leq B_{\mathsf{plain}}, \|\mathbf{s}_j^{(i)}\|_\infty \leq B_{\mathsf{rand}}$ where $j \in [1 \ldots \mathrm{sec}]$. Then P_i uses the random coins $\mathbf{s}^{(i)}$ to compute $\mathbf{a}^{(i)} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}^{(i)}, \mathbf{s}^{(i)})$ and broadcasts $\mathbf{a}^{(i)}$.
3. The parties use $\mathcal{F}_{\mathsf{Rand}}$ to sample $\mathbf{e} \in \{0,1\}^{\mathrm{sec}}$.
4. Each party P_i computes $\mathbf{z}^{(i)\intercal} = \mathbf{y}^{(i)\intercal} + M_{\mathbf{e}} \cdot \mathbf{x}^{(i)\intercal}$ and $T^{(i)} = \mathbf{s}^{(i)} + M_{\mathbf{e}} \cdot \mathbf{r}^{(i)}$ and broadcasts $(\mathbf{z}^{(i)}, T^{(i)})$.
5. Each party P_i computes $\mathbf{d}^{(i)} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}^{(i)}, \mathbf{t})$ where $\mathbf{t}$ ranges through all rows of $T^{(i)}$ then store the sum $\mathbf{d} = \sum_{i=1}^{n} \mathbf{d}^{(i)}$.
6. The parties compute $E = \sum_i E^{(i)}$ $\mathbf{a} = \sum_i \mathbf{a}^{(i)}$, $\mathbf{z} = \sum_i \mathbf{z}^{(i)}$ and $T = \sum_i T^{(i)}$ and conduct the checks (allowing the norms to be $2n$ times bigger to accommodate the summations):
$$\mathbf{d}^\intercal = \mathbf{a}^\intercal + (M_{\mathbf{e}} \cdot E), \quad \|\mathbf{z}\|_\infty \leq 2 \cdot n \cdot B_{\mathsf{plain}}, \quad \|T\|_\infty \leq 2 \cdot n \cdot B_{\mathsf{rand}}.$$
7. If the check passes, the parties output $\sum_{i=1}^{n} E^{(i)}$.

Fig. 10. Protocol for global proof of knowledge of ciphertext

party has a different secret $\mathbf{x}^{(i)}$. Because both challenges have passed the bound checks during the protocol, we get that:

$$(M_{\mathbf{e}} - M_{\mathbf{e}'}) \cdot E^{\mathsf{T}} = (\mathbf{d} - \mathbf{d}')^{\mathsf{T}}$$

To solve the equation for E notice that $M_{\mathbf{e}} - M_{\mathbf{e}'}$ is a matrix with entries in $\{-1, 0, 1\}$ so we must solve a linear system where $E = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_k, \mathbf{r}_k)$ for $k = 1, \ldots, \mathsf{sec}$. This can be done in two steps: solve the linear system for the first half: $\mathbf{c}_1, \ldots, \mathbf{c}_{\mathsf{sec}/2}$ and then for the second half: $\mathbf{c}_{\mathsf{sec}/2+1}, \ldots, \mathbf{c}_{\mathsf{sec}}$. For the first step identify a square submatrix of $\mathsf{sec} \times \mathsf{sec}$ entries in $M_{\mathbf{e}} - M_{\mathbf{e}'}$ which has a diagonal full of 1's or -1's and it is lower triangular. This can be done since there is at least one component j such that $e_j \neq e'_j$. Recall that the plaintexts $\mathbf{z}_k, \mathbf{z}'_k$ have norm less than B_{plain} and the randomness used for encrypting them, $\mathbf{t}_k, \mathbf{t}'_k$, have norm less than B_{rand} where k ranges through $1, \ldots, \mathsf{sec}$.

Solving the linear system from the top row until the middle row via substitution we obtain in the worst case: $\|\mathbf{x}_k\|_\infty \leq 2^k \cdot n \cdot B_{\mathsf{plain}}$ and $\|\mathbf{y}_k\|_\infty \leq 2^k \cdot n \cdot B_{\mathsf{rand}}$ where k ranges through $1, \ldots, \mathsf{sec}/2$. The second step is similar to the first with the exception that now we have to look for an upper triangular matrix of $\mathsf{sec} \times \mathsf{sec}$. Then solve the linear system from the last row until the middle row. In this way we extract $\mathbf{x}_k, \mathbf{r}_k$ which form $(2^{\mathsf{sec}/2+1} \cdot n \cdot B_{\mathsf{plain}}, 2^{\mathsf{sec}/2+1} \cdot n \cdot B_{\mathsf{rand}})$ or $(2^{3\mathsf{sec}/2+1} \cdot n \cdot \tau, 2^{3\mathsf{sec}/2+1} \cdot n \cdot \rho)$ ciphertexts. This means that the slack is $2^{3\mathsf{sec}/2+1}$.

Honest verifier zero-knowledge. Here we give a simulator $\mathcal{S}$ for an honest verifier (each party P_i acts as one at one point during the protocol). The simulator's purpose is to create a transcript with the verifier which is indistinguishable from the real interaction between the prover and the verifier. To achieve this $\mathcal{S}$ samples uniformly $\mathbf{e} \overset{\$}{\leftarrow} \{0,1\}^{\mathsf{sec}}$ and then create the transcript accordingly: sample $\mathbf{z}^{(i)}$ such that $\|\mathbf{z}^{(i)}\|_\infty \leq B_{\mathsf{plain}}$ and $T^{(i)}$ such that $\|T^{(i)}\|_\infty \leq B_{\mathsf{rand}}$ and then fix $\mathbf{a}^{(i)} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}^{(i)}, T^{(i)}) - (M_{\mathbf{e}} \cdot E^{(i)})$, where the encryption is applied component-wise. Clearly the produced transcript $(\mathbf{a}^{(i)}, \mathbf{e}^{(i)}, \mathbf{z}^{(i)}, T^{(i)})$ passes the final checks and the statistical distance to the real one is $2^{-\mathsf{sec}}$, which is negligible with respect to sec.

Putting things together. In the context of our triple generation, we model Π_{gZKPoK} as $\mathcal{F}^S_{\mathsf{gZKPoK}}$ in Figure 11. We will argue below that Π_{gZKPoK} implements $\mathcal{F}^S_{\mathsf{gZKPoK}}$ with slack $S = 2^{3\mathsf{sec}/2+1}$.

$\mathcal{F}^S_{\mathsf{gZKPoK}}$ does not guarantee the correctness of individual corrupted parties ciphertexts but the correctness of the resulting sum. This suffices because only the latter is used in the protocol. A rewinding simulator still can extract individual inputs, but there is no guarantee that either they are in fact pre-images of the encryptions sent by corrupted parties or they are subject to any bounds. Both properties only hold for the sum. This is modeled by $\mathcal{F}^S_{\mathsf{gZKPoK}}$ only outputting a sum, and it is easy to see that this output suffices for SPDZ.

$\mathcal{S}^S_{\mathsf{gZKPoK}}$ in Figure 12 describes our simulator. The rewinding technique is the same as in the soundness simulator for Σ-protocol and therefore has the same

$\mathcal{F}^{S}_{\mathsf{gZKPoK}}$

Let H denote the set of honest parties. Initially, all parties input pk. Then, the following can happen repeatedly:

1. Every honest party P_i inputs $\mathbf{x}^{(i)}$.
2. Output $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}^{(i)})$ for all $i \in H$ to the adversary.
3. The adversary inputs $\mathbf{x}'$.
4. The functionality sends $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}' + \sum_{i \in H} \mathbf{x}^{(i)})$, all with noise increased by a factor $n \cdot S$ to all parties.

Fig. 11. Functionality for global proof of knowledge of ciphertext

$\mathcal{S}^{S}_{\mathsf{gZKPoK}}$

Let A denote the set of corrupted parties, and H the set of honest ones.

1. Receive $E^{(i)}$ for all $i \in H$.
2. Sample $\mathbf{e} \xleftarrow{\$} \{0,1\}^{\mathsf{sec}}$.
3. Use the honest-verifier zero-knowledge simulator above to generate transcripts $(\mathbf{a}^{(i)}, \mathbf{e}, (\mathbf{z}^{(i)}, T^{(i)}))$ for $i \in H$.
4. Send $\{\mathbf{a}^{(i)}\}_{i \in H}$ to the adversary.
5. Receive $(E^{(i)}, \mathbf{y}^{(i)}, \mathbf{a}^{(i)})$ for every corrupted party P_i from the adversary.
6. Emulating $\mathcal{F}_{\mathsf{Rand}}$, send $\mathbf{e}$ to the adversary.
7. Receive $(\mathbf{z}^{(i)}, T^{(i)})$ for every corrupted party P_i from the adversary.
8. Check whether $\sum_{i \in A} \mathbf{z}^{(i)}$ and $\sum_{i \in A} T^{(i)}$ meets the bounds. Abort if not.
9. Rewinding the adversary, sample $\tilde{\mathbf{e}} \neq \mathbf{e}$ and conduct the same check for the adversary's responses $\{\tilde{\mathbf{z}}^{(i)}, \tilde{T}^{(i)}\}_{i \in A}$ until the check passes.
10. Use the Σ-protocol extractor on $\{(E^{(i)}, \mathbf{y}^{(i)}, \mathbf{a}^{(i)}, \mathbf{e}, \mathbf{z}^{(i)}, T^{(i)}, \tilde{\mathbf{e}}, \tilde{\mathbf{z}}^{(i)}, \tilde{T}^{(i)})\}_{i \in A}$ to compute $\{\mathbf{x}^{(i)}\}_{i \in A}$ and input $\sum_{i \in A} \mathbf{x}^{(i)}$ to $\mathcal{F}^{S}_{\mathsf{gZKPoK}}$.

Fig. 12. Simulator for global proof of knowledge of ciphertext

running time (roughly inverse to the success probability of a corrupted prover). See Section 3 of [?] for details. Similarly, the finishing probability of $\mathcal{S}^{S}_{\mathsf{gZKPoK}}$ is the same as a protocol because it only aborts if the first check fails.

5 Implementation

We have implemented all three approaches to triple generation in this paper and measured the throughputs achieved by them in comparison to previous results with SPDZ [?,?] and MASCOT [?]. We have used the optimized distributed decryption in Appendix A for SPDZ-1, SPDZ-2, and High Gear. Our code is written in C++ and uses MPIR [?] for arithmetic with large integers. We use Montgomery modular multiplication and the Chinese reminder theorem representation of polynomials wherever beneficial. See Gentry et al. [?] for more details.

Note that the parameters chosen by Damgård et al. [?][Appendix A] for the non-interactive zero-knowledge proof imply that the prover has to re-compute the proof with probability 1/32 as part of a technique called rejection sampling. We have increased the parameters to reduce this probability by up to 2^{20} as long as it would not impact the performance, i.e., the number of 64-bit words needed to represent p_o and p_1 would not change.

All previous implementations have benchmarks for two parties on a local network with 1 Gbit/s throughput on commodity hardware. We have have used i7-4790 and i7-3770S CPUs with 16 to 32 GB of RAM, and we have re-run and optimized the code by Damgård et al. [?] for a fairer comparison. Table 2 shows our results in this setting. SDPZ-1 and SPDZ-2 refer to the two different proofs for ciphertexts, the Schnorr-like protocol presented in the original paper [?] and the cut-and-choose protocol in the follow-up work [?], the latter with either covert or active security. k-covert security is defined as a cheating adversary being caught with probability $1/k$, and by k-bit security we mean a statistical security parameter of k. Throughout this section, we will round figures to the two most significant digits for a more legible presentation.

To allow direct comparisons with previous works, we have benchmarked our protocols for several choices of security parameters and field size. The main difference between our implementation of SPDZ with the Schnorr-like protocol to the previous one [?] is the underlying BGV implementation because the protocol is the same.

In Table 3, also analyse the communication per triple of some protocols with active security and compared the actual throughput to the maximum possible on a 1 Gbit/s link (network throughput divided by the communication per triple). The higher the difference between actual and maximum possible, the more time is spent on computation. The figures show that MASCOT has very low computation; the actual throughput is more than 90% of the maximum possible. On the other hand, all BGV-based implementations have a significant gap, which is to be expected. The relative gap increases with increasing security in Low Gear

	Triples/s	Security	BGV impl.	$\log_2(\lvert\mathbb{F}_p\rvert)$
SPDZ-1 [?]	79	40-bit active	NTL	64
SPDZ-2 [?]	158	20-covert	specific	64
SPDZ-2 [?]	36	40-bit active	specific	64
MASCOT [?]	5,100	64-bit active	$\perp$	128
SPDZ-1 (ours)	12,000	40-bit active	specific	64
SPDZ-1 (ours)	6,400	64-bit active	specific	128
SPDZ-1 (ours)	4,200	128-bit active	specific	128
SPDZ-2 (ours)	3,900	20-covert	specific	64
SPDZ-2 (ours)	1,100	40-bit active	specific	64
Low Gear (Section 3)	59,000	40-bit active	specific	64
Low Gear (Section 3)	30,000	64-bit active	specific	128
Low Gear (Section 3)	15,000	128-bit active	specific	128
High Gear (Section 4)	11,000	40-bit active	specific	64
High Gear (Section 4)	5,600	64-bit active	specific	128
High Gear (Section 4)	2,300	128-bit active	specific	128

Table 2. Triple generation for prime field with two parties on a 1 Gbit/s LAN.

because 32 GB of memory does not suffice one generator thread per core, hence there is some computation capacity left unused.

	Communication	Security	$\log_2(\lvert\mathbb{F}_p\rvert)$	Triples/s	Maximum
SPDZ-2	350 kbit	40	64	1,100	2,900
MASCOT [?]	180 kbit	64	128	5,100	5,600
SPDZ-1	23 kbit	40	64	12,000	44,000
SPDZ-1	32 kbit	64	128	6,400	31,000
SPDZ-1	37 kbit	128	128	4,200	27,000
Low Gear (Section 3)	9 kbit	40	64	59,000	110,000
Low Gear (Section 3)	15 kbit	64	128	30,000	68,000
Low Gear (Section 3)	17 kbit	128	128	15,000	60,000
High Gear (Section 4)	24 kbit	40	64	11,000	42,000
High Gear (Section 4)	34 kbit	64	128	5,600	30,000
High Gear (Section 4)	42 kbit	128	128	2,300	24,000

Table 3. Communication per prime field triple (one way) and actual vs maximum throughput with two parties on a 1 Gbit/s link

WAN setting. For a more complete picture, we have also benchmarked our protocols in the same WAN setting as Keller et al. [?], restricting the bandwidth to 50 Mbit/s and imposing a delay of 50 ms to all communication. Table 4 shows our results in similar manner to Table 3. As one would expect, the gap between actual throughput and maximum possible is more narrow because the communication becomes more of a bottleneck, and the performance is closely related to the required communication.

	Communication	Security	$\log_2(\mathbb{F}_p\|)$	Triples/s	Maximum
MASCOT [?]	180 kbit	64	128	214	275
SPDZ-1	23 kbit	40	64	1,800	2,200
SPDZ-1	32 kbit	64	128	1,400	1,600
SPDZ-1	37 kbit	128	128	1,100	1,400
Low Gear (Section 3)	9 kbit	40	64	4,500	5,600
Low Gear (Section 3)	15 kbit	64	128	3,200	3,400
Low Gear (Section 3)	17 kbit	128	128	2,600	3,000
High Gear (Section 4)	24 kbit	40	64	1,600	2,100
High Gear (Section 4)	34 kbit	64	128	1,300	1,500
High Gear (Section 4)	42 kbit	128	128	700	1,200

Table 4. Communication per prime field triple (one way) and actual vs maximum throughput with two parties on a 50 Mbit/s link

Fields of characteristic two. For a more thourough comparison with MASCOT, we have also implemented our protocols for the field of size 2^{40} using the same approach as Damgård et al. [?]. Table 5 shows the low performance of homomorphic encryption-based protocols with fields of characteristic two. This has been observed before: in the above work, the performance for $\mathbb{F}_{2^{40}}$ is an order of magnitude worse than for $\mathbb{F}_p$ with a 64-bit bit prime. The main reason is that BGV lends itself naturally to cleartexts modulo some integer p. The construction for $\mathbb{F}_{2^{40}}$ sets $p = 2$ and uses 40 slots to represent an element whereas an element of $\mathbb{F}_p$ for a prime p only requires one ciphertext slot.

	Triples/s	Security	BGV impl.	$\mathbb{F}_{2^n}$
SPDZ-1 [?]	16	40-bit active	NTL	40
MASCOT [?]	5,100	64-bit active	$\perp$	128
SPDZ-1 (ours)	67	40-bit active	specific	40
SPDZ-2 (ours)	24	20-covert	specific	40
SPDZ-2 (ours)	8	40-bit active	specific	40
Low Gear (Section 3)	117	40-bit active	specific	40
High Gear (Section 4)	67	40-bit active	specific	40

Table 5. Triple generation for characteristic two with two parties on a 1 Gbit/s LAN.

More than two parties. Increasing the number of parties, we have benchmarked our protocols and our implementation of SPDZ with up to 64 r4.16xlarge instances on Amazon Web Services. Figure 13 shows that both Low and High Gear improve over SPDZ-1, with High Gear taking the lead from about ten parties. Missing figures do not indicate failed experiments but rather omitted experiments due to financial reasons.

At the time of writing, one hour on an r4.16xlarge instance in US East costs $4.256. Therefore, the number of triples per Dollar and party varies between 190 million (two parties with Low Gear) and 13 million (64 parties with High Gear).

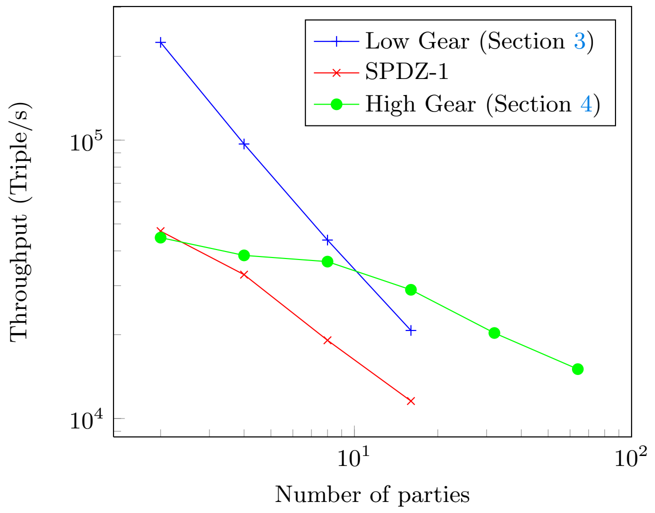

Fig. 13. Triple generation for prime field on AWS r4.16xlarge instances

5.1 Vickrey Auction for 100 Parties

As a motivation for computation with a high number of parties, we have implemented a secure Vickrey second price auction [?], where 100 parties input one bid each. Table 6 shows our online phase timings for two different Amazon Web Services instances.

The Vickrey auction requires 44,571 triples. In Table 7, we compare the offline cost of MASCOT and our High Gear protocol on AWS m3.2xlarge instances.

6 Future work

Recently, there has been an improved zero-knowledge proof of knowledge of bounded pre-images for LWE-style one-way functions [?]. It reduces the extra

AWS instance	Time	Cost per party
t2.nano	9.0 seconds	\$0.000017
c4.8xlarge	1.4 seconds	\$0.000741

Table 6. Online phase of Vickrey auction with 100 parties, each inputting one bid.

	Time	Cost per party
MASCOT [?]	1,300 seconds	\$0.190
High Gear (Section 4)	98 seconds	\$0.014

Table 7. Offline phase of Vickrey auction with 100 parties, each inputting one bid.

ciphertexts per proven ciphertext from two (in our protocol) to any number larger than one dependent on the number of ciphertexts that are proven simultaniously. More concretely, for $u \cdot \mathsf{sec}$ ciphertexts in the one proof (and $u \geq 1$), the prover needs to send $(u+1) \cdot \mathsf{sec}$ ciphertexts in the first round, hence the amortized overhead is $(u+1)/u$. This compares to $2u\mathsf{sec} - 1$ ciphertexts with amortized over $2 - 1/(u \cdot \mathsf{sec})$ in our scheme. However, we estimate that the benefit of the newer proof strongly depend on the parameters and the available memory. For some parameters, we found that our implementation would exhaust 32 GB of memory with less than eight generation threads. We therefore could not exhaust the computational capacity of the CPU. Note that our implementation stores all necessary information for the proof in memory, and consider than one ciphertext takes up to $2^{16} \cdot 700 \cdot 2$ bits or ≈ 11 MBytes. This means that, for 128-bit active security, we require about $(3\mathsf{sec} - 1) \cdot 11$ MBytes or ≈ 4.4 GBytes of storage for the ciphertexts alone (not considering any cleartexts). It would be interesting to see how the newer proof fares and whether using a solid state disk for storage would improve the performance.

References

A More Efficient Decryption to Secret Sharing

SPDZ requires to decrypt values to an additive secret sharing without revealing them (e.g., the encrypted MACs). This is done with a protocol called Reshare, which masks the ciphertext with a secret-shared mask before using public distributed decryption. The secret-shared is then subtracted from the public (masked) cleartext, resulting in a secret sharing of the actual cleartext. The downside of this approach is the cost of the zero-knowledge proof required to prove the correct encryption of the mask. In Figure 14 we propose a protocol that avoids this. Note that it only works in the case where no fresh encryption of the cleartext is required. Therefore, it is only useful for encryption of MAC values, but not for the encryption of ab, where a fresh encryption is required to compute the MAC.

DistDec

Let $(c_0, c_1) \in R_q^2$ be an encryption of $\mathbf{m} \in \mathbb{F}_p^N$ and $s_i \in R_q$ the key share of party i such that $\sum s_i \in R_q$ is the secret key and $\|s_i\|_\infty$ is "small" in usual sense. Furthermore, let B denote a bound on the noise, that is $\|c_0 - s \cdot c_1\|_\infty$.

1. Party i samples $\mathbf{f}_i \overset{\$}{\leftarrow} [0, B \cdot 2^{\mathsf{sec}}]^N$.
2. Party 1 computes $\mathbf{m}'_i := (c_0 - s_1 \cdot c_1) - \mathbf{f}_1 \bmod q$, and every other party i computes $\mathbf{m}'_i := -s_i \cdot c_1$.
3. Party i broadcasts $\mathbf{m}'_i$.
4. Party 1 outputs $\mathbf{m}_1 := (\sum_{i=1}^n \mathbf{m}'_i \bmod q + \mathbf{f}_1) \bmod p$, and every other party i outputs $\mathbf{m}_i := \mathbf{f}_i \bmod p$.

Fig. 14. Distributed decryption to secret sharing

Correctness with honest parties. The protocol is easily seen to be correct if the parties follow it and $B \cdot n \cdot p \cdot 2^{\mathsf{sec}} < q/2$:

$$\begin{aligned}\sum_{i=1}^n \mathbf{m}_i \bmod p &= \Big(\sum_{i=1}^n \mathbf{m}'_i \bmod q + \sum_{i=1}^n \mathbf{f}_i - \sum_{i=1}^n \mathbf{f}_i\Big) \bmod p \\ &= \Big(c_0 - \sum_{i=1}^n s_i \cdot c_1\Big) \bmod q \bmod p \\ &= \mathbf{m}.\end{aligned}$$

The first equation holds because q is large enough, and the last equation is how the decryption is defined.

Privacy. The protocol does not reveal any information because $\mathbf{f}_i$ is chosen from an interval 2^{sec} larger than $c_0 - s \cdot c_1$, which makes $\sum_{i=1}^n \mathbf{m}'_i$ statistically indistinguishable to a random value in an interval of at least size $B \cdot 2^{\mathsf{sec}}$ with respect to sec.

Correctness with malicious parties. Malicious parties can add an error that is independent of the cleartext. This is the same as in the original protocol, and it is dealt with later in the MAC check.