Multipars: Reduced-Communication MPC over $\mathbb{Z}_{2^k}$

Sebastian Hasler, Pascal Reisert, Marc Rivinius, and Ralf Küsters

University of Stuttgart
Stuttgart, Germany
{sebastian.hasler,pascal.reisert,marc.rivinius,ralf.kuesters}@sec.uni-stuttgart.de

Abstract. In recent years, actively secure SPDZ-like protocols for dishonest majority, like SPD$\mathbb{Z}_{2^k}$, Overdrive2k, and MHz2k, over base rings $\mathbb{Z}_{2^k}$ have become more and more efficient. In this paper, we present a new actively secure MPC protocol *Multipars* that outperforms these state-of-the-art protocols over $\mathbb{Z}_{2^k}$ by more than a factor of 2 in the two-party setup in terms of communication. Multipars is the first actively secure N-party protocol over $\mathbb{Z}_{2^k}$ that is based on linear homomorphic encryption (LHE) in the offline phase (instead of oblivious transfer or somewhat homomorphic encryption in previous works). The strong performance of Multipars relies on a new adaptive packing for BGV ciphertexts that allows us to reduce the parameter size of the encryption scheme and the overall communication cost. Additionally, we use modulus switching for further size reduction, a new type of enhanced CPA security over $\mathbb{Z}_{2^k}$, a truncation protocol for Beaver triples, and a new LHE-based offline protocol without sacrificing over $\mathbb{Z}_{2^k}$.
We have implemented Multipars and therewith provide the fastest preprocessing phase over $\mathbb{Z}_{2^k}$. Our evaluation shows that Multipars offers at least a factor of 8 lower communication costs and up to a factor of 15 faster runtime in the WAN setting compared to the currently best available actively secure MPC implementation over $\mathbb{Z}_{2^k}$.

1 Introduction

With multi-party computation (MPC), several parties can compute arbitrary functions or circuits on private data without revealing any information about the inputs apart from the result and what might be inferred from it. While the theoretical foundations of MPC go well back to the last century, MPC has seen a rise in popularity in recent years—both in academia and industry, e.g., for privacy-preserving machine learning [MZ17, CKR$^+$20, HLHD22], which is due to the development of first (reasonably) efficient MPC protocols like SPDZ [DPSZ12]. One of the main features of SPDZ and related protocols like [DKL$^+$13, KOS16, KPR18, BCS19] are their high security guarantees that protect privacy of honest parties' input data even if all other MPC parties act maliciously. To achieve both—high security and efficiency—SPDZ-like protocols use a two-phase approach with an input-independent offline phase which provides structured random data, e.g., Beaver triples [Bea91], to a then lightweight online phase. It therewith allows for an efficient online computation on sensitive input data.

Efficiency of these two-phase MPC protocols is mostly dictated by their communication complexity (cf. [KPR18, CKR$^+$20]). Both the number of communication rounds and the amount of data that needs to be sent are dominating the runtime of protocols if parties communicate over real networks, e.g., the Internet, which introduces communication delay and generally has only a limited bandwidth. Local computation time is usually less critical as it is comparably lower than the communication time and can often be fully parallelized. Apart from their effect on the overall runtime, communication and bandwidth also affect the monetary costs especially when the MPC protocol is deployed on paid (cloud) infrastructures.

One key characteristic that influences the communication cost and overall runtime is the type of primitives used by SPDZ-like protocols in the offline phase: SPDZ [DPSZ12, DKL$^+$13] and Overdrive HighGear [KPR18] use somewhat homomorphic encryption (SHE), MASCOT [KOS16] and SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18] use oblivious transfer, and LowGear [KPR18] uses LHE. The encryption based SPDZ-like protocols use a BGV-type encryption scheme [BGV12] which is naturally linearly homomorphic, i.e., supports the addition of encrypted data and multiplication of encrypted data with plaintext vectors—akin to single instruction multiple data (SIMD) operations. The BGV scheme can become somewhat homomorphic, i.e., also support one ciphertext-ciphertext multiplication. However, this requires a larger ciphertext size.[1] We will refer to protocols that use only the LHE property of the BGV-scheme and therefore have generally smaller ciphertext sizes as LHE-based protocols. Protocols that use BGV ciphertext-ciphertext multiplications are called SHE-based.

Furthermore, SPDZ-like protocols can also be characterized by the underlying arithmetic model they use. The main branch including SPDZ itself and major improvements like [KOS16, KPR18, BCS19] work over finite fields $\mathbb{F}_p$. Finite field arithmetic comes with several strong properties and supports, for instance, aforementioned single instruction multiple data (SIMD) operations in protocols based on homomorphic encryption (HE), since the underlying cyclotomic rings decompose nicely over finite fields.

While finite field arithmetic has a wide variety of applications, it is not ideally suited to work with modern CPUs, which naturally support arithmetic modulo 2^k. Additionally, arithmetic modulo 2^k is better suited to compute binary operations, shifts, truncations, and comparisons [BLW08, CS10, CDE$^+$18]. It is therefore not surprising that a more recent, but still far less developed, branch of SPDZ-like protocols tries to construct secure protocols over $\mathbb{Z}_{2^k}$. As might be expected, this is a non-trivial task given the weaker structural properties available over a base ring. Some of the resulting problems have however been solved in recent years.

For example, the obvious existence of non-trivial non-invertible elements in $R = \mathbb{Z}_{2^k}$ implies that the classical MAC authentication, which guarantees security against malicious adversaries in the $\mathbb{F}_p$-case, no longer provides any protection. This problem has been addressed by SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18], where the authors extend the MAC formalism of SPDZ by computing MACs no longer in R directly but in some larger ring $R' = \mathbb{Z}_{2^{k+s}}$ and therewith circumvent the invertibility issues to regain security against malicious adversaries.

Another issue over $\mathbb{Z}_{2^k}$ is the incompatibility of 2^k-based cyclotomic rings (commonly used in the triple production phase of LHE/SHE offline phases) with SIMD instructions.[2] Overdrive2k [OSV20] addresses this problem and presents the first SHE-based protocol. To this end, the authors of [OSV20] construct a new packing method, which uses about 20% of the available slots, and hence, 20% of the SIMD potential of the encryption scheme. While this is still far less efficient than in the field case, where all ciphertext slots, i.e., 100%, can be used, it presented a first reasonably efficient HE-based offline phase in the $\mathbb{Z}_{2^k}$ setup and sparked further investigation in this kind of protocol. The currently best ring-based protocol MHz2k [CKL21] is SHE-based (just as [OSV20]), i.e., it relies on SHE, but uses a new packing that increases the number of slots that can be used to approximately 50%.

The focus of our paper is to further improve the efficiency of SPDZ-like protocols over $\mathbb{Z}_{2^k}$ with a special focus on the highly relevant low-party setups. Low-party setups are well-suited to cloud-based applications of MPC in real-world applications [JVC18, RRK$^+$20, TKTW21, HLHD22]. Motivated by the superior performance of Overdrive LowGear in the case of a base field, we create a $\mathbb{Z}_{2^k}$ version of LowGear that outperforms the SHE-based $\mathbb{Z}_{2^k}$-protocol [CKL21], e.g., by around 2.2× in the two-party setup analyzed in [CKL21]. We call our protocol Multipa**rs** (**multi-par**ty **rus**t) because we implemented it in the Rust programming language.

The efficiency of Multipars is based on several technical improvements of independent interest. First, we use a new adaptive packing which changes depending on the operations it is used in. For some ciphertexts we use the *tweaked interpolation packing* from [CKL21], since it has better packing efficiency than the original packing from Overdrive2k [OSV20]. When we do not need to perform a slotwise plaintext-ciphertext multiplication, we instead use the coefficient packing, which allows us to avoid the still significant factor 2 packing overhead from [CKL21]. That is, we use 50% of the ciphertext slots with the tweaked interpolation packing in our multiplication subprotocol (Figure 2) and 100% with the coefficient packing in our authentication subprotocol (Figure 3). Similarly, the BGV parameters are chosen separately (and hence optimally) for different subprotocols.

[1] We remark that these SHE schemes additionally need some key switching material not needed for LHE—we refer to [DKL$^+$13] for further details.

[2] Note that SPD$\mathbb{Z}_{2^k}$ uses oblivious transfer which does not use SIMD instructions even in the field case, i.e., the incompatibility issue does not occur, but also the advantage of more recent SIMD-based constructions is not used, which ultimately leads to a lower communication efficiency than, e.g., Overdrive2k [OSV20].

Second, we lift the recent optimized LHE-based protocol of [RRKK23] to the $\mathbb{Z}_{2^k}$ setup. LowGear 2.0 [RRKK23] avoids the costly sacrificing step in LowGear by intertwining the triple production and triple authentication subprotocols. On its own, this approach is no longer secure if we work over $\mathbb{Z}_{2^k}$, since an adversary can deviate from the honest packing technique. However, the tweaked interpolation packing from [CKL21] comes with a zero-knowledge proof of message knowledge (ZKPoMK) which prevents such an attack. Additionally, [RRKK23] as well as [KPR18] use the enhanced CPA security of the underlying cryptosystem to protect against selective failure attacks (in the MAC checks). Again, this property does not transfer to our cryptosystems over $\mathbb{Z}_{2^k}$. In fact, [CRFG20] showed that enhanced CPA security does not hold for any HE scheme over $\mathbb{Z}_{2^k}$. We circumvent this problem by introducing a slightly relaxed form of enhanced CPA security, which we call 2^s*-enhanced CPA security*. We prove that our cryptosystems provide this type of security. In particular, a new truncation protocol for Beaver triples allows us to use this 2^s-enhanced CPA security to construct a maliciously secure triple generation protocol.

Third, we employ modulus switching to reduce the size of (most) ciphertexts, which leads to about 40% less overall communication. To use modulus switching for size reduction (instead of its traditional use as noise reduction) is a useful tool that might be applicable in other SPDZ-like setups.

Last, we construct a new zero-knowledge proof of plaintext knowledge (ZKPoPK) for BGV with plaintext modulus 2^k that—in contrast to prior work [BCS19, CKL21]—does not require post-processing the ciphertexts while attaining the same security parameters. In our concrete instantiation of the ZKPoPK, we employ the rejection sampling idea from [Lyu09] in the non-interactive setting and therewith reduce the soundness slack and as a byproduct achieve perfect (instead of statistical) zero-knowledge.

We formally prove the security of our new protocols and have implemented Multipars in Rust. We therewith provide the currently fastest software implementation of a preprocessing phase for SPD$\mathbb{Z}_{2^k}$. Our implementation is available at [Has23].

We have evaluated Multipars in terms of communication per triple produced and compare it against prior works in the $\mathbb{Z}_{2^k}$ setting: In the two-party[3] setup Multipars outperforms MHz2k [CKL21] by around 2.2×, Overdrive2k [OSV20] by around 11×, and SPD$\mathbb{Z}_{2^k}$ [CDE+18] by 8–30×. Our benchmarks show that, even in fast networks (LAN, 1 Gbps, 0.2 ms RTT, 16 local threads, 64-bit statistical security and plaintext size) where computation is the bottleneck, Multipars can produce more triples per second than the SPD$\mathbb{Z}_{2^k}$ preprocessing phase implemented in MP-SPDZ [DEF+19]—the only SPDZ-like protocol over $\mathbb{Z}_{2^k}$ with a publicly available implementation besides ours. Our advantage compared to SPD$\mathbb{Z}_{2^k}$ is especially pronounced in the WAN setting (100 ms RTT) with a speedup of up to 13.7× at 50 Mbps or 15.2× at 500 Mbps.

Contributions.

– We present a new SPDZ-like triple generation protocol Multipars over $\mathbb{Z}_{2^k}$ that outperforms the prior best protocol in the same setting, MHz2k, by a factor of 2.2 in terms of communication.
– With the newly introduced concept of 2^s*-enhanced CPA security* and a new truncation technique for Beaver triples, we provide an efficient protection against selective failure attacks.
– We introduce technical tools of independent interest (adaptive packing, new modulus switching based techniques, maliciously secure key generation) which contribute to the security and the high efficiency of Multipars compared to previous protocols over $\mathbb{Z}_{2^k}$.
– We present a new software implementation of an HE-based preprocessing phase for SPD$\mathbb{Z}_{2^k}$. Previous HE-based protocols like MHz2k [CKL21] or Overdrive2k [OSV20] do currently not provide an implementation. Our implementation computes triples up to 15.2 times faster than the fastest existing ring-based implementation in MP-SPDZ [DEF+19].

Structure of the Paper. After introducing preliminaries in Section 2, we present our instantiation of the BGV encryption scheme and a secure key generation protocol in Section 3. We show the 2^s-enhanced CPA security of this BGV instantiation in Section 4. Our main contribution, the new LHE-based triple generation, is presented in Section 5 with a security proof in Appendix A. Next, we introduce our new

[3] In the related works we compare against, only the two-party setup has been analyzed.

zero-knowledge proofs in Section 6. Finally, we choose concrete parameters and evaluate our protocol and implementation based on these parameters in Section 7. More details can be found in the appendix and our implementation is provided at [Has23].

2 Preliminaries

Notation. For $n \in \mathbb{N}$, let $[n] := \{0, \dots, n-1\}$, let $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ and let $\mathbb{Z}_n^*$ be its group of units. We use bold letters for vectors ($\boldsymbol{v}$) and capital letters for matrices (A). For vectors $\boldsymbol{v}, \boldsymbol{w}$ we denote their concatenation by $\boldsymbol{v}||\boldsymbol{w}$. For $x \equiv y \pmod{2^k}$ we also use the shorthand notation $x \equiv_k y$.

2.1 Secret Sharing Scheme

We use the authenticated additive secret sharing scheme introduced in [CDE+18] for base rings $\mathbb{Z}_{2^k}$: Given a security parameter s, each party P_i receives a $[\alpha]_i \in \mathbb{Z}_{2^s}$ such that $\alpha \equiv_{k+s} \sum_i [\alpha]_i$ for a secret MAC key $\alpha \in \mathbb{Z}_{2^{k+s}}$. For a secret $x \in \mathbb{Z}_{2^k}$ each party P_i receives an *additive share* $[x]_i \in \mathbb{Z}_{2^{k+s}}$ such that $x \equiv_k \sum_i [x]_i$. To *open* a shared $[x]$, each party P_i broadcasts $[x]_i$ and all parties can reconstruct x. The scheme is full threshold, i.e., if an adversary controls all but one share, they still cannot deduce any information on the secret value. In addition to the actual share $[x]_i$, each party P_i also gets a share $[\gamma_x]_i \in \mathbb{Z}_{2^{k+s}}$ such that $\sum_i [\gamma_x]_i \equiv_{k+s} \alpha \sum_i [x]_i$. $\gamma_x \in \mathbb{Z}_{2^{k+s}}$ is called *MAC* of x. The MACs are used in our MAC check protocol $\Pi_{\mathsf{SingleCheck}}$ (cf. Figure 6) to verify the integrity of opened shares and to detect malicious behavior. We call the pair $[\![x]\!]_i = ([x]_i, [\gamma_x]_i)$ an *authenticated share* of x at party P_i. Please note that this (authenticated) secret sharing scheme allows local computation of linear combinations of (authenticated) shares, i.e., $[x]_i + [y]_i = [x+y]_i$ or $c \cdot [x]_i = [c \cdot x]_i$ for shares of x and y and a public known constant c. The same properties also apply to authenticated shares, e.g., $[\![x+y]\!]_i = [\![x]\!]_i + [\![y]\!]_i$. For the multiplication of two shared value $[x]$ and $[y]$ we use Beaver triples $([\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i)$ where $c = ab$ and $a, b \in \mathbb{Z}_{2^k}$ are uniformly random. Each party P_i first opens the masked inputs $[x]_i - [a]_i, [y]_i - [b]_i$ and reconstructs $x-a, y-b$. A share of the product is then computed as $[\![x \cdot y]\!]_i = (x-a) \cdot [\![y]\!]_i + [\![a]\!]_i \cdot (y-b) + [\![c]\!]_i$. The Beaver triples are generated in the offline phase.

2.2 Cyclotomic Ring of Integers

For $m \in \mathbb{N}$, Φ_m denotes the m-th cyclotomic polynomial. In this work, we require m to be a prime number, in which case $\Phi_m(X) = (X^m - 1)/(X - 1) = \sum_{i=0}^{m-1} X^i$. We define $\mathcal{R} := \mathbb{Z}[X]/\Phi_m(X)$ and $\mathcal{R}_n := \mathcal{R}/n\mathcal{R}$ for a natural number n.

Power Basis. $B^{\text{power}} := (X^i)_{i \in \mathbb{Z}_m^*} = (X^1, \dots, X^{m-1})$ is a $\mathbb{Z}$-basis of $\mathcal{R}$. We call it the *power basis.*[4] Unless otherwise stated, the term *coefficients* refers to the coefficients in the power basis. For $B \in \mathbb{N}$ we denote by $\mathcal{R}_{\leq B}$ the set of polynomials from $\mathcal{R}$ where all coefficients are within $[-B, B)$. We can pack $m-1$ elements $a_1, \dots, a_{m-1}$ of $\mathbb{Z}$ into the coefficients of a polynomial $f \in \mathcal{R}$ in the power basis, i.e., $f(X) = \sum_{i=1}^{m-1} a_i X^i$. We call this trivial packing the *coefficient packing* of $a_1, \dots, a_{m-1}$.

Statistically Close Distributions. For $\delta \in \mathbb{R}_{\geq 0}$ and two distributions $\mathcal{X}, \mathcal{Y}$ that output an element or a tuple of elements from $\mathcal{R}$, we denote by $\mathcal{X} \approx_\delta \mathcal{Y}$ that each (power basis) coefficient of the output of $\mathcal{X}$ is distributed within statistical distance $\leq \delta$ of the corresponding coefficient of the output of $\mathcal{Y}$.

[4] Note that this is not the traditional power basis as used in [LPR13]: instead of the constant X^0, we include the element $X^{m-1} \equiv -\sum_{i=0}^{m-2} X^i$. This makes it easier to change between power basis and CRT basis via Rader's FFT algorithm, as the constant coefficient is always 0.

CRT Basis. Let q be a prime number and let d be the order of q in $\mathbb{Z}_m^*$. Let $r := \varphi(m)/d, \varphi(m) = m-1$ and ζ be a primitive m-th root of unity. Then $\Phi_m \bmod q$ factors into r distinct irreducible monic (and hence pairwise coprime) polynomials $f_i \in \mathbb{F}_q[X]$, i.e., $\Phi_m(X) \equiv \prod_{i=0}^{r-1} f_i(X) \pmod{q}$. Each of those factors has degree d. By the Chinese Remainder Theorem (CRT) it follows that $\mathcal{R}_q \cong \bigtimes_{i=0}^{r-1} \mathbb{F}_q[X]/(f_i)$. We call the preimage $B^{\mathrm{CRT}} := (b_i^{\mathrm{CRT}})_{i\in[r]} \in \mathcal{R}_q^r$ of the standard basis of $\bigtimes_{i=0}^{r-1} \mathbb{F}_q[X]/(f_i) \simeq (\mathbb{F}_{q^d})^r$ in $\mathcal{R}_q$ the *CRT basis* of $\mathcal{R}_q$, i.e., the preimage of $(1, 0, \dots, 0), \dots, (0, \dots, 0, 1) \in (\mathbb{F}_{q^d})^r$. We call the coefficients $\mathrm{CRT}(a) := (a_i)_{i\in[r]} \in \mathbb{Z}_q[X]/(f_i) \simeq \mathbb{F}_{q^d}$ of the an element $a \in R_q$ in the CRT basis the CRT representation of a, i.e., $a = \langle B^{\mathrm{CRT}}, (a_i)_{i\in[r]}\rangle$.

Furthermore note that while $\mathsf{Gal}(\mathbb{F}_q(\zeta)/\mathbb{F}_q) = \mathsf{Aut}_{\mathbb{F}_q}(\mathbb{F}_q(\zeta)) \simeq \langle q\rangle \subset \mathbb{Z}_m^*$ preserves the irreducible factors $\mathbb{F}_q[X]/(f_i)$, the Galois group $\mathsf{Gal}(\mathbb{Q}(\zeta)/\mathbb{Q}) \simeq \mathbb{Z}_m^*$ acts by $\kappa_i : \mathcal{R} \to \mathcal{R}, f(X) \mapsto f(X^i)$ transitively on the set of components $\{\mathbb{Z}[X]/(f_i) : 0 \le i < r\}$. The quotient $\mathbb{Z}_m^*/\langle q\rangle$ then acts freely and transitively on the respective set over $\mathbb{F}_q$, i.e., we find for each two components exactly one element $h \in \mathbb{Z}_m^*/\langle q\rangle$ such that κ_h maps one component to the other. Since m is prime, $\mathbb{Z}_m^*$ is cyclic and we find a generator $g \in \mathbb{Z}_m^*$, i.e., $h = g^k$ for some k. This allows us to fix an order of the irreducible factors $\mathbb{F}_q[X]/(f_i)$ such that $f_{i+k \bmod r}(X) = \gcd(f_i(X^{g^k}), \Phi_m(X))$ (cf. [GHS12a]). We will use this order in the rest of the paper.

Finally, we use Hensel's lemma to lift the construction to prime powers q^T, i.e., we find monic irreducible polynomials $\tilde{f}_i \in \mathbb{Z}_{q^T}[X]$ such that $\tilde{f}_i \bmod q = f_i$, $\deg(f_i) = \deg(\tilde{f}_i)$, $\Phi_m \bmod q^T = \prod_{i=0}^{r-1} \tilde{f}_i$, and $\mathbb{Z}_{q^T}[X]/(\Phi_m) \simeq \bigtimes_{i=0}^{r-1} \mathbb{Z}_{q^T}[X]/(\tilde{f}_i)$.

2.3 Tweaked Interpolation Packing

Let $t := k + 2s$. We want to use the previously described spaces $\mathcal{R}_{q^T}$ in the special case $q = 2$ and for some suitable natural number T to compute multiplications of elements from $\mathbb{Z}_{2^t}$ in a SIMD way. A direct approach to do this is to choose $T = t$ and to encode numbers from $\mathbb{Z}_{2^t}$ in the constant coefficient of $\mathbb{Z}_{2^T}[X]/(\tilde{f}_i)$. Since $\mathcal{R}_{2^T}$ contains r components $\mathbb{Z}_{2^T}[X]/(\tilde{f}_i)$ we can encode an r-tuple of numbers from $\mathbb{Z}_{2^t}$ in one element of $\mathcal{R}_{2^T}$. Now multiplication in $\mathcal{R}_{2^T}$ results in a component-wise multiplication of the r-tuples, i.e., we can simultaneously perform r multiplications of elements in $\mathbb{Z}_{2^t}$. However, this direct approach only uses $1/d$ of the available coefficients, i.e., only the constant terms of a degree d polynomial in $\mathbb{Z}_{2^T}[X]/(\tilde{f}_i)$. To better use the full potential of $\mathcal{R}_{2^T}$ we have to use more evolved *packing methods* to pack more elements from $\mathbb{Z}_{2^t}$ into slots of $\mathcal{R}_{2^T}$ such that multiplication in $\mathcal{R}_{2^T}$ still corresponds to component-wise multiplication of (then larger) tuples from $\mathbb{Z}_{2^t}$. A first non-trivial (but still simple) packing method was introduced in Overdrive2k [OSV20] which allows to use approximately $d^{0.6}$ of the d polynomial coefficients (in each slot). Overdrive2k [OSV20] encodes several values from $\mathbb{Z}_{2^t}$ in a sparse polynomial in a way that (some) coefficients of the polynomial product are simple multiplications of coefficients of the factors (rather than sums over coefficients), e.g., $(aX + a')(bX + b') = abX^2 + (ab' + a'b)X + a'b'$ contains the products ab and $a'b'$ as 2- and 0-coefficient.

MHz2k [CKL21] proposes a far more evolved and far more efficient packing method that uses $1/2$ of the available slots. The idea of this so-called *tweaked interpolation packing* is to pack values from $\mathbb{Z}_{2^t}$ into evaluation points of the polynomials in different slots. More concretely, [CKL21] introduces maps

$$\mathsf{pack} : \mathbb{Z}_{2^t}^M \to \mathcal{R}_{2^T}, \quad \mathsf{unpack} : \mathcal{R}_{2^T} \to \mathbb{Z}_{2^t}^M$$

for $T = t + 2\delta$, $\delta > 0$ suitably large, $D := \lfloor (d+1)/2 \rfloor$, and $M := D \cdot r$ such that for $\boldsymbol{x} = (x_{(i,j)})_{(i,j)\in[r]\times[D]}$ we have

$$\mathrm{CRT}(\mathsf{pack}(\boldsymbol{x}))_i(j) = 2^\delta \cdot x_{(i,j)} \bmod 2^{t+\delta}, \tag{1}$$

i.e., pack maps M numbers $x_{(i,j)} \in \mathbb{Z}_{2^t}$ to a polynomial $\mathsf{pack}(\boldsymbol{x}) \in \mathcal{R}_{2^T}$ whose i-th component polynomial $\mathrm{CRT}(\mathsf{pack}(\boldsymbol{x}))_i = \mathsf{pack}(\boldsymbol{x}) \pmod{\tilde{f}_i}$ evaluates at the point j to $2^\delta \cdot x_{(i,j)} \bmod 2^{t+\delta}$. [CKL21] shows that for a suitably large δ such a packing exists. Furthermore, the choice $D := \lfloor (d+1)/2 \rfloor$ guarantees that the polynomial degree of the product of packed values in each component $\mathbb{Z}_{2^T}[X]/(\tilde{f}_i)$ remains smaller than d and does not wrap around. The evaluation values $2^\delta \cdot x_{(i,j)} \bmod 2^{t+\delta}$ are chosen such that for the product of two values $\mu_{ij} = 2^\delta \cdot x_{(i,j)} \bmod 2^{t+\delta}$ and $\nu_{ij} = 2^\delta \cdot y_{(i,j)} \bmod 2^{t+\delta}$ one has $\mu_{ij}\nu_{ij} \equiv_T z_{(i,j)} 2^{2\delta}$

for $z_{(i,j)} = x_{(i,j)}y_{(i,j)} \bmod 2^t$, i.e., the product of our values in $\mathbb{Z}_{2^t}$ can be reconstructed from (the values of the) polynomial product. In particular, to unpack a product one simply evaluates at each $j \in [D]$ and then removes the factor $2^{2\delta}$ from the value. As in [CKL21], in order to later use a more efficient ZKPoMK with the BGV encryption scheme, we work with $T = t + 2\delta + E$ (instead of $T = t + 2\delta$) for some small $E \in \mathbb{N}$. The unpack operation then has to first remove these E additional bits and then unpack products as described before, i.e., by evaluation and removing the $2^{2\delta}$ factor. Overall, pack and unpack have the following homomorphic property:

$$\mathsf{unpack}(\mathsf{pack}(\boldsymbol{a}) \cdot \mathsf{pack}(\boldsymbol{b})) = \boldsymbol{a} \odot \boldsymbol{b}.$$

where $\odot$ denotes the component-wise multiplication in $\mathbb{Z}_{2^t}^M$. For proofs and further details we refer to [CKL21].

In our protocol Π_{Triple} in Figure 4 we will have to mask a product of packed values $\mathsf{pack}(\boldsymbol{a}) \cdot \mathsf{pack}(\boldsymbol{b})$. To find a suitable mask, observe that each such product is in the group $\mathcal{G}$ of $\mathbb{Z}_{2^T}$-polynomials of degree smaller than d whose values on $[D]$ are in $2^{2\delta}\mathbb{Z}_{2^t}$. Our unpacking procedure, i.e., evaluate on $[D]$ and multiply by $2^{-2\delta}$, is then a surjective group homomorphism $\mathcal{G} \to \mathbb{Z}_{2^t}^M$.[5] We denote its fiber, i.e., the preimage over $\boldsymbol{e} \in \mathbb{Z}_{2^t}^M$ by $\mathsf{pack}'(\boldsymbol{e})$. Now the masks are sampled uniformly from $\mathcal{G}$ and added to a product $\mathsf{pack}(\boldsymbol{a}) \cdot \mathsf{pack}(\boldsymbol{b})$. Since we are in the group $\mathcal{G}$ this is a one-time-pad encryption and therefore (information-theoretically) secure. If we unpack the masked product we get $\boldsymbol{a} \odot \boldsymbol{b} + \boldsymbol{e}$ if the mask was taken from $\mathsf{pack}'(\boldsymbol{e})$. Note that the uniform distribution on $\mathcal{G}$ induces the uniform distribution on $\{(\boldsymbol{e}, x) \mid \boldsymbol{e} \in \mathbb{Z}_{2^t}^M, x \in \mathsf{pack}'(\boldsymbol{e})\}$ and we can therefore also sample $\boldsymbol{e} \in \mathbb{Z}_{2^t}^M$ uniformly at random first and then sample an element of the fiber $\mathsf{pack}'(\boldsymbol{e})$. In slight abuse of notation we will denote the uniform sample from the fiber also by $\mathsf{pack}'(\boldsymbol{e})$.

3 BGV over $\mathbb{Z}_{2^k}$

In this section we describe the LHE scheme BGV used in our secure offline protocol. We explain the homomorphic properties of BGV and how they can be used in a secure multiplication protocol. Furthermore, we add a short description of a secure key generation usually not discussed in the literature.

Our instantiation of the BGV encryption scheme [BGV12] for a plaintext space $\mathcal{R}_{2^k}$ is the same as in [OSV20] and is based on [GHS12b]. We similarly use the NewHope [ADPS16] approximation of the discrete Gaussian distribution. That is, we replace the discrete Gaussian by the centered binomial distribution $\mathcal{CB}$ of the same variance, which is easier to sample in software.

We describe the scheme for plaintext space $\mathcal{R}_{2^T}$. In our protocol (Section 5) we instantiate the scheme with two different values of T, depending on the packing used. Our BGV parameters for concrete instantiations of our protocol can be found in Section 7.2.

Following [OSV20], in order to enable modulus switching, we use two moduli $q_0 := p_0$ and $q_1 := p_0 p_1$ where p_0 and p_1 are primes with $p_1 \equiv 1 \pmod{2^T}$ and $p_0 \equiv p_1 \equiv 1 \pmod{m}$.

Distributions. Apart from the uniform distribution over $\mathcal{R}_{q_1}$, the BGV scheme also requires to sample the following distributions.

- $\mathcal{HWT}(h)$: This samples a random polynomial from $\mathcal{R}$ with h non-zero coefficients, i.e., with Hamming weight h, (in power basis) where all coefficients are from $\{-1, 0, 1\}$ (as in [DKL+13, KPR18, OSV20]).
- $\mathcal{CB}(\sigma^2)$: This samples a random polynomial from $\mathcal{R}$ where all coefficients (in power basis) are sampled from the centered binomial distribution with variance σ^2.

Typically, $\sigma = 3.2$ and $h = 64 + \mathsf{Stat\_sec}$ are used [DPSZ12, DKL+13, KPR18]. We use $\sigma^2 = 10$ as [OSV20, CKL21].

[5] Surjective since $\mathsf{pack}((1, \ldots, 1)) \cdot \mathsf{pack}(\boldsymbol{a})$ maps back to $\boldsymbol{a}$ for each $\boldsymbol{a} \in \mathbb{Z}_{2^t}^M$; linear since the value at some $j \in [D]$ of the sum of 2 polynomials is the sum of their values at j.

Key Generation, Encryption, and Decryption.

– $\mathsf{KeyGen}()$: Sample $\mathsf{sk} \leftarrow \mathcal{HWT}(h)$, $a \xleftarrow{\$} \mathcal{R}_{q_1}$, $e \leftarrow \mathcal{CB}(\sigma^2)$ and set $b := a \cdot \mathsf{sk} + 2^T e$. Return the secret key sk and public key $\mathsf{pk} := (a, b) \in \mathcal{R}_{q_1}^2$.
– $\mathsf{Enc}_{\mathsf{pk}}(m \in \mathcal{R}_{2^T})$: To encrypt m, sample small polynomials $v \leftarrow \mathcal{CB}(0.5)$ and $e_0, e_1 \leftarrow \mathcal{CB}(\sigma^2)$. Return the ciphertext $\boldsymbol{c} := (c_0, c_1) := (bv + 2^T e_0 + m, av + 2^T e_1) \in \mathcal{R}_{q_1}^2$.
– $\mathsf{SwitchMod}(\boldsymbol{c} \in \mathcal{R}_{q_1}^2)$: Compute $\boldsymbol{d} \in \mathcal{R}^2$ with $\boldsymbol{d} \equiv \boldsymbol{c} \pmod{p_1}$, $\boldsymbol{d} \equiv 0 \pmod{2^T}$, $\|\boldsymbol{d}\|_\infty \leq 2^{T-1} p_1$. Return $\boldsymbol{c}' := (\boldsymbol{c} - \boldsymbol{d})/p_1 \in \mathcal{R}_{q_0}^2$.
– $\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c} \in \mathcal{R}_{q_i}^2)$ for $i \in \{0, 1\}$: To decrypt $\boldsymbol{c} = (c_0, c_1)$, compute $m' := c_0 - \mathsf{sk} \cdot c_1$ and return the plaintext $m := (m' \text{ cmod } q_i) \bmod 2^T \in \mathcal{R}_{2^T}$. Here cmod denotes the centered modular reduction that produces a polynomial with coefficients in $(-q_i/2, q_i/2]$.

Homomorphic Operations and Noise. Let $\boldsymbol{c} = (c_0, c_1)$ and $\boldsymbol{c}' = (c_0', c_1')$ be ciphertexts that encode m and m', respectively. We can add two ciphertexts $\boldsymbol{c} + \boldsymbol{c}' := (c_0 + c_0', c_1 + c_1')$, obtaining a ciphertext that encodes $m + m'$. Subtraction works analogously. Multiplication in our linearly homomorphic variant of BGV is only supported by a plaintext polynomial. For a publicly known $f \in \mathcal{R}_{2^T}$, we can multiply $f \cdot \boldsymbol{c} := (fc_0, fc_1)$, obtaining a ciphertext that encodes fm.

Note that correct decryption of the ciphertexts obtained through these homomorphic operations is only guaranteed as long as the noise of the ciphertext is not too large. The noise of a ciphertext $\boldsymbol{c} = (c_0, c_1)$ is defined as $\mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}) := c_0 - \mathsf{sk} \cdot c_1$.

Modulus Switching. The *modulus switching* protocol $\mathsf{SwitchMod}$ described above, allows a party to convert a ciphertext $\boldsymbol{c}$ under a modulus q_1 into a ciphertext $\boldsymbol{c}'$ under a smaller modulus q_0. $\mathsf{SwitchMod}$ follows the classical constructions from [BGV12] and [GHS12b, Appendix D]. When employing modulus switching, we have to be careful that the target modulus is still large enough to guarantee correct decryption. This is captured by the following lemma, which can be proven analogously to [BGV12, Lemma 1].

Lemma 1. *Let $q_0 = p_0$ and $q_1 = p_0 p_1$ where p_0 and p_1 are prime numbers with $p_1 \equiv 1 \pmod{2^T}$ and $p_0 \equiv p_1 \equiv 1 \pmod{m}$. Let $\boldsymbol{c} \in \mathcal{R}_{q_1}^2$ and $\boldsymbol{c}' := \mathsf{SwitchMod}(\boldsymbol{c})$. Then, for any sk with $\|\mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c})\|_\infty < q_1/2 - 2^{T-1} \cdot p_1 \cdot \ell_1(\mathsf{sk})$, we have $\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}) = \mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}')$.*

3.1 Drowning

As in the original LowGear protocol [KPR18], we likewise need to apply so-called drowning to certain ciphertexts to make sure that the ciphertexts' noise does not leak sensitive information (to someone who knows the secret key). That is, the party who knows the secret key should be able to learn the plaintext value (by decrypting) but should not be able to infer, e.g., how this value was computed. Drowning is performed using a modified encryption algorithm Enc'. In contrast to [KPR18], we explicitly define Enc' and formalize (in Theorem 1) the security guarantees we obtain from it. Our definition corresponds to what is currently used in practice [Kel20]. Enc' is defined as follows.

– $\mathsf{Enc}'_{\mathsf{pk}}(m \in \mathcal{R}_{2^T}, B \in \mathbb{N})$: To encrypt and drown m with noise bound B, proceed as in $\mathsf{Enc}_{\mathsf{pk}}(m)$, except that e_0 is sampled uniformly at random from the set of polynomials with coefficients in $[-B, B)$.

Intuitively, the security guarantee from drowning is that any two drowned ciphertexts are indistinguishable from each other, even for the party who knows the secret key, as long as both encrypt the same plaintext m. This is formalized in the following theorem.

Theorem 1. *Let $S \in \mathbb{R}_{\geq 1}$, let $\mathsf{sk}, e \in \mathcal{R}$ with $\|\mathsf{sk}\|_\infty \leq S$ and $\|e\|_\infty \leq 2\sigma^2 S$, and let $\boldsymbol{c}_0, \boldsymbol{c}_1 \in \mathcal{R}_{q_0}^2$ be ciphertexts with $\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}_0) = \mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}_1)$. For any $a \in \mathcal{R}_{q_1}$, define $\mathsf{pk}(a) := (a,\ a \cdot \mathsf{sk} + 2^T e)$. For statistical security parameter $\mathsf{Stat\_sec}$ and*

$$B \geq 2^{\mathsf{Stat\_sec}}(\|\lfloor \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}_i)/2^T \rfloor\|_\infty + \varphi(m) 4\sigma^2 S) \quad \forall i \in \{0, 1\},$$

the two distributions

$$\left\{ \left(a \xleftarrow{\$} \mathcal{R}_{q_1}, \boldsymbol{c}_i + \mathit{Enc}'_{\mathit{pk}(a)}(0, B) \right) \right\}, \quad i \in \{0,1\},$$

are computationally indistinguishable.

Proof. Let $\boldsymbol{c} := \boldsymbol{c}_i$ for any $i \in \{0,1\}$. We show that the distribution for i is computationally indistinguishable from

$$\left\{ \left(a \xleftarrow{\$} \mathcal{R}_{q_1}, \left(\mathsf{sk} \cdot u + 2^T e_0 + m, u \right) \right) : e_0 \xleftarrow{\$} \mathcal{R}_{[-B,B)}, u \xleftarrow{\$} \mathcal{R}_{q_1} \right\}.$$

Then the theorem follows by transitivity.

Let $\boldsymbol{x} = (x_0, x_1) := \boldsymbol{c} + \mathsf{Enc}'_{\mathsf{pk}(a)}(0, B)$. By definition and linearity of noise, we have

$$\begin{aligned} x_0 &= \mathsf{sk} \cdot x_1 + \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{x}) \\ &= \mathsf{sk} \cdot x_1 + \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}) + \mathsf{noise}_{\mathsf{sk}}\left(\mathsf{Enc}'_{\mathsf{pk}(a)}(0, B) \right) \\ &= \mathsf{sk} \cdot x_1 + \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}) + 2^T (ev + e_0 - \mathsf{sk} \cdot e_1) \\ &= \mathsf{sk} \cdot x_1 + m + 2^T (e_0 + \tilde{e}) \end{aligned}$$

where v, e_0, e_1 are sampled as in the definition of Enc' and $\tilde{e} := \lfloor \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c})/2^T \rfloor + ev - \mathsf{sk} \cdot e_1$. In Theorem 1, we chose B specifically such that $B \geq 2^{\mathsf{Stat\_sec}} \left\| \tilde{e} \right\|_\infty$. As e_0 is sampled uniformly at random from $\mathcal{R}_{[-B,B)}$, it follows that $e_0 \approx_\delta e_0 + \tilde{e}$ for $\delta := 2^{-\mathsf{Stat\_sec}}$. Hence

$$(a, \boldsymbol{x}) \approx_\delta \left(a, \ \left(\mathsf{sk} \cdot x_1 + 2^T e_0 + m, \ x_1 \right) \right). \tag{2}$$

By definition of Enc', we have $x_1 = c_1 + av + 2^T e_1$. By a standard hybrid argument using the $\mathcal{R}$-LWE assumption, it follows that we can replace x_1 by a uniformly random element $u \xleftarrow{\$} \mathcal{R}_{q_1}$. That is, the right-hand side of Equation (2) is computationally indistinguishable from $(a, \ (\mathsf{sk} \cdot u + 2^T e_0 + m, \ u))$. □

3.2 Secure Key Generation

In contrast to prior works [DPSZ12, KPR18, OSV20, CKL21] which assume that HE keys are generated honestly by a trusted dealer, we construct a secure key generation protocol for our triple generation. In Section 7, we present how this affects our parameters and evaluation.

In order to use Theorem 1, our secure key generation has to ensure that the assumptions of the theorem still hold, namely that $\mathsf{pk} = (a, \ a \cdot \mathsf{sk} + 2^T e)$ for $a \xleftarrow{\$} \mathcal{R}_{q_1}$ and some $\mathsf{sk}, e \in \mathcal{R}$ with $\left\| \mathsf{sk} \right\|_\infty \leq S$ and $\left\| e \right\|_\infty \leq 2\sigma^2 S$. Here the *slack* S introduces a trade-off: increasing S relaxes the requirements on the key but at the same time increases the noise of the drowned ciphertext.

Now in order to prevent cheating, every party P_i generates her public key using an interactive protocol that proves to the other parties that the public key conforms to the above-mentioned requirements. This protocol is defined as follows. First, the component a of the public key is sampled using a secure coin-flipping protocol over $\mathcal{R}_{q_1}$. Appendix G shows how this (standard) protocol can be realized. Then P_i computes and broadcasts the second component $b = a \cdot \mathsf{sk} + 2^T e$. Finally, P_i engages with each other party P_j ($j \neq i$) in a zero-knowledge proof of secret key knowledge (ZKPoSKK) where P_i proves to know a witness $(\mathsf{sk}, e) \in \mathcal{R}^2$ subject to above-mentioned bounds such that $b = a \cdot \mathsf{sk} + 2^T e$.

A description of our ZKPoSKK is included in Section 6.2. We designed our ZKPoSKK with a small slack on (sk, e). Please note that more efficient ZKPoSKKs are available (e.g., similar to our construction of ZKPoPK in Section 6) but generally result in a larger slack on (sk, e). This larger slack on the other hand impacts the noise bounds of *every* ciphertext. The negative effect of a larger slack on the overall efficiency is therefore much larger than the a slightly less efficient ZKPoSKK used only once in the key generation.

4 2^s-Enhanced CPA Security

Similar to previous LHE-based protocols [KPR18, RRKK23], our triple generation protocol uses a MAC check to detect misbehavior. However, the outcome of the MAC check (i.e., pass or fail) might leak information on certain homomorphically encrypted values.

Previous LHE protocols [KPR18, RRKK23] (in the base field setting) protect against this type of *selective failure* leakage by using encryption schemes, i.e., BGV in the field case, that satisfy an *enhanced CPA* security [KPR18]. Unfortunately, enhanced CPA security does no longer hold over $\mathbb{Z}_{2^T}$ for our BGV-type scheme. Even more, [CRFG20] showed that an adversary in the enhanced CPA security game can always deduce the last bit of a plaintext with non-negligible advantage for *any* homomorphic encryption scheme with plaintext space $\mathbb{Z}_{2^T}$. We extend this result below (see Remark 1) and show that an adversary can deduce the s least significant bits of a plaintext with success probability $1/2^s$.

Fortunately, under the same (classical LHE) assumptions as in [KPR18], i.e., that an adversary can only successfully apply affine operations to ciphertext(s), this is the best an adversary can do. Namely, with a high probability of at least $1-2^{-s}$ an adversary gains no information on the $T-s$ most significant bits. Formally, we capture this property of BGV ciphertexts with a new security notation called 2^s-*enhanced CPA* security:

Definition 1. *Let $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure asymmetric encryption scheme with a packing scheme $(\mathsf{pack} : \mathbb{Z}_{2^t}^M \to \mathcal{R}_{2^T}, \mathsf{unpack} : \mathcal{R}_{2^T} \to \mathbb{Z}_{2^t}^M)$. $\mathcal{S}$ has* 2^s-enhanced CPA security *if for each probabilistic and polynomial time (ppt.) adversary $\mathcal{A}$ there is an event E such that*

(i) if $\mathsf{Pr}[E] \neq 0$ then the game $\mathcal{G}_{2^s\text{-}\mathsf{CPA+}}$ (Figure 1) aborts with probability at least $1-2^{-s}$ over all runs in E, i.e., $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1-2^{-s}$,

(ii) if $\mathsf{Pr}[E] \neq 1$ then the advantage over all runs in $\neg E$ is negligible in the computational security parameter $\mathsf{Comp\_sec}$ of $\mathcal{S}$, i.e., $|\mathsf{Pr}[b'=1 \mid b=1, \neg E] - \mathsf{Pr}[b'=1 \mid b=0, \neg E]|$ is negligible in $\mathsf{Comp\_sec}$.

$\mathcal{G}_{2^s\text{-}\mathsf{CPA+}}$

1. The challenger $\mathcal{C}$ generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$ and sends pk to the adversary $\mathcal{A}$.
2. $\mathcal{C}$ samples $\boldsymbol{m} \xleftarrow{\$} \mathbb{Z}_{2^t}^M$ and sends the ciphertext $\boldsymbol{c} := \mathsf{Enc}_{\mathsf{pk}}(\mathsf{pack}(\boldsymbol{m}))$ to $\mathcal{A}$. $\mathcal{C}$ samples $b \xleftarrow{\$} \{0,1\}$.
3. For $j \in \mathsf{poly}(s)$:
   (a) $\mathcal{A}$ sends some $\boldsymbol{c}_j$ to $\mathcal{C}$.
   (b) If $\mathsf{unpack}(\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}_j)) = \boldsymbol{0}$, $\mathcal{C}$ sends OK to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sends abort to $\mathcal{A}$ and aborts and sets $b' = 0$.
4. If $b = 0$, $\mathcal{C}$ sends $\boldsymbol{m}_0 := \boldsymbol{m}$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ samples $\tilde{\boldsymbol{m}} \xleftarrow{\$} \mathbb{Z}_{2^{t-s}}^M$ and sends $\boldsymbol{m}_1 := 2^s\tilde{\boldsymbol{m}} + (\boldsymbol{m} \bmod 2^s)$ to $\mathcal{A}$.
5. $\mathcal{A}$ outputs $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ is defined as $\mathsf{adv}_{\mathcal{A}} = |\mathsf{Pr}[b'=1 \mid b=1] - \mathsf{Pr}[b'=1 \mid b=0]|$.

Fig. 1: 2^s-enhanced CPA security game.

Intuitively, E is the event where the adversary can gain meaningful information on the $T-s$ most significant bits from the oracle queries, i.e., $|\mathsf{Pr}[b'=1 \mid b=1, E] - \mathsf{Pr}[b'=1 \mid b=0, E]|$ is non-negligible in $\mathsf{Comp\_sec}$. In this case, we demand that the challenger aborts with high probability $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1-2^{-s}$. In the other case ($\neg E$), we demand that the adversary has negligible advantage.

We remark that part (i) is a statistical security notion which is independent of the computational power of an adversary. In particular, an adversary cannot retry to improve his chances. We therefore choose s as in the SPD$\mathbb{Z}_{2^k}$ MAC scheme (see Section 2.1), such that we obtain the same statistical security.[6] A selective failure attack then has the same success probability ($\leq 2^{-s}$) as cheating by guessing the correct MAC key (cf. [CDE+18]).

[6] That is, $\mathsf{sec} := \lfloor s - \log_2(s+1) \rfloor$ (see [CDE+18, Theorem 1]).

2^s-enhanced CPA-security of BGV. We now show that the BGV scheme satisfies Definition 1. For the proof, we require the established assumption that BGV satisfies linear targeted malleability (cf. Definition 3 or [BCI+13]). The argument in the $\mathbb{Z}_{2^T}$ case is identical to the field case discussed in [KPR18]. This still holds when combining BGV with the tweaked interpolation packing where encryption becomes $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{pack}(\cdot))$ and decryption becomes $\mathsf{unpack}(\mathsf{Dec}_{\mathsf{sk}}(\cdot))$. We refer to Appendix B for further details.

Theorem 2. *If the BGV scheme over $\mathbb{Z}_{2^T}$ (cf. Section 3) satisfies linear targeted malleability, then it also satisfies 2^s-enhanced CPA security (cf. Definition 1).*

Proof. If the adversary's advantage (over all runs) is negligible in $\mathsf{Comp\_sec}$, then we choose $E = \emptyset$ in order to get 2^s-enhanced CPA security. Hence, we focus on adversaries $\mathcal{A}$ with non-negligible advantage (over all runs) and need to find an appropriate $E \neq \emptyset$ to satisfy Definition 1. Obviously, such an adversary needs to make queries in $\mathcal{G}_{2^s\text{-}\mathsf{CPA}+}$ (3) or we can directly reduce to the standard CPA game.

With linear targeted malleability, $\mathcal{G}_{2^s-\mathsf{CPA}+}$ can be reduced to a security game where the adversary sends $B \in \mathbb{Z}_{2^T}^{K\times M}$ and $\boldsymbol{b} \in \mathbb{Z}_{2^T}^{K}$ to the challenger $\mathcal{C}$ and $\mathcal{C}$ returns OK if the corresponding affine function $f_{B,\boldsymbol{b}}(\boldsymbol{m}) = B\boldsymbol{m} - \boldsymbol{b} = \boldsymbol{0}$ or else aborts (cf. Figure 15). Linear algebra tells us that B has a Smith normal form [Bro92], i.e., there are invertible matrices $S \in \mathbb{Z}_{2^T}^{K\times K}, R \in \mathbb{Z}_{2^T}^{M\times M}$ and an $0 \le r \le \min\{K, M\}$ such that $S^{-1}BR^{-1} = D = (\delta_{ij}2^{\beta_j}\delta_{j<r})_{0\le i<K, 0\le j<M}$ with $0 \le \beta_1 \le \cdots \le \beta_r < T$ and $\beta_i \in \mathbb{N}$. Thus $B\boldsymbol{m} - \boldsymbol{b} = 0$ is equivalent to $D(R\boldsymbol{m}) = S^{-1}\boldsymbol{b}$. We need one further property: if $R\boldsymbol{m} = \boldsymbol{n}2^j + \boldsymbol{k}$ for some $1 \le j \le s$ and $\boldsymbol{n}, \boldsymbol{k} \in \mathbb{Z}_{2^T}^{M}$, then $\boldsymbol{m} \bmod 2^j = R^{-1}\boldsymbol{k} \bmod 2^j$, i.e., the least j significant bits of $\boldsymbol{k}$ and $\boldsymbol{m}$ determine each other completely. On the other hand if $\boldsymbol{n}$ is uniformly random so are the $T - j$ most significant bits of $\boldsymbol{m}$ since $R^{-1}\boldsymbol{n}2^j$ has uniformly random $T - j$ most significant bits. In summary, if an adversary gets access to the j least significant bits of $R\boldsymbol{m}$ he can deduce the least j significant bits of $\boldsymbol{m}$. However, as long as he learns nothing about the more significant bits of $R\boldsymbol{m}$, he also cannot deduce information about the more significant bits of $\boldsymbol{m}$.

We want to now look at the information an adversary can actually deduce from a successful query. Consider 1-dimensional affine functions $f_{\beta,b}(m) = 2^\beta m - b$ for $\beta \in \mathbb{N}, b \in \mathbb{Z}_{2^T}$ and $m \in \mathbb{Z}_{2^T}$ uniformly random, i.e., $\beta = \beta_j, m = (R\boldsymbol{m})_j, b = (S^{-1}\boldsymbol{b})_j$ for some $1 \le j \le r$. Then $f_{\beta,b}(m) = 0$ only happens if $2^\beta m = b$. If $m = \sum_{i=0}^{T-1} m_i 2^i$ and $b = \sum_{i=0}^{T-1} b_i 2^i$, this condition amounts to $b_i = 0$ for $i < \beta$ and $b_i = m_{i-\beta}$ for $T > i \ge \beta$. If m is sampled randomly this case occurs with probability $2^{\beta-T}$.[7] If this case occurs, the adversary learns the $T - \beta$ least significant bits of m but *nothing* about the more significant bits, since they do not affect the outcome of the zero check.

Hence, in order to gain information on the $T - s$ most significant bits (to improve the winning changes in $\mathcal{G}_{2^s\text{-}\mathsf{CPA}+}$) the adversary needs to choose $T - \beta > s \Leftrightarrow \beta < T - s$. But then the game aborts with probability $1 - 2^{\beta-T} > 1 - 2^{-s}$. Please note that sending several queries does not improve the winning chance. Namely, if the adversary uses the decryption oracle again with $f_{2^{\beta'},b'}$. Obviously, he chooses $\beta' < \beta$ or else he can gain no new information. He should also set $b'_j = m_{j-\beta'}$ for $0 \le j - \beta' < T - \beta$ or else he always fails (since these $m_{j-\beta'}$ are already determined by the first round). If he does so, the second round does not abort with probability $2^{\beta'-\beta}$. He then knows the $t - \beta'$ least significant bits. If $T - \beta' > s$ then the overall abort probability is $1 - 2^{\beta-T}2^{\beta'-\beta} = 1 - 2^{\beta'-T} > 1 - 2^s$.

In summary, we see that we can choose E as the event where at least one query of $\mathcal{A}$ is of the form $f_{B,\boldsymbol{b}}$ with $B = SDR$, $D = (\delta_{ij}2^{\beta_j}\delta_{j<r})_{0\le i<K,0\le j<M}$ as before and $\beta_1 < T - s$. Then, for runs in E, we get an abort probability $\mathsf{Pr}[\mathsf{abort} \mid E] \ge 1 - 2^{-s}$; for runs in $\neg E$, we have $\beta_j \ge \beta_1 \ge T - s$ for all $1 \le j \le r$, so the adversary cannot deduce any information on the $T - s$ most significant bits from querying and it follows that $|\mathsf{Pr}[b' = 1 \mid b = 1, \neg E] - \mathsf{Pr}[b' = 1 \mid b = 0, \neg E]|$ is negligible in $\mathsf{Comp\_sec}$. These two statements together mean that BGV satisfies 2^s-enhanced CPA-security (for every choice $0 \le s < T$). ☐

Remark 1. The previous proof contains an attack against (classical) enhanced CPA-security similar to the one in [CRFG20]. An adversary that queries $2^{T-\beta}\mathsf{Enc}_{\mathsf{pk}}(m)$ gains the β least significant bits with probability (at least) $2^{-\beta}, 0 < \beta \le T$. As in [CRFG20] our attack extends to every LHE scheme with plaintext space $\mathbb{Z}_{2^T}$.

[7] $(R\boldsymbol{m})_j$ is obviously distributed uniformly at random, if $\boldsymbol{m}$ was sampled randomly.

Next we want to explain how 2^s-enhanced CPA-security can be used to construct a secure triple generation. We obviously cannot use the last s bits of an encrypted value since these bits are not protected under 2^s-enhanced CPA-security. In our triple generation protocol, this requires us to instantiate the packing scheme over a slightly larger ring with $t = k + 2s$ (as opposed to $t = k + s$ in [OSV20, CKL21]), and to later "throw away" the s least significant bits.

5 LHE-based Triple Generation

In this section we present our new LHE-based Beaver triple generation protocol. This is the core of our offline phase and, combined with the online phase of, e.g., [CDE$^+$18], enables secure computation over $\mathbb{Z}_{2^k}$. As in Overdrive LowGear, we first construct a pairwise *vector oblivious linear evaluation (VOLE)* subroutine Π_{VOLE} based on the linear homomorphic properties of the BGV scheme (cf. Section 3). In our offline phase, the subroutine is then used in the construction of $c = a \cdot b$, $\alpha \cdot a$ and $\alpha \cdot c$ in a Beaver triple $([\![a]\!], [\![b]\!], [\![c]\!])$. Π_{VOLE} not only combines the state-of-the-art tweaked interpolation packing with LowGear but also adds a modulus switching step that reduces the overall communication by about 40%. Moreover, we present our new authentication subprotocol Π_{Auth}. Finally, we use Π_{VOLE} and Π_{Auth} in our triple generation protocol Π_{Triple}. This extends the recent optimization [RRKK23] of LowGear to the $\mathbb{Z}_{2^k}$ setup and solves the additional security issues that arise in the $\mathbb{Z}_{2^k}$ setup.

The protocols make use of three ideal functionalities. $\mathcal{F}_{\mathsf{Rand}}$ is a (standard) functionality that provides all parties with a (common) uniformly random value (or vector of values). The other two functionalities $\mathcal{F}_{\mathsf{ZKPoK}}$ and $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$ ensure correctly encrypted values are sent to the receiving parties, i.e., $\mathcal{F}_{\mathsf{ZKPoK}}$ ensures a valid encryption of a correctly packed message and $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$ ensures a valid encryption of a constant polynomial (i.e., diagonal plaintext). For a formal definition of these two functionalities, see Appendix A. They are realized with the respective ZKPoPKs (Section 6). Moreover, the realization of $\mathcal{F}_{\mathsf{ZKPoK}}$ requires an additional ZKPoMK in order to guarantee correct packing.

5.1 Vector Oblivious Linear Evaluation (VOLE)

A central subprotocol in the triple generation is our VOLE protocol Π_{VOLE} (Figure 2). This two-party protocol multiplies the vector input $\boldsymbol{a}$ of one party $\mathcal{A}$ component-wise with each of the n vector inputs $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}$ of another party $\mathcal{B}$. The results are shares $\boldsymbol{d}_i$ at $\mathcal{A}$ and shares $\boldsymbol{e}_i$ at $\mathcal{B}$ such that $\boldsymbol{d}_i + \boldsymbol{e}_i = \boldsymbol{a} \odot \boldsymbol{b}_i$ for $i \in [n]$. As $\boldsymbol{a}$ is the same for each $\boldsymbol{b}_i$, the protocol only needs to provide it once (together with a single ZKPoPK) for arbitrarily many $\boldsymbol{b}_i$.

Two-Party Protocol Π_{VOLE}

$\mathcal{A}(\boldsymbol{a} \in \mathbb{Z}_{2^t}^M)$:		$\mathcal{B}(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1} \in \mathbb{Z}_{2^t}^M)$:
$a := \mathsf{pack}(\boldsymbol{a})$	$\rightarrow \boxed{\mathcal{F}_{\mathsf{ZKPoK}}}$	$\rightarrow \boldsymbol{c}_a$
		$\forall i \in [n]$:
		$\boldsymbol{e}_i \xleftarrow{\$} \mathbb{Z}_{2^{t+s}}^M$
		$m_i \xleftarrow{\$} 2^{t+s+2\delta}\mathcal{R}_{2^E}$
	$\xleftarrow{(\boldsymbol{c}'_{d_i})_i}$	$\boldsymbol{c}_{d_i} \leftarrow \boldsymbol{c}_a b_i - \mathsf{Enc}'_{\mathsf{pk}_{\mathcal{A}}}\left(\mathsf{pack}'(\boldsymbol{e}_i) + m_i\right)$
$\forall i \in [n]$:		$\boldsymbol{c}'_{d_i} := \mathsf{SwitchMod}(\boldsymbol{c}_{d_i})$
$d_i := \mathsf{Dec}_{\mathsf{sk}_{\mathcal{A}}}(\boldsymbol{c}'_{d_i})$		**Return** $(\boldsymbol{e}_i)_i$
$\boldsymbol{d}_i := \mathsf{unpack}(d_i)$ (or abort if not a valid packing)		
Return $(\boldsymbol{d}_i)_i$		

Fig. 2: Protocol for vector oblivious linear evaluation.

Π_{VOLE} is modelled following the LowGear pairwise multiplication protocol in [KPR18, Figure 7, **Multiply**, step 2]. However, Π_{VOLE} has several novelties: Most notably, the inputs are vectors over $\mathbb{Z}_{2^t}$ instead of a finite field. In order to pack many elements from $\mathbb{Z}_{2^t}$ into a single BGV ciphertext, $\mathit{\Pi}_{\mathsf{VOLE}}$ uses the tweaked interpolation packing with M denoting the number of slots. Note that the tweaked interpolation packing is not surjective onto the plaintext space. As a consequence, just like in [CKL21], the functionality $\mathcal{F}_{\mathsf{ZKPoK}}$ needs to guarantee not only that $\boldsymbol{c}_a$ is a valid ciphertext but also that the underlying plaintext is a valid packing of some message. That is, in the realization of $\mathcal{F}_{\mathsf{ZKPoK}}$, the prover $\mathcal{A}$ needs to perform not only a ZKPoPK but also a ZKPoMK.

So far, what we described applies the techniques from [CKL21] to the LowGear pairwise multiplication protocol. Furthermore, our protocol $\mathit{\Pi}_{\mathsf{VOLE}}$ contains a new optimization: Before returning the ciphertexts $\boldsymbol{c}_{d_i}$ to party $\mathcal{A}$, we apply modulus switching in order to significantly reduce the size of most transmitted ciphertexts, i.e., by 54–66% in our setup (cf. Section 7.2). Note that we must start with a large enough modulus q_1 such that we can apply drowning with sufficiently large noise, i.e., the noise of Enc'. After the noise is added, we can switch to a smaller modulus q_0. The only constraint is that q_0 must still be large enough to ensure correct decryption (cf. Lemma 1). This optimization reduces the overall communication of the triple generation, i.e., including those ciphertexts that cannot be modulus-switched and including other values that need to be transmitted, by about 40%.

Note that, similar to [CKL21], we use a ZKPoMK with slack in the exponent for a more efficient ZKPoMK. Therefore, the plaintext modulus of BGV is chosen as 2^{T+E}, having E bits of additional slack. To avoid that the (homomorphically evaluated) product $\boldsymbol{a} \odot \boldsymbol{b}_i$ leaks information from overflowing into these additional E bits, a masking m_i is added in Figure 2 that hides any information that could be in these upper bits. Finally, the products $\boldsymbol{a} \odot \boldsymbol{b}_i$ are masked using our uniform distribution on $\mathsf{pack}'(\boldsymbol{e})$ described in Section 2.3.

5.2 Authentication

The authentication subprotocol $\mathit{\Pi}_{\mathsf{Auth}}$ which generates authenticated shares $[\![\boldsymbol{b}]\!]$ for a shared vector $\boldsymbol{b}$ is presented in Figure 3. It first computes $[\alpha]_i \cdot [\boldsymbol{b}]_j$ for each ordered pair of parties (P_i, P_j). As the MAC key α is fixed, this is a simple scalar multiplication. In contrast to $\mathit{\Pi}_{\mathsf{VOLE}}$ this allows us to use the more efficient coefficient packing for $[\boldsymbol{b}]_j$ instead of the tweaked interpolation packing in $\mathit{\Pi}_{\mathsf{VOLE}}$. This way, we achieve the optimal packing efficiency for the authentication, which is over a factor of 2 more efficient than the tweaked interpolation packing.

We briefly describe $\mathit{\Pi}_{\mathsf{Auth}}$ in more detail. First note that all parties need to obtain encrypted shares of α only once during the offline phase. Thus, $\mathit{\Pi}_{\mathsf{Auth}}$ performs the first step **Init** (sending an encryption of $[\alpha]_i$) only once over all invocations of **Auth**. In **Init** the prover $\mathcal{A}$ needs to prove that $\boldsymbol{c}_{[\alpha]_i}$ is a valid ciphertext that encrypts some *diagonal* element (i.e., constant polynomial). This is captured by the functionality $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$ which can be realized using the diagonal version of our ZKPoPK. It does not require a ZKPoMK.

In $\mathit{\Pi}_{\mathsf{Auth}}$ we use a modified version of $\mathit{\Pi}_{\mathsf{VOLE}}$ that is invoked by each ordered pair of parties (P_i, P_j) such that they obtain pairwise shares $(\boldsymbol{d}^{(i,j)}, \boldsymbol{e}^{(i,j)})$ with $\boldsymbol{d}^{(i,j)} + \boldsymbol{e}^{(i,j)} = [\alpha]_i[\boldsymbol{x}]_j$. By the linearity of shares, we can combine all pairwise shares to get shares of $\alpha \cdot \boldsymbol{x}$. More specifically, due to

$$\alpha \cdot \boldsymbol{x} = \sum_{i,j\in[N]} [\alpha]_i[\boldsymbol{x}]_j = \sum_{i\in[N]} [\alpha]_i[\boldsymbol{x}]_i + \sum_{\substack{i,j\in[N]\\ j\neq i}} (\boldsymbol{d}^{(i,j)} + \boldsymbol{e}^{(i,j)}) = \sum_{i\in[N]} [\alpha]_i[\boldsymbol{x}]_i + \sum_{\substack{i,j\in[N]\\ j\neq i}} (\boldsymbol{d}^{(i,j)} + \boldsymbol{e}^{(j,i)}),$$

we can use $[\alpha]_i[\boldsymbol{x}]_i + \sum_{j\in[N]\setminus\{i\}} (\boldsymbol{d}^{(i,j)} + \boldsymbol{e}^{(j,i)})$ as P_i's share of $\alpha \cdot \boldsymbol{x}$. Note that the summand $[\alpha]_i[\boldsymbol{x}]_i$ is computed locally.

5.3 Triple Generation Without Sacrificing

In [RRKK23] the authors construct a LHE-based triple generation protocol without *sacrificing* for the prime field setting. Prior Overdrive-based triple generation protocols [KPR18, BCS19, OSV20, CKL21] need to sacrifice one triple in order to verify correctness of another triple. Thus, almost half of the

```
N-Party Protocol $\Pi_{\mathsf{Auth}}$
Init: Each party $P_i$ does the following:
  1. $[\alpha]_i \xleftarrow{\$} \mathbb{Z}_{2^s}$
  2. For each party $P_j$ with $j \neq i$, use $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$ to send an encryption $\boldsymbol{c}_{[\alpha]_i}$ of $[\alpha]_i$ to $P_j$.
Auth: Each party $P_i$ has input $[\boldsymbol{x}]_i \in \mathbb{Z}_{2^k}^{\varphi(m)}$ and does:
  1. $x_i := \mathsf{coeffPack}([\boldsymbol{x}]_i)$
  2. For each party $P_j$ with $j \neq i$:
     (a) $\boldsymbol{e}^{(j,i)} \xleftarrow{\$} \mathbb{Z}_{2^{t+s}}^{\varphi(m)}$
     (b) $e^{(j,i)} := \mathsf{coeffPack}(\boldsymbol{e}^{(j,i)})$, i.e., the coefficient packing.
     (c) $\boldsymbol{c}_{d^{(j,i)}} \leftarrow \boldsymbol{c}_{[\alpha]_j} x_i - \mathsf{Enc}'_{\mathsf{pk}_j}(e^{(j,i)})$
     (d) $\boldsymbol{c}'_{d^{(j,i)}} := \mathsf{SwitchMod}(\boldsymbol{c}_{d^{(j,i)}})$
     (e) Send $\boldsymbol{c}'_{d^{(j,i)}}$ to $P_j$, receive $\boldsymbol{c}'_{d^{(i,j)}}$.
     (f) $d^{(i,j)} := \mathsf{Dec}_{\mathsf{sk}_i}(\boldsymbol{c}'_{d^{(i,j)}})$
     (g) $\boldsymbol{d}^{(i,j)} := \mathsf{coeffUnpack}(d^{(i,j)})$
  3. $[\boldsymbol{\gamma_x}]_i := [\alpha]_i \cdot [\boldsymbol{x}]_i + \sum_{j \neq i} \left(\boldsymbol{d}^{(i,j)} + \boldsymbol{e}^{(j,i)}\right)$
  4. Return $[\![\boldsymbol{x}]\!]$
```

Fig. 3: Protocol for share authentication.

computed triples are thrown away for sacrificing.[8] In contrast, [RRKK23] presents a variation to LowGear, called LowGear 2.0, where sacrificing can be omitted while maintaining security. Here we show that this variation can be transferred to the $\mathbb{Z}_{2^k}$ setting, too.

However, packed plaintexts over $\mathbb{Z}_{2^k}$ come with a certain structure that can be exploited by an adversary. E.g., the Overdrive2k packing [OSV20] when used with [RRKK23] is insecure: This packing requires certain coefficients to be zero. Without any additional ZKPoMK an adversary can however inject an arbitrary value $\hat{a} \in \mathbb{Z}_{2^k}$ in one of these zero coefficients such that one coefficient of the product becomes $ab + \hat{a}b'$. For example, instead of our previous example $(aX + a')(bX + b') = abX^2 + (ab' + a'b)X + a'b'$ (cf. Section 2) the adversary now forces the computation $(\hat{a}X^2 + aX + a')(bX + b') = \hat{a}bX^3 + (ab + \hat{a}b')X^2 + (ab' + a'b)X + a'b'$ and gets $ab + \hat{a}b'$ as 2-coefficient instead of ab. The triple production would then finish successfully with an authenticated triple $([\![a + \hat{a}]\!], [\![b]\!], [\![ab + \hat{a}b']\!])$, i.e., this triple is correctly authenticated and passes the MAC check, but the multiplicative relation is not fulfilled. By using ZKPoMKs, we can guarantee that $\boldsymbol{c}_a$ encodes a correctly packed message, thus preventing such attacks. For the full security proof see Appendix A.

In Figure 4 we present our triple generation protocol Π_{Triple} which is modelled following [RRKK23]. Given (vectors of) $[a]$ and $[b]$, it needs to compute $[\alpha a]$, $[\alpha b]$, $[ab]$, and $[\alpha ab]$. Observe that all $[\alpha a]$, $[ab]$, and $[\alpha ab]$ are products with a. Therefore, we first authenticate $[b]$ to obtain $[\alpha b]$ and then use the VOLE protocol Π_{VOLE}. Finally, we combine the pairwise products similarly to what is done in Π_{Auth}. Intertwining the authentication and component-wise multiplication in this special way prevents the attack that required prior protocols to check triples using the sacrificing technique. We obtain a protocol that guarantees the multiplicative relation $c = ab$ *without sacrificing* as we prove in Appendix A. This optimization reduces the communication cost (see Section 7.4) as well as the round complexity of Multipars compared to a sacrificing-based approach. As mentioned before, this proof is specially designed for the case $\mathbb{Z}_{2^k}$ since the original proof in [RRKK23] does not directly carry over to $\mathbb{Z}_{2^k}$.

[8] In fact, 2 entries have to be sacrificed in modern protocols starting from [KOS16]. Original SPDZ still sacrificed all 3 entries; MP-SPDZ [DEF+19, Kel20] uses this non-optimized sacrificing.

N-Party Protocol Π_{Triple}

Init: The parties initialize the Π_{Auth} subprotocol.

Triple:

1. Each party P_i samples $[\overline{\boldsymbol{a}}]_i \xleftarrow{\$} \mathbb{Z}_{2^{k+s}}^M$, $[\boldsymbol{b}]_i \xleftarrow{\$} \mathbb{Z}_{2^k}^M$, and $[r_0]_i, [r_1]_i, [r_2]_i \xleftarrow{\$} \mathbb{Z}_{2^k}$.
2. They authenticate $[\boldsymbol{b}||r_0||r_1||r_2]$ over $\mathbb{Z}_{2^{k+s}}$ using a single invocation of Π_{Auth}, obtaining $[\![\boldsymbol{b}]\!]$, $[\![m]\!]$, and $[\![r]\!]$.
3. Let $\boldsymbol{\alpha}$ be the vector of length M containing α in each entry. Each ordered pair of parties (P_i, P_j) runs $\Pi_{\mathsf{VOLE}}([\overline{\boldsymbol{a}}]_i, ([\boldsymbol{\alpha}]_j, [\boldsymbol{b}]_j, [\boldsymbol{\gamma_b}]_j))$ where P_i obtains $(\boldsymbol{d}_0^{(i,j)}, \boldsymbol{d}_1^{(i,j)}, \boldsymbol{d}_2^{(i,j)})$ and P_j obtains $(\boldsymbol{e}_0^{(i,j)}, \boldsymbol{e}_1^{(i,j)}, \boldsymbol{e}_2^{(i,j)})$.
4. Each party P_i computes (modulo 2^{k+2s})
$$[\boldsymbol{\gamma_{\overline{a}}}]_i := [\overline{\boldsymbol{a}}]_i \cdot [\alpha]_i + \sum\nolimits_{j \neq i} \left(\boldsymbol{d}_0^{(i,j)} + \boldsymbol{e}_0^{(j,i)}\right)$$
$$[\overline{\boldsymbol{c}}]_i := [\overline{\boldsymbol{a}}]_i \odot [\boldsymbol{b}]_i + \sum\nolimits_{j \neq i} \left(\boldsymbol{d}_1^{(i,j)} + \boldsymbol{e}_1^{(j,i)}\right)$$
$$[\boldsymbol{\gamma_{\overline{c}}}]_i := [\overline{\boldsymbol{a}}]_i \odot [\boldsymbol{\gamma_b}]_i + \sum\nolimits_{j \neq i} \left(\boldsymbol{d}_2^{(i,j)} + \boldsymbol{e}_2^{(j,i)}\right).$$
5. They use $\mathcal{F}_{\mathsf{Rand}}$ to obtain (public) $\boldsymbol{t} \xleftarrow{\$} \mathbb{Z}_{2^s}^{3M}$.
6. $[\![y]\!] := \langle \boldsymbol{t}, [\![\overline{\boldsymbol{a}}||\boldsymbol{b}||\overline{\boldsymbol{c}}]\!]\rangle + [\![r_0]\!] + 2^k[\![r_1]\!] + 2^{k+s}[\![r_2]\!]$
7. They open y and run $\Pi_{\mathsf{SingleCheck}}$ on $(y, [\gamma_y])$. If the check fails, abort.
8. $([\![\boldsymbol{a}]\!], [\![\boldsymbol{c}]\!]) \leftarrow \Pi_{\mathsf{Trunc}}([\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!])$
9. **Return** $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!])$.

Fig. 4: Protocol for Beaver triple generation.

N-Party Protocol Π_{Trunc}

Let $[\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$ be vectors of authenticated shares with modulus 2^{t+s}.

1. Each party P_i reveals $[\overline{\boldsymbol{a}}]_i \bmod 2^s$.
2. Each party P_i computes
$$\boldsymbol{\Sigma a} := \sum\nolimits_j ([\overline{\boldsymbol{a}}]_j \bmod 2^s)$$
$$[\boldsymbol{\gamma_{\hat{a}}}]_i := [\boldsymbol{\gamma_{\overline{a}}}]_i - \boldsymbol{\Sigma a} \cdot [\alpha]_i \qquad [\hat{\boldsymbol{c}}]_i := [\overline{\boldsymbol{c}}]_i - \boldsymbol{\Sigma a} \odot [\boldsymbol{b}]_i \qquad [\boldsymbol{\gamma_{\hat{c}}}]_i := [\boldsymbol{\gamma_{\overline{c}}}]_i - \boldsymbol{\Sigma a} \odot [\boldsymbol{\gamma_b}]_i.$$
Formally, set $\hat{\boldsymbol{a}} := \overline{\boldsymbol{a}} - \boldsymbol{\Sigma a}$ (not known to any party).
3. Each party P_i commits to $([\boldsymbol{\gamma_{\hat{a}}}]_i \bmod 2^s, [\hat{\boldsymbol{c}}]_i \bmod 2^s, [\boldsymbol{\gamma_{\hat{c}}}]_i \bmod 2^s)$.
4. The parties open their commitments and compute
$$\boldsymbol{\Sigma\gamma_{\hat{a}}} := \sum\nolimits_j ([\boldsymbol{\gamma_{\hat{a}}}]_j \bmod 2^s) \qquad \boldsymbol{\Sigma\hat{c}} := \sum\nolimits_j ([\hat{\boldsymbol{c}}]_j \bmod 2^s) \qquad \boldsymbol{\Sigma\gamma_{\hat{c}}} := \sum\nolimits_j ([\boldsymbol{\gamma_{\hat{c}}}]_j \bmod 2^s).$$
5. The parties check that $\boldsymbol{\Sigma\gamma_{\hat{a}}} \equiv \boldsymbol{\Sigma\hat{c}} \equiv \boldsymbol{\Sigma\gamma_{\hat{c}}} \equiv \boldsymbol{0} \pmod{2^s}$. If the check fails, abort.
6. Each party P_i computes
$$[\boldsymbol{a}]_i := \lfloor[\overline{\boldsymbol{a}}]_i/2^s\rfloor = ([\overline{\boldsymbol{a}}]_i - ([\overline{\boldsymbol{a}}]_i \bmod 2^s))/2^s$$
$$[\boldsymbol{\gamma_a}]_i := \lfloor[\boldsymbol{\gamma_{\hat{a}}}]_i/2^s\rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{a}}}/2^s$$
$$[\boldsymbol{c}]_i := \lfloor[\hat{\boldsymbol{c}}]_i/2^s\rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\hat{c}}/2^s$$
$$[\boldsymbol{\gamma_c}]_i := \lfloor[\boldsymbol{\gamma_{\hat{c}}}]_i/2^s\rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{c}}}/2^s,$$
where only party P_0 adds the summands on the right.
7. **Return** $([\![\boldsymbol{a}]\!], [\![\boldsymbol{c}]\!])$.

Fig. 5: Protocol for truncation of Beaver triples.

Truncation. In order to use 2^s-enhanced CPA security in our triple generation protocol Π_{Triple} (cf. Figure 4), we create larger (Beaver) triples $\overline{a}, b, \overline{c}$ first, where $\overline{a}, \overline{c}$ are from the larger domain $\mathbb{Z}_{2^{t+s}}$ and b from $\mathbb{Z}_{2^t}$. We then employ a truncation subprotocol Π_{Trunc} presented in Figure 5 to remove the additional s bits in the first and third component of the triple. In Π_{Trunc} each party publishes the s least significant bits of her shares, where commitments ensure that an adversary cannot choose his shares depending on the honest parties' shares. Each party can then reconstruct the sum of these s bits of $\overline{a}$ and $\overline{c}$ (and of the corresponding MAC-shares). Please note that due to the 2^s-enhanced CPA security of the BGV scheme, these bits of $\overline{a}, \overline{c}$ were constructed independently of the more significant bits, i.e., they do not leak any information on the more significant bits (see Appendix A for a formal proof).[9] The parties use the opened s bits to locally compute shares of a truncation of $\overline{a}, \overline{c}$. We remark that due to possible overflow the actual outputs (a, c) of Π_{Trunc} can slightly differ from the truncation of $\overline{a}, \overline{c}$. However, the protocol ensures that all relations between (a, b, c), namely $ab \equiv_k c$, $\gamma_a \equiv_t \alpha a$, $\gamma_b \equiv_t \alpha b$, and $\gamma_c \equiv_t \alpha c$, are guaranteed and that (apart from these relations) the values are uniformly random and no information on them is leaked. We prove the correctness of Π_{Trunc} in Appendix A.

Adaptive Packing. In contrast to [RRKK23] (as well as the original LowGear protocol [KPR18]) where the same packing technique is used for both authentication and component-wise multiplication, we use our subprotocols Π_{Auth} and Π_{VOLE} which utilize different packing techniques. Please note that we had to adapt the BGV parameters to fully use the potentials of both packings, i.e., tweaked interpolation packing in Π_{VOLE} and coefficient packing in Π_{Auth}. To be more precise: Π_{Auth} can authenticate up to $\varphi(m)$ values per invocation (due to our usage of the coefficient packing), but in Π_{Triple} it is invoked with only $M+3$ values where $M \approx \varphi(m)/2$ is the capacity of the tweaked interpolation packing. We solve this by using a different BGV parameter set for Π_{Auth} (than for Π_{VOLE}) with prime cyclotomic index $\hat{m}$ just large enough such that we have $\varphi(\hat{m}) \geq M + 3$ coefficients.

Batched MAC Check. As mentioned before, we need to perform a (batched) MAC check in the triple generation protocol, in order to guarantee that all shares have been authenticated correctly. The batched MAC check is performed using a standard technique where a random linear combination $\langle \boldsymbol{t}, [\![\overline{\boldsymbol{a}}]\!] || \boldsymbol{b} || [\![\overline{\boldsymbol{c}}]\!] \rangle \in \mathbb{Z}_{2^{k+2s}}$ of the shares is checked. Additionally, this linear combination is masked, such that no information gets leaked. While in [RRKK23] (i.e., in the prime field setting) this required only a single mask m, in our setting we need multiple masks $r_0, r_1, r_2 \in \mathbb{Z}_{2^k}$ in order to mask the upper $2s$ bits of the random linear combination, too. Note that this requires $s \leq k$, which is given for typical values of s and k [CDE+18, OSV20, CKL21]. For the MAC check itself, we invoke the protocol $\Pi_{\mathsf{SingleCheck}}$ presented in Figure 6. It is the same as the SingleCheck procedure in [CDE+18], except that in our protocol the input is already masked and opened before calling $\Pi_{\mathsf{SingleCheck}}$. This way, we can produce the authenticated masks $[\![r_0]\!], [\![r_1]\!], [\![r_2]\!]$ within Π_{Triple} for no extra cost, instead of invoking Π_{Auth} once more within the MAC check subprotocol.

N-Party Protocol $\Pi_{\mathsf{SingleCheck}}$

The parties have common input $y \in \mathbb{Z}_{2^{t+s}}$ and each party P_i additionally has $[\gamma_y]_i$.

1. Each party P_i commits to $z_i := [\gamma_y]_i - y \cdot [\alpha]_i \bmod 2^{t+s}$.
2. The parties open their commitments and check that $\sum_i z_i \equiv_{t+s} 0$.

Fig. 6: MAC check protocol for a single value.

Theorem 3. *For any number of parties $N \in \mathbb{N}$, Π_{Triple} realizes $\mathcal{F}_{\mathsf{Triple}}$ in the standalone model with rewinding black-box simulator.*

[9] Additionally, we show in Appendix A that similar selective failure attacks on Π_{Auth} do also not present a security issue.

We prove the security of Π_{Triple}, stated here as Theorem 3, in Appendix A in the standalone model with rewinding black-box simulator [Gol04, Lin17]. This is the same security model as in the Overdrive paper [KPR18].[10] The core idea of the proof is the following. Firstly, an adversary cannot infer any information from the messages of honest parties, thus a simulation with (encrypted) dummy values is sufficient. Secondly, the simulator is in full control of the functionalities used in the simulation and can thus extract data from corrupted parties. Lastly, a successful MAC check in the simulation gives us guarantees about the correlation between shared values (with overwhelming probability) and thus not aborting in the simulation implies that we have correctly correlated shares.

6 Zero-Knowledge Proofs

In this section we describe our improved version of the ZKPoPK from [CKL21], the current state-of-the-art ZKPoPK for BGV ciphertexts with plaintext modulus 2^T. We also present our ZKPoSKK, which works very similar to the ZKPoPK. Recall that our ZKPoPK is used for each invocation of Π_{VOLE} (together with a ZKPoMK) in the triple generation. It is therefore an essential part of the whole offline phase, both security- and efficiency-wise.

The ZKPoPK from [CKL21] (in its original form) is not directly applicable in our setup, since it only allows us to prove statements about a multiple $m \cdot \boldsymbol{c}$ of a ciphertext $\boldsymbol{c}$ and not about $\boldsymbol{c}$ itself. This weaker nature of their ZKPoPK seems to have been overlooked by the authors of [CKL21]. However, there is a standard solution to these kind of problems, where the ciphertext $\boldsymbol{c}$ is replaced by its multiple $m \cdot \boldsymbol{c}$ after the ZKPoPK finished successfully.[11]

In this paper we follow a different approach that allows us to circumvent this problem entirely by using a new challenge space introduced in [AL21]. In Appendix C we show our instantiation of this challenge space and prove the necessary properties. In particular, we do not need to change the ciphertexts (and hence the corresponding protocols they are used in) after the ZKPoPK. Our improved ZKPoPK presented below reduces the communication by reducing the number of values that need to be transmitted. Additionally, we reduce the slack by employing the rejection sampling idea from [Lyu09] in the interactive setting. The reduced slack in turn enables the usage of smaller BGV parameters, which reduces overall communication of our triple generation protocol by 3–7% (compared to 80-bit statistical zero-knowledge without rejection sampling[12]) in the setup used for our evaluation (cf. Section 7). At the same time, rejection sampling endows our ZKPoPK with *perfect* instead of statistical zero-knowledge. Please also note, that our ZKPoPK is not an N-prover zero-knowledge proof of knowledge (ZKPoK), e.g., as the ZKPoPK in [BCS19, CKL21], but a classical (1-prover) ZKPoK.

Our ZKPoPK Protocol. In a ZKPoPK, on common input $x = (\boldsymbol{c}, \mathsf{pk})$, the prover proves to the verifier that the ciphertext $\boldsymbol{c}$ was computed honestly from a plaintext and randomness that meet certain bounds and additionally that the prover *knows* such a small plaintext and randomness (instead of, e.g., reusing a valid ciphertext of another party). To improve efficiency, the proof is done for a batch of ciphertexts. As usual [DPSZ12, KPR18, BCS19, CKL21], to achieve this amortization, we use a $U \times 2$ ciphertext matrix C where each of its U rows is a ciphertext. The proof has a parameter $\mathsf{flag} \in \{\perp, \mathsf{Diag}\}$ that determines whether the prover wants to additionally prove that (for each of the U ciphertexts) the plaintext is a so-called *diagonal* element, i.e., a constant polynomial. That is, the plaintext space is defined by $\varrho(\mathsf{flag}) := \mathcal{R}_{2^T}$ if $\mathsf{flag} = \perp$ and $\varrho(\mathsf{flag}) := \mathbb{Z}_{2^T}$ if $\mathsf{flag} = \mathsf{Diag}$.

Our ZKPoPK and the upcoming proofs make use of deterministic BGV encryption for given randomness: let $\mathsf{Enc}_{\mathsf{pk}}(\cdot, v, e_0, e_1)$ denote deterministic encryption with randomness v, e_0, e_1. We also use the natural extension of this to encrypt vectors of plaintexts with vectors of randomness.

[10] In [KPR18], the authors call this model a limited version of the UC model [Can01].
[11] This is the approach taken in [BCS19], where the factor is only 2.
[12] We compare against 80-bit statistical zero-knowledge as in [BCS19], because using the statistical security parameter ($\approx s$) instead would be too low, as also explained in [BCS19].

A honest prover that computed all ciphertexts honestly knows a witness w such that (x, w) belongs to the relation

$$\begin{aligned}\mathbb{L} := \{((C, \mathsf{pk}), (\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)) :\\ C \in \mathcal{R}_q^{U\times 2}, \mathsf{pk} \in \mathcal{R}_q^2, \boldsymbol{m} \in \varrho(\mathsf{flag})^U, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1 \in \mathcal{R}^U,\\ \|\boldsymbol{v}\|_\infty \le 1, \|\boldsymbol{e}_0\|_\infty, \|\boldsymbol{e}_1\|_\infty \le 2\sigma^2, C = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)\}.\end{aligned}$$

As our ZKPoPK has some slack, it only guarantees knowledge soundness for the following relation where the bounds on the witness are relaxed by a factor of $S_{\mathsf{ZKPoPK}} \in \mathbb{R}_{\ge 1}$:

$$\begin{aligned}\mathbb{L}_{S_{\mathsf{ZKPoPK}}} := \{((C, \mathsf{pk}), (\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)) :\\ C \in \mathcal{R}_q^{U\times 2}, \mathsf{pk} \in \mathcal{R}_q^2, \boldsymbol{m} \in \varrho(\mathsf{flag})^U, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1 \in \mathcal{R}^U,\\ \|\boldsymbol{v}\|_\infty \le S_{\mathsf{ZKPoPK}}, \|\boldsymbol{e}_0\|_\infty \le S_{\mathsf{ZKPoPK}} \cdot (2\sigma^2 + 1),\\ \|\boldsymbol{e}_1\|_\infty \le S_{\mathsf{ZKPoPK}} \cdot 2\sigma^2, C = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)\}.\end{aligned}$$

In Figure 7 we present our ZKPoK for $(\mathbb{L}, \mathbb{L}_{S_{\mathsf{ZKPoPK}}})$. Note that, in contrast to prior ZKPoPKs for BGV [DPSZ12, BCS19, CKL21], the message $\boldsymbol{m}$ doesn't have any slack in the dishonest relation $\mathbb{L}_{S_{\mathsf{ZKPoPK}}}$. This is due to the following observation: As the encryption algorithm adds $2^T\boldsymbol{e}_0 + \boldsymbol{m}$, we can simply interpret the parts of $\boldsymbol{m}$ that exceed the bound 2^T as being part of the noise $\boldsymbol{e}_0$. The formal proof of this intuition can be found in Appendix E, where we prove the *knowledge soundness* property of our ZKPoPK. We also use the same observation to combine $2^T\boldsymbol{e}_0 + \boldsymbol{m}$ into a single value in the protocol itself in order to slightly reduce communication.

Protocol Π_{ZKPoPK}, parameterized by $U, V, Z \in \mathbb{N}, \sigma^2 \in \mathbb{R}, \mathsf{flag} \in \{\perp, \mathsf{Diag}\}$

For $C \in \mathcal{R}_q^{U\times 2}, \mathsf{pk} \in \mathcal{R}^2, \boldsymbol{m} \in \mathcal{R}_{2^T}^U, \boldsymbol{v}, \boldsymbol{e_0}, \boldsymbol{e_1} \in \mathcal{R}^U$

$\mathcal{P}\left((C, \mathsf{pk}), (\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e_0}, \boldsymbol{e_1})\right)$: $\mathcal{V}\left((C, \mathsf{pk})\right)$:

Let $S := \vartheta(\mathsf{flag})UZ$.

$\tilde{\boldsymbol{m}} \xleftarrow{\$} \varrho(\mathsf{flag})^V$, $\tilde{\boldsymbol{v}} \leftarrow \mathcal{R}_{\le S}^V$ $W \xleftarrow{\$} \mathsf{Chal}(\mathsf{flag})^{V\times U}$

$\tilde{\boldsymbol{e}}_0 \leftarrow \mathcal{R}^V_{\le S\cdot(2\sigma^2+1)}$, $\tilde{\boldsymbol{e}}_1 \leftarrow \mathcal{R}^V_{\le S\cdot 2\sigma^2}$

$A := \mathsf{Enc}_{\mathsf{pk}}(\tilde{\boldsymbol{m}}, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{e}}_0, \tilde{\boldsymbol{e}}_1)$ $\xrightarrow{A}$

$\xleftarrow{W}$

$\hat{\boldsymbol{v}} := \tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}$

$\hat{\boldsymbol{m}} := 2^T\tilde{\boldsymbol{e}}_0 + \tilde{\boldsymbol{m}} + W \cdot (2^T\boldsymbol{e}_0 + \boldsymbol{m})$

$\hat{\boldsymbol{e}}_1 := \tilde{\boldsymbol{e}}_1 + W \cdot \boldsymbol{e}_1$ $\xrightarrow{\hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1}$ $D := \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}, \hat{\boldsymbol{v}}, 0, \hat{\boldsymbol{e}}_1)$

Check whether:

$\|\hat{\boldsymbol{v}}\|_\infty \le S$ $\|\hat{\boldsymbol{e}}_1\|_\infty \le S \cdot 2\sigma^2$

$\|\hat{\boldsymbol{m}}\|_\infty \le S \cdot 2^T(2\sigma^2 + 1)$ $D = A + W \cdot C$

If $\mathsf{flag} = \mathsf{Diag}$, additionally check whether each entry of $\hat{\boldsymbol{m}} \bmod 2^T$ contains a constant polynomial.

If all checks passed, then accept. Otherwise, reject.

Fig. 7: Our ZKPoPK for BGV ciphertexts over prime cyclotomics and plaintext space $\mathcal{R}_{2^T}$.

Diagonal Plaintexts. For parameter $\mathsf{flag} = \mathsf{Diag}$, our ZKPoPK additionally proves that each entry in the plaintext vector $\boldsymbol{m}$ is a constant polynomial. The challenge space Chal from Appendix C cannot be used in the $\mathsf{flag} = \mathsf{Diag}$ case. In this case, we have to resort to the challenge space $\{0, 1\}$, as usual [BCS19, CKL21]. The usage of the challenge space $\{0, 1\}$ decreases efficiency (compared to the $\mathsf{flag} = \perp$ case), as one has to choose a much larger number V of transmitted ciphertexts to achieve the same knowledge error. On the positive side, $\mathsf{flag} = \mathsf{Diag}$ decreases the slack by $\varphi(m)^2$.

Security. The security proof for $\mathit{\Pi}_{\mathsf{ZKPoPK}}$ can be found in Appendix E, while the security definitions can be found in Appendix D. The protocol provides security as stated in the following theorem. To distinguish the two challenge spaces, we use the notations $\mathsf{Chal}(\mathsf{Diag}) := \{0, 1\}$ and $\mathsf{Chal}(\perp) := \{w_i \mid 0 \leq i < m\}$. Furthermore, we denote by $\vartheta(\mathsf{Diag}) := 1$ and $\vartheta(\perp) := \varphi(m)$ the maximum factor by which the norm of a polynomial can grow when multiplied by a challenge.

Theorem 4. *If $\mathsf{flag} = \perp$, $V \geq (\mathsf{Snd\_sec} + 2)/\log_2|\mathsf{Chal}(\mathsf{flag})|$, and $Z \geq 2^{\mathsf{ZK\_sec}}$ then $\mathit{\Pi}_{\mathsf{ZKPoPK}}$ from Figure 7 (in Appendix E) is*

- *complete (Definition 4) for $\mathbb{L}$ with negligible completeness error,*
- *knowledge sound (Definition 5) for $\mathbb{L}_{S_{\mathsf{ZKPoPK}}}$ with $S_{\mathsf{ZKPoPK}} = 2\vartheta(\mathsf{flag})^2 UZ$ and knowledge error $2^{-\mathsf{Snd\_sec}}$,*
- *special honest verifier zero-knowledge (Definition 6) for $\mathbb{L}$ with statistical distance $2^{-\mathsf{ZK\_sec}}$.*

Security Against Malicious Verifiers. Note that Theorem 4 only provides the zero-knowledge property of our ZKPoPK for honest verifiers. We therefore add (similar to, e.g., [BBC+18]) a secure coin-flipping protocol to sample the challenge W and therewith make our protocols secure against malicious verifiers, too. Details on the secure coin-flip protocol are presented in Appendix G.

An alternative standard approach to security against malicious verifiers is the use of the Fiat-Shamir transform [FS86] in the random oracle model (ROM). However, when using the Fiat-Shamir transform, the prover can retry the proof many times (within the prover's computation budget) until he is successful. When implementing the protocol in practice, this implies that one has to choose the knowledge error much more conservative than in an interactive proof where the prover only has a single attempt. Specifically, in a non-interactive proof one should then set $\mathsf{Snd\_sec}$ to the computational security parameter, which would blow up the size of our proofs by a factor of 2 to 4. For this reason, we decided to stick with an interactive protocol.

6.1 Rejection Sampling

The slack of our ZKPoPK (Figure 7) can be reduced with the rejection sampling idea from [Lyu09]. In Figure 8 we present our ZKPoPK with rejection sampling which we use in our Multipars implementation. Using rejection sampling in this context was already done in SPDZ [DPSZ12]. However, we apply the idea to interactive proofs (similar to what is done in [BDLN16]) instead of non-interactive ones and we thus need to take certain precautions to maintain security. $\mathit{\Pi}^{\mathsf{RS}}_{\mathsf{ZKPoPK}}$ has a new parameter P that introduces a trade-off between failure probability (i.e., completeness error) and slack. Again, for $\mathsf{flag} = \mathsf{Diag}$, $\mathit{\Pi}^{\mathsf{RS}}_{\mathsf{ZKPoPK}}$ falls back to $\mathsf{Chal}(\mathsf{Diag}) = \{0, 1\}$ which increases the requirement on the parameter V.

In practice, we choose a small P in order to keep the slack low, which, on its own, leads to an unacceptable failure probability. To reduce the failure probability, the verifier allows the prover to repeat the proof rep times and accepts as soon as an attempt succeeds. There still needs to be a limit on the number of attempts, in order to bound the knowledge error. In particular, when allowing the prover rep attempts, $\mathit{\Pi}^{\mathsf{RS}}_{\mathsf{ZKPoPK}}$ achieves completeness error $1/P^{\mathsf{rep}}$ and knowledge error $2^{-\mathsf{Snd\_sec}} \cdot \mathsf{rep}$. By using

$$V = \left\lceil \frac{\mathsf{Snd\_sec} + \log_2(\mathsf{rep}) + 2}{\log_2|\mathsf{Chal}(\mathsf{flag})|} \right\rceil, \tag{3}$$

we reestablish knowledge error $2^{-\mathsf{Snd\_sec}}$. In our setup, concretely with $P = 256$, this construction increases the expected runtime of the ZKPoPK only by 0.4% while it reduces the size of our BGV parameters such that the overall communication of our triple generation protocol is reduced by 3–7% (compared to 80-bit statistical zero-knowledge without rejection sampling as shown in Table 3b).

More specifically, $\mathit{\Pi}^{\mathsf{RS}}_{\mathsf{ZKPoPK}}$ has security properties as stated in the following theorem. The corresponding proof is contained in Appendix F.

Protocol $\Pi_{\mathsf{ZKPoPK}}^{\mathsf{RS}}$, parameterized by $U, V, P \in \mathbb{N}, \sigma^2 \in \mathbb{R}, \mathsf{flag} \in \{\perp, \mathsf{Diag}\}$

For $C \in \mathcal{R}_q^{U \times 2}, \mathsf{pk} \in \mathcal{R}^2, \boldsymbol{m} \in \mathcal{R}_{2^T}^U, \boldsymbol{v}, \boldsymbol{e_0}, \boldsymbol{e_1} \in \mathcal{R}^U$

$\mathcal{P}\left((C, \mathsf{pk}), (\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e_0}, \boldsymbol{e_1})\right):$ $\mathcal{V}\left((C, \mathsf{pk})\right):$

Let $S := 3\varphi(m)\vartheta(\mathsf{flag})UVP,\ S' := (3\varphi(m)VP + 1) \cdot \vartheta(\mathsf{flag})U.$

$\tilde{\boldsymbol{m}} \xleftarrow{\$} \varrho(\mathsf{flag})^V,\ \ \tilde{\boldsymbol{v}} \leftarrow \mathcal{R}_{\leq S'}^V$ $W \xleftarrow{\$} \mathsf{Chal}(\mathsf{flag})^{V \times U}$

$\tilde{\boldsymbol{e}}_0 \leftarrow \mathcal{R}_{\leq S' \cdot (2\sigma^2+1)}^V,\ \ \tilde{\boldsymbol{e}}_1 \leftarrow \mathcal{R}_{\leq S' \cdot 2\sigma^2}^V$

$A := \mathsf{Enc}_{\mathsf{pk}}(\tilde{\boldsymbol{m}}, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{e}}_0, \tilde{\boldsymbol{e}}_1)$

$r \xleftarrow{\$} \{0,1\}^{\mathsf{Comp\_sec}},\ \ c := \mathsf{Commit}^r(A)$ $\xrightarrow{c}$

$\xleftarrow{W}$

$\hat{\boldsymbol{v}} := \tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}$

$\hat{\boldsymbol{m}} := 2^T \tilde{\boldsymbol{e}}_0 + \tilde{\boldsymbol{m}} + W \cdot (2^T \boldsymbol{e}_0 + \boldsymbol{m})$

$\hat{\boldsymbol{e}}_1 := \tilde{\boldsymbol{e}}_1 + W \cdot \boldsymbol{e}_1$

Abort if one of the following checks fails:

$$\hat{\boldsymbol{v}} \in \mathcal{R}_{\leq S}^V, \quad \hat{\boldsymbol{e}}_1 \in \mathcal{R}_{\leq S \cdot 2\sigma^2}^V$$
$$\hat{\boldsymbol{m}} \in \mathcal{R}_{\leq S \cdot 2^T(2\sigma^2+1)}^V.$$

$\xrightarrow{r, A, \hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1}$ $D := \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}, \hat{\boldsymbol{v}}, 0, \hat{\boldsymbol{e}}_1)$

Check whether:

$$\|\hat{\boldsymbol{v}}\|_\infty \leq S \qquad \|\hat{\boldsymbol{m}}\|_\infty \leq S \cdot 2^T(2\sigma^2+1)$$
$$\|\hat{\boldsymbol{e}}_1\|_\infty \leq S \cdot 2\sigma^2 \qquad D = A + W \cdot C$$
$$c = \mathsf{Commit}^r(A)$$

If $\mathsf{flag} = \mathsf{Diag}$, additionally check whether each entry of $\hat{\boldsymbol{m}} \bmod 2^T$ contains a constant polynomial.

If all checks passed, then accept. Otherwise, reject.

Fig. 8: Our ZKPoPK with rejection sampling

Theorem 5. *If $\mathit{flag} = \perp$ and $V \geq (\mathit{Snd\_sec} + 2)/\log_2|\mathit{Chal}(\mathit{flag})|$, then $\Pi^{\mathit{RS}}_{\mathit{ZKPoPK}}$ from Figure 8 (in Appendix F) is*

- *complete (Definition 4) for $\mathbb{L}$ with completeness error $1/P$,*
- *knowledge sound (Definition 5) for $\mathbb{L}_{S_{\mathit{ZKPoPK}}}$ with $S_{\mathit{ZKPoPK}} = 6\varphi(m)\vartheta(\mathit{flag})^2UVP$ and knowledge error $2^{-\mathit{Snd\_sec}}$,*
- *special honest verifier zero-knowledge (Definition 6) for $\mathbb{L}$ with statistical distance 0.*

6.2 Zero-Knowledge Proof of Secret Key Knowledge

Recall that our secure key generation for BGV (Section 3.2) requires a ZKPoSKK. The ZKPoSKK works very similar to our ZKPoPK. Recall that the ZKPoPK proves $C = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1) = (b\boldsymbol{v} + 2^T\boldsymbol{e_0} + m, a\boldsymbol{v} + 2^T\boldsymbol{e_1})$ where $(\boldsymbol{v}, 2^T\boldsymbol{e}_0 + \boldsymbol{m}, \boldsymbol{e}_1)$ is the witness. The ZKPoSKK instead proves $b = a\mathsf{sk} + 2^T e$ where (sk, e) is the witness. In both cases we have a linear equation that must be solved by the witness and the witness needs to have small coefficients. Therefore, the construction of the ZKPoSKK works analogously to the ZKPoPK. In the ZKPoSKK, the witness only consists of 2 (instead of $3U$) ring elements. Hence, the slack of the ZKPoSKK is better by a factor of $2/(3U)$.

In our triple generation, the performance of the ZKPoSKK is not as important as achieving a small slack, because the ZKPoSKK is only used during initial setup while the slack affects parameter choices for the entire protocol. For this reason, we use the challenge space $\{0, 1\}$ for the ZKPoSKK, as this eliminates another factor of $\varphi(m)^2$ from the slack (identically to what happens when instantiating our ZKPoPK with $\mathsf{flag} = \mathsf{Diag}$).

We instantiate the ZKPoSKK with rejection sampling and hence achieve a slack of $S_{\mathsf{ZKPoSKK}} = 4\varphi(m)VP$ (by taking Theorem 5 and factoring in the two above-mentioned slack improvements).

7 Evaluation

In this section, we present the results of our evaluation of Multipars and compare them against state-of-the-art SPDZ-like protocols. We start with determining suitable parameters for our protocol in Sections 7.1 and 7.2.

7.1 Noise Analysis

In order to choose suitable BGV parameters, we first need to analyze the maximum noise of a ciphertext that passes the ZKPoPK with rejection sampling (Appendix F). Let $\boldsymbol{c}$ be such a ciphertext. Note that both the adversary's key and $\boldsymbol{c}$ might be computed maliciously. Nevertheless, due to our ZKPoSKK (Section 6.2) we know that the key has the correct form for $S_{\mathsf{ZKPoSKK}} = 4\varphi(m)(\mathsf{Snd\_sec} + \log_2(\mathsf{rep}) + 2)P$. Similarly, our ZKPoPK guarantees that there exist small $v, e_0, e_1 \in \mathcal{R}$ with slack $S_{\mathsf{ZKPoPK}} = 6\varphi(m)^3UVP$ such that $\boldsymbol{c} = (bv + 2^Te_0 + m, av + 2^Te_1)$. It follows that $\mathsf{noise}(\boldsymbol{c})$ equals

$$(b - a \cdot \mathsf{sk})v + 2^T(e_0 - e_1 \cdot \mathsf{sk}) + m = 2^T(ev + e_0 - e_1 \cdot \mathsf{sk}) + m.$$

Using

$$\|ev\|_\infty \leq \varphi(m)S_{\mathsf{ZKPoSKK}}\sigma^2 S_{\mathsf{ZKPoPK}}, \quad \|e_0\|_\infty \leq S_{\mathsf{ZKPoPK}}\sigma^2,$$
$$\|e_1 \cdot \mathsf{sk}\|_\infty \leq \varphi(m)S_{\mathsf{ZKPoPK}}\sigma^2 S_{\mathsf{ZKPoSKK}}, \quad \|m\|_\infty < 2^T,$$

we obtain

$$\|\mathsf{noise}(\boldsymbol{c})\|_\infty \;<\; 2^T((2\varphi(m)S_{\mathsf{ZKPoSKK}} + 1)S_{\mathsf{ZKPoPK}}\sigma^2 + 1) \;\approx\; 2^{T+1}\varphi(m)S_{\mathsf{ZKPoSKK}}S_{\mathsf{ZKPoPK}}\sigma^2.$$

7.2 Parameter Choice

For our ZKPoPKs, we use the variants with rejection sampling with $P = 256$ as suggested in Appendix F. We limit the number of attempts to $\mathsf{rep} = 16$ which guarantees completeness error 2^{-128}.

In Π_{VOLE} (Figure 2), when drowning the noise of $\boldsymbol{c}_a b_i$ with Enc', we must choose the noise bound B at least as large as required by Theorem 1. That is,

$$B \geq 2^{\mathsf{Stat\_sec}} \left(\left\| \lfloor \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}_a b_i)/2^T \rfloor \right\|_\infty + \varphi(m) 4 \sigma^2 S_{\mathsf{ZKPoSKK}} \right)$$

where S_{ZKPoSKK} is an upper bound on the slack of e and sk. From our noise analysis above, we know

$$\left\| \lfloor \mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c}_a b_i)/2^T \rfloor \right\|_\infty + \varphi(m) 4 \sigma^2 S_{\mathsf{ZKPoSKK}} \lessapprox 2^{T+1} \varphi(m)^2 S_{\mathsf{ZKPoSKK}} S_{\mathsf{ZKPoPK}} \sigma^2.$$

Therefore we can choose B slightly larger than $2^{\mathsf{Stat\_sec}+T+1} \varphi(m)^2 S_{\mathsf{ZKPoSKK}} S_{\mathsf{ZKPoPK}} \sigma^2$ by rounding up to the next power of two. Choosing B as a power of two allows for an efficient implementation of the uniform distribution over $\mathcal{R}_{\leq B}$.

Next, we describe the choice of our modulus chain $(q_0 = p_0,\ q_1 = p_0 p_1)$. The smaller modulus $q_0 = p_0$ must be chosen large enough such that ciphertexts can be correctly decrypted after modulus switching. Assuming $\|\mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c})\|_\infty \leq q_1/3$, we can guarantee correct decryption of $\mathsf{SwitchMod}(\boldsymbol{c})$ according to Lemma 1 by choosing $p_0 > 3 \cdot 2^T \cdot \ell_1(\mathsf{sk})$. Since we have $\ell_1(\mathsf{sk}) \leq h$ for honestly generated keys we can choose p_0 as a prime with $T + \lceil \log_2(h) \rceil + 2$ bits. The larger modulus q_1 must be chosen to satisfy our aforementioned assumption $\|\mathsf{noise}_{\mathsf{sk}}(\boldsymbol{c})\|_\infty \leq q_1/3$ for all ciphertexts after drowning, so we choose q_1 with $T + \log_2(B) + 2$ bits.

Our protocol Π_{Triple} uses two different sets of BGV parameters for the Π_{VOLE} and Π_{Auth} subprotocols. The above analysis for choosing B, q_0, and q_1 also holds for Π_{Auth}, except that in B we can replace the factor 2^T by 2^k, because Π_{Auth} uses the coefficient packing where all coefficients are in $\mathbb{Z}_{2^k}$. However, the concrete values of m, T, and S_{ZKPoPK} differ. From now on, to distinguish the parameter sets, we use $\hat{\ }$ notation to refer to parameters for Π_{Auth}.

Parameters for $\boldsymbol{\Pi_{\mathsf{VOLE}}}$. The cyclotomic index m needs to be chosen large enough, i.e., in the order of thousands, such that the corresponding $\mathcal{R}$-LWE problem is hard. To work with our challenge space, we also require m to be prime (required by Lemma 7). We remark that this choice also simplifies our software implementation. We choose $m = 43691$, because it yields a relatively low $d = 34$ (hence $\delta = 15$) and therefore a relatively low overhead in the plaintext length of 16–41%, depending on (k, s). With this m, the tweaked interpolation packing has capacity $M = 21845$.

The plaintext length is $T = k + 2s + 2\delta + E$ so that we can pack a $\mathbb{Z}_{2^{k+2s}}$ message under tweaked interpolation packing and have additional E bits to manage the slackness of the ZKPoMK. E is chosen such that the ZKPoMK achieves the same soundness security level $\mathsf{Snd\_sec}$ as the ZKPoPK when using the same parameter V for both. We choose V according to Equation (3) and we compute the remaining parameters assuming $U = 4V$. That is, with our parameters, the ZKPoPKs can be amortized over up to $4V$ parallel instances of Π_{Triple}. We say *up to*, because our parameter sets can also be used in combination with smaller values of U, as this does not degrade security.[13] In our evaluation we consider the cases $U \in \{2V, 4V\}$ in order to compare against previous work [BCS19, CKL21].

Parameters for $\boldsymbol{\Pi_{\mathsf{Auth}}}$. For the cyclotomic index $\hat{m}$ we use the smallest prime that fulfills our requirement $\varphi(\hat{m}) \geq M + 3$, i.e., $\hat{m} = 21851$. The plaintext length is $\hat{T} = k + 2s$, because Π_{Auth} uses the coefficient packing instead of the tweaked interpolation packing. Contrary to Π_{VOLE}, the ZKPoPK in Π_{Auth} runs with $\mathsf{flag} = \mathsf{Diag}$ and hence has to transmit a larger number $\hat{V}$ of ciphertexts. However, this instantiation of the ZKPoPK runs only once (per pair of parties) during the one-time initialization of Π_{Auth} and therefore has a minor impact on the overall runtime. For the same reason we can choose $\hat{U} = 1$.

[13] A larger U slightly increases the slack of the ZKPoPK and hence slightly increases the minimum required ciphertext modulus. But using a smaller U with otherwise identical parameters is not a problem.

In Table 1 we present the resulting parameter sets for different combinations of k and s. To allow for a fair comparison with prior works [DPSZ12, KPR18, OSV20, CKL21], those parameter sets are constructed under the assumption that HE keys are generated honestly by a trusted dealer, i.e., $S_{\mathsf{ZKPoSKK}} = 1/(64 + \mathsf{Stat\_sec})$. In Table 2, we show how certain parameters (namely B and q_1) need to change when using our secure key generation (which introduces some slack) instead. For each combination, the top and bottom rows display the parameters for $\mathit{\Pi}_{\mathsf{VOLE}}$ and $\mathit{\Pi}_{\mathsf{Auth}}$, respectively. The security parameters $\mathsf{Snd\_sec}$ for zero-knowledge proofs and $\mathsf{Stat\_sec}$ for drowning are set to be consistent with the statistical security of SPD$\mathbb{Z}_{2^k}$ MACs, i.e., $\mathsf{sec} := \lfloor s - \log_2(s+1) \rfloor$ (see [CDE+18, Theorem 1]). All of our parameter sets provide at least 103 bits of computational security with $\sigma^2 = 10$ according to the LWE estimator [APS15].

Table 1: Our parameter sets assuming trusted key generation.

k	s	sec	m	E	$\log_2 S_{\mathsf{ZKPoPK}}$	V	B (bit)	q_1 (bit)	q_0 (bit)	Comp. Sec. [APS15]
32	32	26	43691	9	62.00	3	250	387	144	≈ 259
			21851	–	30.00	32	186	220	105	≈ 238
64	64	57	43691	11	63.47	5	381	616	242	≈ 191
			21851	–	30.98	63	314	380	201	≈ 156
128	64	57	43691	11	63.47	5	445	744	306	≈ 158
			21851	–	30.98	63	378	508	265	≈ 110

Table 2: Updates to our parameter sets when employing secure key generation.

k	s	sec	m	$\log_2 S_{\mathsf{ZKPoSKK}}$	B (bit)	q_1 (bit)	Comp. Sec. [APS15]
32	32	26	43691	30.42	289	426	≈ 273
			21851	29.42	215	249	≈ 214
64	64	57	43691	31.39	421	656	≈ 178
			21851	30.39	344	410	≈ 142
128	64	57	43691	31.39	485	784	≈ 148
			21851	30.39	408	538	≈ 103

7.3 Implementation

We implemented our LHE-based triple generation protocol as an open source software in the Rust programming language [Has23]. Our implementation covers connection establishment, secure key generation, and the triple generation itself, i.e., we provide a full implementation. Almost all optimizations presented in this work are incorporated in the software with the exception of modulus switching. Adding modulus switching will reduce our communication cost by 40%. In order to multiplex concurrent subprotocols and even multiple parallel sessions of the triple generation protocol over the same connection, we use the (TLS-secured) QUIC transport protocol [IT21] which allows us to multiplex multiple concurrent streams over UDP. For constant-time arithmetic with big integers, we employ the crypto-bigint Rust library [Aut22].

7.4 Comparison

We compare our work to prior protocols for Beaver triple generation over $\mathbb{Z}_{2^k}$ [CDE$^+$18, OSV20, CRFG20, CKL21]. Note that these works provide numbers in the two-party setting only. Our comparison covers the communication cost of the protocols as well as experimental runtime benchmarks (in the same two-party setup).

Communication Cost. In Table 3a we show the communication cost per triple. For Overdrive2k, as also noted and explained in [CKL21], we include the numbers from [CKL21] since the communication costs originally given in [OSV20] were hard to reproduce. For our protocol Multipars, we give numbers based on both of our parameter sets, i.e., with trusted and with secure key generation. Table 3a shows that Multipars outperforms MHz2k [CKL21] by around 2.2×, Overdrive2k [OSV20] by around 11×, and SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18] by 8–30×, where the exact factor depends on the parameters (k, s). Moreover, the usage of secure key generation only accounts for an overhead of around 0.5 kbit per triple produced. In setups with $N > 2$ parties, the communication costs of SPD$\mathbb{Z}_{2^k}$ and Multipars given in Table 3a need to be multiplied by $N(N-1)/2$. Mon$\mathbb{Z}_{2^k}$a only works for two parties. In contrast, Overdrive2k and MHz2k scale linearly with N, so they are better suited for large numbers of parties.

Table 3b describes the effect of the different optimizations and packings on the communication per produced triple. The values of Table 3b should be compared to Multipars (without secure KeyGen) in Table 3a. We see that disabling modulus switching leads to a significant increase in communication due to the transmission of larger ciphertexts. Namely, our modulus switching optimization is responsible for a 40% reduction in communication. In contrast, the rejection sampling and LowGear 2.0 optimizations account for a relatively small reduction. We remark that the advantage of LowGear 2.0 (i.e., avoiding sacrificing) is not as significant as in the field case since our other optimizations (modulus switching and adaptive packing) strongly improve the authentication step which is used more often in classical LowGear than in LowGear 2.0. Still, LowGear 2.0 reduces the round complexity (and therewith improves runtime performance) and software complexity, as we did not need to implement sacrificing. Finally, Table 3b shows that Multipars's adaptive packing (i.e., tweaked interpolation packing in Π_{VOLE} and coefficient packing in Π_{Auth}) is superior to using a single packing technique.

Table 3: Total amortized communication in kbit per triple produced with $N = 2$ parties.

(a) Comparison to prior work.

k	s	SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18]	Mon$\mathbb{Z}_{2^k}$a [CRFG20]	Overdrive2k [OSV20]	MHz2k-TG2k [CKL21]		Multipars		Multipars (secure KeyGen)	
					$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$
32	32	79.87	59.07	101.8	26.4	20.1	**11.4**	**9.2**	12.1	9.7
64	64	319.49	175.49	171.4	43.3	31.9	**18.2**	**14.9**	18.9	15.4
128	64	557.06	176.64	190.4	55.0	40.9	**22.3**	**18.4**	23.0	18.9

(b) Comparison of Multipars with certain optimizations disabled or different packing techniques.

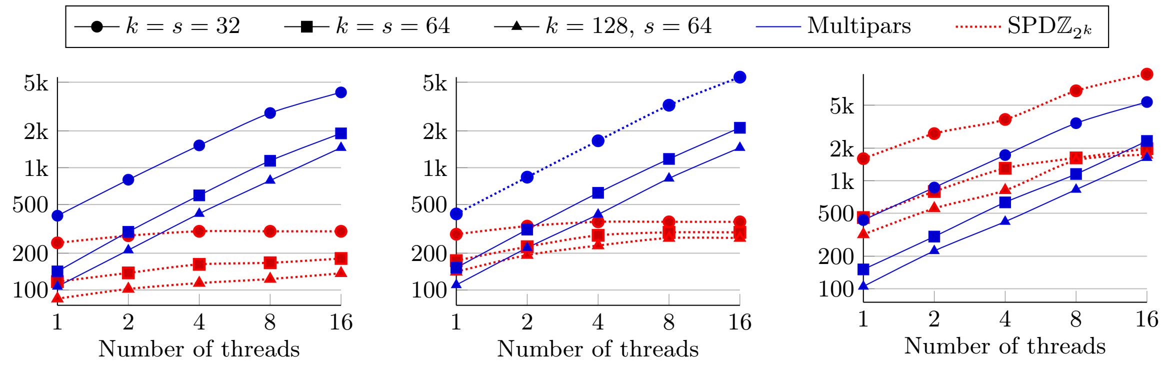
k	s	Without modulus switching		Without rejection sampling		Without LowGear 2.0		Only Overdrive2k packing		Only tweaked interpol. packing	
		$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$	$U = 2V$	$U = 4V$
32	32	17.7	15.5	12.3	9.9	12.1	9.9	20.1	16.3	12.1	9.9
64	64	27.9	24.6	19.0	15.5	19.8	16.5	35.2	29.0	19.3	16.0
128	64	33.8	29.9	23.1	19.0	24.5	20.6	44.4	36.9	23.6	19.7

Runtime Performance. Figure 9 illustrates how our protocol (implemented for $s = 64$) performs in practice. For this, we have measured the throughput of the triple generation for Multipars and SPD$\mathbb{Z}_{2^k}$ (implemented in MP-SPDZ [DEF$^+$19]) in two common [KOS16, KPR18] network settings. The other $\mathbb{Z}_{2^k}$ protocols Overdrive2k and MHz2k currently do not provide an implementation. Available numbers for Mon$\mathbb{Z}_{2^k}$a indicate a substantially slower runtime, e.g., only 19.14 triples per second in the LAN setting for $k = s = 64$. We used the Multipars variant with trusted key generation for our benchmarks. Our protocol does not employ primitives that are computationally more costly than what is used in the (not implemented) $\mathbb{Z}_{2^k}$ SHE-based protocols, while our parameters and communication cost are lower (as shown above). Thus, we expect our protocol to outperform Mon$\mathbb{Z}_{2^k}$a, Overdrive2k, and MHz2k w.r.t. runtime performance.

The results show that we can outperform SPD$\mathbb{Z}_{2^k}$ with our prototype implementation by up to 15.2×. Once modulus switching and (runtime) performance oriented optimization are added, our advantage will only increase. Our low communication overhead allows us to perform multiple parallel triple generations when the required computational resources (number of threads) are available. That is, we can utilize the computational resources better than protocols with higher communication cost, allowing us to generate more triples per second. This is apparent in the WAN setting (50 Mbps bandwidth, 100 ms RTT) where, using 16 threads, we achieve an advantage of 13.7× ($k = s = 32$), 10.5× ($k = s = 64$), or 10.6× ($k = 128$, $s = 64$), respectively. When increasing the WAN bandwidth to 500 Mbps, we have an advantage of 15.2× ($k = s = 32$), 7.2× ($k = s = 64$), or 5.5× ($k = 128$, $s = 64$). In this setting, SPD$\mathbb{Z}_{2^k}$ can neither fully utilize the CPU nor the network bandwidth, which shows that it is bottlenecked by the network round trip time. In the LAN setting (1 Gbps, 0.2 ms RTT), Multipars is at a disadvantage due to CPU bottleneck, while SPD$\mathbb{Z}_{2^k}$ is bandwidth-bottlenecked. In this setting and using 16 threads, Multipars runs at 0.55× ($k = s = 32$), 1.18× ($k = s = 64$), or 0.94× ($k = 128$, $s = 64$) the speed of SPD$\mathbb{Z}_{2^k}$.

Scaling the computational resources (increasing the number of available threads or using faster CPUs) is rather straightforward and relatively cheap compared to improving the network, especially in the cloud where exhausting a Gbit link (between parties running on different cloud providers) is more than an order of magnitude more costly than both parties running an on-demand compute instance with 16 vCPUs. Our protocol is therefore not only faster but also more suited for real-world deployment.

Overall, our new protocol Multipars has a clear runtime and bandwidth advantage against the most efficient MPC protocols [OSV20, CRFG20] over base rings $\mathbb{Z}_{2^k}$, as well as [DEF$^+$19], which is the best known implementation of a SPDZ-like protocol over $\mathbb{Z}_{2^k}$.



(a) WAN (50 Mbps, 100 ms RTT) (b) WAN (500 Mbps, 100 ms RTT) (c) LAN (1 Gbps, 0.2 ms RTT)

Fig. 9: Throughput in triples per second with $N = 2$ parties, each on a virtual server (Intel Xeon Gold 6130 CPU, 2.1 GHz) emulating the network settings between them.

Acknowledgments. This research was supported by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GS-IMTR) at the University of Stuttgart and by the CRYPTECS project founded by the German Federal Ministry of Education and Research under Grant Agreement No. 16KIS1441 and by the French National Research Agency under Grant Agreement No. ANR-20-CYAL-0006.

References

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security 2016*, pages 327–343. USENIX Association, 2016.

AL21. Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of schnorr-like arguments over lattices. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 519–548. Springer, 2021.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

Aut22. The RustCrypto Authors. Cryptographic big integers. `https://github.com/RustCrypto/crypto-bigint`, 2022.

BBC$^+$18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *CRYPTO 2018*, pages 669–699. Springer, 2018.

BCI$^+$13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *Theory of Cryptography*, pages 315–333. Springer, 2013.

BCS19. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using topgear in overdrive: A more efficient zkpok for SPDZ. In *SAC 2019*, pages 274–302. Springer, 2019.

BDLN16. Carsten Baum, Ivan Damgård, Kasper Green Larsen, and Michael Nielsen. How to Prove Knowledge of Small Secrets. In *CRYPTO 2016*, pages 478–498. Springer, 2016.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO '91*, pages 420–432. Springer, 1991.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, 2012.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS 2008*, pages 192–206. Springer, 2008.

Bro92. W. Brown. *Matrices over Commutative Rings.* Chapman & Hall Pure and Applied Mathematics. Taylor & Francis, 1992.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145. IEEE, 2001.

CDE$^+$18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd$\mathcal{F}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *CRYPTO 2018*, pages 769–798. Springer, 2018.

CF01. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

CKL21. Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. Mhz2k: MPC from HE over $\mathcal{F}_{2^k}$ with new packing, simpler reshare, and better ZKP. In *CRYPTO 2021*, pages 426–456. Springer, 2021.

CKR$^+$20. Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In *ASIACRYPT 2020*, pages 31–59. Springer, 2020.

CRFG20. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. Mon$\mathcal{F}_{2^k}$a: Fast maliciously secure two party computation on $\mathcal{F}_{2^k}$. In *PKC 2020*, pages 357–386. Springer, 2020.

CS10. Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *FC 2010*, pages 35–50. Springer, 2010.

DEF$^+$19. Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *SP 2019*, pages 1102–1120. IEEE, 2019.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS 2013*, pages 1–18. Springer, 2013.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, pages 643–662. Springer, 2012.

DS13. Ivan Damgård and Alessandra Scafuro. Unconditionally secure and universally composable commitments from physical assumptions. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2013.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*, pages 186–194. Springer, 1986.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT 2012*, pages 465–482. Springer, 2012.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, pages 850–867. Springer, 2012.

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

Has23. Sebastian Hasler. Multipars. `https://github.com/haslersn/multipars`, 2023.

HLHD22. Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *USENIX Security 2022*, pages 809–826. USENIX Association, 2022.

IT21. Jana Iyengar (ed.) and Martin Thomson (ed.). QUIC: A UDP-based multiplexed and secure transport. RFC 9000, RFC Editor, 2021.

JVC18. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018*, pages 1651–1669. USENIX Association, 2018.

Kel20. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS 2020*, pages 1575–1590. ACM, 2020.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS 2016*, pages 830–842. ACM, 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT 2018*, pages 158–189. Springer, 2018.

Lin17. Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT 2013*, pages 35–54. Springer, 2013.

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT 2009*, pages 598–616. Springer, 2009.

MZ17. Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *SP 2017*, pages 19–38. IEEE, 2017.

OSV20. Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over F_{2^k} from somewhat homomorphic encryption. In *CT-RSA 2020*, pages 254–283. Springer, 2020.

RRK$^+$20. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS 2020*, pages 325–342. ACM, 2020.

RRKK23. Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Küsters. Overdrive lowgear 2.0: Reduced-bandwidth mpc without sacrifice. In *ACM ASIA Conference on Computer and Communications Security (ASIA CCS '23), July 10–14, 2023, Melbourne, VIC, Australia*, 2023.

TKTW21. Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *SP 2021*. IEEE, 2021.

A Security of our Triple Generation

In this section, we prove the security of our truncation protocol Π_{Trunc} (Figure 5) and our triple generation protocol Π_{Triple} (Figure 4) in the standalone model with rewinding black-box simulator [Gol04, Lin17]. We assume that each party P_i has her own LHE key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and additionally the public keys of all other parties pk_j for $j \neq i$.[14]

The standalone model assumes that there is only a single instance of the protocol and no other protocol running concurrently. In this case, it ensures that a protocol provides the same guarantees as an ideal functionality $\mathcal{F}$ that models the exact security properties that we want. For instance, $\mathcal{F}_{\mathsf{Triple}}$ models that correct shares of authenticated Beaver triples are produced and corrupted parties learn nothing about the shares of honest parties; or the protocol aborts.

We consider a ppt. adversary $\mathcal{A}$ that corrupts some static subset $A \subsetneq [N]$ of parties and fully controls all corrupted parties. In particular, $\mathcal{A}$ can abort the protocol (or functionalities) at any time. To keep the functionalities and protocols concise, we usually avoid to state this abort option explicitly.

Our security proofs follow the standard real-ideal (world) paradigm. Namely, we construct for each adversary $\mathcal{A}$ a simulator $\mathcal{S}$. Then the output of an ideal execution between $\mathcal{S}$ and the functionality $\mathcal{F}$ should be (computationally) indistinguishable from the output of a real execution between $\mathcal{A}$ and the honest parties. More formally, we use the following setup:

Ideal Execution. In the ideal model, we consider an execution between a simulator $\mathcal{S}$ on auxiliary input z and an ideal functionality $\mathcal{F}$ where $\mathcal{S}$ provides the corrupted parties' inputs and obtains the corrupted parties' outputs. For security parameter λ, we denote by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z),A}(\lambda)$ the tuple consisting of the simulator's output and the honest parties' outputs as specified by $\mathcal{F}$. Note that the simulator's output doesn't need to match the corrupted parties' outputs, which the simulator obtained from $\mathcal{F}$. Instead, the simulator can output anything computable in polynomial time from the information obtained.

Real Execution. In the real model, we consider an execution between the adversary $\mathcal{A}$ on auxiliary input z and the honest parties following a protocol Π. Here $\mathcal{A}$ acts on behalf of the corrupted parties. For security parameter λ, we denote by $\mathsf{REAL}_{\Pi,\mathcal{A}(z),A}(\lambda)$ the tuple consisting of the adversary's output and the honest parties' outputs.

Definition 2 (Standalone Security). *A protocol Π realizes a functionality $\mathcal{F}$ in the standalone model if, for every ppt. adversary $\mathcal{A}$ corrupting $A \subsetneq [N]$, there exists a ppt. simulator $\mathcal{S}$ such that $\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z),A}(\lambda)\}_{z,\lambda}$ is computationally indistinguishable from $\{\mathsf{REAL}_{\Pi,\mathcal{A}(z),A}(\lambda)\}_{z,\lambda}$ where $z \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$. In case where a fixed simulator $\mathcal{S}$ satisfies the definition for all ppt. adversaries $\mathcal{A}$ (given that $\mathcal{S}$ has rewinding black-box access to $\mathcal{A}$) we call $\mathcal{S}$ a* rewinding black-box simulator.

A.1 Security of Π_{Trunc}

In Figure 10 we present the functionality $\mathcal{F}_{\mathsf{Trunc}}$ for the truncation of Beaver triples. We will show that Π_{Trunc} is a realization of $\mathcal{F}_{\mathsf{Trunc}}$ and later use this fact in order to replace Π_{Trunc} by $\mathcal{F}_{\mathsf{Trunc}}$ in our analysis of Π_{Triple}.

We admit that the functionality $\mathcal{F}_{\mathsf{Trunc}}$ is relatively low level, i.e., it closely resembles the steps of Π_{Trunc}. The sole difference is that in Π_{Trunc} commitments ensure that the adversary has to choose $([\gamma_{\hat{\mathbf{a}}}]_i, [\hat{\mathbf{c}}]_i, [\gamma_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ without knowing the honest parties' respective values. $\mathcal{F}_{\mathsf{Trunc}}$ has this property by definition.

We get the following security result:

Theorem 6. *We assume that Π_{Trunc} is initiated with a computationally hiding, computationally binding, extractable, and equivocal commitment scheme. Then for any number of parties $N \in \mathbb{N}$, Π_{Trunc} realizes $\mathcal{F}_{\mathsf{Trunc}}$ in the standalone model with black-box simulator.*

[14] We assume the same security guarantees on $(\mathsf{sk}_i, \mathsf{pk}_i)$ as in [KPR18, RRKK23], e.g., that the public key is sufficiently random for all parties.

Functionality $\mathcal{F}_{\mathsf{Trunc}}$

Let A be the set of corrupted parties.

1. For $i \notin A$, receive $[\alpha]_i, [\![\overline{\boldsymbol{a}}]\!]_i, [\![\boldsymbol{b}]\!]_i, [\![\overline{\boldsymbol{c}}]\!]_i$ from P_i.
2. For $i \notin A$, send $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ to the adversary.
3. For $i \in A$, receive $([\overline{\boldsymbol{a}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ from the adversary.
4. For $i \notin A$, define $([\boldsymbol{\gamma}_{\hat{\mathbf{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ as in an honest execution of Π_{Trunc} and send it to the adversary.
5. Define $\boldsymbol{\Sigma\gamma}_{\hat{\mathbf{a}}}$, $\boldsymbol{\Sigma}\hat{\boldsymbol{c}}$, and $\boldsymbol{\Sigma\gamma}_{\hat{\mathbf{c}}}$ as in an honest execution of Π_{Trunc} and check that
$$\boldsymbol{\Sigma\gamma}_{\hat{\mathbf{a}}} \equiv \boldsymbol{\Sigma}\hat{\boldsymbol{c}} \equiv \boldsymbol{\Sigma\gamma}_{\hat{\mathbf{c}}} \equiv \mathbf{0} \pmod{2^s}$$
If any check fails, abort.
6. For $i \notin A$, compute $[\![\boldsymbol{a}]\!]_i$ and $[\![\boldsymbol{c}]\!]_i$ as in an honest execution of Π_{Trunc}.
7. Output $([\![\boldsymbol{a}]\!]_i, [\![\boldsymbol{c}]\!])_i$ to each honest party P_i, $i \notin A$.

Fig. 10: Functionality for truncation of Beaver triples

The theorem assumes a commitment scheme that is computationally hiding, computationally binding, extractable, and equivocal. The *extractability* property means that the simulator is able to extract the value from any commitment sent by the adversary, even if the adversary has not (yet) opened the commitment. The *equivocality* property means that the simulator can create fake-commitments which are indistinguishable from "real" commitments and can be opened by the simulator to any value. The computational hiding and binding properties provide the security guarantees, i.e., that an adversary cannot reconstruct a value from its commitment and that he cannot construct two values with the same commitment (in polynomial time). We remark that commitment schemes with all four properties exist [DS13] and that these assumptions are regularly used in simulation-based security proofs [CF01].

Simulator $\mathcal{S}_{\mathsf{Trunc}}(z)$

For $i \notin A$, the simulator receives $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ from $\mathcal{F}_{\mathsf{Trunc}}$. The simulator then starts the adversary $\mathcal{A}$ on auxiliary input z and emulates Π_{Trunc} between the honest parties and $\mathcal{A}$:

(1) In step (1) of the emulation, the simulator reveals the values $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ for $i \notin A$ (which were received from $\mathcal{F}_{\mathsf{Trunc}}$) and $\mathcal{A}$ reveals $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ for $i \in A$. Define $\boldsymbol{\Sigma a} := \sum_j ([\overline{\boldsymbol{a}}]_j \bmod 2^s)$.

(2-3) The simulator skips step (2) and proceeds with step (3) where the simulator (on behalf of the honest parties) sends fake-commitments to $\mathcal{A}$ and in return receives commitments from $\mathcal{A}$. The simulator extracts the received commitments, obtaining $([\boldsymbol{\gamma}_{\hat{\mathbf{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ for $i \in A$.

For $i \in A$, the simulator sends $([\overline{\boldsymbol{a}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ to $\mathcal{F}_{\mathsf{Trunc}}$. For $i \notin A$, the simulator receives $([\boldsymbol{\gamma}_{\hat{\mathbf{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\mathbf{c}}}]_i) \bmod 2^s$ from $\mathcal{F}_{\mathsf{Trunc}}$.

(4) The simulator opens the honest parties' fake-commitments to the values received from $\mathcal{F}_{\mathsf{Trunc}}$.

(5) The simulator continues the emulation until after step (5), where it stops the emulation.

If the emulation aborted, the simulator sends abort to $\mathcal{F}_{\mathsf{Trunc}}$. In any case, the simulator runs $\mathcal{A}$ to completion and outputs whatever $\mathcal{A}$ outputs.

Fig. 11: Simulator for Π_{Trunc} realizing $\mathcal{F}_{\mathsf{Trunc}}$

Proof. In Figure 11 we present our simulator $\mathcal{S}_{\mathsf{Trunc}}$ (with black-box access to the real adversary $\mathcal{A}$) who acts on behalf of the corrupted parties towards $\mathcal{F}_{\mathsf{Trunc}}$. We need to prove that the output tuple of $\mathcal{S}_{\mathsf{Trunc}}$ and the honest parties (as specified by $\mathcal{F}_{\mathsf{Trunc}}$) in the ideal execution is computationally indistinguishable from the output tuple of $\mathcal{A}$ and the honest parties in the real execution.

With $\mathcal{S}_{\mathsf{Trunc}}$, the ideal execution constitutes an exact simulation of the real execution, except for the use of fake-commitments. However, the fake-commitments are indistinguishable from the "real" commitments in the real execution by the choice of our commitment scheme. More precisely, binding and extractability ensure that the adversary cannot change the values in the commitments after the commitments were exchanged (up to negligible probability) and that the simulator extracts exactly these values. Hiding ensures that all commitments look random. Finally, equivocality ensures that the simulator can match the values in the honest parties' commitments to be consistent with the output of the functionality. Hence, the output tuples are computationally indistinguishable, as required. □

A.2 Security of Π_{Triple}

In Figure 12 we present the functionality $\mathcal{F}_{\mathsf{Triple}}$ for Beaver triple generation. In our security analysis, we only consider a single invocation of **Init** followed by a single invocation of **Triple**. Though, the analysis can easily be extended to a polynomial number of parallel or sequential invocations of **Triple** (using only a single initialization in the beginning of the protocol). As above, we avoid to note explicitly in each step of Figure 12 that the adversary can abort any time.

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

This functionality generates shares of the global MAC key and then provides an interface to generate M authenticated Beaver triples. Let A be the set of corrupted parties.

Init:

1. Receive $[\alpha]_i \in \mathbb{Z}_{2^{k+s}}$ for $i \in A$ from the adversary.
2. Sample $[\alpha]_i \overset{\$}{\leftarrow} \mathbb{Z}_{2^s}$ for $i \notin A$.
3. Store $\alpha := \sum_{i\in[N]}[\alpha]_i \bmod 2^{k+s}$.

Triple:

1. Receive $([\boldsymbol{a}]_i, [\boldsymbol{b}]_i, [\boldsymbol{c}]_i) \in \mathbb{Z}_{2^{k+s}}^M$ and $([\boldsymbol{\gamma_a}]_i, [\boldsymbol{\gamma_b}]_i, [\boldsymbol{\gamma_c}]_i) \in \mathbb{Z}_{2^{k+s}}^M$ for $i \in A$ from the adversary.
2. Sample honest shares $[\boldsymbol{a}]_i, [\boldsymbol{b}]_i \overset{\$}{\leftarrow} \mathbb{Z}_{2^k}^M$ for $i \notin A$.
3. For $i \notin A$, sample $[\boldsymbol{c}]_i \in \mathbb{Z}_{2^{k+s}}^M$ uniformly at random subject to $\boldsymbol{c} \equiv_k \boldsymbol{a} \odot \boldsymbol{b}$, and $[\boldsymbol{\gamma_a}]_i, [\boldsymbol{\gamma_b}]_i, [\boldsymbol{\gamma_c}]_i \in \mathbb{Z}_{2^{k+s}}^M$ uniformly at random subject to, for each $\boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$, $\boldsymbol{\gamma_x} \equiv_{k+s} \alpha\boldsymbol{x}$.
4. Output $[\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!]$.

Fig. 12: Functionality for Beaver triple generation

Functionality $\mathcal{F}_{\mathsf{ZKPoK}}$

1. Let $\mathcal{A}$ be the sender and $\mathcal{B}$ be the receiver. Receive $\boldsymbol{x} \in \mathbb{Z}_{2^{k+2s}}^M$ from $\mathcal{A}$ or from the adversary if $\mathcal{A}$ is corrupted.
2. Send $\mathsf{Enc}_{\mathsf{pk}_{\mathcal{B}}}(\mathsf{pack}(\boldsymbol{x}))$ to $\mathcal{B}$ where the encryption randomness is chosen with additional slack S_{ZKPoPK} if $\mathcal{A}$ is corrupted.

(a) Functionality for ZKPoPK and ZKPoMK.

Functionality $\mathcal{F}_{\mathsf{ZKPoK}}^{\mathsf{Diag}}$

1. Let $\mathcal{A}$ be the sender and $\mathcal{B}$ be the receiver. Receive $x \in \mathbb{Z}_{2^k}$ from $\mathcal{A}$ or from the adversary if $\mathcal{A}$ is corrupted.
2. Send $\mathsf{Enc}_{\mathsf{pk}_{\mathcal{B}}}(\mathsf{coeffPack}((x, 0, \ldots)))$ to $\mathcal{B}$ where the encryption randomness is chosen with additional slack S_{ZKPoPK} if $\mathcal{A}$ is corrupted.

(b) Functionality for ZKPoPK of diagonal plaintexts.

Fig. 13: Functionalities for zero-knowledge proofs.

Now we proceed with the theorem for the security of Π_{Triple} and its formal proof.

Theorem 3. *For any number of parties $N \in \mathbb{N}$, Π_{Triple} realizes $\mathcal{F}_{\mathsf{Triple}}$ in the standalone model with rewinding black-box simulator.*

Proof. As Π_{Trunc} is a realization of $\mathcal{F}_{\mathsf{Trunc}}$, we replace the subprotocol Π_{Trunc} by an invocation of $\mathcal{F}_{\mathsf{Trunc}}$ in the following analysis of Π_{Triple}. In Figure 14 we present our simulator $\mathcal{S}_{\mathsf{Triple}}$ (with rewinding black-box access to the real adversary $\mathcal{A}$) who acts on behalf of the corrupted parties towards $\mathcal{F}_{\mathsf{Triple}}$. We need to prove that the output tuple of $\mathcal{S}_{\mathsf{Triple}}$ and the honest parties (as specified by $\mathcal{F}_{\mathsf{Triple}}$) in the ideal execution is computationally indistinguishable from the output tuple of $\mathcal{A}$ and the honest parties in the real execution. In both executions, $\mathcal{A}$ runs on auxiliary input z.

Simulator $\mathcal{S}_{\mathsf{Triple}}(z)$

Init: The simulator starts the adversary $\mathcal{A}$ on auxiliary input z. It then emulates the **Init** subprotocol of Π_{Triple} between the honest parties, $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$, and $\mathcal{A}$. Since the simulator emulates $\mathcal{F}^{\mathsf{Diag}}_{\mathsf{ZKPoK}}$, it obtains all plaintexts $[\alpha]_i$ and can compute $\alpha := \sum_{i\in[N]}[\alpha]_i \bmod 2^{k+s}$. The simulator sends $[\alpha]_i$ for $i \in A$ to $\mathcal{F}_{\mathsf{Triple}}$.

Triple: The simulator emulates the **Triple** subprotocol of Π_{Triple} between the honest parties, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ZKPoK}}$, $\mathcal{F}_{\mathsf{Trunc}}$, and $\mathcal{A}$. If the protocol aborts, the simulator sends abort to $\mathcal{F}_{\mathsf{Triple}}$, runs $\mathcal{A}$ to completion and outputs whatever $\mathcal{A}$ outputs. Otherwise: For $i \notin A$, let $[\![\boldsymbol{a}]\!]_i$, $[\![\boldsymbol{b}]\!]_i$, and $[\![\boldsymbol{c}]\!]_i$ denote the shares computed on behalf of the honest parties in the emulation. By rewinding the adversary until before step (5), the simulator obtains enough pairs $(\boldsymbol{t}, [y]_i)$ in order to reconstruct $[\boldsymbol{a}||\boldsymbol{b}||\boldsymbol{c}]_i$ for $i \in A$. The simulator chooses arbitrary MACs $[\boldsymbol{\gamma_a}]_i, [\boldsymbol{\gamma_b}]_i, [\boldsymbol{\gamma_c}]_i$ for $i \in A$ such that, for each $\boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$, $\sum_i [\boldsymbol{\gamma_x}]_i = \alpha \cdot \boldsymbol{x}$. The simulator sends $([\boldsymbol{a}]_i, [\boldsymbol{b}]_i, [\boldsymbol{c}]_i)$ and $([\boldsymbol{\gamma_a}]_i, [\boldsymbol{\gamma_b}]_i, [\boldsymbol{\gamma_c}]_i)$ for $i \in A$ to $\mathcal{F}_{\mathsf{Triple}}$. The simulator then runs $\mathcal{A}$ to completion and outputs whatever $\mathcal{A}$ outputs.

Fig. 14: Simulator for Π_{Triple} realizing $\mathcal{F}_{\mathsf{Triple}}$

Note that the ideal execution aborts with the same probability as the real execution and, in this case, the ideal execution constitutes an exact simulation of the real execution, because there are no inputs and outputs. Thus, it remains to prove that above-mentioned output tuples are indistinguishable conditioned on the event that the protocol does not abort.

Let $\mathcal{A}$'s shares of $\overline{\boldsymbol{a}}$, $\boldsymbol{b}$, and $\overline{\boldsymbol{c}}$ used in the MAC check be fixed. Under this condition, we first show that in both the ideal and real execution, the honest parties' outputs are distributed (computationally close to) uniformly at random subject to $\boldsymbol{c} \equiv_k \boldsymbol{a} \odot \boldsymbol{b}$ and, for each $\boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$, $\boldsymbol{\gamma_x} \equiv_t \alpha\boldsymbol{x}$ where $t := k + s$. In the ideal execution, this statement is obvious from the way the honest parties' outputs are sampled by $\mathcal{F}_{\mathsf{Triple}}$. Now we turn to the real execution: According to our assumption, the MAC check passes, so we have $\boldsymbol{c} \equiv_k \boldsymbol{a} \odot \boldsymbol{b}$ and $\forall \boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\} : \alpha\boldsymbol{x} \equiv_t \boldsymbol{\gamma_x}$ with high probability by Lemma 4 below. We also require that the honest parties' outputs in the real execution are distributed uniformly at random subject to these conditions. Note that, in Π_{Triple}, the honest parties sample their shares of $\boldsymbol{a}$ and $\boldsymbol{b}$ uniformly at random, but this uniform distribution is not necessarily preserved as we condition on the event that the protocol doesn't abort. Indeed, a selective failure attack is possible where some information about $\overline{\boldsymbol{a}}$ (i.e., the value before truncation) is leaked. Nonetheless, we can show that $\boldsymbol{a}$ is still distributed uniformly at random: By Lemma 5 below, there exists an event E with $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1 - 2^{-s}$ such that, conditioned on $\neg E$, $\boldsymbol{a}$ is distributed uniformly at random. We ignore the event "$E \wedge \neg\mathsf{abort}$" as it happens with statistically small probability. In the remaining cases, either the protocol aborts (event "abort") or $\boldsymbol{a}$ is distributed uniformly at random (event "$\neg E \wedge \neg\mathsf{abort}$").

As the simulator emulates the protocol, we know that $\mathcal{A}$'s shares of $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are distributed identically between the ideal and real execution. It follows that the honest parties' outputs are computationally indistinguishable between the ideal and real execution, even without fixing $\mathcal{A}$'s shares.

Next, we inductively show that, when adding $\mathcal{A}$'s view-so-far (i.e., $\mathcal{A}$'s view of the transcript up to a certain step in the protocol Π_{Triple}) to the output tuple, that tuple is computationally indistinguishable between the ideal and real execution. After the last step, this concludes our proof, as we can assume without loss of generality that $\mathcal{A}$ outputs its own view.

Init In the initialization step, the received ciphertexts $\boldsymbol{c}_{[\alpha]_i}$ are computationally indistinguishable by the semantic security of the encryption scheme.

(2) During authentication (step (2) of **Triple**), the received ciphertexts $\boldsymbol{c}'_{d^{(i,j)}}$ have been drowned and decrypt to a uniformly random plaintext (as the plaintexts have been masked by uniformly random $e^{(j,i)} \in \mathcal{R}_{2^{k+s}}$). It follows by Theorem 1 that they are computationally indistinguishable between the ideal and real execution.

(3) In Π_{VOLE} (step (3) of **Triple**), ciphertexts $\boldsymbol{c}_a$ and $(\boldsymbol{c}'_{d_i})_i$ get exchanged. For $\boldsymbol{c}_a$, we can argue again with the semantic security of the encryption scheme. The returned ciphertexts $(\boldsymbol{c}'_{d_i})_i$ have been drowned, but they do not decrypt to a uniformly random plaintext: Only the upper E bits, which have been masked by m_i, are distributed uniformly at random. However, the ZKPoMK (modelled as $\mathcal{F}_{\mathsf{ZKPoK}}$, cf. Figure 13a) guarantees that $\boldsymbol{c}_a$ encodes some correct packing of a message from $\mathbb{Z}^M_{2^{k+2s}}$. The addition of $\mathsf{pack}'(\boldsymbol{e}_i)$ ensures that the plaintext of $(\boldsymbol{c}'_{d_i})_i$ is distributed uniformly at random over the set of all correct packings. guarantees

(5) Recall that in our analysis we replace Π_{Trunc} (in step (5) of **Triple**) by $\mathcal{F}_{\mathsf{Trunc}}$. Here, the adversary receives the values $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ and $([\boldsymbol{\gamma}_{\hat{\boldsymbol{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\boldsymbol{c}}}]_i) \bmod 2^s$ for $i \notin A$ from $\mathcal{F}_{\mathsf{Trunc}}$. The value $[\overline{\boldsymbol{a}}]_i \bmod 2^s$ is distributed uniformly at random as it was sampled in step (1) of Triple. The values $([\boldsymbol{\gamma}_{\hat{\boldsymbol{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\boldsymbol{c}}}]_i) \bmod 2^s$ are distributed uniformly at random subject to having a certain sum (mod 2^s) over $i \notin A$, and that sum is indistinguishable between the ideal and real execution, as we show below.

(6) In step (6), the vector $\boldsymbol{t}$ obtained from $\mathcal{F}_{\mathsf{Rand}}$ is distributed uniformly at random.

(8) In step (8), each received share $[y]_i$ is distributed uniformly at random over $\mathbb{Z}_{2^{k+s}}$ (as it has been masked with $[m]_i + 2^k[r]_i$). In $\Pi_{\mathsf{SingleCheck}}$ the opened values are distributed uniformly at random over $\mathbb{Z}_{2^{k+s}}$ subject to their sum being 0.

Hence, the view of $\mathcal{A}$ is computationally indistinguishable between both executions. It follows that the output of $\mathcal{S}_{\mathsf{Triple}}$ in the ideal execution is computationally indistinguishable from the output of $\mathcal{A}$ in the real execution.

As mentioned above, for this implication to hold, we still need to show that the values $([\boldsymbol{\gamma}_{\hat{\boldsymbol{a}}}]_i, [\hat{\boldsymbol{c}}]_i, [\boldsymbol{\gamma}_{\hat{\boldsymbol{c}}}]_i) \bmod 2^s$ are distributed uniformly at random subject to having a certain sum (mod 2^s) over $i \notin A$. We explicitly show this fact only for $([\hat{\boldsymbol{c}}]_i \bmod 2^s)_{i \notin A}$ and the same argument can be applied to $[\boldsymbol{\gamma}_{\hat{\boldsymbol{a}}}]_i$ and $[\boldsymbol{\gamma}_{\hat{\boldsymbol{c}}}]_i$, too. We conditioned on the event that the protocol does not abort, so we have $\sum_i [\hat{\boldsymbol{c}}]_i \equiv_s \boldsymbol{\Sigma}\hat{\boldsymbol{c}} \equiv_s 0$. It follows that $\hat{\boldsymbol{\sigma}} := \sum_{i \notin A} [\hat{\boldsymbol{c}}]_i \bmod 2^s \equiv_s -\sum_{i \in A} [\hat{\boldsymbol{c}}]_i$, where we observe that the right-hand side only depends on $\mathcal{A}$'s view-so-far at the point where $\mathcal{F}_{\mathsf{Trunc}}$ did not yet send honest parties' shares of $\hat{\boldsymbol{c}}$ (mod 2^s) to $\mathcal{A}$.[15] We already showed that the view-so-far up to that point is computationally indistinguishable between the ideal and real execution, so the same holds for $\hat{\boldsymbol{\sigma}}$. We need to show that the individual summands $([\hat{\boldsymbol{c}}]_i \bmod 2^s)_{i \notin A}$ are distributed uniformly at random subject to having the sum $\hat{\boldsymbol{\sigma}}$ (mod 2^s). Let P_h be an arbitrary honest party. Then it suffices to show that $([\hat{\boldsymbol{c}}]_i \bmod 2^s)_{i \notin A \cup \{h\}}$ is distributed uniformly at random, because (given $\hat{\boldsymbol{\sigma}}$) these values uniquely determine $[\hat{\boldsymbol{c}}]_h \bmod 2^s$. Consider the masks $\boldsymbol{e}_1^{i,h}$ (notation from Π_{Triple}, cf. Figure 4) which P_h sampled in the Π_{VOLE} subprotocol. These masks affect the honest parties' shares $[\boldsymbol{c}]_i$, but they do not affect the overall sum $\sum_{i \notin A} [\boldsymbol{c}]_i$, so these masks are also independent of $\hat{\boldsymbol{\sigma}}$. It follows that, under our condition that the protocol doesn't abort, these masks are still distributed uniformly at random. Therefore, these masks induce a uniform distribution of $([\hat{\boldsymbol{c}}]_i \bmod 2^s)_{i \notin A \cup \{h\}}$, as required. □

Proof of Correct Multiplication. Here we show that the adversary cannot tamper with its shares in any way inducing $\boldsymbol{c} \not\equiv_k \boldsymbol{a} \odot \boldsymbol{b}$. This is similar to the "Proof of Correct Multiplication" in [RRKK23], but here we need to additionally handle the truncation subprotocol Π_{Trunc}. As we perform the MAC check before the truncation, we initially only have a guarantee about the shares $[\![\overline{\boldsymbol{a}}]\!], [\![\overline{\boldsymbol{c}}]\!]$. This guarantee is captured by the following lemma (cf. [CDE+18, Claim 1]).

[15] This is the place in the proof where it's important that Π_{Trunc} uses commitments, which we modelled in $\mathcal{F}_{\mathsf{Trunc}}$ by $\mathcal{A}$ needing to send its shares of $\hat{\boldsymbol{c}}$ (mod 2^s) before receiving the honest parties' shares. Without this, a rushing adversary could choose its shares of $\hat{\boldsymbol{c}}$ (mod 2^s) dependent on honest parties' shares, so we would not be able to deduce that $\hat{\boldsymbol{\sigma}}$ depends only on $\mathcal{A}$'s previous view-so-far.

Lemma 2. *Let $t = k+s$ and let $\mathcal{A}$ be a ppt. adversary attacking Π_{Triple}. Given rewinding black box access to $\mathcal{A}$, one can extract malicious parties' shares of $[\alpha]$, $[\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$ such that, with probability $\geq 1-2^{-s}$, either the MAC check fails or*

$$\boldsymbol{\gamma}_{\overline{\boldsymbol{a}}} \equiv_{t+s} \alpha\overline{\boldsymbol{a}} \tag{4}$$

$$\boldsymbol{\gamma}_{\boldsymbol{b}} \equiv_{t+s} \alpha\boldsymbol{b} \tag{5}$$

$$\boldsymbol{\gamma}_{\overline{\boldsymbol{c}}} \equiv_{t+s} \alpha\overline{\boldsymbol{c}}. \tag{6}$$

Now we proceed with the proof of correct multiplication, first for the shares before truncation (Lemma 3) and then for the shares after truncation (Lemma 4).

Lemma 3. *Let $t, [\alpha], [\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$ as in Lemma 2. Then, with probability $\geq 1-2^{-s}$, either the MAC check fails or $\overline{\boldsymbol{a}} \odot \boldsymbol{b} \equiv_t \overline{\boldsymbol{c}}$.*

Proof. For simplicity, we only consider a single malicious party and identify it by $\mathcal{A}$, but the argument can easily be extended to an arbitrary number of malicious parties.

Let $\tilde{\alpha}$ and $\tilde{\boldsymbol{b}}$ be the accumulated values supplied to Π_{Auth} and let $\tilde{\boldsymbol{a}}$ be the accumulated "$\boldsymbol{a}$"-value supplied to Π_{VOLE}. Let $\tilde{\boldsymbol{\alpha}}$ be the vector of length M containing $\tilde{\alpha}$ in each entry. $\mathcal{A}$ can tamper with the shares of $(\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{b}}, \boldsymbol{\gamma}_{\tilde{\boldsymbol{b}}})$ supplied to Π_{VOLE}, namely by effectively supplying shares of $(\tilde{\boldsymbol{\alpha}}+\boldsymbol{\delta}_{\boldsymbol{\alpha}}, \tilde{\boldsymbol{b}}+\boldsymbol{\delta}_{\boldsymbol{b}}, \tilde{\alpha}\tilde{\boldsymbol{b}}+\boldsymbol{\delta}_{\boldsymbol{\gamma_b}})$ instead (for some $\boldsymbol{\delta}$-values chosen by $\mathcal{A}$). Afterwards, the parties have shares of $[\tilde{\alpha}]$ and

$$[\![\tilde{\boldsymbol{a}}]\!] = ([\tilde{\boldsymbol{a}}], [\tilde{\boldsymbol{a}} \odot (\tilde{\boldsymbol{\alpha}}+\boldsymbol{\delta}_{\boldsymbol{\alpha}})]) \qquad [\![\tilde{\boldsymbol{b}}]\!] = ([\tilde{\boldsymbol{b}}], [\tilde{\alpha}\tilde{\boldsymbol{b}}]) \qquad [\![\tilde{\boldsymbol{c}}]\!] = ([\tilde{\boldsymbol{a}} \odot (\tilde{\boldsymbol{b}}+\boldsymbol{\delta}_{\boldsymbol{b}})], [\tilde{\boldsymbol{a}} \odot (\tilde{\alpha}\tilde{\boldsymbol{b}}+\boldsymbol{\delta}_{\boldsymbol{\gamma_b}})]).$$

By Lemma 2, with probability $\geq 1-2^{-s}$ and assuming the MAC check passes, we can extract $\mathcal{A}$'s shares of $[\alpha], [\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$ conforming to Equations (4) to (6), with honest parties' shares given by $[\alpha]_i = [\tilde{\alpha}]_i$, $[\![\overline{\boldsymbol{a}}]\!]_i = [\![\tilde{\boldsymbol{a}}]\!]_i$, $[\![\boldsymbol{b}]\!]_i = [\![\tilde{\boldsymbol{b}}]\!]_i$, and $[\![\overline{\boldsymbol{c}}]\!]_i = [\![\tilde{\boldsymbol{c}}]\!]_i$ for $i \notin A$. Let

$$\Delta_\alpha := [\alpha]_A - [\tilde{\alpha}]_A \qquad \begin{aligned} \boldsymbol{\Delta}_{\boldsymbol{a}} &:= [\overline{\boldsymbol{a}}]_A - [\tilde{\boldsymbol{a}}]_A & \boldsymbol{\Delta}_{\boldsymbol{\gamma_a}} &:= [\boldsymbol{\gamma}_{\overline{\boldsymbol{a}}}]_A - [\boldsymbol{\gamma}_{\tilde{\boldsymbol{a}}}]_A \\ \boldsymbol{\Delta}_{\boldsymbol{b}} &:= [\boldsymbol{b}]_A - [\tilde{\boldsymbol{b}}]_A & \boldsymbol{\Delta}_{\boldsymbol{\gamma_b}} &:= [\boldsymbol{\gamma}_{\boldsymbol{b}}]_A - [\boldsymbol{\gamma}_{\tilde{\boldsymbol{b}}}]_A \\ \boldsymbol{\Delta}_{\boldsymbol{c}} &:= [\overline{\boldsymbol{c}}]_a - [\tilde{\boldsymbol{c}}]_A & \boldsymbol{\Delta}_{\boldsymbol{\gamma_c}} &:= [\boldsymbol{\gamma}_{\overline{\boldsymbol{c}}}]_A - [\boldsymbol{\gamma}_{\tilde{\boldsymbol{c}}}]_A. \end{aligned}$$

Substitute this into Equation (4) and rearrange, so we obtain

$$\alpha\boldsymbol{\Delta}_{\boldsymbol{a}} \equiv_{t+s} \boldsymbol{\Delta}_{\boldsymbol{\gamma_a}} - (\Delta_\alpha - \boldsymbol{\delta}_{\boldsymbol{\alpha}}) \odot \tilde{\boldsymbol{a}}.$$

As the adversary has no information on α, it follows that $\boldsymbol{\Delta}_{\boldsymbol{a}} \equiv_t \boldsymbol{0}$ with overwhelming probability. Therefore $\tilde{\boldsymbol{a}} \equiv_t \tilde{\boldsymbol{a}} + \boldsymbol{\Delta}_{\boldsymbol{a}} = \overline{\boldsymbol{a}}$. By doing the same with Equation (5), we obtain

$$\alpha\boldsymbol{\Delta}_{\boldsymbol{b}} \equiv_{t+s} \boldsymbol{\Delta}_{\boldsymbol{\gamma_b}} - \Delta_\alpha\tilde{\boldsymbol{b}},$$

so it follows that $\boldsymbol{\Delta}_{\boldsymbol{b}} \equiv_t \boldsymbol{0}$ with overwhelming probability. By doing the same with Equation (6), we obtain

$$\alpha(\tilde{\boldsymbol{a}} \odot \boldsymbol{\delta}_{\boldsymbol{b}} + \boldsymbol{\Delta}_{\boldsymbol{c}}) \equiv_{t+s} \tilde{\boldsymbol{a}} \odot \boldsymbol{\delta}_{\boldsymbol{\gamma_b}} + \boldsymbol{\Delta}_{\boldsymbol{\gamma_c}} - \Delta_\alpha\tilde{\boldsymbol{a}} \odot \tilde{\boldsymbol{b}})$$

so it follows that $\tilde{\boldsymbol{a}} \odot \boldsymbol{\delta}_{\boldsymbol{b}} + \boldsymbol{\Delta}_{\boldsymbol{c}} \equiv_t \boldsymbol{0}$ with overwhelming probability. We conclude that

$$\overline{\boldsymbol{c}} = \tilde{\boldsymbol{c}} + \boldsymbol{\Delta}_{\boldsymbol{c}} = \tilde{\boldsymbol{a}} \odot (\tilde{\boldsymbol{b}}+\boldsymbol{\delta}_{\boldsymbol{b}}) + \boldsymbol{\Delta}_{\boldsymbol{c}} \equiv_t \tilde{\boldsymbol{a}} \odot \tilde{\boldsymbol{b}} \equiv_t \overline{\boldsymbol{a}} \odot \boldsymbol{b}.$$

□

Lemma 4. *Let $t = k+s$ and let $\mathcal{A}$ be a ppt. adversary attacking Π_{Triple}. Given rewinding black box access to $\mathcal{A}$, one can extract malicious parties' shares of $[\alpha], [\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!]$ such that, with probability $\geq 1-2^{-s}$, either the MAC check fails or $\boldsymbol{a} \odot \boldsymbol{b} \equiv_k \boldsymbol{c}$ and $\forall \boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\} : \alpha\boldsymbol{x} \equiv_t \boldsymbol{\gamma}_{\boldsymbol{x}}$.*

Proof. Let $[\alpha], [\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$ as in Lemma 2. Assume we are in the case where the MAC check in Π_{Triple} passes and (with probability $> 1 - 2^{-s}$) above-mentioned shares satisfy Equations (4) to (6) and, by Lemma 3, $\overline{\boldsymbol{a}} \odot \boldsymbol{b} \equiv_t \overline{\boldsymbol{c}}$. For honest parties' shares (i.e., $i \notin A$) we have

$$\begin{aligned}
[\boldsymbol{a}]_i &= \lfloor [\overline{\boldsymbol{a}}]_i / 2^s \rfloor \\
[\boldsymbol{\gamma_a}]_i &= \lfloor [\boldsymbol{\gamma_{\hat{a}}}]_i / 2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{a}}} / 2^s &&= \lfloor ([\boldsymbol{\gamma_{\overline{a}}}]_i - \boldsymbol{\Sigma a} \cdot [\alpha]_i)/2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{a}}}/2^s \\
[\boldsymbol{c}]_i &= \lfloor [\hat{\boldsymbol{c}}]_i / 2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma}\hat{\boldsymbol{c}} / 2^s &&= \lfloor ([\overline{\boldsymbol{c}}]_i - \boldsymbol{\Sigma a} \odot [\boldsymbol{b}]_i)/2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma}\hat{\boldsymbol{c}}/2^s \\
[\boldsymbol{\gamma_c}]_i &= \lfloor [\boldsymbol{\gamma_{\hat{c}}}]_i / 2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{c}}} / 2^s &&= \lfloor ([\boldsymbol{\gamma_{\overline{c}}}]_i - \boldsymbol{\Sigma a} \odot [\boldsymbol{\gamma_b}]_i)/2^s \rfloor + 1_{i=0} \cdot \boldsymbol{\Sigma\gamma_{\hat{c}}}/2^s.
\end{aligned}$$

We can compute the corrupted parties' shares in the same way, taking $[\boldsymbol{a}]_i \bmod 2^s$, $[\boldsymbol{\gamma_{\hat{a}}}]_i \bmod 2^s$, $[\hat{\boldsymbol{c}}]_i \bmod 2^s$ and $[\boldsymbol{\gamma_{\hat{c}}}]_i \bmod 2^s$ from the transcript and computing all $\boldsymbol{\Sigma}$-values based on those. Then we have

$$\begin{aligned}
\boldsymbol{a} &= (\overline{\boldsymbol{a}} - \boldsymbol{\Sigma a})/2^s \\
\boldsymbol{\gamma_a} &= (\boldsymbol{\gamma_{\overline{a}}} - \boldsymbol{\Sigma a} \cdot \alpha)/2^s \equiv_t (\alpha\overline{\boldsymbol{a}} - \boldsymbol{\Sigma a} \cdot \alpha)/2^s = \alpha\boldsymbol{a} \\
\boldsymbol{c} &= (\overline{\boldsymbol{c}} - \boldsymbol{\Sigma a} \odot \boldsymbol{b})/2^s \equiv_k (\overline{\boldsymbol{a}} \odot \boldsymbol{b} - \boldsymbol{\Sigma a} \odot \boldsymbol{b})/2^s = \boldsymbol{a} \odot \boldsymbol{b} \\
\boldsymbol{\gamma_c} &= (\boldsymbol{\gamma_{\overline{c}}} - \boldsymbol{\Sigma a} \odot \boldsymbol{\gamma_b})/2^s \equiv_t (\alpha\overline{\boldsymbol{c}} - \boldsymbol{\Sigma a} \odot \alpha\boldsymbol{b})/2^s = \alpha\boldsymbol{c}.
\end{aligned}$$

□

Selective Failure Attacks. In LHE-based protocols, selective failure attacks might be possible whenever an adversary can manipulate a ciphertext in a certain way such that the fact that the manipulated ciphertext decrypts to zero reveals something about the original plaintext. In our protocol, this can happen in two places:

1. In Π_{VOLE} there is a ciphertext $\boldsymbol{c}_a$ and a malicious party could add some derived ciphertext $\boldsymbol{c}'$ to the returned ciphertexts.
2. In Π_{Auth} there is a ciphertext $\boldsymbol{c}_{[\alpha]_i}$ and a malicious party could add some derived ciphertext $\boldsymbol{c}'$ to the ciphertext returned to P_i.

In both cases, shares will be altered only if $\mathsf{unpack}(\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}')) \neq \boldsymbol{0}$, leading to a MAC check failure. Hence, a passing MAC check reveals to the adversary that $\mathsf{unpack}(\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}')) = \boldsymbol{0}$, which could reveal something about the original plaintext, i.e., $[\overline{\boldsymbol{a}}]_i$ or $[\alpha]_i$.

Here we show that such selective failure attacks on Π_{VOLE} cannot reveal anything about the truncated shares $[\boldsymbol{a}]_i$, which is what we care about.

Lemma 5. *Let $\mathcal{A}$ be a ppt. adversary attacking Π_{Triple}. There exists an event E such that (i) if $\mathsf{Pr}[E] \neq 0$, then $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1 - 2^{-s}$ and (ii) if $\mathsf{Pr}[E] \neq 1$, then, conditioned on $\neg E$, honest parties' shares $([\boldsymbol{a}]_i)_{i \notin A}$ are distributed computationally indistinguishable from the uniform distribution and independent of $([\overline{\boldsymbol{a}}]_i \bmod 2^s)_{i \notin A}$.*

Proof. In this proof we ignore the modulus switching which is performed in Π_{VOLE} as an optimization. Adding modulus switching cannot degrade the security of the protocol, as the modulus-switched ciphertext cannot reveal more information than the original ciphertext.

Observe that in Π_{Triple} we have

$$\begin{aligned}
\sum\nolimits_{j \neq i} \boldsymbol{d}_0^{(i,j)} &= [\boldsymbol{\gamma_{\overline{a}}}]_i - [\overline{\boldsymbol{a}}]_i \cdot [\alpha]_i - \sum\nolimits_{j \neq i} \boldsymbol{e}_0^{(j,i)} \\
\sum\nolimits_{j \neq i} \boldsymbol{d}_1^{(i,j)} &= [\overline{\boldsymbol{c}}]_i - [\overline{\boldsymbol{a}}]_i \odot [\boldsymbol{b}]_i - \sum\nolimits_{j \neq i} \boldsymbol{e}_1^{(j,i)} \\
\sum\nolimits_{j \neq i} \boldsymbol{d}_2^{(i,j)} &= [\boldsymbol{\gamma_{\overline{c}}}]_i - [\overline{\boldsymbol{a}}]_i \odot [\boldsymbol{\gamma_b}]_i - \sum\nolimits_{j \neq i} \boldsymbol{e}_2^{(j,i)}.
\end{aligned}$$

Hence, given (a ciphertext of) all shares, we can compute a ciphertext $\boldsymbol{c}_l$ of $\sum_{j \neq i} \boldsymbol{d}_l^{(i,j)}$ for $l \in \{0, 1, 2\}$.

Let P_h ($h \notin A$) be an arbitrary honest party and, for any ppt. distinguisher D, let $\mathcal{A}^D$ be the following attacker on the 2^s-enhanced CPA security game (cf. Figure 1):

1. Receive pk and $\boldsymbol{c}$ from the challenger.
2. Emulate Π_{Triple} between $\mathcal{A}$ and the honest parties until after step (7). Replace P_h's public key used in Π_{VOLE} by pk and replace P_h's ciphertext $\boldsymbol{c}_a$ by $\boldsymbol{c}_a := \boldsymbol{c}$. Use Lemma 2 to extract malicious parties' shares of $[\alpha], [\![\overline{\boldsymbol{a}}]\!], [\![\boldsymbol{b}]\!], [\![\overline{\boldsymbol{c}}]\!]$.
3. Note that $\boldsymbol{c}_a$ is a ciphertext of $[\overline{\boldsymbol{a}}]_h$. Based on the extracted shares, honest parties' shares, and $\boldsymbol{c}_a$, compute ciphertexts $\boldsymbol{c}_l$ for $l \in \{0,1,2\}$ as described above.
4. Let $\tilde{\boldsymbol{c}}_l$ for $l \in \{0,1,2\}$ be the actual accumulated ciphertexts returned to P_h in the emulation.
5. Send $\hat{\boldsymbol{c}}_l := \tilde{\boldsymbol{c}}_l - \boldsymbol{c}_l$ for $l \in \{0,1,2\}$ to the challenger for zero-checking. If the challenger aborts on any of those queries, then abort.
6. Receive $\boldsymbol{m}_b$ from the challenger.
7. Output $D(\boldsymbol{m}_b)$.

We assume linear targeted malleability of BGV, so Theorem 2 states that there exists an event E such that, (i) if $\mathsf{Pr}[E] \neq 0$, then $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1 - 2^{-s}$ and (ii) if $\mathsf{Pr}[E] \neq 1$, then, conditioned on $\neg E$, $\mathcal{A}^D$ has a negligible advantage in the security game. As this holds for all ppt. distinguishers D, it follows that $\boldsymbol{m}_0$ $(= \overline{\boldsymbol{a}})$ is computationally indistinguishable from $\boldsymbol{m}_1$. Therefore, conditioned on $\neg E$, $[\boldsymbol{a}]_h = \lfloor [\overline{\boldsymbol{a}}]_h / 2^s \rfloor = \lfloor \boldsymbol{m}_0 / 2^s \rfloor$ is computationally indistinguishable from $\tilde{\boldsymbol{m}}$, which is uniformly distributed and independent of $[\overline{\boldsymbol{a}}]_h \bmod 2^s = \boldsymbol{m}_0 \bmod 2^s$. Furthermore, observe that the MAC check in the emulation passes only if $\forall l \in \{0,1,2\} : \mathsf{unpack}(\mathsf{Dec}_{\mathsf{sk}}(\hat{\boldsymbol{c}}_l)) = \boldsymbol{0}$, i.e., only if the challenger does not abort in the security game. Hence the property $\mathsf{Pr}[\mathsf{abort} \mid E] \geq 1 - 2^{-s}$ carries over from the security game to the emulation.

The same argument can be repeated independently for each honest party P_h $(h \notin A)$. □

A similar selective failure attack on Π_{Auth} might reveal information about $[\alpha]_i$ and hence about the secret MAC key α. In the following section about 2^s-enhanced CPA security, we argue that the success probability of learning ℓ bits of $[\alpha]_i$ is only $2^{-\ell}$. With the same probability, an adversary can learn ℓ bits of α during the MPC online phase, and nonetheless, the online phase is secure (since an adversary needs to predict all bits of α in order to successfully cheat). As it doesn't matter whether an adversary learns this information in the offline phase or online phase, it follows that this selective failure attack does not break security.

B Properties of BGV

This appendix contains technical definitions and proofs related to BGV.

Linear Targeted Malleability of BGV. As in [KPR18] the BGV scheme over $\mathbb{Z}_{2^t}$ comes with linear targeted malleability:

Definition 3. *An encryption scheme has the linear targeted malleability property if for any polynomial-size adversary $\mathcal{A}$ and any plaintext generator $\mathcal{M}$ there is a polynomial-size simulator $\mathcal{S}$ 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:*

$$\left\{ \begin{array}{l|r} \mathsf{pk}, & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ s, a_1, \ldots, a_m & s, a_1, \ldots, a_m \leftarrow \mathcal{M}(\mathsf{pk}) \\ & \boldsymbol{c}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a_i), \forall 1 \leq i \leq m \\ \mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}'_j), \forall 1 \leq j \leq k & (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_k) \leftarrow \mathcal{A}(\mathsf{pk}, c_1, \ldots, c_m, z) \\ & \textit{with } \boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_k \textit{ in the domain of the decryption algorithm} \end{array} \right\}$$

and

$$\left\{ \begin{array}{l|r} \mathsf{pk}, & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ s, a_1, \ldots, a_m & s, a_1, \ldots, a_m \leftarrow \mathcal{M}(\mathsf{pk}) \\ & (B, \boldsymbol{b}) \leftarrow \mathcal{S}(\mathsf{pk}, z) \\ a'_j, \forall 1 \leq j \leq k & (a'_1, \ldots, a'_k)^T \leftarrow B(a_1, \ldots, a_m)^T + \boldsymbol{b} \end{array} \right\}$$

for a $k \times m$ matrix B, a k-dimensional vector $\boldsymbol{b}$, and s some arbitrary string (possibly correlated with the plaintexts).

The argument in the $\mathbb{Z}_{2^T}$ case is identical (without any non-notational adaptions). Namely, for non-affine operations on $\mathsf{Enc}_{\mathsf{pk}}(m)$, e.g., for higher order polynomial evaluations, the necessary key switching material (e.g., as in [DKL$^+$13]) is not available in a purely LHE-scheme. Additionally, if $\mathcal{A}$ himself encrypts any self-chosen values, $\mathcal{S}$ can do the same. Finally, if $\mathcal{A}$ uses elements of the ciphertext space not derived by encryption, e.g., $(0, 1)$ or $(0, 2^\ell)$ for some $\ell \in \mathbb{N}$, $\mathcal{S}$ can sample a (dummy) secret key sk' uniformly at random and decrypt using sk'. As in [KPR18] we then have to increase the entropy of the secret key by $\mathsf{Stat\_sec}$ bits to account for the leakage of up to $\mathsf{Stat\_sec}$ bits of information about sk with probability $2^{-\mathsf{Stat\_sec}}$. For further details we refer to [KPR18].

As a result of linear targeted malleability, we may assume (as in [KPR18]) that an adversary against our security game in Figure 1 can only query whether an affine function $f_{B,\boldsymbol{b}}(\boldsymbol{m}) = B\boldsymbol{m} - \boldsymbol{b}$ for some $a, b \in \mathbb{Z}_T$ is $\mathbf{0}$ or not (in which case the protocol aborts). The resulting game is included as Figure 15.

$\tilde{\mathcal{G}}_{2^s\text{-CPA+}}$

1. $\mathcal{C}$ samples $\boldsymbol{m} \xleftarrow{\$} \mathbb{Z}_{2^t}^M$. $\mathcal{C}$ samples $b \xleftarrow{\$} \{0, 1\}$.
2. For $j \in \mathsf{poly}(s)$:
   (a) $\mathcal{A}$ sends some $B \in \mathbb{Z}_{2^t}^{K \times M}$ to $\mathcal{C}$ and $\boldsymbol{b} \in \mathbb{Z}_{2^t}^K$ for some $K \in \mathbb{N}$.
   (b) If $B\boldsymbol{m} + b = \mathbf{0}$, $\mathcal{C}$ sends OK to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sends abort to $\mathcal{A}$ and aborts and sets $b' = 0$.
3. If $b = 0$, $\mathcal{C}$ sends $\boldsymbol{m}_0 := \boldsymbol{m}$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ samples $\tilde{\boldsymbol{m}} \xleftarrow{\$} \mathbb{Z}_{2^{t-s}}^M$ and sends $\boldsymbol{m}_1 := 2^s \tilde{\boldsymbol{m}} + (\boldsymbol{m} \bmod 2^s)$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

The advantage of $\mathcal{A}$ is defined as in Figure 1.

Fig. 15: Transformed 2^s-enhanced CPA security game.

Composition of Affine Queries. We further add a technical lemma that shows that replacing multiple affine queries in the proof of 2^s-enhanced security of BGV by their composition generally leads to lower leakage, i.e., the elementary divisor exponent β_1 is larger for the composition. Of course the composition then also has a lower abort probability.

Lemma 6. *Let $B \in \mathbb{Z}_{2^k}^{K \times M}$ be a matrix with Smith normal form $B = SDR$ for invertible matrices S, R, $0 \le r \le \min\{K, M\}$, and $D = (\delta_{ij} 2^{\beta_j} \delta_{j<r})_{0 \le i < K, 0 \le j < M}$ with $\beta_1 \le \cdots \le \beta_r$. Analogously let $B' \in \mathbb{Z}_{2^k}^{M \times L}$ with Smith normal form $B' = S'D'R'$ for invertible matrices S', R', $0 \le r' \le \min\{M, L\}$, and $D' = (\delta_{jk} 2^{\beta'_k} \delta_{k<r'})_{0 \le j < M, 0 \le k < L}$ with $\beta'_1 \le \cdots \le \beta'_{r'}$ for $1 \le j < r'$. Then there are invertible matrices S'', R'', and $0 \le r'' \le \min\{K, L\}$ such that $B'' := BB' = S''D''R''$ for $D = (\delta_{ik} 2^{\beta''_k} \delta_{k<r''})_{0 \le i < K, 0 \le k < L}$ and $\beta''_1 \le \cdots \le \beta''_{r''}$ for $1 \le j < r''$. We then have $\beta_1 + \beta'_1 \le \beta''_1$.*

Proof. By definition the 2^{β_1} are the first elementary divisors defined as the gcd over all matrix entries, i.e. $2^{\beta_1} = \gcd(b_{ij} : 0 \le i < K, 0 \le j < M)$ for $B = (b_{ij})_{0 \le i < K, 0 \le j < M}$. Analogously for $B' = (b'_{jk})_{0 \le j < M, 0 \le k < L}$ and $2^{\beta'_1}$. But obviously $\gcd(b_{ij} : 0 \le i < K, 0 \le j < M) \gcd(b'_{ij} : 0 \le i < K, 0 \le j < M)$ divides each $\sum_{j=0}^{M-1} b_{ij} b'_{jk}$, i.e., each entry of B''. Thus we also have $\gcd(b_{ij} : 0 \le i < K, 0 \le j < M) \gcd(b'_{ij} : 0 \le i < K, 0 \le j < M) \mid \gcd(b''_{ik} : 0 \le i < K, 0 \le k < L)$ for $BB' = B'' = (b''_{ik})_{0 \le i < K, 0 \le k < L}$. This implies $2^{\beta_1} 2^{\beta'_1} \mid 2^{\beta''_1}$ and thus $\beta_1 + \beta'_1 \le \beta''_2$ as claimed. □

C The Challenge Space

In our zero-knowledge proof we have to show the *knowledge soundness* property (cf. Definition 5). This proof requires us to extract a witness, which can be reduced to the task of finding a short inverse of

$w - w'$ for any two distinct challenges $w, w' \in \mathsf{Chal}$. We therefore choose the challenge space $\mathsf{Chal} := \{w_i \mid 0 \leq i < m\}$ for

$$w_i(X) := \sum_{k=0}^{i-1} X^k \in \mathcal{R} \tag{7}$$

from [AL21]. Lemma 7 shows that Chal comes in fact with the two necessary properties, i.e., invertibility of $w - w'$ and a sufficiently small norm bound.

Remark. The challenge space from [BCS19, CKL21] contained elements w, w' such that $w - w'$ was not invertible, but only had a scaled inverse, i.e., an element r with $r(w - w') = m$. As a result this original ZKPoPK could only prove a statement about a multiple of the original ciphertext.

Lemma 7. *Let $\mathcal{R} = \mathbb{Z}[X]/\Phi_m(X)$ with m prime. For all $i \neq j$ it holds true that $w_j - w_i \in \mathcal{R}$ is invertible and $\left\|(w_j - w_i)^{-1}\right\|_\infty = 1$.*

Proof. Below, we prove the statement for $0 \leq i < j < m$, while the remaining cases follow from $w_i - w_j = (-1) \cdot (w_j - w_i)$. Let l be the multiplicative inverse of $j - i \bmod m$. Hence there exists $n \in \mathbb{N}$ such that $(j - i)l = nm + 1$. Let $h(X) := X^{m-i} w_l(X^{j-i})$. Using, $w_a(X) w_b(X^a) = w_{ab}(X)$, $X^m = 1$, and $w_m(X) = \Phi_m(X) = 0$, we obtain

$$\begin{aligned}(w_j(X) - w_i(X))h(X) &= \sum_{k=i}^{j-1} X^k X^{m-i} w_l(X^{j-i}) = w_{j-i}(X) w_l(X^{j-i}) = w_{(j-i)l}(X) \\ &= X^{nm} + w_{nm}(X) = 1 + w_n(X^m) w_m(X) = 1,\end{aligned}$$

hence $(w_j - w_i)^{-1} = h$. For computing the norm of h, first observe that $w_l(X)$ in power basis only has coefficients in $\{0, 1\}$. Therefore, the same holds for $w_l(X^{j-i})$, as m doesn't divide $j - i$. By Lemma 8 (see below) it follows that $\|h\|_\infty \leq 1$. As $h \neq 0$, it must be that $\|h\|_\infty = 1$. □

The proof of Lemma 7 relies on the following Lemma 8:

Lemma 8. *Let $l \leq 0 \leq u \in \mathbb{Z}$ and let $f(X) = \sum_{k=1}^{m-1} a_k X^k$ for some coefficients $a_k \in [l, u]$. Then, for all $i \in \mathbb{N}$, it holds true that $X^i f(X)$ in the power basis has coefficients in $[l - u, u]$.*

Proof. As $X^i = X^{i \bmod m}$, assume without loss of generality that $0 < i \leq m$. Define $a_0 := 0$. We conclude that

$$\begin{aligned}X^i f(X) &= \sum_{k=0}^{m-1} a_k X^{k+i \bmod m} = \sum_{k=0}^{m-1} a_{k-i \bmod m} X^k \\ &= a_{m-i} + \sum_{k=1}^{m-1} a_{k-i \bmod m} X^k \overset{(*)}{=} \sum_{k=1}^{m-1} (a_{k-i \bmod m} - a_{m-i}) X^k\end{aligned}$$

where $(*)$ follows by subtracting $a_{m-i}\Phi_m(X) = 0 \in \mathcal{R}$. □

In [BCS19, CKL21], the multiplication of a polynomial with a challenge is an operation linear in m, since their challenges are simple monomials X^j. While our challenges $w_i \in \mathsf{Chal}$ are more complex, multiplication with a challenge however stays similarly efficient. In fact, in Algorithm 1 we present a simple algorithm that multiplies a polynomial by $w_i \in \mathsf{Chal}$ and requires only $m - 1$ additions and subtractions, independent of i.

Note that we can also run Algorithm 1 on $f \in \mathcal{R}_q$. In that case, all arithmetic in the algorithm is modulo q.

Lemma 9. *On input $f \in \mathcal{R}$ and $0 \leq i < m$, Algorithm 1 correctly outputs $f w_i$.*

Algorithm 1 Multiplication by a challenge.

```
Input: $f \in \mathcal{R}$ in power basis, challenge index $0 \leq i < m$
Output: res $= fw_i$ in power basis
  res $:= 0 \in \mathcal{R}$
  sum $:= 0 \in \mathbb{Z}$
  for $j = 1, \ldots, m-1$ do
    sum $:=$ sum $+ f.$coeff$[X^j] - f.$coeff$[X^{j-i \bmod m}]$
    res.coeff$[X^j] :=$ sum
  end for
  return res
```

Proof. Let $f(X) = \sum_{j=1}^{m-1} a_j X^j$ for some coefficients $a_i \in \mathbb{Z}$. Define $a_0 := 0$. We obtain

$$f(X)w_i(X) = \sum_{k=0}^{i-1}\sum_{j=0}^{m-1} a_j X^{j+k \bmod m} = \sum_{k=0}^{i-1}\sum_{j=0}^{m-1} a_{j-k \bmod m} X^j.$$

Define $\hat{b}_j := \sum_{k=0}^{i-1} a_{j-k \bmod m}$ and $b_j := \hat{b}_j - \hat{b}_0$. Using $\Phi_m(X) = 0$, we obtain

$$f(X)w_i(X) = \sum_{j=0}^{m-1} \hat{b}_j X^j = \left(\sum_{j=0}^{m-1} \hat{b}_j X^j\right) - \hat{b}_0 \Phi_m(X) = \sum_{j=0}^{m-1} b_j X^j.$$

Observe that $b_0 = 0$ and, for $1 \leq j < m$,

$$b_j \;=\; \sum_{k=0}^{i-1} a_{j-k \bmod m} - \hat{b}_0 \;=\; \sum_{k=0}^{i-1} a_{j-1-k \bmod m} - \hat{b}_0 + a_j - a_{j-i \bmod m} \;=\; b_{j-1} + a_j - a_{j-i \bmod m}.$$

Inductively it follows that each res.coeff$[X^j]$ in Algorithm 1 is computed correctly. □

D Security of Zero-Knowledge Proofs

We analyze the security of our proof system based on the following security definitions.

Definition 4 (Completeness). *A proof system $(\mathcal{P}, \mathcal{V})$ is* complete *for the relation R with* completeness error $\rho \in [0, 1]$ *if for all $(x, w) \in R$ it holds true that $Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x)\rangle = 0] \leq \rho$.*

Definition 5 (Knowledge Soundness). *A proof system $(\mathcal{P}, \mathcal{V})$ is* knowledge sound *for the relation R with* soundness error $\kappa \in [0, 1]$ *if there exists a polynomial $q(\cdot)$ and probabilistic extractor $\mathcal{E}$ such that for all deterministic adversaries $\mathcal{P}^*$ and $x \in \{0, 1\}^*$ with*

$$\varepsilon := Pr[\langle \mathcal{P}^*(x), \mathcal{V}(x)\rangle = 1] > \kappa$$

it holds true that $\mathcal{E}^{\mathcal{P}^*(x)}(x)$ outputs a witness $w \in R(x)$ in expected time $\leq q(|x|)/(\varepsilon - \kappa)$.

Definition 6 (Special Honest-Verifier Zero-Knowledge). *A proof system $(\mathcal{P}, \mathcal{V})$ is* special honest-verifier zero-knowledge *for the relation R with statistical distance δ if there exists a probabilistic simulator $\mathcal{S}$ such that for all $(x, w) \in R$ and randomness α it holds true that $\mathcal{S}(x, \alpha) \approx_\delta Transcr\langle \mathcal{P}(x, w), \mathcal{V}^\alpha(x)\rangle$.*

Here $Transcr\langle \mathcal{P}(x, w), \mathcal{V}^\alpha(x)\rangle$ is the random variable over the random coins of $\mathcal{P}$ that outputs the transcript of $\langle \mathcal{P}(x, w), \mathcal{V}^\alpha(x)\rangle$.

E Zero-Knowledge Proof of Plaintext Knowledge

In this appendix, we restate and prove the security theorem of our ZKPoPK (without rejection sampling). The proof is presented for the case $\mathsf{flag} = \bot$. The other case $\mathsf{flag} = \mathsf{Diag}$ can be proven analogously.

Theorem 4. *If $\mathsf{flag} = \bot$, $V \geq (\mathsf{Snd\_sec} + 2)/\log_2|\mathsf{Chal}(\mathsf{flag})|$, and $Z \geq 2^{\mathsf{ZK\_sec}}$ then Π_{ZKPoPK} from Figure 7 (in Appendix E) is*

- *complete (Definition 4) for $\mathbb{L}$ with negligible completeness error,*
- *knowledge sound (Definition 5) for $\mathbb{L}_{S_{\mathsf{ZKPoPK}}}$ with $S_{\mathsf{ZKPoPK}} = 2\vartheta(\mathsf{flag})^2 UZ$ and knowledge error $2^{-\mathsf{Snd\_sec}}$,*
- *special honest verifier zero-knowledge (Definition 6) for $\mathbb{L}$ with statistical distance $2^{-\mathsf{ZK\_sec}}$.*

Completeness. For proving completeness, both parties are assumed to behave honestly. We show that, with overwhelming probability, all checks by the verifier pass when running Π_{ZKPoPK} on input $((C, \mathsf{pk}), (\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)) \in \mathbb{L}$.

As the rows of W consist of U elements from $\mathsf{Chal}(\bot) = \{w_i \mid 0 \leq i < m\}$, it follows that

$$\|W \cdot \boldsymbol{v}\|_\infty \leq \varphi(m)U \|\boldsymbol{v}\|_\infty = \varphi(m)U.$$

As $\tilde{\boldsymbol{v}}$ is sampled uniformly at random with an exponentially larger bound $\varphi(m)UZ$, it follows that $\hat{\boldsymbol{v}} = \tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}$ exceeds this bound only with negligible probability. Hence, the verifier's check on $\|\hat{\boldsymbol{v}}\|_\infty$ passes with overwhelming probability. The same can similarly be shown for $\|\hat{\boldsymbol{m}}\|_\infty$ and $\|\hat{\boldsymbol{e}}_1\|_\infty$.

It remains to show that the verifier's last check $D = A + W \cdot C$ passes. Using $(a, b) := \mathsf{pk}$, we obtain

$$\begin{aligned} D &= \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}, \hat{\boldsymbol{v}}, 0, \hat{\boldsymbol{e}}_1) = \big(b\hat{\boldsymbol{v}} + 2^T \cdot 0 + \hat{\boldsymbol{m}}, a\hat{\boldsymbol{v}} + 2^T \hat{\boldsymbol{e}}_1\big) \\ &= \big(b(\tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}) + 2^T \tilde{\boldsymbol{e}}_0 + \tilde{\boldsymbol{m}} + W \cdot (2^T \boldsymbol{e}_0 + \boldsymbol{m}),\ a(\tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}) + 2^T(\tilde{\boldsymbol{e}}_1 + W \cdot \boldsymbol{e}_1)\big) \\ &= \big(b\tilde{\boldsymbol{v}} + 2^T \tilde{\boldsymbol{e}}_0 + \tilde{\boldsymbol{m}}, a\tilde{\boldsymbol{v}} + 2^T \tilde{\boldsymbol{e}}_1) + W \cdot (b\boldsymbol{v} + 2^T \boldsymbol{e}_0 + \boldsymbol{m}, a\boldsymbol{v} + 2^T \boldsymbol{e}_1\big) \\ &= \mathsf{Enc}_{\mathsf{pk}}(\tilde{\boldsymbol{m}}, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{e}}_0, \tilde{\boldsymbol{e}}_1) + W \cdot \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}, \boldsymbol{v}, \boldsymbol{e}_0, \boldsymbol{e}_1)\ =\ A + W \cdot C. \end{aligned}$$

Knowledge Soundness. On input $x = (C, \mathsf{pk})$ and given oracle access to a deterministic prover $\mathcal{P}^*$ that succeeds on x with probability $\varepsilon > 2^{-\mathsf{Snd\_sec}}$, the extractor has to output a witness $w \in \mathbb{L}_{S_{\mathsf{ZKPoPK}}}(x)$.

Let $j \in [U]$. Because $V \geq (\mathsf{Snd\_sec} + 2)/\log_2|\mathsf{Chal}|$, we can follow the method from [BBC+18, Lemma 3], to extract two accepting transcripts $(A, W, (\hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1))$ and $(A, W', (\hat{\boldsymbol{v}}', \hat{\boldsymbol{m}}', \hat{\boldsymbol{e}}_1'))$ where W and W' are equal except for the j-th column. This takes expected time $\mathsf{poly}(|x|)/\varepsilon$.

As the transcripts are accepting, we have

$$\begin{aligned} D &:= \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}, \hat{\boldsymbol{v}}, 0, \hat{\boldsymbol{e}}_1) = A + W \cdot C \\ D' &:= \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}', \hat{\boldsymbol{v}}', 0, \hat{\boldsymbol{e}}_1') = A + W' \cdot C. \end{aligned}$$

Subtracting both equations, we obtain

$$D^{(i,\cdot)} - D'^{(i,\cdot)} = (W^{(i,j)} - W'^{(i,j)}) \cdot C^{(j,\cdot)}$$

for $i \in [V]$. Now choose the row index i such that $W^{(i,j)} \neq W'^{(i,j)}$ and let h be the inverse of $W^{(i,j)} - W'^{(i,j)} \in \mathcal{R}$. By Lemma 7, h exists and $\|h\|_\infty = 1$. It follows that

$$\begin{aligned} C^{j,\cdot} &= h \cdot (D^{(i,\cdot)} - D'^{(i,\cdot)}) \\ &= h \cdot \big(\mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}^{(i)}, \hat{\boldsymbol{v}}^{(i)}, 0, \hat{\boldsymbol{e}}_1^{(i)}) - \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}'^{(i)}, \hat{\boldsymbol{v}}'^{(i)}, 0, \hat{\boldsymbol{e}}_1'^{(i)})\big) \\ &= \mathsf{Enc}_{\mathsf{pk}}\big(h \cdot (\hat{\boldsymbol{m}}^{(i)} - \hat{\boldsymbol{m}}'^{(i)}), h \cdot (\hat{\boldsymbol{v}}^{(i)} - \hat{\boldsymbol{v}}'^{(i)}), 0, h \cdot (\hat{\boldsymbol{e}}_1^{(i)} - \hat{\boldsymbol{e}}_1'^{(i)})\big) \\ &= \mathsf{Enc}_{\mathsf{pk}}(m, v, e_0, e_1) \end{aligned}$$

where

$$m := h \cdot (\hat{\boldsymbol{m}}^{(i)} - \hat{\boldsymbol{m}}'^{(i)}) \bmod 2^T, \qquad v := h \cdot (\hat{\boldsymbol{v}}^{(i)} - \hat{\boldsymbol{v}}'^{(i)}),$$
$$e_0 := h \cdot (\hat{\boldsymbol{m}}^{(i)} - \hat{\boldsymbol{m}}'^{(i)}) \operatorname{div} 2^T, \qquad e_1 := h \cdot (\hat{\boldsymbol{e}}_1^{(i)} - \hat{\boldsymbol{e}}_1'^{(i)}).$$

Recall that $S_{\mathsf{ZKPoPK}} = 2\varphi(m)^2 UZ$. Since the extracted transcripts are accepting, we have $\left\|\hat{\boldsymbol{v}}^{(i)} - \hat{\boldsymbol{v}}'^{(i)}\right\|_\infty \leq 2\varphi(m)UZ$. Using $\|h\|_\infty = 1$, we conclude that $\|v\|_\infty \leq S_{\mathsf{ZKPoPK}}$. Similarly, it can easily be shown that $\|e_0\|_\infty \leq S_{\mathsf{ZKPoPK}} \cdot (2\sigma^2 + 1)$ and $\|e_1\|_\infty \leq S_{\mathsf{ZKPoPK}} \cdot 2\sigma^2$.

By repeating this procedure to compute (m, v, e_0, e_1) for each $j \in [U]$, we can extract a valid witness $w \in \mathbb{L}_{S_{\mathsf{ZKPoPK}}}(x)$. The expected runtime of the extractor is still $\mathsf{poly}(|x|)/\varepsilon < \mathsf{poly}(|x|)/(\varepsilon - \kappa)$, meeting the required bound.

Special Honest-Verifier Zero-Knowledge. In Figure 16 we present our simulator $\mathcal{S}_{\mathsf{ZKPoPK}}$. Let $x \in L_{\mathbb{L}}$ be a common input and let $W \in \mathsf{Chal}(\perp)$ be the verifier's randomness. We show that $\mathcal{S}_{\mathsf{ZKPoPK}}(x, W)$ is distributed statistically close to the transcript of $\langle \mathcal{P}(x), \mathcal{V}^W(x) \rangle$.

First, observe that we can ignore the component A in the transcript, as in both worlds A is uniquely determined by $(W, \hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1)$. As already argued in the completeness proof, we have $\|W \cdot \boldsymbol{v}\|_\infty \leq \varphi(m)U$. It follows that the coefficients of $\hat{\boldsymbol{v}} = \tilde{\boldsymbol{v}} + W \cdot \boldsymbol{v}$ in the real execution are distributed within statistical distance $\leq 1/Z$ from the coefficients of $\hat{\boldsymbol{v}}$ in the simulation. Similarly this can be shown for $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{e}}_1$. Clearly, W is distributed identically in both worlds. □

```
                              Simulator $\mathcal{S}_{\mathsf{ZKPoPK}}$
Input: $x = (C, \mathsf{pk}), W \in \mathsf{Chal}(\perp)^{V \times U}$
Output: $A, W, (\hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1)$
  Sample $(\hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1) \leftarrow \mathcal{R}^{V \times 3}$ uniformly at random subject to:
      $\|\hat{\boldsymbol{v}}\|_\infty \leq \varphi(m)UZ$   $\|\hat{\boldsymbol{m}}\|_\infty \leq \varphi(m)UZ \cdot 2^T(2\sigma^2 + 1)$   $\|\hat{\boldsymbol{e}}_1\|_\infty \leq \varphi(m)UZ \cdot 2\sigma^2$
  $D := \mathsf{Enc}_{\mathsf{pk}}(\hat{\boldsymbol{m}}, \hat{\boldsymbol{v}}, 0, \hat{\boldsymbol{e}}_1)$
  $A := D - W \cdot C \bmod q$
  return $A, W, (\hat{\boldsymbol{v}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{e}}_1)$
```

Fig. 16: Simulator for $\mathit{\Pi}_{\mathsf{ZKPoPK}}$

F ZKPoPK with Rejection Sampling

In this appendix, we restate and prove the security theorem of our ZKPoPK *with* rejection sampling.

Theorem 5. *If $\mathsf{flag} = \perp$ and $V \geq (\mathsf{Snd\_sec} + 2)/\log_2|\mathsf{Chal}(\mathsf{flag})|$, then $\mathit{\Pi}^{\mathsf{RS}}_{\mathsf{ZKPoPK}}$ from Figure 8 (in Appendix F) is*

- *complete (Definition 4) for $\mathbb{L}$ with completeness error $1/P$,*
- *knowledge sound (Definition 5) for $\mathbb{L}_{S_{\mathsf{ZKPoPK}}}$ with $S_{\mathsf{ZKPoPK}} = 6\varphi(m)\vartheta(\mathsf{flag})^2 UVP$ and knowledge error $2^{-\mathsf{Snd\_sec}}$,*
- *special honest verifier zero-knowledge (Definition 6) for $\mathbb{L}$ with statistical distance 0.*

Proof. Again, we prove the theorem for the case $\mathsf{flag} = \perp$. The other case $\mathsf{flag} = \mathsf{Diag}$ can be proven analogously.

Completeness: For any bound $B \in \mathbb{N}$, define $\mathcal{I}(B) := [-B, B)$. The coefficients of $\tilde{\boldsymbol{v}}$ are distributed uniformly at random in $\mathcal{I}((3\varphi(m)VP + 1) \cdot \varphi(m)U)$. From $\|W \cdot \boldsymbol{v}\|_\infty \leq \varphi(m)U$, it follows for fixed W

that each coefficient of $\hat{\boldsymbol{v}}$ is distributed uniformly at random in some superset of $\mathcal{I}_v := \mathcal{I}(3\varphi(m)^2UVP)$. For each coefficient of $\hat{\boldsymbol{v}}$, the probability that it passes the prover's check, i.e., lies within $\mathcal{I}_v$, is

$$\frac{3\varphi(m)^2UVP}{(3\varphi(m)VP+1)\cdot\varphi(m)U} = \frac{3\varphi(m)VP}{3\varphi(m)VP+1} > \frac{3\varphi(m)VP-1}{3\varphi(m)VP} = 1 - 1/(3\varphi(m)VP).$$

The same probability can be shown for the coefficients of $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{e}}_1$ and respective intervals $\mathcal{I}_m$ and $\mathcal{I}_{e_1}$. By a union bound over all $3\varphi(m)V$ coefficients, it follows that the prover aborts with probability less than $1/P$.

For the zero-knowledge property, observe that the abort probability does not depend on the inputs and challenge W. Hence, a simulator can proceed as follows: First determine whether the run should abort or not. If not, sample $\hat{\boldsymbol{v}}$, $\hat{\boldsymbol{m}}$, and $\hat{\boldsymbol{e}}_1$ uniformly at random from $\mathcal{I}_v$, $\mathcal{I}_m$, and $\mathcal{I}_{e_1}$, respectively, and compute A just like $\mathcal{S}_{\mathsf{ZKPoPK}}$ (Figure 16). In the abort case, output an aborting transcript instead. The usage of a computationally hiding commitment scheme makes sure that the aborting transcript is computationally indistinguishable from an aborting transcript in the real world.

Knowledge soundness follows analogously to the proof of Theorem 4. $\square$

G Secure Coin-Flip for Public-Coin Zero-Knowledge Proofs

One way to perform a secure coin-flip over some challenge space $\mathcal{C}$ is the following: First the prover commits to a random challenge index $i_{\mathcal{P}}$ and sends the commitment to the verifier. Afterwards the verifier samples a random challenge index $i_{\mathcal{V}}$, too, and sends it to the prover. Finally, the prover opens the commitment and both parties compute the challenge by indexing with $i_{\mathcal{P}} + i_{\mathcal{V}} \bmod |\mathcal{C}|$ into the challenge space.

Figure 17 displays our protocol $\Pi_{\mathsf{coin\text{-}flip}}$ that implements this approach in the ROM. It uses a random oracle $H : \{0,1\}^* \to \{0,1\}^{\mathsf{Comp\_sec}}$ which the prover uses to commit to $i_{\mathcal{P}}$.

By using $\Pi_{\mathsf{coin\text{-}flip}}$ to sample the challenges in a public-coin honest-verifier zero-knowledge proof, we obtain a zero-knowledge proof that is secure against malicious verifiers. This transform also preserves the completeness and knowledge soundness properties, except that the knowledge soundness property now only holds for computationally bounded provers that run in polynomial time.

Protocol $\Pi_{\mathsf{coin\text{-}flip}}$

$$\begin{array}{lcl}
\mathcal{P}: & & \mathcal{V}: \\
\hline
i_{\mathcal{P}} \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|} & & i_{\mathcal{V}} \xleftarrow{\$} \mathbb{Z}_{|\mathcal{C}|} \\
r \xleftarrow{\$} \{0,1\}^{\mathsf{Comp\_sec}} & & \\
c := H(r, i_{\mathcal{P}}) & \xrightarrow{c} & \\
 & \xleftarrow{i_{\mathcal{V}}} & \\
W := \mathcal{C}[i_{\mathcal{P}} + i_{\mathcal{V}}] & \xrightarrow{r,\, i_{\mathcal{P}}} & \text{Reject if } c \neq H(r, i_{\mathcal{P}}). \\
 & & W := \mathcal{C}[i_{\mathcal{P}} + i_{\mathcal{V}}]
\end{array}$$

Fig. 17: Our secure coin-flipping protocol