Overdrive LowGear 2.0:
Reduced-Bandwidth MPC without Sacrifice

Sebastian Hasler
sebastian.hasler@sec.uni-stuttgart.de
Institute of Information Security
University of Stuttgart, Germany

Toomas Krips
toomas.krips@ut.ee
Institute of Computer Science
University of Tartu, Estonia

Ralf Küsters
ralf.kuesters@sec.uni-stuttgart.de
Institute of Information Security
University of Stuttgart, Germany

Pascal Reisert
pascal.reisert@sec.uni-stuttgart.de
Institute of Information Security
University of Stuttgart, Germany

Marc Rivinius
marc.rivinius@sec.uni-stuttgart.de
Institute of Information Security
University of Stuttgart, Germany

ABSTRACT

Some of the most efficient protocols for Multi-Party Computation (MPC) follow a two-phase approach where correlated randomness, in particular Beaver triples, is generated in the offline phase and then used to speed up the online phase. Recently, more complex correlations have been introduced to optimize certain operations even further, such as matrix triples for matrix multiplications. In this paper, our goal is to improve the efficiency of the triple generation in general and in particular for classical field values as well as matrix operations. To this end, we modify the Overdrive LowGear protocol to remove the costly sacrificing step and therewith reduce the round complexity and the bandwidth. We extend the state-of-the-art MP-SPDZ implementation with our new protocols and show that the new offline phase outperforms state-of-the-art protocols for the generation of Beaver triples and matrix triples. For example, we save 33 % in bandwidth compared to Overdrive LowGear.

KEYWORDS

Multi-party computation; implementation

1 INTRODUCTION

Multi-Party Computation (MPC) allows several parties to compute an arithmetic circuit on private inputs without revealing information about the inputs apart from the result. Modern two-phase protocols, like SPDZ [19] and related protocols [18, 30, 31], consist of an offline phase, where (structured) random data, classically in the form of Beaver triples [3], is precomputed, and an online phase, where the precomputed data is used to compute the desired output from the private inputs. This general design principle allows parties to speed up the online phase considerably. A reasonably less efficient offline phase is usually considered acceptable since preprocessing can start well before the input data becomes available. Efficiency in these types of two-phase protocols and generally in MPC protocols heavily depends on the number of communication rounds needed and the bandwidth, i.e. the amount of data that has to be sent. Local computation times are often considered less relevant for real-world applications as long as hardware requirements, e.g. memory requirements, do not get out of hand. Apart from their effect on the overall runtime of an MPC scheme, communication costs, memory requirements, and bandwidth also contribute to the incurring (financial) costs when the scheme is actually deployed, e.g. on commercial cloud infrastructure. For SPDZ-like protocols, bandwidth currently is most expensive given the high prices for outgoing traffic.

SPDZ and its variants are state-of-the-art actively secure protocols. As long as at least one MPC party is honest, i.e. even if an adversary controls all but one honest party and deviates arbitrarily from the protocol, the adversary cannot gain any information on the honest party's inputs. Furthermore, the honest party is guaranteed that an output is correct—if not, the protocol aborts.[1]

ML Applications. The high security guarantees as well as the steadily improving efficiency of SPDZ-like protocols in recent years has led to a growing interest in using MPC in industrial grade application like privacy-preserving cloud computing and Machine Learning (ML) [20, 36, 45, 46]. ML usually requires a large amount of diversified data, often more than a single company can contribute. MPC offers a solution for distrustful industry competitors to train and evaluate an ML model without revealing private input data, which could for example contain business secrets or costumer data protected by law. While some applications like large ML training algorithms are still out of reach of current MPC protocols, evaluating neural networks or secure decision trees has been performed successfully in reasonable time [16, 27, 37, 41, 43, 44]. The large application potential of MPC in ML has also led to targeted improvements of the underlying MPC protocols themselves. One focus of these optimizations are operations that often occur in ML tasks, like matrix multiplications or tensor convolutions, and have therefore, a large impact on the performance of an ML algorithm.

Beaver Triples and Matrix Triples. Classically in the SPDZ framework, matrix multiplications (and similarly convolutions) are reduced to a series of sums and multiplications in the underlying finite field $K = \mathbb{F}_{p^d}$, e.g. for $X = (x_{ij}) \in K^{u\times v}$ and $Y = (y_{jk}) \in K^{v\times w}$ one gets $Z = X \cdot Y = (z_{ik}) \in K^{u\times w}$ as $z_{ik} = \sum_{j=1}^{v} x_{ij}y_{jk}$—a sum over v multiplications in K. To perform these multiplication in a secure way, SPDZ uses classical Beaver triples, i.e. shared random triples (a, b, c) with $ab = c$, which are used to mask the inputs x_{ij}, y_{jk} and compute a secure field multiplication. For a product of a $u\times v$ and a $v\times w$ matrix, this leads to uvw Beaver triples and therewith to much more (shared) masks than inputs. An easy solution to this overproduction of correlated random data was presented by [37] for a semi-honest MPC protocol. Namely, the parties directly construct a shared random *matrix triple* (A, B, C) with $C = AB$ as

[1] Both properties, i.e. privacy and correctness, are guaranteed with overwhelming probability in the security parameter.

a matrix product in the offline phase and use this matrix triple to mask X, Y and compute $Z = XY$. This approach then only needs an equivalent of $uv + vw + uw$ field elements in the online phase. Furthermore, [37] showed that matrix triples can be produced by a linear homomorphic encryption scheme in the offline phase. This construction was still only passively-secure and instantiated with the cryptosystems of Paillier [40] or Damgard-Geisler-Kroigaard [17], rather than a lattice-based cryptosystem as in SPDZ. A transfer to an actively secure protocol compatible with the SPDZ framework is, in fact, possible—as we show as a byproduct of our work. In particular, this seems very promising, as Overdrive LowGear [31] – the fastest SPDZ-like protocol (for a small number of parties) – is also based on the linear homomorphic properties of the BGV encryption scheme [9]. Unfortunately, the resulting lattice-based actively secure version of [37] is not the most efficient way to produce higher-structured randomness like matrix triples.

In fact, the first and currently best actively secure matrix triple generation protocol appeared in [11] and is based on the levelled homomorphic structure of BFV [8, 21]. The protocol reaches a very low bandwidth per generated matrix triple and is therewith not only significantly faster than a naive application of Overdrive with Beaver triples, but also faster than the before mentioned lattice-based linear homomorphic approach (along the lines of [37]). While most of the improvements compared to Overdrive are certainly based on the use of matrix triples instead of Beaver triples, [11] gains an additional advantage by using suitable matrix packing techniques and by avoiding a costly *sacrificing* technique inherent in all SPDZ-like protocols since SPDZ [19] itself. To this end, the BFV ciphertext modulus is extended significantly to support a larger number of homomorphic multiplications. The negative effect of this extension is compensated by removing the need for sacrificing.

Our Approach. The success of [11] leads to the question, whether (for matrix triples or even more general) lattice-based linear homomorphic encryption should be abandoned and replaced by a levelled homomorphic approach as in [11] or whether there is a better linear homomorphic solution. In this paper we show that the latter is the case by constructing a new linear homomorphic offline phase that, e.g. outperforms [11] for matrix triples by approximately 39 % in bandwidth and 62 % in runtime in the 2-party setup.

Our new offline phase is based on the following easy exchange protocol between two parties P_1 and P_2, just as [31, 37]: P_1 has a and sends $\mathrm{Enc}_{\mathrm{pk}_1}(a)$ to P_2 for Enc a suitable homomorphic (asymmetric) encryption scheme and pk_1 the public key of P_1. P_2 has pk_1, b, computes $\mathrm{Enc}(c_1) = \mathrm{Enc}_{\mathrm{pk}_1}(a)b - \mathrm{Enc}_{\mathrm{pk}_1}(r)$ for some random element r and sends $\mathrm{Enc}(c_1)$ to P_1. P_1 can decrypt this to $c_1 = ab - r$ since the encryption scheme is linear homomorphic.[2] As P_2 knows $c_2 = r$, the two parties have a sharing of $c = ab = c_1 + c_2$. This pairwise routine can be easily extended to construct a sharing of $c = ab$ from shares of a and b in an n-party protocol, i.e. to construct a shared structure random triple (a, b, c) where each party P_i holds a share $[a]_i, [b]_i, [c]_i$ of (a, b, c). If a, b are field values, we get a classical *Beaver triple*. If they are matrices, one gets a *matrix triple*.

Now in order to reach active security (as in [31]), triple shares are authenticated, i.e. each party P_i receives additional shares $[\alpha a]_i$, $[\alpha b]_i$, $[\alpha c]_i$ for some secret α. It can be shown that an adversary that deviates from the protocol will cause a protocol abort with overwhelming probability in the security parameter as long as he does not know α. However, he can change the shares or corrupted parties after the pairwise production—but before the authentication. In this way, he can get correctly authenticated shares which no longer add up to $c = ab$. In order to prevent this misbehavior, [31] (and all SPDZ-like protocols) use a sacrificing step. This sacrificing step results in a bandwidth overhead of 80 % to 100 % depending on the actual sacrificing protocol. We avoid this sacrificing step by intertwining the pairwise multiplication protocol and the authentication. In this way, the adversary no longer has the choice to enter different values into the authentication and the integrity of the triple is guaranteed without further sacrificing. Together with some further optimizations of the pairwise protocol itself we reduce the bandwidth compared to Overdrive [31] generally by 33 %.

We also extend the general approach to evaluate bilinear operation (e.g., matrix multiplications) to allow for even more efficient computations of special (matrix) operations. Notably, this includes mm (squaring of matrices) and $mm^{\top}$ (e.g., inner products) for matrices m. The corresponding reduced online phase only needs a *structured pair* instead of a triple—our bandwidth advantage in these cases is consequentially in the online phase around 33 %, a suitably constructed offline saves around 50 % bandwidth.

Furthermore, our offline protocols reduce the number of communication rounds per (batch of) *triples* produced from 3 down to 2. The effect of our round reduction becomes the more significant the slower the connection becomes, e.g. given a large geographical distance between compute parties. Please note that apart from a correspondingly smaller runtime, the lower traffic will also result in smaller monetary costs if the MPC protocols runs paid clouds.

We have implemented our new protocols as an extension to the MP-SPDZ framework [29]. It therefore contains all subprotocols already available in MP-SPDZ, including key generation, offline/online protocols (unless replaced by our optimized versions), the zero-knowledge proofs (ZKPs) from [31] and [2], MAC checks, and so on. In particular, it is compatible with MP-SPDZ based cloud platforms like Carbyne Stack [12]. Our implementation will profit from future improvements of MP-SPDZ [29]. We compare our protocols to a state-of-the-art Beaver triple based protocol [31] (Overdrive LowGear) for generic field operations, matrix operations and sample ML algorithms. We further run our triple production protocol against the best known matrix triple generation protocol [11]. Our benchmarks confirm our theoretical advantage.

Contributions.

- We present a new offline phase based on linear homomorphic encryption that reduces the bandwidth and number of communications rounds compared to the state-of-the-art MPC protocol for a small number of parties – Overdrive LowGear [31] – by around 33%. We prove security of the new protocols.
- We develop new offline and online protocols for specialized operations like matrix squares and inner products. The offline bandwidth advantage in the case of computations in a field compared to [31] is then even 50%.
- We apply our results to the production of matrix triples to save 39% in bandwidth in a 2-party setup compared to the best known

[2] Please see Section 3 for details on the sharing scheme and the linear homomorphic property. Appendix A repeats the BGV scheme used in SPDZ like protocols like ours.

matrix triple production in [11]. For specialized operations, this advantage is between 55% and 62%.

- We modify and extend the MP-SPDZ implementation [29] with the before mentioned protocols. We give benchmarks for generic field and matrix operations and benchmarks for Machine Learning applications. We compare our results to state-of-the-art MPC solutions like [31] (including improved TopGear zero-knowledge proofs [2]) and [11]. Our evaluation results confirm the theoretical predictions, e.g. in our ML benchmarks we save 28 %-74 % in runtime and 57 %-66 % bandwidth on average against Overdrive; we save 62 % in runtime against [11] for matrix operations and 58 % in bandwidth for ResNet-50 in the 2-party setup.[3]

2 RELATED WORK

We place our work in a series of SPDZ-like protocols, e.g. [2, 11, 18, 19, 30, 31] (see also [38]). Therefore, we concentrate our discussion on recent progress applicable to SPDZ-like papers, rather than classical theoretical results like [1, 3, 4, 13, 24].

In addition to the already mentioned development of secure matrix operations for MPC discussed in Section 1, first small optimizations of the Beaver triple-based online phase in SPDZ already appeared in [18], where square pairs are used to improve the squaring of secret shared values—we extend this optimization to higher-dimensional operations. This original idea has been picked up by Morton Dahl who describes in [15] a variety of generalized structured randomness that can improve the online phase. In particular, Dahl presents matrix triples and convolution triples, which have also been discussed in [37] in the passively secure domain. Matrix (and convolution) triples have since then seen further attention and are by now available as part of the actively secure protocol [11]. As mentioned before, the actively secure triple production in [11] relies on leveled homomorphic BFV encryption scheme [8, 21] together with classical packing methods like in [25, 28, 34]. Due to the large impact of matrix operations on the overall performance of ML algorithms, the secure evaluation of matrix products has a natural application in privacy-preserving ML ([11, 37]). Recent progress in the field of privacy-preserving ML with MPC include, e.g., [16, 27, 41, 43–46].

Finally, note that with a pseudo-random generator, as, e.g., in [6], structured randomness can be produced with only a small amount of initial communication. Special solutions exist on a theoretical level for other structured random data like matrix triples [7], too.

3 PRELIMINARIES

In this work, we will generally work over a commutative ring R and R-modules M. All rings are assumed to be unitary.

Since we focus on MPC in the dishonest majority setting, we use a classical additive secret-sharing, denoted by $[\cdot]$ on R-modules. A secret m in an R-module M is then shared among n parties $P_1, \dots, P_n$ such that $m = \sum_{i=1}^{n} [m]_i$ where $[m]_i$ is the share of party P_i. All shares are needed to reconstruct a secret and $n-1$ or less shares do not reveal any information. This secret sharing scheme is R-linear, i.e., we can set $[m+m']_i := [m]_i + [m']_i$, $[rm]_i := r \cdot [m]_i$, $[m+c]_i := [m]_i + \delta_{i1} \cdot c$ for shared values $m, m' \in M$ and a publicly known constants $r \in R$, $c \in M$, where δ_{ij} is the Kronecker delta. To open (or reconstruct) a secret-shared value, parties simply broadcast their shares and compute the sum of all shares.

In SPDZ and related protocols, shares are additionally authenticated to verify the outputs of the protocol using a MAC key [18, 19]. The MAC key $\alpha \in R$ is shared in the preprocessing phase. Elements m of an R-module M (e.g. like protocol inputs or structured randomness) are authenticated in the offline phase by adding a sharing of αm. We use $[\![m]\!] = ([m], [\alpha m])$ to denote the authenticated shares of m and $[\![\boldsymbol{m}]\!] = ([\![m_1]\!], \dots, [\![m_k]\!])$ for a tuple $\boldsymbol{m} = (m_1, \dots, m_k)$. Linear operations on authenticated shares are a trivial extension of linear operations on shares with except for $[\![m+c]\!]_i := ([m+c]_i, [\alpha m]_i + c \cdot [\alpha]_i)$. In slight abuse of notation we write $[m]_i \in M$ if each share $[m]_i$ is an element of M and analogously for authenticated shares.

A MAC check enables parties to verify the integrity of previously opened shares (cf. [18, 19] or Protocol 7). The soundness of the MAC check is proportional to $1/|R|$, e.g. $2/|\mathbb{F}_{p^d}|$ in [19], can be aggregated over many opened values, and does not reveal the MAC key [18]. We chose R such that $1/|R|$ is negligible in the security parameter.

When we analyze the theoretical performance of our protocols, bandwidth is measured in the number of ring elements sent. Analogously, the size of the structured randomness needed in the online phase, i.e. the tuple/triple size, is the number of ring elements contained in the triple/tuple which have to be provided by the offline phase. The round complexity of a protocol is the number of communication rounds. One communication round consists of all information that can be sent in parallel. In particular, if in a protocol party P_1 has to wait for a message from P_2 before P_1 can send her message, the protocol has round complexity 2. We note that the opening phase in actively secure SPDZ-like protocols comes with an additional invocation of a MAC check, which can be amortized for all openings in the online phase and does therefore not significantly influence the round complexity.

Finally, we use the UC model [10] to prove our schemes secure against malicious, static adversaries, except for proofs of knowledge where we allow rewinding to extract inputs from the adversary to avoid sending additional ciphertexts. The same limited UC model is also used in SPDZ-like protocols like Overdrive [31].

4 THE ONLINE PHASE

In this section, we first describe the general construction used to evaluate *bilinear authenticated* operations in the online phase. The most prominent example of these operations is classical (authenticated) multiplication in a finite field. However, the general construction also works for matrix products and tensor convolutions (cf. [11, 37]). We therefore present the online multiplication protocol in a universal form for modules over commutative rings—however, we mostly use the protocol for vector spaces over finite fields. Furthermore, we discuss special cases of this construction that occur if certain constraints are satisfied.

Abstract Form of Bilinear Evaluation. Let R be a commutative ring and M, M', M'' R-modules together with a bilinear map $\cdot : M \times M' \to M''$, $(m, m') \mapsto m \cdot m' =: mm'$.[4] In the online phase each party P_i gets shares $[\![m]\!]_i, [\![m']\!]_i$ of m, m' and the parties want to compute $[\![mm']\!]$ together. To this end, one extends the

[3] Note that for more than 2 parties, [11] provides no runtimes or implementation.

[4] $\cdot$ refers here to an arbitrary bilinear map and not a specific product in a field.

previously known classical constructions from Beaver [3] used in [19] and more recent results from [5] or [11] on bilinear triples.[5]

In these protocols it is common that the parties $P_1, \ldots, P_n$ have access to structured random data provided by the offline phase in the form of a shared tuple. The tuple usually contains shared input masks $[\![a]\!]$ and $[\![a']\!]$ for uniformly random $a \in M$, $a' \in M'$. The parties then locally compute $[\![m - a]\!]_i$ and $[\![m' - a']\!]_i$ and publish their share. After this initial round of communication, each party can locally add up the shares to get $m - a \in M$ and $m' - a' \in M'$. As long as the masks a, a' are not known to *any* party, the inputs m, m' are hidden information theoretically.

The random triple contains a further shared entry $[\![aa']\!]$ which depends on the masks a and a'. Overall we have the shared triple $([\![a]\!], [\![a']\!], [\![aa']\!]) \in M \times M' \times M''$ which can be used by the parties to compute a share of mm' as follows:

$$[\![mm']\!]_i = (m - a)[\![m']\!]_i + [\![a]\!]_i(m' - a') + [\![aa']\!]_i \qquad (1)$$

The correctness of this construction follows from the definition. Note in particular, that in the second component we use R-bilinearity to get $\sum_{i=1}^{n}(m - a)[\alpha m']_i = (m - a) \cdot (\alpha m') = \alpha((m - a)m')$.

Remark 1. *While this construction works in general for applications in maliciously secure MPC, the orbits of the R-actions should not become too small. E.g. for $M = M' = M''$, and the trivial R-actions, i.e., $r \cdot m = m$, we get independence of the MAC key and hence trivially not malicious security.*

Remark 2. *Note that by local operations, the parties can locally create any linear term in M and M'. Thus, any quadratic term in the inputs m, m' can be created with a shared triple $([\![a]\!], [\![a']\!], [\![aa']\!])$ in one round of communication.*

Special Tuples. In some applications, the different spaces M and M' obey certain constraints. One exceptionally useful case is when M and M' admit an R-linear map $\phi : M \longrightarrow M'$. Then we only need the structured pair $([\![a]\!], [\![a\phi(a)]\!]) \in M \times M''$ to compute $m\phi(m)$ for any $m \in M$—we call this kind of structured randomness special tuples/pairs. The corresponding protocol reduces to

$$[\![m\phi(m)]\!]_i = (m - a)\phi([\![m]\!]_i) + [\![a]\!]_i\phi(m - a) + [\![a\phi(a)]\!]_i. \qquad (2)$$

where $\phi([\![m]\!]_i) := (\phi([m]_i), \phi([\alpha m]_i))$.

Example 1. *The most simple example is that of squares in a field $\mathbb{F}_{p^d}$ introduced in [18]. In this case, choose $R = \mathbb{F}_{p^d} = M = M' = M''$, where R acts by field multiplication. Choosing $\phi = \mathrm{id}_R$,* (2) *can be used to compute the square of a shared value. In the field case, $\phi = \mathrm{id}_R$ is up to some scalar multiplication the only possible homomorphism of the underlying additive groups. While $\phi = z \cdot \mathrm{id}_R, z \in \mathbb{Z}$ can theoretically be used to compute $[\![zm^2]\!]$, it is much more practical to compute $[\![m^2]\!]$ and multiply the constant z locally.*

The previous example trivially extends to R-matrices in $M = R^{u \times v}, M' = R^{v \times w}, M'' = R^{u \times v}$ for some dimensions $u, v, w \in \mathbb{N}_{\geq 1}$ with the natural R-scalar multiplication in the matrix rings. For $\phi = \mathrm{id}_R$ (and hence $u = v = w$), we get matrix squares. However, in this higher dimensional case there are more possible choices for ϕ, i.e. $\phi \in \mathrm{Hom}_R(R^{u \times v}, R^{v \times w}) \simeq R^{uv^2w}$ (as R-modules). When we recall that by local computations we can also construct any linear term in the matrix entries locally, we see that a pair $(a, a\phi(a)) \in M \times M''$ is in fact enough to compute any quadratic term in the matrix entries.

One particularly nice application exists for $w = u$ and $\phi(m) = m^{\mathsf{T}}$ for $m \in M = R^{u \times v}$. In this case, (2) computes mm^{T}, i.e. the standard (non-degenerated) bilinear form $\langle m_i, m_j \rangle := m_i m_j^{\mathsf{T}}$ evaluated on the rows $m_1, \ldots, m_u$ of M for all $1 \leq i, j \leq u$. Even more, for any symmetric matrix $H \in R^{v \times v}$ and $\phi(m) = Hm^{\mathsf{T}}$ we get the corresponding non-degenerated symmetric bilinear form $\langle m_i, m_j \rangle_H = m_i H m_j^{\mathsf{T}}$. In the case of $u = 1$, we only need a tuple of size $v + 1$ to compute an inner product. Over $\mathbb{R}$ one gets scalar products and mm^{T} or $m^{\mathsf{T}}m$, respectively, is called Gram matrix. In particular, this special pairs can be used to compute scalar products and norms on fixed-point representations of real-valued data. The analogous construction works for scalar products and norms over $\mathbb{C}$.

Remark 3. *We concentrate in this paper on matrix squares and inner products, since the resulting special tuples for matrices are particularly interesting and have to our knowledge not been introduced. Since constraints must preserve R-linearity to apply* (2) *in an MPC protocol, other special setups can usually be reduced to tuples $(a, \phi(a))$.*

Now that we have seen how to evaluate bilinear maps with bilinear triple $([\![a]\!], [\![a']\!], [\![aa']\!])$ and special products of the form $m\phi(m)$ with special pairs $([\![a]\!], [\![a\phi(a)]\!])$ in the online phase, we have to explain how this structured randomness can be produced efficiently in the offline phase next.

5 THE OFFLINE PHASE

In this section we explain how to construct the randomness consumed in the online phase of Section 4 in an actively secure offline phase for generic bilinear triples and special tuples. In particular, this section provides an offline phase to generate classical Beaver triples and squares, matrix triples and special matrix pairs, but also other forms of correlated randomness, like e.g. triples for tensor convolutions. For matrix triples we introduce additional optimizations in Section 6.

We base our triple construction on the linear homomorphic properties of the BGV encryption scheme, like in [31]. Consequentially, our starting point is the LowGear protocol [31] which is still considered the state-of-the art SPDZ-like protocol for a low number of compute parties, e.g. $n = 2$ or 3. We consider a recent variant of this protocol with the more efficient zero-knowledge proofs introduced in [2]. We ultimately improve the triple production in [31] by intertwining the triple production and the authentication of the produced triples in one protocol. This allows us to avoid the sacrificing step common in SPDZ-like protocols like [31].

For this section we choose $R = \mathbb{F}_{p^d}$ a finite field.[6] Let $\Phi_m \in \mathbb{Z}[X]$ be the m-th cyclotomic polynomial of degree $N = \varphi(m)$. Let $A = \mathbb{Z}[X]/(\Phi_m)$ the ring of integers of the algebraic number field $\mathbb{Q}(\zeta_m)$ for ζ_m a primitive m-th root of unity, e.g. $\zeta_m = \exp(2\pi i/m) \in \mathbb{C}$. Let p be a prime and d the order of p in $\mathbb{Z}_m^*$. Then $\Phi_m \bmod p$ is the product of $s = N/d$ irreducible polynomials $f_1, \ldots, f_s$ such that $A_p := \mathbb{F}_p[X]/(\Phi_m \bmod p) \simeq \mathbb{Z}[X]/(\Phi_m, p) \simeq \bigtimes_{j=1}^{s} \mathbb{F}_p[X]/(f_j) \simeq \bigtimes_{j=1}^{s} \mathbb{F}_{p^d}$. We consider elements $m \in A_p$ as $s = N/d$-dimensional

[5] [5] interprets (finitely-generated) abelian groups as $\mathbb{Z}$-modules.

[6] An extension to base rings $R = \mathbb{Z}_{2^\nu}$ similar to [14, 39] seems possible. To check compatibility with modified MAC checks and packing methods over $\mathbb{Z}_{2^\nu}$ is however out of scope of this paper.

Π_{return}

P_j has input $(\text{Enc}_{\text{pk}_i}(a), (b_1, r_1), \ldots, (b_l, r_l))$ with $(b_k, r_k) \in A_p^{\lambda} \times A_p^{\mu}$ for $1 \le k \le l$ for some $l \in \mathbb{N}_{\ge 1}$.

1. P_j sends $\text{Enc}_{\text{pk}_i}(a)b_k - \text{Enc}^{\chi}_{\text{pk}_i}(r_k)$ to P_i for $1 \le k \le l$.
2. P_i decrypts this to get $d_k = ab_k - r_k$ for $1 \le k \le l$.

Protocol 1: Pairwise return protocol.

vectors $(m \bmod f_j)_{j=1}^{s}$.[7] This allows to use SIMD (Single Instruction Multiple Data) operations, e.g. we can evaluate an $\mathbb{F}_{p^d}$-bilinear operation $\cdot : M \times M' \longrightarrow M''$ in the *online phase* for $M = \mathbb{F}_{p^d}^{\kappa}, M' = \mathbb{F}_{p^d}^{\lambda}, M'' = \mathbb{F}_{p^d}^{\mu}$ on s pairs simultaneously. In more detail, we get the natural induced bilinear map

$$M^s \times (M')^s \longrightarrow (M'')^s, ((m_j)_{j=1}^s, (m'_j)_{j=1}^s) \mapsto (m_j m'_j)_{j=1}^s.$$

Since $M^s = A_p^{\kappa}, (M')^s = A_p^{\lambda}, (M'')^s = A_p^{\mu}$ this is a bilinear map $A_p^{\kappa} \times A_p^{\lambda} \longrightarrow A_p^{\mu}$. Now we can use the BGV [9] encryption scheme on the plaintext space A_p to securely evaluate this bilinear map and hence to evaluate s $\mathbb{F}_{p^d}$-bilinear operations $M \times M' \longrightarrow M''$.

The CPA-secure BGV [9] encryption scheme $(\text{Enc}^{\theta}_{\text{pk}}, \text{Dec}_{sk})$ on A_p is common in SPDZ-like protocols [19, 31], where θ denotes the distribution from which the encryption randomness is chosen. The ciphertext space is A_q^2 with $A_q := \mathbb{Z}[X]/(\Phi_m, q)$ for some natural number q and $p \nmid q$. We will usually drop the θ if we sample from the standard distribution used in [31] (cf. Appendix A). However, as in [31] we will occasionally sample from a larger distribution χ, i.e. set $\theta = \chi$—the exact form of this distribution χ depends on how the encryption scheme is employed. Most importantly for our applications the BGV scheme is linearly homomorphic (for suitably chosen parameters and suitable χ), i.e. $\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(r)r' + \text{Enc}^{\chi}_{\text{pk}}(r'')) = rr' + r''$ for $r, r', r'' \in A_p$.[8] For example $\text{Enc}_{\text{pk}}(a)b_k - \text{Enc}^{\chi}_{\text{pk}}(r_k)$ in Protocol 1 uses the standard distribution from [31] for the encryption of a and a larger distribution χ for the encryption of r_k (see also Remark 5 for further details on the distributions). Furthermore, χ depends on the operations to be evaluated. For example, if we use the encryption scheme with matrix operations the distribution must be chosen differently than for simple field operations—we will determine the exact size for matrix operations in Section 6 to guarantee the same security level as in [31]. For more details on the encryption scheme, see the reference literature or Appendix A. We remark that our construction does not rely on the specifics of this BGV-type encryption scheme and should be applicable for other linearly homomorphic encryption schemes. When we work over finite-dimensional free A_p-modules A_p^{κ} we apply the encryption scheme on A_p component-wise.

Finally we also use several ideal functionalities which mostly coincide with those in [31]—these are included in Appendix E.

Π_{pair}

P_i has input $[a]_i \in A_p^{\kappa}$, P_j has input $([b_k]_j)_{1 \le k \le l} \in (A_p^{\lambda})^l$ for some $l \in \mathbb{N}_{\ge 1}$.

1. P_i sends $\text{Enc}_{\text{pk}_i}([a]_i)$ to P_j using $\mathcal{F}_{\text{ZKP}}$ (cf. Protocol 10).
2. P_j samples $(r_{ijk})_{1 \le k \le l} \in (A_p^{\mu})^l$ and invokes Π_{return} with input $(\text{Enc}_{\text{pk}_i}([a]_i), ([b_k]_i, r_{ijk})_{1 \le k \le l})$ and P_i receives d_{ijk}.

Protocol 2: Pairwise multiplication protocol in [31].

5.1 Classical Triple Production with Sacrificing

We will first present the classical triple production from [31]. For the triple production we assume that the parties $P_1, \ldots, P_n$ already went through a setup phase such that P_i possesses the following data: Her own private keys $\text{sk}_i, \text{sk}'_i$, her own share $[\alpha]_i$ of the MAC key, the public keys[9] of all parties $\text{pk}_j, \text{pk}'_j$ for all $1 \le j \le n$, and encryptions of all MAC key shares $\text{Enc}_{\text{pk}'_j}([\boldsymbol{\alpha}]_j)$ with $[\boldsymbol{\alpha}]_j := ([\alpha]_j, \ldots, [\alpha]_j)$, and $[\alpha]_j \in A_p$ for all $1 \le j \le n$. Possible realizations of a secure setup phase are discussed, e.g. in [18] or [42].

The classical triple production Π_{LowGear} in Overdrive LowGear is given in Protocol 9.[10] We subdivided the original protocol [31, Fig. 7] into two (sub)protocols 1 and 2 in order to reuse these two pairwise subroutines in our new protocols later. Recall that to compute $c_1 = ab_1 = (\sum_{i=1}^n [a]_i)(\sum_{j=1}^n [b_1]_j)$ it is enough to compute each summand $[a]_i[b_1]_j$. Π_{pair} does just that, i.e. for two parties P_i with $[a]_i$ and P_j with $[b_1]_j$ the protocol outputs a (2-party) sharing of $[a]_i[b_1]_j$. By using a zero-knowledge functionality $\mathcal{F}_{\text{ZKP}}$ (cf. [31] or Protocol 10) for BGV ciphertexts, Π_{pair} furthermore guarantees that the parties know suitable plaintexts $[a]_i$ and that the ciphertext noise stays small enough and can be masked securely. If all pairs of parties run this subroutine Π_{pair} all mixed terms $[a]_i[b_1]_j$ can be constructed and then added as in Π_{LowGear}, step 3, to get a sharing of the product $c_1 = ab_1$. Overall, Π_{LowGear} outputs an authenticated 5-tuple $([\![a]\!]_j, [\![b_1]\!]_j, [\![c_1]\!]_j, [\![b_2]\!]_j, [\![c_2]\!]_j)$ in A_p such that $c_1 = ab_1$ and $c_2 = ab_2$.

Unfortunately, malicious parties can easily manipulate the product entries c_1, c_2 before authentication, i.e. a malicious party P_j simply modifies his shares to $[a]_j, [b_1]_j, [c_1]_j + \Delta_1, [b_2]_j, [c_2]_j + \Delta_2$. Now the parties hold a sharing of $(a, b_1, c_1 + \Delta_1, b_2, c_2 + \Delta_2)$. The authentication subprotocol in step 4 of Π_{LowGear} will authenticate this tuple to $([\![a]\!]_j, [\![b_1]\!]_j, [\![c_1 + \Delta_1]\!]_j, [\![b_2]\!]_j, [\![c_2 + \Delta_2]\!]_j)$. In particular, the MAC checks in the online phase cannot detect this modification. To prevent against this kind of attack, the parties have to run an additional *sacrificing* step. The usual sacrificing step is, e.g., described in [18] and also given in Appendix B. As a result, if the check goes through, the parties get one triple $([\![a]\!], [\![b_1]\!], [\![c_1]\!])$ such that with overwhelming probability $c_1 = ab_1$. The sacrificing step is the reason why we have to generate the other two entries b_2, c_2 in the first place. After the sacrificing, these cannot be used anymore (without compromising the security of the protocol), which creates a significant overhead no longer necessary in our protocol.

[7] See Appendix A for further details.

[8] Lattice-based encryption schemes like BGV usually do not have the homomorphic property unconditionally, i.e. applying several linear transformations on ciphertexts without intermediate decryption might lead to a wrong decryption results. By a correct parameter choice, we will ensure that this does not happen in our protocols (cf. Secion 6 and Section 7) for the discussed use cases.

[9] We require the same security guarantees from the setup phase as [31], e.g. for $\text{pk} = (a, b) \in A_q^2$, a sufficiently random for all parties. As in [29] we do not use seperate keys for each pair of parties. The security proofs of [31] still hold. We use the second key pk' mostly for the MAC keys for security reasons (cf. Appendix E, security proof).

[10] Note that original LowGear only considered Beaver triples, i.e. $\kappa = \lambda = \mu = 1$.

Π_{reduced}

P_j has input $\text{Enc}_{\text{pk}_i}([a]_i), \text{Enc}_{\text{pk}_i}([b]_i), ([a]_j, [b]_j, r_{ij}) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$.

1. P_j sends $\text{Enc}_{\text{pk}_i}([a]_i)[b]_j + [a]_j \, \text{Enc}_{\text{pk}_i}([b]_i) - \text{Enc}_{\text{pk}_i}^{\chi'}(r_{ij})$ to P_i (cf. Remark 5 for details on χ').
2. P_i decrypts this to get $d_{ij} = [a]_i[b]_j + [a]_j[b]_i - r_{ij}$.

Protocol 3: Reduced return protocol.

$\Pi_{\text{red-pair}}$

P_i has input $[a]_i \in A_p^\kappa$, $[b]_i \in A_p^\lambda$, P_j has input $[a]_j \in A_p^\kappa$, $[b]_j \in A_p^\lambda$.

1. P_i sends $\text{Enc}_{\text{pk}_i}([a]_i), \text{Enc}_{\text{pk}_i}([b]_i)$ to P_j using $\mathcal{F}_{\text{ZKP}}$.
2. P_j samples $r_{ij} \in A_p$ and invokes Π_{reduced} with input $(\text{Enc}_{\text{pk}_i}([a]_i), \text{Enc}_{\text{pk}_i}([b]_i), [a]_j, [b]_j, r_{ij})$. P_i receives d_{ij}.

Protocol 4: Reduced pairwise protocol.

Π_{Triple}

Generate. P_i has input $[\boldsymbol{\alpha}]_i, \text{Enc}_{\text{pk}'_j}([\boldsymbol{\alpha}]_j)$ for all $1 \leq j \leq n$.

1. P_i samples $[a]_i \in A_p^\kappa$, $[b]_i \in A_p^\lambda$.
2. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where P_j inputs $(\text{Enc}_{\text{pk}'_i}([\boldsymbol{\alpha}]_i), [b]_j)$. P_i receives d_{ij}. P_j gets r_{ij}.
3. The results are locally combined by P_i to $[\![b]\!]_i$.
4. Each (ordered) pair (P_i, P_j) invokes Π_{pair} where P_i has input $[a]_i$ and P_j has input $[\alpha]_j, [b]_j, [\alpha b]_j$. P_i gets d_{ijk}, P_j gets r_{ijk} for $k = 1, 2, 3$.
5. P_i locally combines the outputs to get $([\![a]\!], [\![b]\!], [\![c]\!]) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$.

Check. Each party P_i samples $[y_0]_i \in R$. The parties invoke $\mathcal{F}_{[\![\cdot]\!]}$ to authenticate $y_0 = \sum_{i=1}^n [y_0]_i$. Each party receives $[\![y_0]\!]_i$. To check the MACs of l (components of) triple entries $[\![y_k]\!]$, $1 \leq k \leq l$, the parties use $\mathcal{F}_{\text{random}}$ to generate $t \in R^l$. P_i opens $[z]_i = [y_0]_i + \sum_{k=1}^l t_k [y_k]_i$. The parties run $\mathcal{F}_{[\![\cdot]\!]}$.Check with input $z = \sum_{i=1}^n [z]_i$. If the check fails, abort.

Triple. The parties invoke **Generate** and receive s triples $([\![a]\!], [\![b]\!], [\![c]\!]) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$. The parties invoke **Check** on these triples. They store the s authenticated triples if the check succeeds.

Protocol 5: New LowGear-type Protocol.

5.2 Reduced Communication Protocol

Before we present our new offline protocol in the next subsection, we first discuss how the parties can use all available information to reduce the bandwidth in the pairwise subprotocol Π_{pair}.

Namely, if in Π_{return} (Protocol 1), P_j has access to not only $\text{Enc}_{\text{pk}_i}([a]_i)$ and $([b]_j, r_{ij})$ but also to $\text{Enc}_{\text{pk}_i}([b]_i)$, $[a]_j$ then they can return $\text{Enc}_{\text{pk}_i}([a]_i)[b]_j + [a]_j \, \text{Enc}_{\text{pk}_i}([b]_i) - \text{Enc}_{\text{pk}_i}^{\chi'}(r_{ij})$ which decrypts to $[a]_i[b]_j + [a]_j[b]_i - r_{ij}$. Note that we use a different distribution χ' (instead of χ in Π_{return}) to mask sums $\text{Enc}_{\text{pk}_i}([a]_i)[b]_j + [a]_j \, \text{Enc}_{\text{pk}_i}([b]_i)$—we will discuss choices for χ, χ' in more detail in Remark 5. Since we use the pairwise protocol to compute the mixed terms $[a]_i[b]_j + [a]_j[b]_i$ in the sum $ab = (\sum_{i=1}^n [a]_i)(\sum_{j=1}^n [b]_j)$, the parties can use this fact to reduce from two invocations of the pairwise subprotocol – each party acts once as sender and once as receiver – down to one invocation, where one party P_i provides $\text{Enc}_{\text{pk}_i}([a]_i)$ and $\text{Enc}_{\text{pk}_i}([b]_i)$ and the other one returns an encryption of $[a]_i[b]_j + [a]_j[b]_i - r_{ij}$.

In more detail: Let offset $\leftarrow j - i \bmod n$. Note that $S = \{(i, j) \in \{0, \ldots, n-1\}^2 : (0 < 2 \cdot \text{offset} < n) \vee (2 \cdot \text{offset} = n \wedge j > i)\}$ has exactly $n(n-1)/2$ elements. Moreover, if $(i, j) \in S$ than $(j, i) \notin S$, i.e. for each (i, j) with $i \neq j$ either $(i, j) \in S$ or $(j, i) \in S$. Now each pair (P_i, P_j) with $(i, j) \in S$ runs a pairwise protocol Π_{reduced} (Protocol 3) such that P_i receives $d_{ij} = [a]_i[b]_j + [a]_j[b]_i - r_{ij}$ and P_j receives r_{ij}. Oberve that each $[a]_k[b]_{k'}$ for any $0 \leq k, k' < n$ occurs exactly once in one of the d_{ij} for $(i, j) \in S$. Locally P_i combines the outputs of the different pairwise protocols to $e_i = [a]_i[b]_i + \sum_{j:(i,j) \in S} d_{ij}$, $f_i = \sum_{j:(i,j) \in S} r_{ij}$ and $[c]_i = [ab]_i = e_i + f_i$. As expected, we have $c = ab$.

This reduced pairwise protocol has a clear bandwidth advantage if both $\text{Enc}_{\text{pk}_i}([a]_i)$ and $\text{Enc}_{\text{pk}_i}([b]_i)$ are already available to the second party P_j, e.g. since they have been sent (and verified) in another subprotocol. For example, this is the case when $a = b$ and can be used to compute a sharing of a^2 (cf. Π_{Special} in Protocol 11).

If one encryption is not yet available and has to be provided additionally by party P_i, the situation is far less clear and often the standard Protocol 1 is still the best choice, since to send a new ciphertext also means to invoke a usually expensive zero-knowledge proof to realize $\mathcal{F}_{\text{ZKP}}$. We will discuss this issue in more detail once our new protocol Π_{Triple} has been established (cf. Remark 4).

5.3 Triple Production without Sacrificing

Our new offline protocol is presented as Π_{Triple} in Protocol 5. In contrast to Π_{LowGear} we construct the MAC share of $c = ab$ not from c itself but we use a and αb, i.e. $\alpha c = a(\alpha b)$. By this changed authentication of the third triple entry c, we avoid the attack which required previous protocols to add the sacrificing technique. Malicious parties can no longer enter a modified c into the authentication step, since the authentication and multiplication are now intertwined. In particular, each party has to commit to her share of a and her share of αb before c and αc are constructed—if a party modifies his shares of a (or αb) after $\text{Enc}_{\text{pk}_i}([a]_i)$ was sent in step 4 of Π_{Triple}, he also must change αa (or b) suitably.[11] As long as he does not know the MAC key he will fail to do so (with overwhelming probability)—the MAC check then fails and the overall protocol aborts. The MACs are checked in Π_{Triple}. **Check** in the usual way on field values. This means **Generate** outputs s triples $([\![a_\nu]\!], [\![b_\nu]\!], [\![c_\nu]\!]) \in R^\kappa \times R^\lambda \times R^\mu$ for $1 \leq \nu \leq s$. Now the new variable y_k runs through all components of the $[\![a_\nu]\!], [\![b_\nu]\!], [\![c_\nu]\!]$, e.g. $[\![a_\nu]\!] = ([\![a_{\nu\eta}]\!])$ for $1 \leq \eta \leq \kappa$ has the components $[a_{\nu\eta}] \in R$. The MAC check then takes an arbitrary linear combination of the y_k and adds a secret mask $y_0 \in R$ such that no information is leaked

[11] In Appendix E we shortly repeat the standard argument, why sending $\text{Enc}_{\text{pk}_i}([a]_i)$ results in the same $[a]_i$ being authenticated in step 4.

when the linear combination $z = [y_0] + \sum_{k=1}^{l} t_k [y_k]$ is opened. The opened value z is checked with the standard functionality $\mathcal{F}_{[\![\cdot]\!]}$ from [31]. $\mathcal{F}_{[\![\cdot]\!]}$ (cf. Protocol 15) is the standard MPC functionality from [31] that allows one to input values, perform local computations, and receive (verified) outputs. $\mathcal{F}_{[\![\cdot]\!]}$ can be realized by applying the input protocol $\Pi_{[\![\cdot]\!]}$ from [31] suitably, e.g. component-wise for every A_p-valued component of an input. We will prove security and privacy of Π_{Triple} below.

Remark 4 (Variants). *In* Π_{Triple} *(cf. Protocol 5), the reduced communication protocol* Π_{reduced} *is not used. An alternative protocol is included as* $\Pi_{\text{Triple-2}}$ *(Protocol 12). Namely, the* P_i *sends* $\text{Enc}_{\text{pk}'_i}([a]_i)$, $\text{Enc}_{\text{pk}'_i}([b]_i)$ *and later* $\text{Enc}_{\text{pk}'_i}([\alpha b]_i)$ *only for* $(0 < 2\,\text{offset} < n) \vee (2\,\text{offset} = n \wedge j > i)$*. We remark that step 1 and step 2 of* $\Pi_{\text{Triple-2}}$ *can be combined.* P_i *can then send* $\text{Enc}_{\text{pk}'}([\boldsymbol{\alpha}]_i[b]_i + \sum_{j \neq i}(d_{ij2} + r'_{ji2} + r''_{ji2}))$ *and together with step 2b the other parties can construct* $\text{Enc}_{\text{pk}'_i}([\alpha b]_i)$*, i.e. the protocol has in fact communication complexity* 3*. Since the reduced return protocol is now used 2 times this reduces the communication by 1 ciphertext. However, the additional ciphertexts to be provided by* P_i*, add half a ciphertext (per pair of parties) which has to be verified with a zero-knowledge proof. Using the a zero-knowledge proof from [2, 19, 31] requires another half ciphertext plus some plaintexts, i.e. overall* $\Pi_{\text{Triple-2}}$ *has slightly larger bandwidth than* Π_{Triple}*. Furthermore,* $\Pi_{\text{Triple-2}}$ *needs one more round of communication. We will therefore stick with Protocol 5 and also run our evaluation with* Π_{Triple}*. Nevertheless,* $\Pi_{\text{Triple-2}}$ *might be of some interest in cases where only a small number of triples is produced. Namely, the zero-knowledge proofs usually amortize over several ciphertexts.*[12] *If only a small number of triples and therewith a small number of* $\text{Enc}_{\text{pk}_i}([a]_i)$ *has to be verified, one invocation of the zero-knowledge proof is enough to prove* $\text{Enc}_{\text{pk}_i}([a]_i)$ *and* $\text{Enc}_{\text{pk}_i}([b]_i)$ *simultaneously. Without additional costs for the ZKP,* $\Pi_{\text{Triple-2}}$ *retains a slight advantage of half a ciphertext over* Π_{Triple} *(per pair of parties).*

Theoretical Performance. In Table 1 we give a short comparison of our new protocols to Overdrive's LowGear protocol [31]. To simplify the comparison, we present the number of ciphertexts sent. The number in brackets denotes the number of additional ciphertexts sent in a TopGear-style ZKP [2]. Please note that in this ZKP an amortized amount of two plaintexts per ciphertext verification are sent additionally. Due to the small size of a plaintext in A_p compared to a ciphertext in A_q^2 (for our applications with $p \ll q$) we did not include the plaintexts in the comparison. The same holds for plaintexts sent during the MAC checks and the sacrificing protocol. Moreover, we remark that ciphertexts in the different protocols can be of different modulus q due to security considerations. However, the difference is small, i.e. protocols that use the reduced pairwise protocol need a one bit larger q due to the larger noise χ'. Furthermore, the ciphertext modulus depends on $A_p^{\kappa}, A_p^{\lambda}, A_p^{\mu}$—more details for the special case of matrices can be found in Section 6.

Table 1: Comparison of triple production. Bandwidth in number of ciphertexts send per ordered pair (P_i, P_j), $i \neq j$. Ciphertext sizes may variate slightly between the different protocols (cf. Remark 5).

Approach	Comm. Rounds	Bandwidth	
Π_{Triple} (Protocol 5)	2	5	(+1)
$\Pi_{\text{Triple-2}}$ (Protocol 12)	3	4.5	(+1.5)
Π_{Special} (Protocol 11)	3	3.5	(+1)
$\Pi_{\text{Special-2}}$ (Protocol 14)	2	3.5	(+1)
Π_{LowGear} (Protocol 9, [31])	3	8	(+1)

Finally, please note that in the round count in Table 1 we already parallelized steps that do not depend on each other. E.g., Step 2 and Step 1 of Π_{pair} (as part of Step 4) can be run in parallel in Π_{Triple}, which reduces the number of communication rounds from 3 down to 2. As a result, we see that our new protocols have the same or a smaller round complexity and come with a significant bandwidth reduction of around 33% compared to LowGear in the generic case and up to 50% for special operations.

Proof of Correct Multiplication. We have to check that an adversary can not tamper with the input of Step 4 of Π_{Triple} (cf. Protocol 5). The security of [31] then guarantees that the outputs are correct, i.e. on input a, b', b'', Step 6 will output $[\![a]\!]$, $[\![b]\!]$, $([ab'], [ab''])$. More formally, we have the following security game:

Security Game. The challenger C samples $\alpha \in R$. The adversary $\mathcal{A}$ sends $\delta, \delta', \Delta, \Delta'$ to C. C checks whether $\alpha(a(b + \delta) + \Delta) = a(\alpha b + \delta') + \Delta'$, i.e. whether the MAC check succeeds. The adversary wins if the check goes through and $a\delta + \Delta \neq 0$, i.e. if $ab \neq c$ for $([\![a]\!], [\![b]\!], [\![c]\!])$ the produced shared triple. We claim that $\mathcal{A}$ wins the game with probability smaller or equal $\frac{1}{|R|}$.

Proof. We first note that the check by C is equivalent to $\alpha(a\delta + \Delta) = a\delta' + \Delta'$. Now if $\mathcal{A}$ chooses $a\delta + \Delta = 0$ he looses. If $\mathcal{A}$ chooses $a\delta + \Delta \neq 0$ then he only wins if $\alpha = \frac{\delta' a + \Delta'}{\delta a + \Delta}$. The probability for this case is $\frac{1}{|R|1}$ since $\mathcal{A}$ has no information on the uniformly sampled MAC key α. Thus, the probability that the adversary wins the game is bounded by $\frac{1}{|R|}$. □

Please note that the argument directly extends (as in [19, 30]) to the linear combination used in **Check** of Π_{Triple}.

Remark 5 (Ciphertext Noise Size). *For privacy we require that* χ *and* χ' *in* Π_{Triple} *and* Π_{Special} *(or subprotocols thereof) are chosen such that* $\text{Enc}^{\chi}_{\text{pk}_i}(r)$ *and* $\text{Enc}_{\text{pk}_i}([a]_i)[b]_j - \text{Enc}^{\chi}_{\text{pk}_i}(r_{ij})$ *are statistically indistinguishable (w.r.t. the security parameter* sec*) for every valid output* $\text{Enc}^{\chi}_{\text{pk}_i}([a]_i)$ *of* $\mathcal{F}_{ZKP}$ *and any* $[b]_j \in A_p^{\lambda}$ *and* r, r_{ij} *sampled uniformly from* A_p^{μ}*. If* slack *is the slack of the zero-knowledge proof, we get* $\|[a]_i\|_\infty \leq \frac{p}{2}\,\text{slack}$ *where* $\|\cdot\|_\infty$ *denotes the largest absolute value among all* polynomial *coefficients in any entry of* $a \in A_q$*.*[13] *In particular, in the case* $R = \mathbb{F}_{p^d}$ *the indistinguishability is guaranteed if we choose the encryption noise* χ *in* $\text{Enc}^{\chi}_{\text{pk}_i}$ *by a factor* $2^{\text{sec}} p\,\text{slack}$ *larger than the encryption noise in* Enc_{pk_i}*. Furthermore,* χ' *must be twice the size of* χ *in the reduced protocol* $\Pi_{\text{red-pair}}$ *to compensate the additional sum* $\text{Enc}([a]_i)[b]_j + [a]_j\,\text{Enc}([b]_i)$*. From now on, we will only use* χ, χ' *which guarantee security as described*

[12] For our benchmarks we use (comparable to [29]) a TopGear zero-knowledge proof with amortization over 6 ciphertexts.

[13] More details on the norm are included e.g. in [19]. Here, $(a_l)_{1 \leq l \leq \kappa} \in A_q^{\kappa}$ and $\|a\|_\infty = \max_{1 \leq l \leq \kappa} \|a_l\|_\infty$ where a_l is identified with $\sum_{j=0}^{N-1} a_{lj} X^j \in \mathbb{F}_q[X]/(\Phi_m)$ and $\|a_l\|_\infty = \max_{0 \leq j < N} |a_{lj}|$.

above. Section 6 will present the concrete values necessary for matrix rings over A_p.

Theorem 5.1. Π_{Triple} *implements* $\mathcal{F}_{\text{Triple}}$ *in the* $(\mathcal{F}_{[\![\cdot]\!]}, \mathcal{F}_{\text{rand}})$*-hybrid model with at least one honest party.*

Proof. We use the simulator $\mathcal{S}_{\text{Triple}}$ from Protocol 13, which replaces all inputs by honest parties with the value 0 and then runs the **Generate** and **Check** part of Π_{Triple} honestly. Since by assumption $\text{Enc}_{\text{pk}_i}([a]_i)[b_j] - \text{Enc}^{\chi}_{\text{pk}_i}(r_{ij}) \approx -\text{Enc}^{\chi}_{\text{pk}_i}(r'_{ij})$ for $r_{ij}, r'_{ij} \leftarrow A_p^{\mu}$ uniformely, the adversary cannot distinguish the real and ideal world instances of Π_{return}. Note that by $\mathcal{F}_{\text{ZKP}}$, he still needs to provide an encryption $\text{Enc}_{\text{pk}_i}([a]_i)$ for a sufficiently small input $[a]_i \in A_q$, i.e., $\|[a]_i\|_\infty \leq \frac{p}{2} \cdot$ slack, such that the indistinguishability is guaranteed. Furthermore, the adversary cannot distinguish $\text{Enc}_{\text{pk}_i}([a]_i)$ for a share $[a]_i$ in the real world from $\text{Enc}_{\text{pk}_i}(0)$ by the CPA-security of the encryption scheme. Moreover, observe that $[z]_i$ is distributed randomly, since y_0 is random.

Now the adversary is commited to (the sum of) his shares by the MAC check and the simulator can retrieve the (sum of the) inputs by the rewinding step. Hence, $\mathcal{S}$ can adapt the dummy shares $([\tilde{a}]_i, [\alpha\tilde{a}]_i, [\tilde{b}]_i, [\alpha\tilde{b}]_i, [\tilde{c}]_i, [\alpha\tilde{c}]_i)$ of honest parties $i \in H$ used in **Generate** to correct shares for the random outputs a, b, c from $\mathcal{F}_{\text{Triple}}$. More precisely, for $\tilde{a} = \sum_{i=1}^n [\tilde{a}]_i$, $\tilde{b} = \sum_{i=1}^n [\tilde{b}]_i$, $\tilde{c} = \sum_{i=1}^n [\tilde{c}]_i$, the output of **Generate** and $i_0 \in H$ set $[\![a]\!]_i = [\![\tilde{a}]\!]_i$, $[\![b]\!]_i = [\![\tilde{b}]\!]_i$, $[\![c]\!]_i = [\![\tilde{c}]\!]_i$ for all $i \neq i_0$ and $[a]_{i_0} = [\tilde{a}]_0 + a - \tilde{a}$, $[b]_{i_0} = [\tilde{b}]_0 + b - \tilde{b}$, $[c]_{i_0} = [\tilde{c}]_0 + c - \tilde{c}$. Since $\mathcal{S}$ has access to the MAC key α, $\mathcal{S}$ futher sets $[\alpha a]_{i_0} = [\alpha\tilde{a}]_0 + \alpha(a - \tilde{a})$, $[\alpha b]_{i_0} = [\alpha\tilde{b}]_0 + \alpha(b - \tilde{b})$, $[\alpha c]_{i_0} = [\alpha\tilde{c}]_0 + \alpha(c - \tilde{c})$. □

Remark 6. *Please also note that we do not prove security of the subroutines* Π_{pair} *or* Π_{return} *but of the complete protocol* Π_{Triple}*. In particular,* Π_{Triple} *is not the composition of these subroutines in the UC sense. However, the whole protocol* Π_{Triple} *has been proven secure above and can be composed with other UC secure protocols, e.g. the secure online protocol from [19], within our limited UC framework.*

Modifications for Special Tuples. For the production of special tuples, one can further optimize Π_{Triple}. As before in Section 4 we assume that Enc_{pk_i} is R-linear and $\phi : M \longrightarrow M'$ is a R-linear map, i.e. can be represented as $\phi(\sum_{i=1}^k r_i m_i) = \sum_{i=1}^k r_i \sum_{j=1}^l \phi_{ij} m_j$ for some $\phi_{ij} \in R$ and a basis m_i of the finite-dimensional free R-module M. Again we get an induced R-linear map $A_p^{\kappa} \longrightarrow A_p^{\lambda}$ which we denote in slight abuse of notation also by ϕ. Then a party P_j that is in possession of $\text{Enc}_{\text{pk}_i}([a]_i)$ can locally compute $\phi(\text{Enc}_{\text{pk}_i}([a]_i)) = \text{Enc}_{\text{pk}_i}(\phi([a]_i))$. Thus, the parties can use the reduced protocol $\Pi_{\text{red-pair}}$ to compute $[a\phi(a)]$. The detailed protocol Π_{Special} is included as Protocol 11—please note that Steps 3 and 4 can be run in parallel to reduce the number of communication rounds. The theoretical advantage for the generation of special pairs can be found in Table 1. The security of Π_{Special} can be proven analogously to the proof of Theorem 5.1. Please also note that, similar to Remark 4, other combinations of the subprotocols might be considered. One example is given as $\Pi_{\text{Special-2}}$ in Protocol 14, with its theoretical performance given in Table 1.

6 MATRIX TRIPLES

Besides improving the performance of Overdrive itself, another focus of this paper is to optimize MPC protocols for bilinear operations on higher-dimensional R-modules (which are in our case usually vector spaces over finite fields) and not only for field elements. We concentrate on matrix triples but note that similar results will also hold more general, e.g. for tensor convolutions.

Let $M = R^{u \times v}, M' = R^{v \times w}$ and $M'' = R^{u \times w}$ where the bilinear map $M \times M' \longrightarrow M''$ is the usual matrix multiplication. Over the plaintext space we get the induced matrix multiplication $A_p^{u \times v} \times A_p^{v \times w} \longrightarrow A_p^{u \times w}$, i.e. in the general notation from Section 5 $\kappa = u \times v, \lambda = v \times w, \mu = u \times w$. Recall that the security of our protocol relies on the indistinguishability of ciphertexts $\text{Enc}^{\chi}_{\text{pk}_i}(r)$ and $\text{Enc}_{\text{pk}_i}([a]_i)[b]_j - \text{Enc}^{\chi}_{\text{pk}_i}(r_{ij})$ where $r, r_{ij} \leftarrow A_p^{u \times w}$. Since the matrix multiplication $[a]_i[b]_j$ is defined by a sum over v products of field elements, we choose χ v-times larger than in the field case.

In more detail: Let $a = (a_{\nu\eta})_{\nu\eta} \in A_p^{u \times v}$ and $b = (b_{\eta\pi})_{\eta\pi} \in A_p^{v \times w}$. $\mathcal{F}_{\text{ZKP}}$ guarantees that $\|a_{\nu\eta}\|_\infty \leq \frac{p}{2}$ slack. Hence, $\|\sum_{\nu=1}^{v} a_{\nu\eta} b_{\eta\pi}\|_\infty \leq v\frac{p^2}{4}$ slack. In particular, these terms can be up to a factor v larger than simple multiplication in A_p. Thus, we also have to increase the encryption noise suitably. We choose χ $2^{\text{sec}} p$ slack $\cdot v$ times larger than normal encryption noise in Enc and therewith a factor v larger than necessary for simple field multiplications (cf. [31]). χ' must be adapted accordingly, i.e. a factor 2 larger than χ. Please note that the size of χ and χ', respectively, does affect the ciphertext modulus q. However, the ciphertext modulus increases only by v compared to field multiplication, e.g for usual matrix sizes like $v = 1024$ one needs a $\log_2(v) = 10$ bit longer ciphertext modulus. In particular, if we apply our protocol to matrices, our ciphertexts are slightly larger than those used in [31]. The effect on the bandwidth comparison (cf. Table 1) is minor.

For special matrix operations like matrix squares $\phi = \text{id}_M$ $(u = v = w)$ Π_{Special} (or variations therof like $\Pi_{\text{Special-2}}$) should be used depending on the setup, i.e. for a strict bandwidth restriction one uses the bandwidth optimized protocol Π_{Special}, for setup with high network delays one decreases the number of communication rounds with $\Pi_{\text{Special-2}}$. In some setups the properties of ϕ and $x\phi(x)$, respectively, allow further optimizations. E.g. for matrices of scalar products, i.e. the case $\phi(x) = x^{\top}$ $(u = w)$, $x\phi(x)$ is a symmetric $u \times u$ matrix. Hence, it is enough to return the $\frac{u(u+1)}{2}$ entries above and on the diagonal to compute the upper part of $a\phi(a)$. Then each party can locally recover the remaining $\frac{u(u-1)}{2}$ entries below the diagonal. Hence, the bandwidth can be reduced in Steps 3 and 5 of Π_{Special} and similiarly in $\Pi_{\text{Special-2}}$. Compared to classical matrix triples where we need a bandwidth of $3uv + vw + 2uw = 4uv + 2u^2$ modulus q ciphertexts, with specialized pairs for $\phi(x) = x^{\top}$ one only needs $\approx 3uv + 0.75uw = 3uv + 0.75u^2$ ciphertexts per pair of parties—the later ciphertexts are one bit longer.

Packing methods. As already recognized in previous work in [37] and [11] our paper shows again that matrix triples have a clear bandwidth advantage compared to the generic Beaver triples approach, where for each of the uvw products that occur in a $u \times v$ times $u \times w$ matrix multiplication one Beaver triple has to be created and later consumed in the online phase. Furthermore, the lattice based BGV ciphertexts allow for producing a larger number of (matrix) triples, namely $s = N/d$ in one round of our protocol. While this is an advantage for many real world use cases, for some applications only a small number of matrix triples is needed. Additionally, the production of a larger number of matrix triples can lead to memory issues on small machines. While this problem exists for all kinds of triples and in particular for Beaver triples produced in [31], for large dimensional matrices it is more pressing. We therefore use the well-known packing method from [25] to pack one dimension of the matrix into the slots of the ciphertext, i.e. we only produce s/u matrix triples at once. For details and alternatives see App. C.

We remark that the packing becomes particularly efficient for the production of matrix triples with Π_{Triple} since matrix multiplication can then be soley based on plaintext manipulations of the $[b]_i$. For special matrix pairs in order to use the reduced pairwise protocols the parties must also be able to manipulate ciphertexts which is slightly less efficient and requires some additional key switching material. Again, details can be found in Appendix C.

Finally, note that since ciphertexts are usually verified by the ZKPs in batches, we also pack the second matrix dimension into these batches. E.g. if the ZKP amortizes as in [31] over 40 ciphertexts (for sec = 40) and the parties want to compute 10×10 matrix triples they should run 4 instances of Π_{Triple} in parallel and use only one invocation of the ZKP to verify all 40 ciphertexts $\text{Enc}([a]_i)$. The effect of this packing is less significant for larger matrix dimensions or smaller ZKP amortization.

Table 2: ResNet-50. We compare the effect of matrix triples and ReLUs on the 2-party evaluation of ResNet-50 based on benchmarks.

Protocol	Matrix Mult.	ReLUs	Overall
[11]	51 GB	1007 GB	1058 GB
Ours	31 GB	413 GB	444 GB

7 IMPLEMENTATION AND EVALUATION

To illustrate the practicality of our approach, we have implemented our protocols in the MP-SPDZ framework [29] and run several benchmarks. This includes implementations of Π_{Triple} for matrices of arbitrary size and field elements, Π_{Special} for matrix squares and Gram matrices—both of arbitrary size, as well as the new online protocols for special matrix tuples. As a result, we have fully usable implementations of MPC protocols for the online phase, the offline phase, and the full protocol (online and offline phase combined).

Please note that the total runtime is dominated by the triple/pair generation and hence benchmarks of the full protocol are only slightly larger than the actual offline runtime. Therefore, we only benchmark the online phase and the full protocol (with the exception of offline-only benchmarks for the throughput in Table 8). Our implementation is available at [33].

We compare our results to the MP-SPDZ implementations of Overdrive LowGear, as well as the implementation of [11]. As far as we can see, the currently available implementation of [11] does not include an implementation to benchmark all parts of their protocol—only the homomorphic multiplication of (encrypted) matrices can be benchmarked. Therefore, we compare our implementation only to the numbers given in [11].

Table 3: Benchmarks for secure matrix multiplication of square matrices. Timings for LowGear [31] and [11] are taken from [11]. All experiments are for $n = 2$ parties. Our experiments emulate the network settings of the experiments in [11]: LAN setting with 10 ms network delay and 5 Gbit s^{-1} network bandwidth, and WAN setting with 35 ms delay and 320 Mbit s^{-1} bandwidth.

Network	Matrix Dims.	LowGear	[11]	Ours
LAN	128×128	128 s	36.1 s	8.54 s
	256×256	900 s	214.5 s	64.1 s
	384×384	46.8 min	653.6 s	216.3 s
	512×512	105 min	24.5 min	500.0 s
	1024×1024	735 min	173 min	67.1 min
WAN	128×128	24.6 min	38.15 s	9.13 s
	256×256	172.8 min	222.6 s	67.3 s
	384×384	540 min	672 s	221.1 s
	512×512	20.2 h	25 min	509.5 s
	1024×1024	141.1 h	175.1 min	67.4 min

Setup. In all our benchmarks we use a prime p of size 128 bits, $N = 8132$ and $p = 1 \bmod 2N$, i.e. Φ_m decomposes mod p into linear factors. We can in particular process $N = 8192$ $\mathbb{F}_p$-elements with one plaintext or ciphertext, respectively. Furthermore, we use TopGear [2] zero-knowledge proof with $V = 12$ and $U = 6$ to get soundness security of more than 128 bits. Overall we have computational security of at least 128 bits. As default in [29] we use statistical security parameter 40. We choose a ciphertext modulus of 383 bits to guarantee the computational security as well as correct decryption in all our protocols. We run [29] with the same setup (which is the default for its LowGear implementation). Note that [11] uses a slightly larger $V = 16$ given their larger cyclotomic polynomial with $N = 32768$.

We ran the benchmarks on two different computers: PC 1 (AMD EPYC 7443, 512 GB RAM) for Tables 3, 4, 5, 6, 7 and PC 2 (Intel i7-10700K, 80 GB RAM) for Tables 8, 9, 10. For better comparability we ran all our benchmarks on a single core. Please note however, that our triple production can be parallelized completely, i.e. we expect that the throughput increases linearly in the number of threads used. To simulate realistic MPC settings, we ran all our benchmarks for no delay and no imposed bandwidth restriction (on a single machine), with a network delay of 10 ms/bandwidth restriction 5 Gbit/s (LAN setup) and with a network delay of 35 ms/bandwidth rate restriction 320 Mbit/s.

Remark 7. *All benchmarks in this section are in a* 2*-party setup to stay comparable with the reference literature. For [11] only benchmark for the* 2*-party setup are available. Our comparison against Overdrive [31] presented here in the* 2*-party case directly transfers to more than two parties since both protocols are based on a pairwise subprotocol and scale linearly in the number of pairs. We remark that our implementation can be run (just as the LowGear implementation in MP-SPDZ) with an arbitrary number of parties.*

Comparison for Matrix Triples. [11] sends around 15.41 MB per party for the generation of one $(u, v, w) = (128, 128, 128)$ matrix triple. This number is slightly higher than originally claimed by the authors, since in their theoretical analysis (Equation (10) of [11]) two ciphertexts are missing (2 ciphertexts equivalent to $4A_q$ elements in step 2 of the matrix production in Fig. 1 and $4A_q$-elements from the distributed decryption lead to $8A_q$-elements instead of 6.) At the same time, our bandwidth amounts to around 9.44MB per $(128, 128, 128)$ matrix triple per party in a 2-party setup. Hence in the 2-party setup our protocol has around 39 % less bandwidth. As can be seen from our benchmarks in Table 3 we even have a larger runtime advantage against [11], i.e. in average by around 62 %. As mentioned before, there is currently no data available for more than two parties for [11]. We expect however, that as usual for pairwise protocols, the levelled homomorphic approach [11], which is linear in the number of parties n, will outperform our protocol for large n.

To stay comparable to [11] we shortly discuss the effect of our new protocols on an private interference of ResNet-50 [26]. As in [11] we rewrite the convolutions as matrix multiplications—we provide more details in Appendix D. As a result one gets 3298 multiplications of 128×128 matrices. Additionally, we compare the effect of our protocol on the 9,608,704 ReLUs of ResNet-50. Each ReLU requires 122 Beaver triples and some shared bits that have a minor impact on the bandwidth. We remark that ReLU layers (just like other types of layers, e.g. batch normalization, pooling layers) are implemented in [29]. Since for [11] only data on the convolutions and the ReLUs is available, we also restrict to these two layers for this paragraph. In the next paragraph, we then provide benchmark for small ML algorithms which consist of different type layers (including convolutions, dense layers and ReLUs). The result of the comparison with [11] on ResNet-50 can be seen in Table 2. We assumed 6.874 64 kbit per Beaver triple (cf. Table 8).

Special Matrix Pairs. The Tables 5 and 7 compare the runtime and bandwidth costs for an online matrix multiplication with classical LowGear, our matrix triples based online phase and for the reduced online phase with special pairs. The Tables 4 and 6 contain the corresponding complete protocol including the triple/pair production. For the special pair benchmarks we used our implementation of transpose pairs for multiplications AA^T, which performs better in the offline phase due to specific further optimizations. For completeness, we also added the bandwidth benchmarks for squares in the offline phase to show that matrix transpose pairs have the expected advantage of around 16 % (cf. Section 6).

We see that although special matrix pairs have a clear bandwidth advantage the effect of the reduced bandwidth on the overall runtime is covered for small matrix dimensions by the network delay in a realistic LAN or WAN setup. Due to the higher number of communication rounds in Π_{Special} compared to Π_{Triple}, special matrix pairs might be slightly slower. However, depending on the setup, a minimally slower runtime can be acceptable given the large bandwidth advantage, e.g. if higher bandwidth comes with higher costs in cloud computing.

Comparison to Generic LowGear. We also compared our protocol to generic Overdrive in Table 8. The comparison confirms the theoretical advantage of our protocols for generic field operations. Please note that the throughput is lower than in the original [31] paper since we only use a single core instead of 8 threads in [31]. As mentioned before the triple generation can be completely parallelized such that with 8 threads our protocol will produce approximately 8 times as many triples per second.

Table 4: Benchmarks for secure special matrix operations of square matrices. We use the same network settings as in Table 3. The "local" setting corresponds to running the code of the parties on the same machine without any network delay or bandwidth restrictions. Timings are amortized over 8192 tuples. For LowGear, we ran normal matrix multiplication as no special matrix operations are available in SPDZ.

Net- work	Matrix Dims.	LowGear	Ours (Mult.)	Ours (Special)
Local	2 × 2	1.12 ms	249.36 µs	152.04 µs
	4 × 4	6.57 ms	820.06 µs	550.15 µs
	8 × 8	47.90 ms	4.14 ms	2.65 ms
	16 × 16	375.76 ms	22.96 ms	15.11 ms
	32 × 32	2.98 s	151.77 ms	98.13 ms
	64 × 64	23.68 s	1.08 s	693.15 ms
LAN	2 × 2	22.38 ms	20.60 ms	20.49 ms
	4 × 4	33.49 ms	21.25 ms	20.92 ms
	8 × 8	123.62 ms	24.83 ms	23.11 ms
	16 × 16	825.47 ms	45.05 ms	36.38 ms
	32 × 32	6.19 s	180.26 ms	124.93 ms
	64 × 64	49.07 s	1.12 s	724.89 ms
WAN	2 × 2	75.90 ms	70.91 ms	70.73 ms
	4 × 4	104.37 ms	72.04 ms	71.41 ms
	8 × 8	323.95 ms	77.72 ms	74.57 ms
	16 × 16	2.01 s	104.89 ms	91.61 ms
	32 × 32	15.33 s	267.52 ms	193.45 ms
	64 × 64	121.72 s	1.34 s	864.30 ms

Machine Learning. Finally we provide benchmarks for ML application. We therefore use 4 benchmark programs available in MP-SPDZ. These programs combine different ML layers to reflect different architecture common in dense and convolutional network:

- Benchmark Net A contains the following layers in this order: Dense, Square, Dense, Square, Dense, Argmax.
- Benchmark Net B contains the following layers in this order: $2d$ Convolution, MaxPool, ReLU, $2d$ Convolution, MaxPool, ReLU, Dense, ReLU, Dense, Argmax.
- Benchmark Net C has layers as B but with different dimensions.
- Benchmark Net D contains the following layers in this order: $2d$ Convolution, ReLU, Dense, ReLU, Dense, Argmax.

For further specifics on the layers, e.g. number of inputs, we refer to the corresponding programs available in our implementation. The results of our evaluation for our protocol as well as Overdrive LowGear are included in Table 10 for the online phase and Table 9 for the overall protocol. Recall that we realize the convolution layers by matrix multiplications as [11] (cf. Appendix D). Furthermore, the matrix operations in the dense layers use matrix triples. If Beaver triples are used, e.g. in ReLU Layers, we use Π_{Triple} for field values. In the online phase we have an average advantage in runtime of 28 %, in the offline phase we are in average approximately a factor

Table 5: Benchmarks for secure special matrix operations of square matrices in the online phase. We use the same network settings as in Table 4. Timings are amortized over 100 multiplications. For LowGear, we ran normal matrix multiplication as no special matrix operations are available in SPDZ.

Network	Matrix Dims.	LowGear	Ours (Mult.)	Ours (Special)
Local	2 × 2	26.24 µs	19.68 µs	16.53 µs
	4 × 4	52.36 µs	23.58 µs	24.18 µs
	8 × 8	288.21 µs	62.86 µs	53.92 µs
	16 × 16	2.07 ms	318.36 µs	284.71 µs
	32 × 32	16.14 ms	2.20 ms	2.04 ms
	64 × 64	127.96 ms	16.37 ms	15.43 ms
LAN	2 × 2	20.60 ms	20.59 ms	20.58 ms
	4 × 4	20.63 ms	20.58 ms	20.61 ms
	8 × 8	20.93 ms	20.64 ms	20.64 ms
	16 × 16	24.82 ms	20.95 ms	20.96 ms
	32 × 32	45.69 ms	23.40 ms	23.12 ms
	64 × 64	265.53 ms	39.41 ms	38.04 ms
WAN	2 × 2	71.83 ms	71.87 ms	71.95 ms
	4 × 4	71.96 ms	71.85 ms	71.85 ms
	8 × 8	73.02 ms	71.98 ms	71.91 ms
	16 × 16	85.16 ms	72.64 ms	72.41 ms
	32 × 32	158.84 ms	76.30 ms	75.26 ms
	64 × 64	842.21 ms	102.36 ms	96.06 ms

Table 6: Bandwidth measurements for secure special matrix operations of square matrices. The measurements were taken in the experiments for Table 4, i.e. in the same setting. Bandwidth is given as average per party.

Matrix Dims.	LowGear	Ours (Mult.)	Ours (Transp.)	Ours (Square)
2 × 2	14.66 kB	3.59 kB	2.03 kB	2.18 kB
4 × 4	88.47 kB	11.27 kB	6.69 kB	7.55 kB
8 × 8	650.11 kB	45.06 kB	25.03 kB	29.06 kB
16 × 16	5.09 MB	161.82 kB	97.81 kB	115.09 kB
32 × 32	40.45 MB	647.28 kB	387.78 kB	459.21 kB
64 × 64	323.17 MB	2.52 MB	1.55 MB	1.84 MB

of 3.80 faster. On average, the bandwidth in our protocol is reduced by a factor of 2.35 in the online phase and 2.95 in the offline phase.

In summary, we have seen that the newly presented protocols and our implementation improve significantly in runtime and bandwidth over Overdrive LowGear, which is the most efficient MPC protocols for generic field operations for a low number parties, as well as [11], the best known protocol for matrix triple generation. Our sample ML inference benchmarks further show the application potential of our protocols.

ACKNOWLEDGMENTS

This research was supported by the CRYPTECS project founded by the German Federal Ministry of Education and Research under Grant Agreement No. 16KIS1441 and from the French National Research Agency under Grant Agreement No. ANR-20-CYAL-0006. Additionally, this research has been funded by European Social Fund via IT Academy programme. We furthermore thank Johannes Schäufele for his help and valuable remarks.

Table 7: Bandwidth measurements for secure special matrix operations of square matrices in the online phase. The measurements were taken in the experiments for Table 5, i.e. in the same setting. Bandwidth is given as average per party.

Matrix Dims.	LowGear	Ours (Mult.)	Ours (Special)
2 × 2	257.36 B	129.36 B	65.36 B
4 × 4	2.05 kB	513.36 B	257.36 B
8 × 8	16.39 kB	2.05 kB	1.03 kB
16 × 16	131.07 kB	8.19 kB	4.10 kB
32 × 32	1.05 MB	32.77 kB	16.39 kB
64 × 64	8.39 MB	131.07 kB	65.54 kB

Table 8: Bandwidth per triple and throughput

Protocol	sec = 40	sec = 64	sec = 128
LowGear	10 283 triple/s	8454 triple/s	6840 triple/s
	14.133 kbit/triple	15.950 kbit/triple	17.767 kbit/triple
Ours	25778 triple/s	21382 triple/s	17012 triple/s
	6.875 kbit/triple	7.783 kbit/triple	8.691 kbit/triple

Table 9: Benchmarks for secure ML inference. We use the same network settings as in Table 3. ML model names (A–D) designate the neural networks in [32, 44] (with the implementation included in [29]).

Protocol	Model	Local	LAN	WAN
LowGear	A	15.65 s	36.76 s	99.25 s
	B	252.31 s	10.06 min	28.50 min
	C	7.88 min	20.01 min	49.01 min
	D	32.57 s	79.45 s	201.35 s
Ours	A	5.17 s	9.69 s	22.53 s
	B	95.18 s	184.45 s	7.56 min
	C	156.24 s	292.46 s	11.59 min
	D	12.43 s	24.75 s	56.21 s

Protocol	A	B	C	D
LowGear	439.90 MB	7.09 GB	13.68 GB	920.01 MB
Ours	141.47 MB	2.72 GB	4.27 GB	361.62 MB

REFERENCES

[1] Judit Bar-Ilan and Donald Beaver. 1989. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *PODC 1989*. ACM, 201–209.

[2] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. 2020. Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ. In *SAC 2019*. Springer, 274–302.

[3] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO '91*. Springer, 420–432.

Table 10: Online benchmarks for secure ML inference. We use the same network settings as in Table 3. ML model names (A–D) designate the neural networks in [32, 44] (with the implementation included in [29]).

Protocol	Model	Local	LAN	WAN
LowGear	A	0.121 s	1.378 s	4.421 s
	B	0.733 s	6.471 s	20.573 s
	C	1.847 s	10.555 s	34.080 s
	D	0.159 s	1.770 s	5.441 s
Ours	A	0.070 s	1.002 s	3.416 s
	B	0.526 s	5.201 s	16.059 s
	C	0.358 s	7.479 s	23.000 s
	D	0.107 s	1.291 s	4.298 s

Protocol	A	B	C	D
LowGear	9.524 MB	80.969 MB	209.219 MB	13.346 MB
Ours	5.768 MB	40.652 MB	77.754 MB	8.925 MB

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC 1988*. ACM, 1–10.

[5] Christina Boura, Ilaria Chillotti, Nicolas Gama, Dimitar Jetchev, Stanislav Peceny, and Alexander Petric. 2018. High-Precision Privacy-Preserving Real-Valued Function Evaluation. In *FC 2018*. Springer, 183–202.

[6] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO 2019*. Springer, 489–518.

[7] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2022. Efficient Pseudorandom Correlation Generators from Ring-LPN. Cryptology ePrint Archive, Paper 2022/1035. https://eprint.iacr.org/2022/1035

[8] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012*. Springer, 868–886.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *ITCS 2012*. ACM, 309–325.

[10] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS 2001*. IEEE, 136–145.

[11] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. 2020. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In *ASIACRYPT 2020*. Springer, 31–59. Implementation: https://github.com/snwagh/ponytail-public/.

[12] Carbyne Stack Contributors. 2021. Carbyne Stack: Open Source Cloud Native Secure Multiparty Computation. Available at https://carbynestack.io/.

[13] Ronald Cramer and Ivan Damgård. 2001. Secure Distributed Linear Algebra in a Constant Number of Rounds. In *CRYPTO 2001*. Springer, 119–136.

[14] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD $\mathbb{Z}_{2^k}$: Efficient MPC Mod 2^k for Dishonest Majority. In *CRYPTO 2018*. Springer, 769–798.

[15] Morten Dahl. 2017. Cryptography and ML. https://mortendahl.github.io

[16] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. Secure Evaluation of Quantized Neural Networks. *PETS* 2020, 4 (2020), 355–375.

[17] Ivan Damgård, Martin Geisler, and Mikkel Kroigard. 2008. Homomorphic Encryption and Secure Comparison. *Int. J. Appl. Cryptol.* 1, 1 (2008), 22–31.

[18] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In *ESORICS 2013*. Springer, 1–18.

[19] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012*. Springer, 643–662.

[20] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. 2019. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *SP 2019*. 1102–1120.

[21] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryp. ePrint Arch.* (2012), 144. https://ia.cr/2012/144

[22] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Fully Homomorphic Encryption with Polylog Overhead. In *EUROCRYPT 2012*. Springer, 465–482.

[23] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic Evaluation of the AES Circuit. In *CRYPTO 2012*. Springer, 850–867.

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC 1987*. ACM, 218–229.

[25] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *CRYPTO 2014*. Springer, 554–571.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR 2016*. IEEE, 770–778.

[27] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. *IACR Cryptol. ePrint Arch.* (2022), 207. https://eprint.iacr.org/2022/207

[28] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *CCS 2018*. ACM, 1209–1222.

[29] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *CCS '20: 2020 ACM, Virtual Event*. ACM, 1575–1590.

[30] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *CCS*. ACM, 830–842.

[31] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ Great Again. In *EUROCRYPT 2018*. Springer, 158–189.

[32] Marcel Keller and Ke Sun. 2022. Secure Quantized Training for Deep Learning. In *ICML 2022*. PMLR, 10912–10938.

[33] Toomas Krips, Ralf Küsters, Pascal Reisert, Marc Rivinius, and Johannes Schäufele. 2022. Overdrive 2.0: Implementation. Available on request.

[34] F. Thomson Leighton. 1991. *Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes*. Elsevier.

[35] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. A Toolkit for Ring-LWE Cryptography. In *EUROCRYPT 2013*. Springer, 35–54.

[36] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *CCS '18*. ACM, 35–52.

[37] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *SP 2017*. IEEE Computer Society, 19–38.

[38] Emmanuela Orsini. 2021. Efficient, Actively Secure MPC with a Dishonest Majority: A Survey. In *Arithmetic of Finite Fields*. Springer, 42–71.

[39] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. 2020. Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption. In *CT-RSA 2020*. Springer, 254–283.

[40] Valerio Pastro. 2013. *Zero-Knowledge Protocols and Multiparty Computation*. Ph. D. Dissertation. Aarhus University. Advisor(s) Damgård, Ivan.

[41] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS 2020*. ACM, 325–342.

[42] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. 2022. Actively Secure Setup for SPDZ. *J. Cryptol.* 35, 1 (2022), 5.

[43] Jinhyun So, Başak Güler, and A. Salman Avestimehr. 2021. CodedPrivateML: A Fast and Privacy-Preserving Framework for Distributed Machine Learning. *J. Sel. Areas Inf. Theory* 2, 1 (2021), 441–451.

[44] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *PETS* 2019, 3 (2019), 26–49.

[45] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PETS* 2021, 1 (2021), 188–208.

[46] Wenting Zheng, Raluca A. Popa, Joseph Gonzalez, and Ion Stoica. 2019. Helen: Maliciously Secure Coopetitive Learning for Linear Models. *SP 2019*, 724–738.

A THE BGV ENCRYPTION SCHEME

In the offline phase we use the [9] cryptosystem $(\mathrm{Enc}_{\mathrm{pk}}, \mathrm{Dec}_{\mathrm{sk}})$.[14] We will stick closely to its use in [19], [18] or [31] in order to be able to compare our approach for the construction of matrix triples with the state of the art protocols. We will describe the cryptosystem on some base ring A_p—the extension to A_p-modules A_p^k is as usual entrywise.

We choose a message space $A_p \coloneqq \mathbb{F}_p[X]/(\Phi_m \bmod p)$ for a m-th cyclotomic polynomial $\Phi_m \in \mathbb{Z}[X]$ of degree $N = \varphi(m)$ and p a prime. Note that $A_p \simeq \bigtimes_{j=1}^{s} \mathbb{F}_{p^d}$ for p of order d modulo m and $s = N/d$. Please recall that Φ_m decomposes into s degree d (pairwise prime) irreducible polynomials over $\mathbb{F}_p$ then. Let $p \nmid q$ and embed A_p into $A_q = (\mathbb{Z}/q\mathbb{Z})[X]/\Phi_m$ in the usual way, i.e. identify $f = \sum_{j=0}^{N} f_j X^j \in A_p$ with $\sum_{j=0}^{N} g_j X^j$ and $g_j = f_j \bmod p$ if $f_j \leq p/2$ and $g_j = f_j \bmod p - p$ for $f_j > p/2$ where $f_j \bmod p$ denotes the representative of $f_j \in \mathbb{F}_p$ in $\{0, \dots, p-1\}$ under the natural quotient map $\mathbb{Z} \to \mathbb{F}_p \simeq \mathbb{Z}/p\mathbb{Z}$. Furthermore note that $\mathbb{Z}_q$ naturally embeds (as a set) in $\mathbb{Z}$ and $\mathbb{Q}$ and that $\mathbb{Q}[X]/\Phi_m$ embeds under the canonical embedding into $\mathbb{C}^N$.

We consider the following distributions on $L = A_p$ or $L = A_q$:

- $\mathcal{U}(L)$ is the uniform distribution on M.
- $\mathcal{D}_\sigma(L)$ the discrete Gaussian distribution of variance σ^2 on L.
- $\mathcal{ZO}_{1/2} = (\mathcal{U}(\{-1,0\}) * \mathcal{U}(\{0,1\}))^N$.[15]
- $\mathcal{HWT}_h = \mathcal{U}(\{x \in \{0, \pm 1\}^N \mid \sum_{j=1}^{N} |x_j| \geq h\})$.[16]

Key Generation. Take $\mathrm{sk} \leftarrow \mathcal{HWT}_h, a \leftarrow A_q, e \leftarrow D_\sigma(A_q)$. Set $b \leftarrow \mathrm{sk}\, a + pe$ and $\mathrm{pk} \leftarrow (a, b)$.
Encryption. Let $m \in A_p$ be a message. Sample $v \leftarrow \mathcal{ZO}_{1/2}, e_0 \rightarrow \chi, e_1 \leftarrow D_\sigma(A_q)$ and set $\mathrm{Enc}_{\mathrm{pk}}(m) = (c_0, c_1)$ with $c_0 = bv + pe_0 + m \in A_q$ and $c_1 = av + pe_1 \in A_q$.
Decryption. Decrypt $(c_0, c_1) \in A_q^2$ using sk by $(c_0 - \mathrm{sk}\, c_1 \bmod q) \bmod p$. Correct decryption is guaranteed for $pe_0 + m - \mathrm{sk}\, pe_1 < q$.

Classically $\chi = D_\sigma(A_q)$. For larger encryption noise adapt σ accordingly, i.e. for $\log_2 v = 10$ times larger encryption noise, sample from $\chi = D_{10\sigma}(A_q)$. For other parameter choice, e.g. σ, p, sec we use the same values as in [31]. If q is the ciphertext modulus in [31] then $q' = q + \log_2(v) + 1$ is the ciphertext modulus needed for $v \times v$ matrix multiplications (cf. Section 5 for the theoretical discussion).

B CLASSICAL SACRIFICING AND MAC CHECK

We present the optimized sacrificing technique Π_{Sac} used in [31] (cf. Protocol 6). We also include the classical MacCheck Π_{MC} from [18] in Protocol 7.

Π_{Sac}

P_i has input $[a]_i \in A_p^\kappa$, $[b_1]_i, [b_2]_i \in A_p^\lambda$, $[c_1]_i, [c_2]_i \in A_p^\mu$.

1. Call $\mathcal{F}_{\mathrm{random}}$ to generate $r \in R$.
2. Call $\mathcal{F}_{[\![\cdot]\!]}$.LinearCombination. Each party P_i receives to $[\![\rho]\!]_i \leftarrow r[\![b_1]\!] - [\![b_2]\!]$. Open ρ with $\mathcal{F}_{[\![\cdot]\!]}$.Open
3. Open $\tau \leftarrow rc_1 - c_2 - \rho a$ with $\mathcal{F}_{[\![\cdot]\!]}$.Open. Abort if $\tau \neq 0$.
4. Call $\mathcal{F}_{[\![\cdot]\!]}$.Check on all opened values, abort if any check fails.

Protocol 6: Sacrificing in [31].

Π_{MC}

Every party P_i has $[\![y_j]\!]_i = ([y_j]_i, [\alpha y_j]_i, [\alpha]_i)$, $1 \leq j \leq l$. $y = (y_1, \dots, y_l) \in R^l$ is public and has to be checked. Denote $[\alpha y]_i \coloneqq ([\alpha y_1]_i, \dots, [\alpha y_l]_i)$.

1. The parties sample a random $r \in R^l$.
2. Each P_i sets $[\sigma]_i = r^t([\alpha y]_i - [\alpha]_i y)$ for r^t transpose of r.
3. Each P_i uses $\mathcal{F}_{\mathrm{commit}}$ to commit to her share $[\sigma]_i$.
4. After each party has committed, call $\mathcal{F}_{\mathrm{commit}}$ to open $[\sigma]_i$.
5. If $\sum_{i=1}^{n} [\sigma]_i \neq 0$ then abort.

Protocol 7: MAC Check in [18].

C PACKING METHODS FOR MATRICES

In this appendix we first repeat the diagonal packing method for matrices presented in [34] and [25].

Let $s = N/d$ be the number of plaintext slots and $r = \lfloor s/u \rfloor$. For $0 \leq t < r$ let $A_r = (a_{t,ij}) \in R^{u \times v}, B_r = (b_{t,jk}) \in R^{v \times w}, C_t = A_t B_t = (c_{t,ik}) \in R^{u \times w}$ where $0 \leq i < u, 0 \leq j < v, 0 \leq k < w$. Let $a_{t,j} = (a_{t,i,i+j \bmod v})_i \in R^u$, i.e. (secondary) diagonals of A. Write $a_{t,j}$ for fixed j into the plaintext slots of some plaintext $\boldsymbol{a}_j \in R^s$, i.e. $\boldsymbol{a}_j = (a_{\lfloor l/u \rfloor, l \bmod u, (l \bmod u + j) \bmod v})_l \in R^s$ for $0 \leq l < s$. Encrypt $\boldsymbol{a}_j$ with the usual BGV encryption scheme to and $\mathrm{Enc}_{\mathrm{pk}}(\boldsymbol{a}_j)$. Furthermore, let $b_{t,j,k} = (b_{t,i+j \bmod v,k})_i \in R^u$ and $\boldsymbol{b}_j = (b_{\lfloor l/u \rfloor, (l \bmod u + j) \bmod v, k})_l \in R^s$, i.e. we rotated the k-th column by j entries. Then the $(tu+i)$-th slot of $\sum_{j=1}^{v} \mathrm{Enc}_{\mathrm{pk}}(\boldsymbol{a}_j)\boldsymbol{b}_{j,k}$ decrypts to $c_{t,i,k} = \sum_{j=1}^{v} a_{t,i,i+j \bmod v} b_{t,i+j \bmod v,k}$ since the encryption scheme is linearly homomorphic. Hence we can use this packing technique to produce r matrix products with vk ciphertext-plaintext multiplications and v plaintext rotations. We use this packing in our implementation of Π_{Triple}.

In Π_{Special} and $\Pi_{\mathrm{Special\text{-}2}}$ we have to compute terms $\mathrm{Enc}_{\mathrm{pk}}(A)B + A\,\mathrm{Enc}_{\mathrm{pk}}(B)$. Unfortunately, the previous packing method cannot trivially be applied to this setup, since now we have to rotate ciphertexts slots of $\mathrm{Enc}_{\mathrm{pk}}(B)$. [22] showed that this is in fact possible for our cyclotomic ring, since the action of the Galois group $\mathrm{Gal}_{\mathbb{Q}}(\mathbb{Q}[\zeta_m])$ for a primitive m-th root ζ rotates the ciphertext slots.[17] Unfortunately, this rotation applied to a plaintext $(c_0, c_1) \in M_q^2$ also shifts the secret key, i.e. $\mathrm{Gal}_{\mathbb{Q}}(\mathbb{Q}[\zeta_m])$ acts as $X \mapsto X^g$ for some $g \in (\mathbb{Z}/m\mathbb{Z})^*$ and hence c_1 is rotated to $\mathrm{sk}(X^g)v(X^g) + w(X^g)p$. [23] solved this problem with their key switching algorithm, i.e. as part of the public key a party also provides a suitably formed term W such that $\mathrm{sk}\, W(\mathrm{sk}(X^g)v(X^g) + w(X^g)p) \bmod q = \mathrm{sk}(X^g)v(X^g) + vp \bmod q$ for a suitably small v that allows decryption. Of course, W cannot leak information on the secret key sk. Given that key switching protocol, we can after each ciphertext rotation, rotate the second component of the ciphertext back by W

[14] For a precise overview of the algebraic and number theoretic background you may consult [35].

[15] $x = (x_i)_i \in \{-1, 0, 1\}^N$ and $x_i = x_{i-} + x_{i+}$ with $x_{i\pm}$ uniformly from $\{\pm 1, 0\}$.

[16] [23] uses $h = 64$, Overdrive [31] adds sec for enhanced CPA-security, i.e. chooses $h' = h + \mathrm{sec}$. We will stick with the Overdrive convention. sec is the security parameter, e.g. sec = 40 or sec = 128.

[17] More precisely, representatives in $\mathrm{Gal}_{\mathbb{Q}}(\mathbb{Q}[\zeta_m])$ of $\mathrm{Gal}_{\mathbb{Q}}(\mathbb{Q}[\zeta_m])/\langle \mathrm{Frob}_{p^n} \rangle$ (since the (generalized) Frobenius automorphism Frob_{p^n} actually preserves the slots).

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

On input $(\text{Triple}, \text{id}_a, \text{id}_b, \text{id}_c)$ sample $a \longleftarrow M, b \longleftarrow M'$ and store $(\text{Val}[\text{id}_a], \text{Val}[\text{id}_b], \text{Val}[\text{id}_c]) = (a, b, c)$ where $c = ab \in M''$.

Protocol 8: Triple Generation.

Π_{LowGear}

P_i has input $[\boldsymbol{\alpha}]_i \in A_p, \text{Enc}_{\text{pk}'_j}([\boldsymbol{\alpha}]_j)$ for all $1 \leq j \leq n$.

1. Each party P_i samples uniformly $[a]_i, [b]_i, [b']_i \in A_p$.
2. Each (ordered) pair of parties (P_i, P_j) runs Π_{pair} where P_i has input $[a]_i$ and P_j has input $[b]_j, [b']_j$. P_i gets d_{ij} and d'_{ij}. P_j gets r_{ij} and r'_{ij}.
3. P_i locally sets $f_i := \sum_{j:j \neq i} r_{ji}, f'_i := \sum_{j:j \neq i} r'_{ji}$ and $e_i := [a]_i[b]_i + \sum_{j \neq i} d_{ij}, e'_i := [a]_i[b]_i + \sum_{j \neq i} d'_{ij}$, which are combined to $[c]_i = [ab]_i := e_i + f_i, [c']_i = [ab']_i := e'_i + f'_i$.
4. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where $\text{Enc}_{\text{pk}'_i}([\boldsymbol{\alpha}]_i)$ is provided by the preprocessing and P_j inputs $[a]_j, [b]_j, [c]_j, [b']_j, [c']_j$. P_i receives d_{ijk} for $k = 1, \ldots, 5$. P_j gets r_{ijk} for $k = 1, \ldots, 5$.
5. The results are locally combined as in Step 2. Each party receives $(\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i, \llbracket b' \rrbracket_i, \llbracket c' \rrbracket_i)$.

Protocol 9: Triple production in [31].

and we remain a ciphertext for sk. The resulting ciphertexts for the same key sk can than be summed up homomorphically as before.

Remark 8. *Please note that other packings might be advantageous for certain matrix dimensions u, v, w. E.g. if we pack rows $(a_{i,k+j})_j$ of a matrix A and diagonals $b_{k+j,k}$ then we get vectors of length w and hence can process s/w matrices in one go. Depending on $s \bmod u$, $w \bmod u$ this might lead to less unused slots.*

D CONVOLUTIONS

An often-used operation in ML are (tensor) convolutions. One way to securely realize a 2D convolution of an input tensor $A \in R^{h \times w \times c}$ and a kernel $B \in R^{c' \times h' \times w' \times c}$ is by representing the convolution as matrix multiplication. This is done, for example, in [11] to get the result

$$R^{h \times w \times c'} \ni C_{i,j,k'} = \sum_{i'=1}^{h'} \sum_{j'=1}^{w'} \sum_{k=1}^{c} A_{i+\delta_{i'},j+\delta_{j'},k} \cdot B_{k',i',j',k}$$

with a single matrix multiplication for $\delta_{i'} = i' - \lfloor h'/2 \rfloor, \delta_{j'} = j' - \lfloor w'/2 \rfloor, A_{i,j,k} := 0$ if $(i, j) \notin \{1, \ldots, h\} \times \{1, \ldots, w\}$. For this, we can define a $hw \times h'w'c$ matrix $X, X_{(i,j),(i',j',k)} = A_{i+\delta_{i'},j+\delta_{j'},k}$ and a $h'w's \times s'$ matrix $Y, Y_{(i',j',k),k'} = B_{k',i',j',k}$. Here, (i, j) uses the canonical mapping of $\{1, \ldots, h\} \times \{1, \ldots, w\}$ to $\{1, \ldots, hw\}$ and (i', j', k) analogously. We then have $C_{i,j,k'} = (X \cdot Y)_{(i,j),k'}$. We implemented convolutions for the networks in Section 7 like this in MP-SPDZ. Different approaches to compute convolutions with homomorphic encryption exists (e.g. [27]) and could be compared to the matrix-based approach in future work.

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

On input $(\text{ZKP}, \text{pk}_i, \text{slack})$ from two parties P_i, P_j the following can happen repeatedly:

1. P_i inputs $x \in A_p$.
2. If P_i is honest P_j receives $\text{Enc}_{\text{pk}_i}(x)$. Otherwise P_i receive $\text{Enc}'_{\text{pk}_i}(x)$ where Enc' has noise at most slack-times as much as regular encryption.
3. The adversary can abort any time.

Protocol 10: Zero-Knowledge Proof of Plaintext Knowledge.

Π_{Special}

Generate. Let $\phi : A_p^\kappa \longrightarrow A_p^\lambda$ be an R-linear map. P_i has input $[\boldsymbol{\alpha}]_i$ for all $1 \leq j \leq n$.

1. P_i samples $[a]_i \in A_p^\kappa$.
2. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 1, where P_i inputs $\text{Enc}_{\text{pk}_i}([a]_i)$. P_j locally computes $\text{Enc}_{\text{pk}_i}(\phi([a]_i))$.
3. Let offset $= j - i \bmod n$. If $(0 < 2\,\text{offset} < n) \vee (2\,\text{offset} = n \wedge j > i)$ the parties (P_i, P_j) run $\Pi_{\text{red-pair}}$, Step 2, where P_j inputs $\text{Enc}_{\text{pk}_i}([a]_i), \text{Enc}_{\text{pk}_i}(\phi([a]_i)), \phi([a]_j), [a]_j$. P_i receive d_{ij}. P_j gets r_{ij}.
4. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where P_j inputs $(\text{Enc}_{\text{pk}_i}([a]_i), [\boldsymbol{\alpha}]_j)$. P_i receive $\tilde{d}_{ij}$. P_j gets $\tilde{r}_{ij}$.
5. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where P_j inputs $(\text{Enc}_{\text{pk}_i}([\phi(a)]_i), [\alpha a]_j)$. P_i receives $\hat{d}_{ij}$. P_j gets $\hat{r}_{ij}$.
6. P_i locally combines the outputs to get $(\llbracket a \rrbracket, \llbracket a\phi(a) \rrbracket) \in A_p^\kappa \times A_p^\mu$.

Check. Each party P_i uses $\mathcal{F}_{\text{random}}$ to sample $[y_0]_i \in M$. The parties emulate $\mathcal{F}_{\llbracket \cdot \rrbracket}$ to authenticate $y_0 = \sum_{i=1}^n [y_0]_i$. Each party receives $\llbracket y_0 \rrbracket_i$. To check the MACs of l (components of) pair entries $\llbracket y_k \rrbracket, 1 \leq k \leq l$, the parties use $\mathcal{F}_{\text{random}}$ to generate $t \in R^l$. P_i opens $[z]_i = [y_0]_i + \sum_{k=1}^l t_k [y_k]_i$. The parties run $\mathcal{F}_{\llbracket \cdot \rrbracket}$.Check with input $z = \sum_{i=1}^n [z]_i$. If the check fails, abort.

Triple. The parties invoke **Generate** and receive s pairs $(\llbracket a \rrbracket, \llbracket a\phi(a) \rrbracket) \in A_p^\kappa \times A_p^\mu$. The parties invoke **Check** on these pairs. They store the s authenticated pairs if the check succeeds.

Protocol 11: Protocol for Special Tuples.

E FUNCTIONALITIES

This appendix contains the additional protocols discussed in Section 5. Also, the simulator for the proof of Theorem 5.1 can be found in Protocol 13. We additionally added ideal functionalities for our protocols which coincide mostly (and intentionally) with those of [31]. The functionalities $\mathcal{F}_{\text{random}}$ outputs the same random element to all parties; the functionality $\mathcal{F}_{\text{commit}}$ is a simple commitment functionality as presented in [18]. Both $\mathcal{F}_{\text{random}}$ and $\mathcal{F}_{\text{commit}}$ can be implemented as in [18]. $\mathcal{F}_{\text{Triple}}$ outputs a bilinear triple.

Finally, we also include a short proof why sending $\text{Enc}_{\text{pk}_i}([a]_1)$ in step 4 of Π_{Triple} forces parties to authenticate the same $[a]_1$ later in the same step:

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

$H = \{i \in \{1, \dots, N\} : P_i$ is honest$\}$ be the set of honest parties.

Generate.

1. For each (i, j) run Π_{pair}, Step 2 with
   - if $j \in H$ input $(\text{Enc}_{\text{pk}'_i}([\boldsymbol{\alpha}]_i), 0)$. Set $[\tilde{b}]_j := 0$.

   For each $i \in H$ receive and store d_{ij}. For each $j \in H$ store r_{ij}.
2. For each $i \in H$ locally combine the results to $[\alpha\tilde{b}]_i$.
3. For each (i, j) run Π_{pair} with:
   - if $i \in H$ input $[\tilde{a}]_i := 0$.
   - if $j \in H$ input $(0, [\alpha\tilde{b}]_j)$.

   Store the outputs d_{ijk} for $i \in H$, and r_{ijk} for $k = 1, 2$ and $j \in H$. Store all ciphertexts $\text{Enc}_{\text{pk}_i}([\tilde{a}]_i)$ for all parties $1 \leq i \leq n$.
4. For each (i, j) run Π_{pair}, Step 2 with:
   - if $j \in H$ input $(\text{Enc}_{\text{pk}_i}([\tilde{a}]_i), \boldsymbol{\alpha}_j)$.

   If $i \in H$ receive and store $\tilde{d}_{ij}$. If $j \in H$ store $\tilde{r}_{ij}$.
5. If $i \in H$ combine the outputs to $(\llbracket\tilde{a}\rrbracket_j, \llbracket\tilde{b}\rrbracket_j, \llbracket\tilde{c}\rrbracket_j)$.

Check.

1. Use $\mathcal{F}_{\text{random}}$ to sample $[y_0]_i \in R$ for each $i \in H$. Emulate $\mathcal{F}_{\llbracket\cdot\rrbracket}$ to authenticate $y_0 = \sum_{i=1}^{n} [y_0]_i$. Store $\llbracket y_0 \rrbracket_i$ for each $i \in H$.
2. Emulate $\mathcal{F}_{\text{random}}$ to generate $t \in R^l$. Send $[z]_i = [y_0]_i + \sum_{k=1}^{l} t_k [y_k]_i$ to the adversary for all $i \in H$. Receive $[z]_j$ from the adversary for all $j \notin H$. Run $\mathcal{F}_{\llbracket\cdot\rrbracket}$.Check with input $z = \sum_{i=1}^{n} [z]_i$. If the check fails, abort.
3. Rewind the adversary for random $t \in R^l$ to reconstruct $\sum_{j \notin H} [y_k]_j$ for each $1 \leq k \leq l$.

Triple. Invoke **Generate** and receive s triples $(\llbracket\tilde{a}_j\rrbracket, \llbracket\tilde{b}_j\rrbracket, \llbracket\tilde{c}_j\rrbracket) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$ for $1 \leq j \leq s$. Invoke **Check** on these triples. For each $1 \leq j \leq s$ emulate $\mathcal{F}_{\text{Triple}}$ to receive $(a_j, b_j, c_j) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$ with $a_j b_j = c_j$. Adapt the shares $(\llbracket a_j \rrbracket_i, \llbracket b_j \rrbracket_i, \llbracket c_j \rrbracket_i)$ of honest parties $i \in H$ suitably (given the shares from adversarial parties determined by rewinding in **Check**). Store the s authenticated triples if the check succeeds, else abort.

Protocol 13: Simulator for Π_{Triple}.

$\Pi_{\text{Special-2}}$

Generate. P_i has input $[\boldsymbol{\alpha}]_i, \text{Enc}_{\text{pk}'_j}([\boldsymbol{\alpha}]_j)$ for all $1 \leq j \leq n$.

1. P_i samples $[a]_i \in A_p^\kappa$.
2. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where P_j inputs $(\text{Enc}_{\text{pk}'_i}([\boldsymbol{\alpha}]_i), [a]_j)$. P_i receives d_{ij}. P_j has r_{ij}.
3. The results are locally combined by P_i to $[\alpha a]_i$.
4. Let offset $= j - i \bmod n$. If $(0 < 2\,\text{offset} < n) \vee (2\,\text{offset} = n \wedge j > i)$ the parties (P_i, P_j) runs $\Pi_{\text{red-pair}}$, where P_i inputs $[a]_i$ and P_j inputs $\phi([a]_j), [a]_j$, and $\text{Enc}_{\text{pk}_i}(\phi([a]_i))$ computed locally after Step 1. P_i receive d_{ij}. P_j has r_{ij}.
5. Each (ordered) pair (P_i, P_j) invokes Π_{pair}, Step 2, where P_j inputs $(\text{Enc}_{\text{pk}_i}([\phi(a)]_i), [\alpha a]_j)$. P_i receives $\hat{d}_{ij}$. P_j has $\hat{r}_{ij}$.
6. P_i combines the outputs to $(\llbracket a \rrbracket, \llbracket a\phi(a) \rrbracket) \in A_p^\kappa \times A_p^\mu$.

Check. Run Π_{Special}.Check.
Triple. Run Π_{Special}.Generate.

Protocol 14: Alternative Protocol for Special Tuples.

$\mathcal{F}_{\llbracket\cdot\rrbracket}$

The dictionary Val keeps track of authenticated values. For simplicity entries of Val cannot be changed. Val is indexed by Val . Keys. Entries of Val are elements of R.

Input. On input $(\text{Input}, \text{id}_1, \dots, \text{id}_m, x_1, \dots, x_m, P_i)$ from P_i and $(\text{Input}, \text{id}_1, \dots, \text{id}_m, P_j)$ from all other parties, set $\text{Val}[\text{id}_j] \leftarrow X_j$ for all $1 \leq j \leq m$.

Linear Combination. Given $(\text{LC}, \text{id}_{\text{LC}}, (\text{id}_j)_{1 \leq j \leq m}, (a_j)_{0 \leq j \leq m})$ for $a_j \in R$ from all parties with $\text{id}_j \in \text{Val}\,.\,\text{Keys}$ for all $1 \leq j \leq m$, set $\text{Val}[\text{id}_{\text{LC}}] = a_0 + \sum_{j=1}^{m} a_j \,\text{Val}[\text{id}_j]$.

Open. If (Open, id) from all parties, send Val[id] to Adv. After receiving X from Adv return X to all parties.

Check. Given $(\text{Check}, (\text{id}_j)_{1 \leq j \leq m}, (x_j)_{1 \leq j \leq m})$ from all parties, if Adv sends Ok and $\text{Val}[\text{id}_j] = x_j$ for all $1 \leq j \leq m$, send Ok to everyone. Else abort $\perp$.

Abort. On input $\perp$ from Adv, send $\perp$ to all parties.

Protocol 15: Input Functionality.

$\Pi_{\text{Triple-2}}$

Generate. P_i has input $[\boldsymbol{\alpha}]_i, \text{Enc}_{\text{pk}_j}([\boldsymbol{\alpha}]_j)$ for all $1 \leq j \leq n$.

1. P_i samples $[a]_i \in A_p^\kappa$, $[b]_i \in A_p^\lambda$.
2. Let offset $= j - i \bmod n$. If $(0 < 2\,\text{offset} < n) \vee (2\,\text{offset} = n \wedge j > i)$ the parties (P_i, P_j) run:
   a. P_i sends $\text{Enc}_{\text{pk}'_i}([a]_i), \text{Enc}_{\text{pk}'_i}([b]_i)$ invoking $\mathcal{F}_{\text{ZKP}}$.
   b. Π_{pair}, Step 2, twice where P_j inputs $(\text{Enc}_{\text{pk}'_i}([a]_i), [\boldsymbol{\alpha}]_j)$ and $(\text{Enc}_{\text{pk}'_i}([b]_i), [\boldsymbol{\alpha}]_j)$. P_i gets d'_{ij1} and d'_{ij2}. P_j gets r'_{ij1} and r'_{ij2}.
   c. Π_{pair}, Step 2, where P_i inputs $(\text{Enc}_{\text{pk}_i}([\boldsymbol{\alpha}]_j), [a]_j, [b]_j)$. P_i gets d''_{ij1} and d''_{ij2}. P_j gets r''_{ij1} and r''_{ij2}.
   d. The results are locally combined by P_i to $\llbracket a \rrbracket_i, \llbracket b \rrbracket_i$. P_i sends $\text{Enc}_{\text{pk}'_i}([\alpha b]_i)$ invoking $\mathcal{F}_{\text{ZKP}}$.
   e. $\Pi_{\text{red-pair}}$, Step 2, where P_j inputs $\text{Enc}_{\text{pk}'_i}([a]_i)$, $\text{Enc}_{\text{pk}'_i}([b]_i), a_j, b_j$. P_i gets d_{ij1}. P_j gets r_{ij1}.
   f. $\Pi_{\text{red-pair}}$, Step 2, where P_j inputs $\text{Enc}_{\text{pk}'_i}([a]_i)$, $\text{Enc}_{\text{pk}'_i}([\alpha b]_i), a_j, \alpha b_j$. P_i gets d_{ij2}. P_j gets r_{ij2}.
3. The results are locally combined by P_i to $\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, [c]_i$.
4. P_i combines the outputs to $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket) \in A_p^\kappa \times A_p^\lambda \times A_p^\mu$.

Check. Invoke Π_{Triple}.Check.
Triple. Invoke Π_{Triple}.Triple.

Protocol 12: New LowGear-type Protocol.

Security Game for Authentication. The challenger C samples $[a]_1, [\alpha]_1, r_{12}$ and sends an encryption $\text{Enc}_{\text{pk}_1}([a]_1)$ under her public key to the adversary $\mathcal{A}$. $\mathcal{A}$ sends $\text{Enc}_{\text{pk}_2}([a]_2)$ with $\mathcal{F}_{\text{ZKP}}$ (under his public key), and additionally $\text{Enc}_{\text{pk}_1}([a]_1)([\alpha]_2 + \delta_2) - r_{21}$. C returns $\text{Enc}_{\text{pk}_1}([a]_2)[\alpha]_1 - r_{12}$. Set $a = [a]_1 + [a]_2$, $\alpha = [\alpha]_1 + [\alpha]_2$. Then $\mathcal{A}$ has to provide $\delta_1 \neq 0, \Delta$ such that $\alpha a + [a]_1\delta_2 + \Delta = \alpha(a + \delta)$. Note that this challenge is equivalent to finding $\delta_1 \neq 0, \delta_2, \Delta$ such that $\alpha\delta_1 = [a]_1\delta_2 + \Delta$, i.e. to guessing α correctly, which has probability $\frac{1}{|R|}$. Also note that $\mathcal{A}$ has no information on $[a]_1$ (and hence on a) by the CPA-security of the encryption scheme; he has no information on a $\text{Enc}_{\text{pk}}(\alpha_1)$ (with pk instead of pk$'$); he has no information on $[\alpha]_1$ (and hence on α) by the information-theoretically secure masking with r_{12}.