Efficient Information-Theoretic Multi-Party Computation over Non-Commutative Rings

Daniel Escudero[1] and Eduardo Soria-Vazquez[2]

[1] Dept. Computer Science, Aarhus University, Denmark.
[2] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE.[†]
escudero@cs.au.dk, eduardo.soria-vazquez@tii.ae

Abstract. We construct the first efficient, unconditionally secure MPC protocol that only requires black-box access to a non-commutative ring R. Previous results in the same setting were efficient only either for a constant number of corruptions or when computing branching programs and formulas. Our techniques are based on a generalization of Shamir's secret sharing to non-commutative rings, which we derive from the work on Reed Solomon codes by Quintin, Barbier and Chabot (*IEEE Transactions on Information Theory, 2013*). When the center of the ring contains a set $A = \{\alpha_0, \ldots, \alpha_n\}$ such that $\forall i \neq j, \alpha_i - \alpha_j \in R^*$, the resulting secret sharing scheme is strongly multiplicative and we can generalize existing constructions over finite fields without much trouble.

Most of our work is devoted to the case where the elements of A do not commute with all of R, but they just commute with each other. For such rings, the secret sharing scheme cannot be linear "on both sides" and furthermore it is not multiplicative. Nevertheless, we are still able to build MPC protocols with a concretely efficient online phase and black-box access to R. As an example we consider the ring $\mathcal{M}_{m\times m}(\mathbb{Z}/2^k\mathbb{Z})$, for which when $m > \log(n+1)$, we obtain protocols that require around $\lceil\log(n+1)\rceil/2$ less communication and $2\lceil\log(n+1)\rceil$ less computation than the state of the art protocol based on Circuit Amortization Friendly Encodings (Dalskov, Lee and Soria-Vazquez, *ASIACRYPT 2020*).

In this setting with a "less commutative" A, our black-box preprocessing phase has a less practical complexity of $\mathsf{poly}(n)$. We fix this by additionally providing specialized, concretely efficient preprocessing protocols for $\mathcal{M}_{m\times m}(\mathbb{Z}/2^k\mathbb{Z})$ that exploit the structure of the matrix ring.

1 Introduction

Multiparty Computation, or MPC for short, is a collection of techniques that enable a set of mutually distrustful parties $P_1, \ldots, P_n$ to securely compute a given function f on private inputs $x_1, \ldots, x_n$, while revealing only the output of the computation. Security is formalized by considering an adversary that corrupts t of the parties, and aims at learning as much as possible from the honest parties' inputs either by only seeing the messages corrupt parties send/receive without changing their behavior (passive adversary), or by arbitrarily deviating from the protocol specification (active adversary). Security requires that the adversary does not learn anything about the honest parties' inputs beyond what is possibly leaked by the output of the computation. MPC protocols exist in a wide variety of settings, and some very interesting ones are the settings in which $t < n/2$ and the stronger one in which $t < n/3$. It is well known that in these scenarios, MPC protocols whose security is completely independent of the hardness of any computational problem can be devised, and with the lack of these computational problems typically more efficiency is gained. These protocols are called information-theoretic protocols.

Many information-theoretic protocols exist in the $t < n/2$ and $t < n/3$ regimes, for which computation is primarily represented as an arithmetic circuit whose gates involve additions and

[†]Work partially done while at Aarhus University, Denmark.

multiplications over a finite ring. Traditionally, this ring has been restricted to be a finite field, since the lack of zero divisors simplifies protocol design and opens for a vast literature of algebraic tricks which can ensure that an active adversary does not cheat during the protocol. A recent line of work [ACD+19, DLS20, CRX19, ED20] designs protocols that operate over non-field rings, namely $\mathbb{Z}_{2^k}$ (integers modulo 2^k), and Galois ring extensions $\mathsf{GR}(2^k, d)$ of these. The use of these rings is well motivated in practice due to their direct compatibility with hardware and their natural affinity with binary-based protocols like binary decomposition, secure comparison or secure truncation for fixed-point arithmetic. It is not hard to generalize the techniques presented in [ACD+19] to more general *commutative* rings, as long as the so-called Lenstra constant[1] of the ring is large enough. However, in spite of the recent progress in the design of MPC protocols over non-field rings, *non-commutative* rings have been mostly overlooked in the literature.

Studying non-commutative rings is a well motivated theoretical question, since it explores what are the minimal assumptions required on an algebraic structure so that MPC protocols can be naturally defined over it. Furthermore, there are some non-commutative rings that are very suitable for practical applications. For instance, matrix rings are very useful for applications based on linear algebra, which include statistics as well as the training and evaluation of different kinds of machine learning models. Another example are quaternion rings, which are particularly advantageous for describing rotations in a three-dimensional space. Due to this feature, quaternions are a useful tool in the domains of computer graphics, robotics and aerospace, including satellite navigation.

Motivated by the above, in this work we attack the question of designing *efficient* information-theoretic MPC protocols which work *directly* over not-necessarily-commutative finite rings.

1.1 Theoretical Contributions

First, we observe that feasibility results for MPC have been established already since the 80s (e.g. [BGW88]), so in principle we could make use of any existing MPC protocol that allows computing *any* function in order to emulate arithmetic over any given ring R. However, we notice that this requires white-box access to the representation of elements in R, and moreover, it is unlikely to lead to efficient protocols if there is no certain "compatibility" between the ring R and the domain used for the underlying MPC protocol. For example, if R is a matrix ring over the integers modulo a prime, and the domain of computation of the given protocol is $\mathbb{Z}_2$, then there is a large overhead incurred in emulating each single addition and multiplication modulo a prime using binary circuits.

Given the above, we propose *efficient* unconditionally secure MPC protocols over rings containing big enough exceptional sets A which satisfy some additional commutativity properties. Our most general results only requires *black-box* access to the ring, which we start by precisely defining.

Definition 1. *We say that a protocol has* black-box access *to a ring R, or simply that it is* black-box*, if it only requires black-box access to the ring operations and the elements of a particular exceptional set A. Furthermore, we assume that it is efficient both to sample elements from R and to invert elements from R^*.*

Our protocols are based on a generalization of Shamir's linear secret sharing scheme to non-commutative rings. Prior to our work, Quintin, Barbier and Chabot [QBC13] showed how to con-

[1] An exceptional set is a subset of ring elements whose non-zero pairwise differences are invertible. The Lenstra constant of a ring is the size of the largest exceptional set.

struct Reed Solomon codes over rings that do not need to be commutative. By reinterpreting their results under the lenses of linear secret-sharing schemes (LSSS's), we first obtain the following result.

Theorem 1 (Theorem 4, restated). *Let R be a ring such that $Z(R)$ contains an exceptional set of size at least $n+1$ and let $t < n/3$. Then, we can define a Shamir-style strongly multiplicative linear secret sharing scheme over R.*

This is discussed in Section 3. Given a ring satisfying the hypothesis of the previous theorem, we can adapt the perfectly secure protocol by Beerliova and Hirt [BTH08].

Corollary 1 (Corollary 2, restated). *Let R be a ring such that $Z(R)$ contains an exceptional set of size at least $2n$. Let $\mathcal{A}$ be an active adversary corrupting $t < n/3$ parties. There exists a perfectly secure, black-box MPC protocol with an amortized communication complexity of $O(n)$ ring elements per gate.*

While interesting, the previous result leaves out of the picture several non-commutative rings. For example, the centre of the ring of matrices over $\mathbb{Z}_{2^k}$, which is widely used in applications involving linear algebra, like machine learning, has a Lenstra constant of 2, which is not large enough to apply the results highlighted above. We fix this by relaxing the commutativity requirements for the elements of the exceptional set: instead of requiring this exceptional set to be a subset of the centre of the ring, we only ask the elements of the exceptional set to commute with each other. It is in this setting that the previous results on Reed-Solomon codes [QBC13] do not help us as much. The resulting code (i.e. secret sharing scheme) is not linear "on both sides", but rather only when multiplied by scalars either on the left or on the right. Even more disastrously, the left-or-right LSSS that we obtain is not multiplicative, which rules out standard techniques to achieve unconditionally secure MPC. Considering all of this, most of our work relates to proving the following Theorem:

Theorem 2 (Informal). *Let R be a ring and $A = \{\alpha_0, \ldots, \alpha_n\} \subseteq R$ an exceptional set such that $\forall \alpha_i, \alpha_j \in A, \alpha_i \cdot \alpha_j = \alpha_j \cdot \alpha_i$. Let $\mathcal{A}$ be an active adversary corrupting t parties. For $t < n/2$ and $t < n/3$, there exist efficient, information-theoretically secure, black-box MPC protocols over R. The amortized communication complexity of their* online phase *is $O(n)$ ring elements per gate.*

The black-box online phase of our protocol is described in Section 4, while the black-box offline phase is presented in Section 5.1.

1.2 Concretely Efficient Protocols for $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$

Beyond their theoretical interest, our techniques also have relevance in the context of concretely efficient MPC. As an example for this, we provide constructions for the important ring $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, the ring of $m \times m$ matrices with entries modulo 2^k, which improve upon the concrete efficiency of the online phase of the state of the art protocols in the same setting but without black-box access to R. As mentioned before, the centre of this ring does not have a large enough exceptional set, so the MPC techniques based on the multiplicativity of Shamir secret sharing (which are quite efficient) cannot be applied.

Given this, our approach is to make use of the black-box protocol from Theorem 2. First, we show that, whenever $m \geq \log(n+1)$, the hypothesis of this theorem is satisfied. This is due to the fact that $\mathsf{GR}(2^k, m)$ is a (commutative) subring of R and the Lenstra constant of $\mathsf{GR}(2^k, m)$

is precisely 2^m. Second, we show how to replace the black-box preprocessing from Theorem 2, which achieves only $\mathsf{poly}(n)$ communication complexity, by a more tailored preprocessing protocol for $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$. This is done in Section 5.2. Finally, we also show how to do efficient error-correction of Shamir shares in this setting, which is described in Section B.1 in the Appendix.

By following Theorem 2 we obtain a very efficient online phase, as described by the (generic) protocols we provide in Section 4. For this part of the protocol execution, the fact of having a secret sharing scheme directly over R is a significant efficiency advantage: each share of a secret is a single element of R, and arithmetic happens at the level of R. However, the price to pay for such an efficient online phase, which overcomes the issues of lacking multiplicativity, is that the offline phase becomes much more complex.

In Section 5.2 we show how to compute the required preprocessing material for $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ by, intuitively, secret-sharing each entry of a matrix and then leveraging existing works on MPC over $\mathbb{Z}_{2^k}$ [ACD+19, DLS20]. As we need to compute the product between secret-shared matrices and retain information-theoretic security, this is our best approach for concrete efficiency. Secret-sharing each entry of a matrix in $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ individually would require us to move to a Galois extension of $\mathbb{Z}_{2^k}$ of degree $d \simeq \log(n+1)$, which would add such overhead in terms of communication and a worse one for computation. Instead of naively secret-sharing each entry, we amortize the asymptotic communication cost of working over such Galois extension by using the Circuit Amortization Friendly Encodings (CAFEs) introduced in [DLS20]. Intuitively, what this means is that chunks of every row/column of each matrix will be secret-shared as a single Galois Ring element.

At this point, one could ask why not work with this type of secret-sharing for the whole protocol execution, rather than just the preprocessing phase. The CAFE for inner products that we use in our preprocessing phase allows to "pack" approximately $d/2$ elements from $\mathbb{Z}_{2^k}$ into $\mathsf{GR}(2^k, d)$, so that seems to be a small overhead. Some arguments for not following this route are as follows. First of all, the protocol from [DLS20] is only fully detailed for double sharings and in the case of security with abort. Our online protocol, on the other hand, has guaranteed output delivery (if $t < n/3$) and uses multiplication triples. Furthermore, it is most efficient when using a function-dependent preprocessing in the style of [BNO19, ED20]. These differences make a fair comparison more difficult, but even assuming an adaptation of [DLS20] to the function-dependent techniques of [BNO19, ED20], our online phase remains more efficient in the following aspects.

- Secret sharing input values: For the adapted [DLS20], when parties provide inputs to the computation, it would be important to check that they are rightly encoded. This issue is not specific to CAFEs, but merely to the fact of having to work over an extension ($\mathsf{GR}(2^k, d)$) of the ring one is actually interested in ($\mathbb{Z}_{2^k}$). The check can be performed by using preprocessing material, but it increases the communication and round complexity of the protocol. Our online phase does not need to perform any such check, as it works directly over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$.
- Computing the product of two secret-shared matrices, communication: In our work, this requires to reconstruct two secrets in $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$. An adapted [DLS20], would need to reconstruct one element of $\mathsf{GR}(2^k, d)$ *per entry* of the matrix. Hence, in terms of communication we are around $d/2$ times better in our work.
- Computing the product of two secret-shared matrices, computation: Our work benefits from the fact that the shares each party holds, as well as the operations they perform on them, are in $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$. This has the advantage that an implementation of our protocol can fully exploit existing libraries for matrix arithmetic, which is quite efficient due to its relevance in multiple practical settings. On the other hand, using a potential extension of the techniques of [DLS20] in

the online phase would require computation on Galois ring elements, which is way less studied than matrix arithmetic and it is also more inefficient. Each multiplications of two Galois ring elements costs d^2 operations in $\mathbb{Z}_{2^k}$, or potentially $d\log(d)$ using FFT-based techniques. This is not very concretely efficient, as explored experimentally in [DEK21], for example.

Instead of using CAFEs, Reverse Multiplication Friendly Embeddings (RMFEs) [CCXY18] could have been chosen. A generalization of the interpolation-based RMFE from [CCXY18] was presented in [DLS20] and the constructions based on algebraic geometric codes were also lifted to rings in [CRX19]. Whereas RMFEs are much computationally heavier than CAFEs, they can provide a slightly better communication complexity. More concretely, if δ is the amount of elements from the base ring that the RMFE can "pack", RMFEs incur in a communication complexity for the product of secret shared matrices that would be only d/δ (rather than $d/2$ as in CAFEs), still worse than ours[2]. On the other hand, using RMFEs requires two sequential openings per matrix multiplication, rather than a single one as CAFEs do. This is due to a resharing operation to compute $\psi(\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}))$ in RMFEs. In [ACE+21], the authors show how to reduce this to a single round, but they just provide a solution for $t < n/2$ which can only achieve security with abort.

In a nutshell, by using our seemingly theoretical tools, we are able to build MPC protocols which have a more efficient online phase than the state of the art protocols, while retaining a comparable preprocessing.

1.3 Related Work

There are a few works on MPC over non-abelian *groups*, rather than rings. Hence, we are not interested in those. These include [DPS+12] and [CDI+13]. Note that [CDI+13] additionally provides constructions for *commutative* rings.

MPC over non-commutative rings has been discussed in [CFIK03], but their results related to MPC from (multiplicative) Monotone Span Programs are restricted to (algebras over) commutative rings. They only seem to take care of the non-commutative case in Sections 4.2. and 4.3., which deal only with branching programs and formulas, rather than circuits.

Although not mentioned explicitly in [BBY20], the basic building blocks (secure addition and multiplication) presented in that work for MPC based on replicated secret-sharing also work over a non-commutative ring. However, these techniques differ from ours in several ways. First, they use computational assumptions (PRFs) in order to improve their overall efficiency. Second, as it is inherent for MPC based on replicated secret-sharing, the communication complexity does not scale well as the number of parties increases. More precisely, each share consists of $\binom{n}{t}$ ring elements, which is exponential in n whenever $t = n/c$ for some $c > 1$, since $\binom{n}{n/c} \geq (c^{1/c})^n$.[3]

2 Preliminaries

Notation. Sometimes, we use $[n]$, where $n \in \mathbb{N}$, to represent $\{1, 2, \ldots, n\}$. We write $x \xleftarrow{\$} \mathcal{X}$ to denote sampling a value x uniformly from the set $\mathcal{X}$. We write $\mathbb{Z}_{p^k}$ to denote the ring of integers modulo p^k and $\mathcal{M}_{r\times c}(R)$ to refer to the ring of $r \times c$ matrices over R.

[2] Asymptotically, [CRX19] shows that $\delta = d/4.92$ by lifting the algebraic geometry based construction from [CCXY18].

[3] Here we use the well known inequality $\binom{a}{b} \geq (a/b)^b$.

2.1 Multiparty Computation

We consider secure evaluation of functions $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ given by arithmetic circuits with addition and multiplication gates defined over a finite ring R, where party P_i is supposed to learn y_i.

The security of our protocols is proven in the UC framework by Cannetti [Can01]. We assume secure, synchronous channels, and we deal with active, static adversaries. In a nutshell, the adversary corrupts a subset of t parties actively, arbitrarily changing their behavior during the execution of the protocol. The adversary, also known as an *environment*, additionally provides the inputs for all the parties.A given protocol Π instantiates a given functionality $\mathcal{F}$, if there exists a simulator $\mathcal{S}$ who, by interacting with the adversary and with the functionality $\mathcal{F}$, creates an execution (called the *ideal* execution) that is indistinguishable to the adversary from the real execution in which the actual honest parties are running the protocol Π.

If the distributions in the two executions are exactly the same, then we say that Π instantiates $\mathcal{F}$ with perfect security. In contrast, if the distributions are only negligibly apart (in some security parameter κ), then we say that Π instantiates $\mathcal{F}$ with statistical security. Finally, sometimes we consider *hybrid* models in which a protocol Π instantiates a functionality $\mathcal{F}$, assuming access to another functionality $\mathcal{F}'$. In this case we say that Π instantiates $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model. See [Can01] for details.

In this work we consider a broadcast functionality $\mathcal{F}_{\text{BC}}$ that receives an input from a designated sender and relays this exact same value to all the parties.

We will take into account two functionalities for MPC. One is $\mathcal{F}_{\text{MPC-GOD}}$, which receives inputs $x_1, \ldots, x_n$ from the parties, computes the given function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, which is represented as an arithmetic circuit over a ring R composed of addition and multiplication gates, and returns the output y_i to each party P_i. The second functionality is $\mathcal{F}_{\text{MPC-abort}}$, which is defined as $\mathcal{F}_{\text{MPC-GOD}}$, except that, before delivering output to the parties, it waits for a message from the adversary. If the message is `abort`, then the functionality sends `abort` to all the parties. Else, if the message if `ok`, then the functionality sends the output y_i to each party P_i. In a real execution, when we say that an honest party "aborts", it means that this party sends an abort signal to all the parties using $\mathcal{F}_{\text{BC}}$ and then outputs `abort`. A party aborts upon receiving an abort signal through the broadcast channel.

2.2 Background in Ring Theory

We turn to recall some useful results from ring theory. Outside of this section, whenever we talk about a ring R, we mean a finite ring with identity $1 \neq 0$ for which we do *not* assume commutativity. During this specific section we do not assume finiteness, so that it is clear which results require such hypothesis.

Working over these general rings hides subtleties which do not appear in the field case. Besides the lack of commutativity, one has to be careful about the fact that the rings we consider contain zero divisors. Moreover, it is important to reconsider what it means to be a unit. We recap some basic definitions and results in this area of algebra.

Definition 2. *Let R be a ring. An element $a \in R$ is a unit if there exists $b \in R$ such that $a \cdot b = b \cdot a = 1$. The set of all units is denoted by R^*.*

An element $a \in R \setminus \{0\}$ is a left (resp. right) zero divisor if $\exists\ b \in R \setminus \{0\}$ such that $a \cdot b = 0$ (resp. $b \cdot a = 0$). In this work, whenever we say that $a \in R \setminus \{0\}$ is a zero divisor we mean that a is both a left and right zero divisor.

Lemma 1. *1. $a \in R^*$ if and only if a is both left-invertible and right-invertible.*
2. If a has a right inverse, then a is not a right zero divisor.
3. If R is finite, then every element which has a right inverse is a unit.

Proof. (1) The first implication is trivial. For the opposite direction, let $b, c \in R$ such that $a \cdot b = c \cdot a = 1$. We have that $c = c \cdot (a \cdot b) = (c \cdot a) \cdot b = b$.

(2) Denote by $b \in R$ the element such that $a \cdot b = 1$. Assume by contradiction the existence of $c \in R \setminus \{0\}$ such that $c \cdot a = 0$. Then we would have that $c = c \cdot (a \cdot b) = (c \cdot a) \cdot b = 0$.

(3) See Appendix A, after understanding the notion of polynomial ring and evaluation maps in Section 2.3. ∎

Lemma 2. *Let R be a finite ring. Then all non-zero elements of R are either a unit or a zero divisor.*

Proof. See Appendix A, after understanding the notion of polynomial ring and evaluation maps in Section 2.3. ∎

Some elements of a non-commutative ring have better commutative properties than other. The two following definitions allow us to name them.

Definition 3. *The center of a ring R, denoted by $Z(R)$ consists of the elements $a \in Z(R)$ such that $\forall b \in R$, $ab = ba$.*

Definition 4 ([QBC13]). *Let $A = \{a_1, \ldots, a_n\} \subset R$. We say that A is a commutative set if $\forall a_i, a_j \in A, a_i \cdot a_j = a_j \cdot a_i$.*

Exceptional sets. Elements which satisfy that their pairwise differences are invertible will be fundamental in our constructions. These have received different names in the literature: 'subtractive sets' in [QBC13], 'exceptional sequences' in [ACD+19] and 'exceptional sets' in [DLS20]. We will stick with the latter denomination.

Definition 5. *Let $A = \{a_1, \ldots, a_n\} \subset R$. We say that A is an exceptional set if $\forall i \neq j, a_i - a_j \in R^*$. We define the Lenstra constant of R to be the maximum size of an exceptional set in R.*

2.3 Polynomials over Non-Commutative Rings

Definition 6 (Polynomial Ring). *Let R be a ring and $A \subseteq R$. The set of polynomials over A of degree at most d is given by $A[\mathtt{X}]_{\leq d} = \{f(\mathtt{X}) = \sum_{i=0}^{d} a_i \cdot \mathtt{X}^i \mid a_i \in A\}$. The set of polynomials over A is $A[\mathtt{X}] = \cup_{d \geq 0} A[\mathtt{X}]$. Given two polynomials $a(\mathtt{X}) = \left(\sum_{i=0}^{d} a_i \cdot \mathtt{X}^i\right)$, $b(\mathtt{X}) = \left(\sum_{j=0}^{d'} b_j \cdot \mathtt{X}^j\right)$, the ring R induces the following operations:*

1. *$c(\mathtt{X}) = a(\mathtt{X}) + b(\mathtt{X}) = \sum_{k=0}^{\max\{d,d'\}} (a_k + b_k) \cdot \mathtt{X}^k$, where $a_k = 0$ for $k > d$ and $b_k = 0$ for $k > d'$.*

2. $c(\mathtt{X}) = a(\mathtt{X}) \cdot b(\mathtt{X}) = \sum_{k=0}^{d+d'} c_k \cdot \mathtt{X}^k$, *where*

$$r_k = \sum_{\substack{i+j=k \\ 0 \leq i \leq d,\ 0 \leq j \leq \tilde{d}}} p_i \cdot q_j.$$

$$\ell_j(X) = \prod_{\substack{i=1 \\ i \neq j}}^{4} (X - \alpha_j) \cdot (\alpha_i - \alpha_j)^{-1}.$$

Furthermore, when A is a ring, so is $A[\mathtt{X}]$.

Our definition of the product in a polynomial ring imposes that "the indeterminate $\mathtt{X}$ commutes with the coefficients". Otherwise, when formally multiplying two polynomials we would encounter terms of the form $a_i \mathtt{X}^i b_j \mathtt{X}^j$, which could not be turned into $a_i b_j \mathtt{X}^{i+j}$. Allowing the indeterminate to commute with coefficients, rather than keeping everything non-commutative, allows us to prove a series of results leading to the existence and uniqueness of interpolating polynomials. On the other hand, granting this small commutativity property to polynomials requires to consider their evaluation more carefully, as we will see next.

Definition 7 (Evaluation Maps). *Let $f = \sum_{i=0}^{d} f_i \mathtt{X}^i \in R[\mathtt{X}]$ and $a \in R$. We define the* evaluation at a on the right *(resp.* left*) map $f^{R}(a)$ (resp. $f^{L}(a)$) as follows:*

$$\cdot^{R}(a) : R[\mathtt{X}] \to R \qquad\qquad \cdot^{L}(a) : R[\mathtt{X}] \to R$$

$$f \mapsto f^{R}(a) = \sum_{i=0}^{d} f_i a^i \qquad\qquad f \mapsto f^{L}(a) = \sum_{i=0}^{d} a^i f_i$$

We say that a is a right (rep. left) root whenever $f^{R}(a) = 0$ (resp. $f^{L}(a) = 0$). We use $f(a)$ to denote $f^{R}(a)$.

The evaluation maps above are additive homomorphisms but, in general, they are not ring homomorphisms. This is because, as mentioned above, in polynomial multiplication the indeterminate $\mathtt{X}$ commutes with the coefficients. It is important to keep in mind that we are dealing with *polynomials* as formal objects of their own, rather than confusing them with *polynomial functions* (where a "variable" $\mathtt{X}$ is "instantiated" with $a \in R$ when evaluating the polynomial) as one usually does in commutative rings. Fortunately, there are some cases in which some notion of multiplicative homomorphism holds for the evaluation maps, as described in the following lemma.

Lemma 3. *Let $f \in R[\mathtt{X}]$.*

1. *Let A be a commutative set. If $g \in (A \cup Z(R))[\mathtt{X}]$ and $a \in A$, then $(f \cdot g)^{R}(a) = f^{R}(a) \cdot g^{R}(a)$ and $(g \cdot f)^{L}(a) = g^{L}(a) \cdot f^{L}(a)$.*
2. *Let $g \in R[\mathtt{X}]$. If $a \in Z(R)$, for all $h \in R[\mathtt{X}]$, $h^{R}(a) = h^{L}(a)$. Furthermore, $(f \cdot g)^{R}(a) = f^{R}(a) \cdot g^{R}(a)$.*

Proof. We only prove the first part of the first statement, as the same reasoning applies for the rest of the claims. Let $a \in A$ and let $f(\mathtt{X}) = \left(\sum_{i=0}^{d} f_i \cdot \mathtt{X}^i\right) \in R[\mathtt{X}]$ and $g(\mathtt{X}) = \left(\sum_{j=0}^{d'} g_j \cdot \mathtt{X}^j\right) \in$

$(A \cup Z(R))[\mathtt{X}]$ be our polynomials.

$$f^{\mathsf{R}}(a) \cdot g^{\mathsf{R}}(a) = \left(\sum_{i=0}^{d} f_i \cdot a^i\right) \cdot \left(\sum_{j=0}^{d'} g_j \cdot a^j\right) = \sum_{i=0}^{d}\sum_{j=0}^{d'} f_i \cdot a^i \cdot g_j \cdot a^j =$$
$$= \sum_{i=0}^{d}\sum_{j=0}^{d'} f_i \cdot a^{i-1} \cdot g_j \cdot a^{j+1} = \sum_{i=0}^{d}\sum_{j=0}^{d'} f_i \cdot g_j \cdot a^{i+j} = (f \cdot g)^{\mathsf{R}}(a). \quad \blacksquare$$

Theorem 3 (Euclidean Algorithm over Rings). *Let $f(\mathtt{X}) \in R[\mathtt{X}]$ be a non-zero polynomial and let $g(\mathtt{X}) \in R[\mathtt{X}]$ be a monic polynomial. There exist unique $q_\ell(\mathtt{X}), r_\ell(\mathtt{X})$ (resp. $q_r(\mathtt{X}), r_r(\mathtt{X})$) such that $f(\mathtt{X}) = q_\ell(\mathtt{X}) \cdot g(\mathtt{X}) + r_\ell(\mathtt{X})$ (resp. $f(\mathtt{X}) = g(\mathtt{X}) \cdot q_r(\mathtt{X}) + r_r(\mathtt{X})$), where $\deg(r_\ell) < \deg(g)$ (resp. $\deg(r_r) < \deg(g)$).*

Proof. We only prove the result for $q_\ell(\mathtt{X})$ and $r_\ell(\mathtt{X})$. The result for $q_r(\mathtt{X})$ and $r_r(\mathtt{X})$ follows in a similar manner.

We prove the statement regarding the existence of $q_\ell(\mathtt{X})$ and $r_\ell(\mathtt{X})$ by induction on the degree of f. Uniqueness is proven at the end. Consider a degree-zero f. If $g(\mathtt{X}) = 1_R$, then we may take $q_\ell(\mathtt{X}) = f(\mathtt{X})$ and $r_\ell(\mathtt{X}) = 0$. If $\deg(g) \geq 1$, then we may take $q_\ell(\mathtt{X}) = 0$ and $r_\ell(\mathtt{X}) = f(\mathtt{X})$.

Suppose the statement is true for polynomials f such that $\deg(f) = i$, and let us denote $\deg(g) = m$. We will now show the statement is also true for f such that $\deg(f) = i + 1$. If $\deg(f) < m$, we are done: we can take $q_\ell(\mathtt{X}) = 0$ and $r_\ell(\mathtt{X}) = f(\mathtt{X})$. Otherwise, let c_{i+1} be the leading coefficient of f and define $\tilde{f}_\ell(\mathtt{X}) = f(\mathtt{X}) - c_{i+1} \cdot \mathtt{X}^{i+1-m} \cdot g(\mathtt{X})$. As $\deg(\tilde{f}_\ell) \leq i$, by the induction hypothesis we have that $\tilde{f}_\ell(\mathtt{X}) = \tilde{q}_\ell(\mathtt{X}) \cdot g(\mathtt{X}) + r_\ell(\mathtt{X})$, where $\deg(r_\ell) < \deg(g)$. We can now conclude the proof:

$$f(\mathtt{X}) = \tilde{f}_\ell(\mathtt{X}) + c_{i+1} \cdot g(\mathtt{X}) \cdot \mathtt{X}^{i+1-m} = \tilde{q}_\ell(\mathtt{X}) \cdot g(\mathtt{X}) + r_\ell(\mathtt{X}) + c_{i+1} \cdot \mathtt{X}^{i+1-m} \cdot g(\mathtt{X})$$
$$= (\tilde{q}_\ell(\mathtt{X}) + c_{i+1} \cdot \mathtt{X}^{i+1-m}) \cdot g(\mathtt{X}) + r_\ell(\mathtt{X}).$$

Regarding uniqueness, suppose that $f(\mathtt{X}) = q_\ell(\mathtt{X}) \cdot g(\mathtt{X}) + r_\ell(\mathtt{X})$ and $f(\mathtt{X}) = q'_\ell(\mathtt{X}) \cdot g(\mathtt{X}) + r'_\ell(\mathtt{X})$, where $\deg(r_\ell) < \deg(g)$ and $\deg(r'_\ell) < \deg(g)$. This implies that $(q'_\ell(\mathtt{X}) - q_\ell(\mathtt{X})) \cdot g(\mathtt{X}) = r_\ell(\mathtt{X}) - r'_\ell(\mathtt{X})$, but if $q_\ell(\mathtt{X}) \neq q'_\ell(\mathtt{X})$ we would have that $\deg((q'_\ell(\mathtt{X}) - q_\ell(\mathtt{X})) \cdot g(\mathtt{X})) \geq \deg(g(\mathtt{X}))$, which is a contradiction with the fact that $\deg(r_\ell(\mathtt{X}) - r'_\ell(\mathtt{X})) \leq \max\{\deg(r_\ell(\mathtt{X})), \deg(r'_\ell(\mathtt{X}))\} < \deg(g(\mathtt{X}))$. $\blacksquare$

Given the two previous results, we can bound the number of roots of a polynomial as it is described in the next Lemma.

Lemma 4. *Let $f \in R[\mathtt{X}]_{\leq n}$ be a non-zero polynomial. Then f has at most n distinct left (resp. right) roots in the same commutative exceptional set $A \subset R$. In other words, if f has at least $n+1$ left (resp. right) roots in A, then it is the zero polynomial.*

Proof. We focus on right roots for the result, and we reason by induction on the degree d of the non-zero polynomial f. The statement is clear when $d = 0$. Assuming the result for $d - 1$, we now look at a degree-d polynomial f. If f does not have any roots, or if it only has one root, then the result clearly holds. Else, let $a, b \in A$ be two different roots of $f(\mathtt{X})$. As $g(\mathtt{X}) = \mathtt{X} - a$ is a monic polynomial, by Theorem 3 there exists $q(\mathtt{X}) \in R[\mathtt{X}]$ and $c \in R$ such that $f(\mathtt{X}) = q(\mathtt{X}) \cdot g(\mathtt{X}) + c$. Observe that $\deg(q) < \deg(f)$.

Now, since $g(\mathtt{X}) \in A[\mathtt{X}]$, by Lemma 3 we have that $f^{\mathsf{R}}(a) = q^{\mathsf{R}}(a)g^{\mathsf{R}}(a) + c$, so $0 = q^{\mathsf{R}}(a) \cdot (a - a) + c = c$. From this, it follows that $0 = f^{\mathsf{R}}(b) = q^{\mathsf{R}}(b)g^{\mathsf{R}}(b) = q^{\mathsf{R}}(b) \cdot (b - a)$. Since $(b - a) \in R^*$, then it has to be that $q^{\mathsf{R}}(b) = 0$.

By the induction hypothesis, $q(\mathtt{X})$ has at most $d-1$ distinct right roots in A, so we can conclude that $f(\mathtt{X})$ has at most d distinct right roots in A. ∎

Lagrange interpolation for sets of points $(x_i, y_i) \in R^2$ can be computed, as long as all the x_i are part of the same commutative exceptional set $A \subset R$. The following result was proved in [QBC13], but it only considered evaluation on the right. We reformulate and extend their result here for completeness and additional precision.

Proposition 1. *Let $A = \{x_1, \ldots, x_{n+1}\} \subset R$ be a commutative exceptional set and let $B = \{y_1, \ldots, y_{n+1}\} \subset R$. Then there exists a unique polynomial $f \in R[\mathtt{X}]$ (resp. $g \in R[\mathtt{X}]$) of degree at most d such that $f^{\mathsf{R}}(x_i) = y_i$ (resp. $g^{\mathsf{L}}(x_i) = y_i$) for $i = 1, \ldots, d+1$. Furthermore, if $A \cup B$ constitutes a commutative set, or if $A \subset Z(R)$, $f(X) = g(X)$.*

Proof. Let $L_i(\mathtt{X}) = \prod_{j \neq i}(\mathtt{X} - x_j) \in A[\mathtt{X}]$. Observe that for all $j = 1, \ldots, d+1$ it holds that $L_i(x_j) \in R^*$, since $(x_i - x_j) \in R^*$.

It is easy to verify, with the help of Lemma 3, that the two following polynomials show the existence of solutions:

$$f(\mathtt{X}) = \sum_{i=1}^{d+1} y_i L_i(x_i)^{-1} L_i(\mathtt{X}); \qquad g(\mathtt{X}) = \sum_{i=1}^{d+1} L_i(\mathtt{X}) L_i(x_i)^{-1} y_i$$

The uniqueness of f (resp. g) is a consequence of Lemma 4. The fact that $f(X) = g(X)$ when $A \cup B$ constitutes a commutative set or $A \subset Z(R)$ follows from inspection. ∎

2.4 Galois Rings

Galois Rings relate to integers modulo a prime power p^k in the same way a Galois Field relates to integers modulo a prime p. They are a fundamental object of study among finite commutative rings.

Definition 8. *A Galois Ring $\mathsf{GR}(p^k, d)$ is a ring of the form $R = \mathbb{Z}_{p^k}[\mathtt{X}]/(h(\mathtt{X}))$, where p is a prime, k a positive integer and $h(X) \in \mathbb{Z}_{p^k}[\mathtt{X}]$ a monic polynomial of degree $d \geq 1$ such that its reduction modulo p is an irreducible polynomial in $\mathbb{F}_p[\mathtt{X}]$.*

Given a base ring $\mathbb{Z}_{p^k}$, there is a unique degree d Galois extension of $\mathbb{Z}_{p^k}$, which is precisely the Galois Ring provided on the previous definition. Note that Galois Rings reconcile the study of finite fields $\mathbb{F}_{p^d} = \mathsf{GR}(p, d)$ and finite rings of the form $\mathbb{Z}_{p^k} = \mathsf{GR}(p^k, 1)$. Every Galois Ring $R = \mathsf{GR}(p^k, d)$ is a local ring and its unique maximal ideal is (p). Hence, all the zero divisors of R are furthermore nilpotent, and they constitute the maximal ideal (p). Furthermore, we have that $R/(p) \cong \mathbb{F}_{p^d}$.

Proposition 2 ([ACD+19]). *The Lenstra constant of $R = GR(p^k, d)$ is p^d.*

Whenever we need to explicitly represent elements $a \in R$, we will consider two options. The first one, which we will denote the *additive representation*, follows from Definition 8 and consists of the residue classes

$$a \equiv a_0 + a_1 \cdot \mathtt{X} + \cdots + a_{d-1} \cdot \mathtt{X}^{d-1} \mod h(\mathtt{X}), \quad a_i \in \mathbb{Z}_{p^k}. \tag{1}$$

The second option is what we shall call the *matrix representation*, which uses the embedding $\iota : \mathsf{GR}(p^k, d) \hookrightarrow \mathcal{M}_{d\times d}(\mathbb{Z}_{p^k})$ and represents a as $\iota(a)$. It will be instructive to discuss this embedding more explicitly for other parts of this work. Let us look at how the product between $a, b \in \mathsf{GR}(p^k, d)$ is computed. If we express a in its additive representation, $a = \sum_{\ell\in[d]} a_\ell \cdot \mathtt{X}^\ell$, multiplication by b can be seen as the homomorphism of free $\mathbb{Z}_{p^k}$-modules $\phi_b : \mathbb{Z}_{p^k}^d \to \mathbb{Z}_{p^k}^d$, which maps the coefficients of a's additive representation to those of $c = \phi_b(a)$. As there is a one-to-one correspondence between homomorphisms of free modules and matrices, we can represent b as the matrix defined by ϕ_b.

Notice that since $\mathsf{Im}(\iota) \simeq \mathsf{GR}(p^k, d)$, we have that $\iota(a) \cdot \iota(b) = \iota(b) \cdot \iota(a)$. In other words, the matrices in $\mathsf{Im}(\iota)$ constitute a commutative subset (as per Definition 4) of $\mathcal{M}_{d\times d}(\mathbb{Z}_{p^k})$.

3 Shamir's Secret Sharing over Non-commutative Rings

Secret sharing schemes (SSS) are one of the most fundamental building blocks in secure computation. There are three properties which we usually want from SSS in MPC. The first one is t-privacy, meaning that no set of at most t shares reveals any information about the secret. The second one is $t+1$-reconstruction, which allows to reconstruct the secret from any subset of $t+1$ correct shares. The third one is *linearity*, which requires talking about specific algebraic structures. In our work, as we will be working over rings for which we do not assume commutativity, we need to distinguish between *left* and *right* linearity.

Definition 9. *Let $C = \{(s, s_1, \ldots, s_n)\} \subseteq R^{n+1}$ be a SSS, where s is a* secret *and $s_1, \ldots, s_n$ are its* shares*. We say that C is a* left *(resp.* right*)* linear secret sharing scheme *if it is a left (resp. right) submodule of R^{n+1}. We will respectively denote the secret sharing of s by $[s], \langle s\rangle$. If C is a bisubmodule of R^{n+1}, then we simply call it a* linear secret sharing scheme*, which we denote as $[\![s]\!]$.*

In Shamir's secret sharing scheme, which was originally restricted to finite fields [Sha79], the submodule C is a Reed-Solomon code, i.e. $[\![s]\!]_t$ would be sampled from $C = \{(s, f(\alpha_1), \ldots, f(\alpha_n)) : f \in \mathbb{F}[\mathtt{X}]_{\leq t} \wedge s = f(\alpha_0)\}$. This was later on generalized to commutative rings containing big enough exceptional sets [ACD+19]. In this work, we observe that Reed-Solomon codes have been constructed even over non-commutative rings [QBC13]. Throughout this section we translate the relevant parts of [QBC13] to the LSSS language. Moreover, we fill some gaps about error correction left by the authors of [QBC13], we generalize standard secret reconstruction procedures from [DN07] and we show where do matrix rings fit in these results.

Beyond linearity, another desirable property for a SSS to have is that of *(strong) multiplicativity*. Briefly, such notion guarantees that (even in the presence of active adversaries) the product of two secrets a, b can be reconstructed as a function of the coordinate-wise product of their shares, $a_i \cdot b_i$. For a formal definition see [CDM00].

Theorem 4. *Let R be a ring such that $Z(R)$ contains an exceptional set $A = \{\alpha_0, \ldots, \alpha_n\}$ and let $t < n/3$. Then, we can define a Shamir-style strongly multiplicative linear secret sharing scheme over R. In more detail, a degree-t sharing $[\![s]\!]_t$ is sampled from:*

$$\{(s, f^R(\alpha_1), \ldots, f^R(\alpha_n)) : f \in R[\mathtt{X}]_{\leq t} \wedge s = f^R(\alpha_0)\}$$

Strong multiplicativity in the previous result woks as usual in Shamir's LSSS. In more detail, given two shared values, $[\![a]\!]_t = (a, a_1, \ldots, a_n)$ using a polynomial $f \in R[\mathtt{X}]_{\leq t}$ and $[\![b]\!]_t = (b, b_1, \ldots, b_n)$ using a polynomial $g \in R[\mathtt{X}]_{\leq t}$, it holds that $[\![a \cdot b]\!]_{2t} = (ab, a_1b_1, \ldots, a_nb_n)$. This is due to the fact that the points α_i where f and g are evaluated at are contained in $Z(R)$, and hence by Lemma 3 it holds that $(f \cdot g)^{\mathsf{R}}(\alpha_i) = f^{\mathsf{R}}(\alpha_i) \cdot g^{\mathsf{R}}(\alpha_i)$, which is not generally the case for non-commutative polynomials.

Given the previous theorem, we can adapt the results of [BTH08, ACD+19] to work over non-commutative rings as the ones of the hypothesis without too much effort. This gives us the following result, which we discuss in more detail in Appendix D. The increase on the size of the exceptional set is due to the use of so-called hyper-invertible matrices.

Corollary 2. *Let R be a ring such that $Z(R)$ contains an exceptional set of size at least $2n$. Let $\mathcal{A}$ be an active adversary corrupting $t < n/3$ parties. There exists a perfectly secure, black-box MPC protocol with an amortized communication complexity of $O(n)$ ring elements per gate.*

If we relax the hypothesis of Theorem 4, so that we only ask from the elements of the exceptional set to commute with each other, rather than being in the centre of the ring, we can still build Shamir-style secret sharing schemes.

Theorem 5. *Let R be a ring containing a commutative, exceptional set $A = \{\alpha_0, \ldots, \alpha_n\}$. Then, we can define a Shamir-style left-LSSS $[\cdot]$ and a Shamir-style right-LSSS $\langle\cdot\rangle$ over R. These secret sharing schemes are not multiplicative.*

We do not provide an explicit proof of the previous Theorem, but in Figure 1 we show how to share a secret for the left-linear scheme $[\cdot]$. The right-linear scheme $\langle\cdot\rangle$ would produce shares in an analogous way, setting instead $s_i = f^{\mathsf{L}}(\alpha_i)$. The t-privacy and $t+1$-reconstruction properties are a consequence of Proposition 1. To see why these schemes are not multiplicative, remember that the evaluation maps are not ring homomorphisms in general, i.e. given $f, g \in R[\mathtt{X}]$, generally $f^{\mathsf{R}}(\alpha_i) \cdot g^{\mathsf{R}}(\alpha_i) \neq (f \cdot g)^{\mathsf{R}}(\alpha_i)$. Hence, in contrast with Theorem 4, given $[a]_t = (a, f^{\mathsf{R}}(\alpha_1), \ldots, f^{\mathsf{R}}(\alpha_n))$ and $[b]_t = (b, g^{\mathsf{R}}(\alpha_1), \ldots, g^{\mathsf{R}}(\alpha_n))$, Lemma 3 is of no help now, since the α_i values are not in the centre any more. Note that we cannot simply impose for the sampled polynomial f in Figure 1 to be in $A[\mathtt{X}]_{\leq d}$. As an example, imagine the case when A is furthermore a subring of R. We would then have that $f^{\mathsf{R}}(\alpha_0) \in A$, effectively restricting the values that can be secret shared to those in the subring A itself.

Protocol $\Pi_{[\cdot]-\text{Share}}(s, d)$

Input: A secret $s \in R$ held by a dealer P_D. A commutative exceptional set $A = \{\alpha_0, \ldots, \alpha_n\}$.
Output: Sharing $[s]_d$.
Protocol: The parties proceed as follows

1. P_D samples a polynomial $f \xleftarrow{\$} R[\mathtt{X}]_{\leq d}$ such that $f^{\mathsf{R}}(\alpha_0) = s$. Define the shares to be $s_i = f^{\mathsf{R}}(\alpha_i)$ for $i \in [n]$.
2. For $i \in [n] \setminus \{D\}$, P_D sends s_i to P_i.

Fig. 1. Sharing a secret using $[\cdot]$.

3.1 Secret Sharing over Matrix Rings

As our more practical results are related to the ring $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, it will be useful to give already a more concrete analysis of how it fits with respect to Theorems 4 and 5, before returning to generic rings. We start by reminding the following basic result.

Lemma 5. *The centre of $\mathcal{M}_{m\times m}(R)$, where R is a commutative ring, is the R-multiples of the identity matrix.*

Besides which elements are in the centre of the ring, it is important that we identify exceptional sets in the ring. As we discussed in Section 2.4, there exists an embedding $\iota : \mathsf{GR}(p^k, m) \hookrightarrow \mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$. Hence, as the Lenstra constant of the former ring is p^m, that of $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ has to be at least p^m. Whereas one could hope that by working over matrix rings (which contain many invertible elements) exceptional sets could be bigger than that, we prove that this is not the case.

Proposition 3. *The Lenstra constant of $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ is p^m.*

Proof. As we can embed $\mathsf{GR}(p^k, m)$ (which has an exceptional set of size p^m) into $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$, we have that the Lenstra constant of $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ is at least p^m. We complete the proof by showing that the Lenstra constant of $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ cannot exceed p^m.

Let us first look at the case $S = \mathcal{M}_{m\times m}(\mathbb{F}_p)$. Assume by contradiction that there is an exceptional set $A \subseteq S$ such that $|A| \geq p^m + 1$. Then, by the pigeonhole principle, there must be at least two matrices $M, N \in A$ which have exactly the same first column. We then have that $M - N$ has an all zeroes column, which implies that $M - N$ is not invertible. Hence, the Lenstra constant of $\mathcal{M}_{m\times m}(\mathbb{F}_p)$ is p^m.

Consider now $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ and $a, b \in R$ such that $a - b \in R^*$. Then, there exists some $x \in R^*$ such that $x \cdot (a - b) = 1_R$. Let $\phi : R \mapsto S$ be the "reduction mod p" map. As it is a ring homomorphism, we have the following:

$$\phi(x \cdot (a - b)) = \phi(1_R) \iff \phi(x) \cdot (\phi(a) - \phi(b)) = 1_S$$

Hence, if $A = \{a_1, \ldots, a_\ell\}$ is an exceptional set over R, so is it the following one over S: $\phi(A) = \{\phi(a_1), \ldots, \phi(a_\ell)\}$. As the Lenstra constant of S is p^m, so it has to be the one of R. ∎

Let us focus on the ring $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$. We know that $Z(R)$ cannot contain exceptional sets of size bigger than two, so Theorem 4 is ruled out. The good news are that, since $\mathsf{GR}(2^k, m)$ (more precisely, $\mathsf{Im}(\iota)$) is a *commutative subring* of R, we can easily identify within R a commutative exceptional set of size 2^m and construct the secret sharing schemes described in Theorem 5 whenever $m > \log(n + 1)$.

3.2 Error Correction and Robust Reconstruction.

Let R be a finite ring and let $A = \{\alpha_0, \alpha_1, \ldots, \alpha_n\} \subseteq R$ be an exceptional commutative set. Let $[s]_d = (s_1, \ldots, s_n)$ be a secret-shared value $s \in R$ using a polynomial of degree at most d. For $i = 1, \ldots, n$, let $s'_i = s_i + \delta_i$, where at most e of the $\delta_i \in R$ are non-zero, with $n > d + 2e$. Our goal is to recover $(s_1, \ldots, s_n)$ from $(s'_1, \ldots, s'_n)$. This is an essential primitive when designing MPC protocols based on Shamir secret-sharing, as it corresponds to reconstructing a secret-shared value from a given set of announced shares among which some of them could be incorrect due

to adversarial behavior. This is achieved by a generalization of the Berlekamp-Welch decoding algorithm for Reed-Solomon codes to the non-commutative setting. Such result was exhibited in [QBC13], although many holes were left due to the general approach taken by the authors. For instance, a crucial step in the decoding algorithm lies in solving a system of linear equations over a non-commutative ring, which as we discuss later on it is not a very well studied area. Hence, concrete algorithms should be developed for each particular instantiation. Motivated by this, and also for the sake of clarity and self-containment, we present below our own version of the generalization of the Berlekamp-Welch algorithm, filling in the holes left in [QBC13]. Below, we let $n' = d + 2e + 1$.

Generalization of the Berlekamp-Welch algorithm. Below we let F denote the subring of R made of finite sums of terms of the form $\alpha_{i_1} \cdot \alpha_{i_2} \cdots \alpha_{i_\ell}$. We say that two polynomials $p(\mathtt{X}), q(\mathtt{X}) \in R[\mathtt{X}]$ satisfy the BW-conditions if

1. $\deg(p) \leq e$;
2. $\deg(q) \leq d + e$;
3. $p(\mathtt{X})$ is monic;
4. $p(\mathtt{X}) \in F[\mathtt{X}]$;
5. For all $i = 1, \ldots, n'$, it holds that $s'_i \cdot p(\alpha_i) = q(\alpha_i)$.

We begin with the following claim.

Claim. There exists a pair $p(\mathtt{X}), q(\mathtt{X}) \in R[\mathtt{X}]$ that satisfies the BW-conditions above.

Proof. Let $f(\mathtt{X}) = \sum_{i=0}^{d} c_i \mathtt{X}^i \in R_{\leq d}[\mathtt{X}]$ such that $f(\alpha_i) = s_i$ for $i = 1, \ldots, n'$, guaranteed by Proposition 1, and define $p(\mathtt{X}) = \prod_{e_i \neq 0} (\mathtt{X} - \alpha_i)$ and $q(\mathtt{X}) = f(\mathtt{X})p(\mathtt{X})$. It can be easily verified, with the help of Lemma 3, that this choice of $p(\mathtt{X})$ and $q(\mathtt{X})$ satisfies the BW-conditions. ∎

The next claim shows that any other pair satisfying the BW-conditions is as good as the one guaranteed from the previous claim for the purpose of recovering $f(\mathtt{X})$.

Claim. Let $p(\mathtt{X}), q(\mathtt{X})$ be defined as in the proof of the previous claim, and suppose that $\hat{p}(\mathtt{X}), \hat{q}(\mathtt{X})$ satisfy the BW-conditions. Then $\hat{p}(\mathtt{X})$ divides $\hat{q}(\mathtt{X})$ and $\hat{q}(\mathtt{X})/\hat{p}(\mathtt{X}) = f(\mathtt{X})$.[4]

Proof. Consider the polynomial $r(\mathtt{X}) = \hat{q}(\mathtt{X})p(\mathtt{X}) - q(\mathtt{X})\hat{p}(\mathtt{X})$. In light of Lemma 3, taking into account that $p(\mathtt{X}) \in F[\mathtt{X}]$, we have that for every $i = 1, \ldots, n'$:

$$r(\alpha_i) = \hat{q}(\alpha_i)p(\alpha_i) - q(\alpha_i)\hat{p}(\alpha_i) = s'_i\hat{p}(\alpha_i)p(\alpha_i) - s'_i p(\alpha_i)\hat{p}(\alpha_i) = 0.$$

Observe that in the last equality we have used the fact that $\hat{p}(\alpha_i)p(\alpha_i) = p(\alpha_i)\hat{p}(\alpha_i)$. Since $\deg(r) \leq d+2e < n'$, it follows from Lemma 4 that $r(\mathtt{X}) \equiv 0$, which shows that $\hat{q}(\mathtt{X})p(\mathtt{X}) = q(\mathtt{X})\hat{p}(\mathtt{X})$. Given that $q(\mathtt{X}) = f(\mathtt{X})p(\mathtt{X})$, we have that $\hat{q}(\mathtt{X})p(\mathtt{X}) = f(\mathtt{X})p(\mathtt{X})\hat{p}(\mathtt{X})$, which implies $(\hat{q}(\mathtt{X}) - f(\mathtt{X})\hat{p}(\mathtt{X})) \cdot p(\mathtt{X}) = 0$.

We claim that $\hat{q}(\mathtt{X}) - f(\mathtt{X})\hat{p}(\mathtt{X}) = 0$, which can be shown by proving that this polynomial evaluates to 0 in at least $d + e + 1$ points on an exceptional set in light of Lemma 4. To see this, consider the evaluation of this polynomial at α_i for all i such that $e_i = 0$. Observe that there are at least $n' - e = d + e + 1$ such evaluation points. It is easy to see that in this case $p(\alpha_i)$ is invertible, so $(\hat{q}(\alpha_i) - f(\alpha_i)\hat{p}(\alpha_i)) \cdot p(\alpha_i) = 0$ implies that $\hat{q}(\alpha_i) - f(\alpha_i)\hat{p}(\alpha_i) = 0$, as required. At this point we see that $\hat{q}(\mathtt{X}) = f(\mathtt{X})\hat{p}(\mathtt{X})$, which concludes the proof of the main claim. ∎

[4] $b(\mathtt{X})$ (right-)divides $a(\mathtt{X})$, if, after dividing a by b using Theorem 3 obtaining $q(\mathtt{X})$ and $r(\mathtt{X})$ such that $a(\mathtt{X}) = q(\mathtt{X}) \cdot b(\mathtt{X}) + r(\mathtt{X})$ with $\deg(r) < \deg(b)$, it holds that $r(\mathtt{X}) = 0$. The quotient $a(\mathtt{X})/b(\mathtt{X})$ is defined as $q(\mathtt{X})$.

Error Detection. Finally, if $n > d + e$ the parties may not be able to perform error correction, but they can still do error detection by checking if all the received shares $(s'_1, \ldots, s'_n)$ are consistent with a polynomial of degree at most d (e.g. by using the first $d + 1$ shares to interpolate such polynomial and checking that the remaining shares are consistent with it). If this is the case, since this polynomial is determined by any set of $d + 1$ shares, it is in particular determined by the $n - e \geq d + 1$ shares without errors.

Solving for the BW-conditions. In order to have an efficient decoder it remains to show how to find at least one pair $p(\mathtt{X}), q(\mathtt{X})$ that satisfies the BW-conditions. First, notice that by treating the coefficients of the unknown polynomials $p(\mathtt{X}), q(\mathtt{X})$ as unknowns, the BW-conditions transform into a system of $n' = d + 2e + 1$ linear equations on $d + 2e + 1$ variables over R.[5] Unfortunately, to the best of our knowledge the theory of linear equations over *general* non-commutative rings is not very well understood, with only a few works considering concrete instantiations of some types of rings (e.g. [Ore31, Son75, DKH+12]). Since it is of particular interest to us, we develop efficient algorithms to solve systems of linear equations for the matrix ring case $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$. This is explained in Appendix B.1.

3.3 Efficient Protocols for Secret Reconstruction

Protocol $\Pi_{\text{PrivOpen}}([s]_d, P_r)$

Input: Sharing $[s]_d$, a receiver party P_r.
Output: P_r learns s.
Protocol: The parties proceed as follows

1. Each party P_j for $j \in \{1, \ldots, n\} \setminus \{r\}$ sends its share of s to P_r.
2. Upon receiving all the shares of $[s]$, P_r defines $s_i = 0$ for every missing share s_i and proceeds as follows.
   - If $0 < n - d \leq t$: Interpolate the unique polynomial $f \in R[\mathtt{X}]_{\leq d}$ such that $f^{\mathtt{L}}(\alpha_i) = s_i$ for $i = 1, \ldots, d+1$. Output $s' = f(\alpha_0)$.
   - If $t < n - d \leq 2t$: Interpolate the unique polynomial $f \in R[\mathtt{X}]_{\leq d}$ such that $f^{\mathtt{L}}(\alpha_i) = s_i$ for $i = 1, \ldots, d+1$. Check if $f^{\mathtt{L}}(\alpha_i) = s_i$ for $i = d + 2, \ldots, d + t + 1$. If this is the case, output $s = f(\alpha_0)$. Else abort.
   - If $2t < n - d$: Apply error correction (Section 3.2) on the shares $(s_1, \ldots, s_{d+2t+1})$ to recover a polynomial $f \in R[\mathtt{X}]_{\leq d}$ such that $f^{\mathtt{L}}(\alpha_i) = s_i$ for at least $d + t + 1$ points, and output $s = f(\alpha_0)$.

Fig. 2. Reconstructing secret-shared values efficiently to a single party.

Given the above, a party that receives n shares of degree $\leq d$, among which at most t can be corrupted by an adversary, can perform error detection if $t < n - d \leq 2t$, and it can perform error correction if $2t < n - d$. We denote by $\Pi_{\text{PrivOpen}}([s]_d, P_r)$ the protocol in which all parties send their share of $[s]_d$ to P_r. If all parties are intended to learn the secret s, we make use of a protocol $\Pi_{\text{PublicOpen}}([s_0]_d, \ldots, [s_d]_d)$ that opens a batch of secrets towards all the parties with an amortized communication complexity that is linear in n. This protocol is achieved by a natural generalization of the equivalent protocol in [DN07], except that great care must be taken when handling the different multiplications and polynomial evaluations when the ring is not commutative. This is described in detail in Appendix B.2.

[5] It is worth noting that some of the unknowns will have coefficients multiplying from both left and right.

Finally, notice that the protocols $\Pi_{\text{PrivOpen}}, \Pi_{\text{PubOpen}}$ are currently described for $[\cdot]$-sharings, but they can be naturally adapted to $\langle\cdot\rangle$-sharings by evaluating the polynomial $f(\mathtt{X})$ on the right and computing $\langle f^{\mathsf{R}}(\alpha_i)\rangle = \sum_{j=0}^{t} \langle s_j \rangle \alpha_i^j$.

4 MPC in the Preprocessing Model

The goal of this section is to leverage the adaptations of Shamir's secret sharing described in Section 3 to build an MPC protocol that operates directly over R, in a black-box way. We assume an active adversary corrupting t out of n parties, where it could hold either that $t < n/3$ or $t < n/2$. In the first case we can obtain guaranteed output delivery with perfect security, and in the second case we achieve perfect security with abort (both in the preprocessing model).

Multiparty computation can be obtained from any linear secret sharing scheme satisfying certain multiplicative properties, as shown in [CDM00]. As we proved in Corollary 2 (and showed in Appendix D), even more modern and efficient techniques for MPC over commutative rings can be adapted to the non-commutative setting, assuming that there is a large enough exceptional set in the centre of the ring. The challenge in this section is, however, that we only assume the existence of a big enough *commutative* exceptional set, i.e. the conditions of Theorem 2.

Losing multiplicativity leads to most existing techniques for secret-sharing based MPC to fail. A clever solution when multilpicativity is lost is to resort to properties of the *dual* of the error-correcting code underlying the secret-sharing scheme [CDM00]. However, although as shown in [QBC13] the usual properties of the dual code of Reed-Solomon codes do carry over to non-commutative rings, this requires that the evaluation points constitute not only an exceptional set, but that they are also contained in $Z(R)$. Unfortunately, this is precisely the assumption we do not want to make (and in fact, as said above, such assumption would yield a multiplicative LSSS *directly*).

Given the hurdles highlighted above, this work takes a different route. Our protocols are set in the offline/online paradigm, in which a set of input-independent correlated information is generated in a preprocessing phase, which is then used in an online phase once the inputs are known. By preprocessing the so-called Beaver triples, the online phase can be executed without relying on any multiplicativity property of the underlying secret-sharing scheme. However, due to non-commutativity, the usual approach to secure multiplication using Beaver triples does not directly work, as we will explain shortly. The rest of this section is then devoted to overcoming these issues and obtain a secure computation protocol in the preprocessing model, where we assume that the input-independent correlated data is given "for free". Our protocols for instantiating such preprocessing phase will be discussed in Section 5.

4.1 A First Approach

We begin by considering the typical approach to Beaver-based multiplication, and discuss why it fails in our setting. Assume for a moment that R is commutative. A Beaver triple is a set of shared values $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ such that $a, b \in R$ are uniformly random and $c = a \cdot b$. Given two shared values $\llbracket x \rrbracket, \llbracket y \rrbracket$, these can be multiplied by means of the following protocol:

1. Parties call $d = \Pi_{\text{PubOpen}}(\llbracket x \rrbracket - \llbracket a \rrbracket)$ and $e = \Pi_{\text{PubOpen}}(\llbracket y \rrbracket - \llbracket b \rrbracket)$.
2. Parties compute locally $\llbracket xy \rrbracket = \llbracket a \rrbracket e + d \llbracket b \rrbracket + \llbracket c \rrbracket + de$.

Privacy follows from the fact that the sensitive values x and y are being masked by uniformly random values a and b that are unknown to the adversary. Correctness follows from the fact that $xy = (d + a)(e + b) = ae + db + ab + de$, a relation that also holds even if R is non-commutative. Here we use the fact that, since $t < n/2$, the calls to Π_{PubOpen} result in the parties learning the correct underlying secret or aborting (and in the stronger case that $t < n/3$ then Π_{PubOpen} does not result in abort).

The issue with a non-commutative R is that, unless $Z(R)$ contains a big enough exceptional set, the secret sharing scheme $[\cdot]$ (resp. $\langle\cdot\rangle$) we can define is just a left (resp. right) submodule of R^n. In particular, the local operation $d \cdot [b]$ can be carried out, but $[a] \cdot e$ does not result in a $[\cdot]$-shared value[6]. To address this complication, let $([a], [b], [c])$ be a triple. Assume the existence of "sextuples", which are just triples of the form $([a], [b], [c])$ enhanced with shares of the form $(\langle a\rangle, [r], \langle r\rangle)$. These are produced by a functionality $\mathcal{F}_{\text{Tuples}}$, which is formally described in Fig. 3. These tuples can be used to multiply $[x]$ and $[y]$ as follows:

1. Parties call $d = \Pi_{\text{PubOpen}}([x]_t - [a]_t)$, $e = \Pi_{\text{PubOpen}}([y]_t - [b]_t)$.
2. Parties call $f = \Pi_{\text{PubOpen}}(\langle a\rangle_t \cdot e + \langle r\rangle_t)$.
3. Parties compute locally $[xy]_t = d \cdot e + d \cdot [b]_t + f - [r]_t + [c]_t$

Privacy follows from the fact that sensitive data x and y is masked by the uniformly random values a and b before opening and also because, before reconstructing $a \cdot e$ (which could potentially leak information about a), the uniformly random mask r is applied. In terms of correctness, we observe that the final expression defining $[xy]_t$ is well defined given that only additions and multiplications *on the left* are used. Furthermore, the computation of f uses multiplication on the right on the sharings $\langle\cdot\rangle$, which admit such mutliplications. The rest is simply a matter of using the definition of d, e, c and f in the final computation: $d \cdot e + d \cdot b + f - r + c = (x - a) \cdot (y - b) + (x - a) \cdot b + a \cdot (y - b) + r - r + a \cdot b = x \cdot y$.

Functionality $\mathcal{F}^{\text{abort}}_{\text{Tuples}}$

The functionality runs with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{A}$ who corrupts t parties. On input $(\texttt{Tuples}, m)$, the functionality proceeds as follows:

1. Wait for a message from $\mathcal{A}$. If the message is `abort`, then the functionality notifies honest parties and aborts.
2. Otherwise, if the message is `ok` then, for $j = 1, \ldots, m$, the functionality samples values $a_j, b_j, r_j \xleftarrow{\$} R$, compute sharings $[a_j]_t, [b_j]_t, [a_j \cdot b_j]_t, \langle a_j\rangle_t, [r_j]_t, \langle r_j\rangle_t$ and distributes them to the parties.

Fig. 3. Function-independent preprocessing functionality.

4.2 Improving Round-Complexity

The protocol sketched in the previous section suffers from the issue that its round complexity is quite high, requiring 4 rounds per multiplication (two sequential calls to Π_{PubOpen}, each requiring 2 rounds). In MPC protocols networking is usually the most scarce resource, and it can be argued that round-count is even more sensitive than communication complexity, specially in wide area networks that have high latency. Therefore, the rest of this section is devoted to lowering the round count of each secure multiplication.

[6] More concretely, if we had $[a]_t$, multiplication by e on the right will *not* result on $[ae]_t$ in general.

In order to achieve secure multiplication with no sequential calls to Π_{PubOpen} in the non-commutative case, we modify the way multiplications are handled. First, each intermediate value of the computation x will not be represented by $[x]$, but rather by a pair $([\lambda_x], \mu_x)$, where $\lambda_x \in R$ is uniformly random and unknown to any party, and $\mu_x = x - \lambda_x$. Notice that this still maintains the privacy of x since the only public value is μ_x, which perfectly hides x as it is being masked by λ_x, that is random and unknown to any party.

Suppose the parties have two shared values $([\lambda_x], \mu_x = x - \lambda_x)$ and $([\lambda_y], \mu_y = y - \lambda_y)$. To obtain a shared representation of their sum, the parties simply locally compute $([\lambda_x + \lambda_y], \mu_x + \mu_y)$. On the other hand, to securely multiply these shared values, the process is as follows. Let $[\lambda_z]$ be the random mask associated to the output of the multiplication. To obtain $([\lambda_z], \mu_z)$, the parties need to get the value $\mu_z = x \cdot y - \lambda_z$ in the clear. This is achieved by noticing that, since $x = \mu_x + \lambda_x$ and $y = \mu_y + \lambda_y$, it holds that $\mu_z = \mu_x\mu_y + \mu_x\lambda_y + \lambda_x\mu_y + \lambda_x\lambda_y - \lambda_z$. Assume that the parties have $[\lambda_x\lambda_y]$, which can be preprocessed as λ_x and λ_y are simply random values. If R was commutative, then the parties could compute

$$[\mu_z] = \mu_x\mu_y + \mu_x[\lambda_y] + [\lambda_x]\mu_y + [\lambda_x\lambda_y] - [\lambda_z],$$

followed by opening μ_z. This approach was followed in [BNO19] in order to improve the communication complexity of secure multiplication in the dishonest majority setting. It was also used in the context of honest majority in [ED20], both to minimize online communication complexity and in order to avoid selective failure attacks.

Unfortunately, when R is non-commutative, this approach cannot be carried out as we find the exact same issue we had in the previous section, namely that $[\lambda_x]\mu_y$ is not well defined as the secret-sharing scheme $[\cdot]$ does not allow multiplication on the right. However, our crucial observation is that, unlike the traditional use of triples from Section 4.1, in this case the task is not to take a combination of sharings in order to obtain a *new shared value* but rather take a linear combination of sharings in order to *open* the result μ_z. This difference turns out to be essential in order to devise a protocol for the non-commutative case that does not require sequential openings, which we describe in detail below. The overall idea of our protocol is that the parties do not really need to convert $\langle\lambda_x\rangle\mu_y$ to $[\lambda_x\mu_y]$ as in Section 4.1, which adds an extra opening round, but rather it is enough to *open* this part separately from the other part that uses the $[\cdot]$-sharing, and then add the two opened values to obtain μ_z. Some masking is necessary to ensure that each separate piece does not leak anything, but this is easily achievable, as we will describe next.

Functionality $\mathcal{F}_{\text{F.D.Prep}}$

The functionality runs with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{A}$ who corrupts t parties. On input $(\texttt{F.D.Prep}, C)$, where C is an arithmetic circuit over R, the functionality proceeds as follows:

1. Wait for a message from $\mathcal{A}$. If the message is `abort`, then the functionality notifies honest parties and aborts.
2. Otherwise, if the message is `ok` then, for every wire in the circuit x, sample $\lambda_x \xleftarrow{\$} R$ and generate shares $[\lambda_x]_t$. Distribute these shares towards the parties.
3. For every input wire x assigned to party P_i, $\mathcal{F}_{\text{F.D.Prep}}$ sends λ_x to P_i.
4. For every multiplication gate with inputs x, y, sample $r \xleftarrow{\$} R$. Generate shares $\langle\lambda_x\rangle_t, [\lambda_x\lambda_y]_t, [r]_t, \langle r\rangle_t$ and distribute them to the parties.

Fig. 4. Function-dependent preprocessing functionality.

Preprocessing functionality. Unfortunately, resorting to this new approach does not allow us to use the functionality $\mathcal{F}_{\text{Tuples}}$ directly. Instead, we must resort to a similar but different type of preprocessing, which is captured by functionality $\mathcal{F}_{\text{F.D.Prep}}$ in Figure 4. In a nutshell, this preprocessing material also consists of tuples $([\lambda_x]_t, [\lambda_y]_t, [\lambda_x \cdot \lambda_y]_t, \langle\lambda_x\rangle_t, [r]_t, \langle r\rangle_t)$, except that, if λ_x (resp. λ_y) is used to mask a value x (resp. y) for a given multiplication, then the same λ_x (resp. λ_y) must be used for all multiplications involving x (resp. y) as a left (resp. right) input. Since the structure of the tuples returned depend on the way multiplications are arranged in the circuit, we refer to this type of preprocessing as *function-dependent preprocessing*. This is in contrast to the preprocessing described in Fig. 3 which only depends on (an upper bound on) the *number* of multiplications in the circuit and not on the way these are arranged.

Protocols for MPC in the $(\mathcal{F}_{\text{F.D.Prep}}, \mathcal{F}_{\text{BC}})$-hybrid model. Now we finally describe our protocol for MPC in the $(\mathcal{F}_{\text{F.D.Prep}}, \mathcal{F}_{\text{BC}})$-hybrid model. The protocol Π_{Online}, described in Fig. 5 achieves guaranteed output delivery with perfect security against an active adversary corrupting $t < n/3$ parties, and it achieves perfect security with abort against an active adversary corrupting $t < n/2$ parties. With the tools we have seen so far, we are able to prove the following.

Protocols Π_{Online}

PREPROCESSING PHASE

The parties call $\mathcal{F}_{\text{F.D.Prep}}$ to get the following.

- For every wire in the circuit x the parties have $[\lambda_x]$.
- Party P_i knows λ_x for every input gate x corresponding to P_i.
- For every multiplication gate with inputs x, y and output z, the parties have $[\lambda_x\lambda_y]$.

ONLINE PHASE

Input Gates. For every input gate x owned by party P_i, the parties do the following:
1. P_i uses $\mathcal{F}_{\text{BC}}$ to send $\mu_x = x - \lambda_x$ to all parties.
2. Upon receiving this value, the parties set the sharing $([\lambda_x], \mu_x)$

Addition Gates. For every addition gate with inputs $([\lambda_x], \mu_x)$ and $([\lambda_y], \mu_y)$, the parties locally get shares of the sum as $([\lambda_x] + [\lambda_y], \mu_x + \mu_y)$.

Multiplication Gates. For every multiplication gate with inputs $([\lambda_x], \mu_x)$ and $([\lambda_y], \mu_y)$, the parties proceed as follows:
1. The parties call $\gamma \leftarrow \Pi_{\text{PubOpen}}(\mu_x\mu_y + \mu_x[\lambda_y] + [\lambda_x\lambda_y] - [\lambda_z] + [r])$ and $\rho \leftarrow \Pi_{\text{PubOpen}}(\langle\lambda_x\rangle\mu_y - \langle r\rangle)$,[a] and, if there was no abort, set $\mu_z = \gamma + \rho$.
2. Output $([\lambda_z], \mu_z)$ as shares of the product.

Output Gates. If the parties did not abort above, then for every output gate $([\lambda_x], \mu_x)$ that is supposed to be learned by P_i, the parties do the following:
1. The parties call $\Pi_{\text{PubOpen}}([\lambda_x], P_i)$.
2. If this call does not result in abort, P_i outputs $\mu_x + \lambda_x$.

[a] Since Π_{PubOpen} takes as inputs *batches* of shares to be opened, this is called for all the multiplication gates on the given layer of the circuit in parallel, doing multiple calls if necessary.

Fig. 5. Online phase of our MPC protocol.

Theorem 6. *Assume that $t < n/3$. Then protocol Π_{Online} implements functionality $\mathcal{F}_{\text{MPC-GOD}}$ in the $(\mathcal{F}_{\text{F.D.Prep}}, \mathcal{F}_{\text{BC}})$-hybrid model with perfect security.*

Proof. Let $\mathcal{A}$ be an adversary actively corrupting t of the parties in the real execution. We will describe a simulator $\mathcal{S}$ who interacts with the adversary and the functionality $\mathcal{F}_{\text{MPC}-\text{GOD}}$ in such a way that the environment $\mathcal{Z}$ cannot distinguish between the two execution.

The simulator $\mathcal{S}$ emulates virtual honest parties that interact with the corrupt parties controlled by $\mathcal{A}$. A crucial property we will make use of throughout this proof is that, given any set of t shares, any secret can be chosen so that a polynomial of degree at most t interpolate the shares and the secret. In particular, since the adversary controls t parties, any shares these parties have can be shares of any secret by choosing appropriately the shares of the honest parties.

Preprocessing. $\mathcal{S}$ begins by emulating the functionality $\mathcal{F}_{\text{F.D.Prep}}$, distributing random shares to the corrupt parties corresponding to the values $[\lambda_x]$ for every gate x, and $([\lambda_x], [\lambda_y], [\lambda_x\lambda_y], \langle\lambda_x\rangle, [r], \langle r\rangle)$ for every multiplication gate with input wires x and y. However, the simulator *does not* sample the λ_x's nor the r's, since as observed above these are not needed to be fixed for the corrupt parties to receive shares of these values. Additionally, as part of the emulation of $\mathcal{F}_{\text{F.D.Prep}}$ the simulator does sample a random $\lambda_x \in R$ for every input wire x corresponding to each corrupt party P_i, and it sends this value to such party.

Input phase. $\mathcal{S}$ emulates $\mathcal{F}_{\text{BC}}$ as part of the emulation of the honest parties in the input phase. $\mathcal{S}$ begins by sending a random value $\mu_x \in R$ through $\mathcal{F}_{\text{BC}}$ on behalf of each honest P_i for every input wire x held by P_i. Notice that the environment cannot distinguish this from the real execution since there $\mu_x = x - \lambda_x$ where x is the real input from party P_i, but λ_x is random and unkown to the adversary so μ_x also looks uniformly random.

As part of the input phase each corrupt party calls $\mathcal{F}_{\text{BC}}$ by providing some value μ_x for each input wire x corresponding to P_i. The simulator, who emulates $\mathcal{F}_{\text{BC}}$ receives μ_x and sets the input of party P_i to be $x := \mu_x + \lambda_x$, where recall λ_x was sent by $\mathcal{S}$ to P_i in the emulation of $\mathcal{F}_{\text{F.D.Prep}}$.

Computation phase. Observe that after the input phase $\mathcal{S}$ knows the shares that the corrupt parties should hold for each input wire, and it also knows the associated μ_x. The same holds for the preprocessing data. The goal is to keep this invariant throughout the computation. For addition gates $\mathcal{S}$ does nothing as this is a local operation, it only records what shares the corrupt parties should compute.

For multiplication gates $\mathcal{S}$ must emulate the calls $\{\gamma_i\}_{i=0}^t \leftarrow \Pi_{\text{PubOpen}}(\{[\gamma_i]\}_{i=0}^t)$ and $\{\rho_i\}_{i=0}^t \leftarrow \Pi_{\text{PubOpen}}(\{\langle\rho_i\rangle\}_{i=0}^t)$. Observe that, since $t < n/3$, it holds that $2t < n - t$, so by the properties of Π_{PubOpen} studied in Section 3.2, the calls to in the real execution of this protocol will result in the parties learning the correct secret-shared values. In the real execution we have each $[\gamma_i]$ and each $\langle\rho_i\rangle$ has the form $\mu_x\mu_y + \mu_x[\lambda_y] + [\lambda_x\lambda_y] - [\lambda_z] + [r]$ and $\langle\lambda_x\rangle\mu_y - \langle r\rangle$, respectively. In particular, $\mathcal{S}$ knows the shares of each $[\gamma_i]$ and $\langle\rho_i\rangle$ that the corrupt parties hold. We begin by discussing how $\mathcal{S}$ emulates the call $\{\gamma_i\}_{i=0}^t \leftarrow \Pi_{\text{PubOpen}}(\{[\gamma_i]\}_{i=0}^t)$. First, $\mathcal{S}$ samples uniformly random values $\gamma_i \in R$ for $i = 0, \ldots, t$ and computes the shares of each $[\gamma_i]$ corresponding to the honest parties so that these are consistent with the shares held by the corrupt parties. Then, $\mathcal{S}$ defines the polynomial $f(\mathtt{X}) = \sum_{j=0}^t \gamma_j \mathtt{X}^j$, and then it computes internally the shares of $[f^{\mathsf{L}}(\alpha_i)] = \sum_{j=0}^t \alpha_i^j [\gamma_j]$ for $i = 1, \ldots, n$ corresponding to all parties.

At this point the emulated honest parties have well-defined shares of $\{[f^{\mathsf{L}}(\alpha_i)]\}_{i=1}^n$, so they execute step 2 in the Π_{PubOpen} procedure by sending their shares of $[f^{\mathsf{L}}(\alpha_i)]$ to party P_i for $i = 1, \ldots, n$. Step 3 in the procedure is emulated by each virtual honest party P_i sending $f(\alpha_i)$ to all parties. We observe that $\mathcal{Z}$ cannot distinguish this emulation from the real execution since the

virtual honest parties are behaving exactly as the real honest parties would, and, like in the ideal execution, the values of γ_i for $i = 0, \ldots, t$ look uniformly random in the real execution since recall these have the form $\mu_x\mu_y + \mu_x\lambda_y + \lambda_x\lambda_y - \lambda_z + r$, which is uniformly random thanks to the addition of the uniformly random value $-\lambda_z$, which is unknown to the adversary. Here we are using the fact that, due to error correction (see Section 3.2), since there are at most t errors and $n > 3t$, each honest party P_i in the real execution learns the correct $f(\alpha_i)$ in the first round of Π_{PubOpen} and therefore they also learn the correct polynomial $f(x)$ in the second round.

The emulation of the call $\{\rho_i\}_{i=0}^t \leftarrow \Pi_{\text{PubOpen}}(\{\langle\rho_i\rangle\}_{i=0}^t)$ is done in a similar way as above, except that this time the indistinguishability argument between real and ideal worlds follows from the fact that each ρ_i has the form $\lambda_x\mu_y - r$, which looks uniformly random to the adversary due to the mask r. At this point the invariant is preserved: $\mathcal{S}$ knows the μ_z associated to the output of the multiplication gates, and it also knows the shares of $[\lambda_z]$ that the adversary should hold.

Output phase. Let x be the input of a corrupt party P_i extracted by $\mathcal{S}$ in the input phase. $\mathcal{S}$ calls the functionality $\mathcal{F}_{\text{MPC}-\text{GOD}}$ using these as the inputs from the corrupt parties, and obtains an output z_j for each party P_j. For each corrupt party P_j the simulator, who knows the shares of $[\lambda_{z_j}]$ that the corrupt parties hold, computes the shares that the honest parties should hold so that the reconstructed value is $\lambda_{z_j} := z_j - \mu_{z_j}$, and the virtual parties send these shares to P_j as part of the emulation of the output phase. Notice that this is indistinguishable to $\mathcal{Z}$ from the real execution since it is easy to see that in the latter μ_{z_j} and λ_{z_j} will be uniformly random conditioned on $z_j = \lambda_{z_j} + \mu_{z_j}$, which is the exact same distribution in the ideal execution. This concludes the proof of the theorem. ■

We recall that the functionality $\mathcal{F}_{\text{BC}}$ can be instantiated with perfect security if $t < n/3$ [LSP82], which, together with Theorem 6, implies that there exists a protocol that instantiates $\mathcal{F}_{\text{MPC}-\text{GOD}}$ with perfect security in the $\mathcal{F}_{\text{F.D.Prep}}$-hybrid model.

Finally, in a similar way as the theorem above, the following can be proved. The main difference lies in the fact that the simulator may send abort signals to the functionality $\mathcal{F}_{\text{MPC}-\text{abort}}$ if it detects that the adversary is sending inconsistent shares. This works since error detection in the $t < n/2$ case is possible.

Theorem 7. *Assume that $t < n/2$. Then protocol Π_{Online} implements functionality $\mathcal{F}_{\text{MPC}-\text{abort}}$ in the $(\mathcal{F}_{\text{F.D.Prep}}, \mathcal{F}_{\text{BC}})$-hybrid model with satistical security.*

5 Preprocessing

In this section we provide different protocols to realize the $\mathcal{F}_{\text{Tuples}}$ functionality when $Z(R)$ does not contain a big enough exceptional set. Our presentation focuses in this simpler functionality, since $\mathcal{F}_{\text{F.D.Prep}}$ can be easily realized either in the $\mathcal{F}_{\text{Tuples}}$-hybrid or by slightly tweaking the protocols that implement $\mathcal{F}_{\text{Tuples}}$. In Section 5.1 we provide a generic protocol that only requires *black-box* access to the ring operations and the ability to sample random ring elements. On the downside, this theoretical result has a complexity of $\mathsf{poly}(n)$, in contrast with the more specialized protocol for matrices over commutative rings we provide in Section 5.2. By additionally getting black-box access to the commutative ring operations, this optimized protocol has $O(n)$ communication complexity and $O(n \log n)$ computational complexity.

5.1 Generic, Black-box Construction

Representing non-commutative ring arithmetic as operations in $\mathbb{G} = \mathsf{GL}_3(R)$. We quickly recap the work of Ben-Or and Cleve [BC92]. Let $a \in R$, where R is a possibly non commutative ring. We will keep the invariant of representing such elements within the group of 3×3 invertible matrices over R, $\mathbb{G} = \mathsf{GL}_3(R)$, as follows:

$$M(a) = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This allows us to compute additions as $M(a+b) = M(a) \cdot M(b)$. Multiplication is a bit more complicated. We can compute $M(a \cdot b) = J_1 \cdot M(b) \cdot J_2 \cdot M(a) \cdot J_3 \cdot M(b) \cdot J_4 \cdot M(a) \cdot J_5$, where the J_i matrices are the following:

$$J_1 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad J_2 = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad J_3 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad J_4 = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad J_5 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Preprocessing for MPC over non-commutative rings Given the previous representation, we can use existing results for efficient MPC over non-abelian groups [DPS+12, CDI+13] in order to implement the $\mathcal{F}_{\text{Prep}}$ functionality required for the online phase in Section 4. In more detail, this can be computed as a constant-depth arithmetic circuit over R which we will represent as a series of products in the group $\mathbb{G} = \mathsf{GL}_3(R)$.

Note that the shares in the group-based protocol are according to the group law (i.e., they are "multiplicative shares"), whereas the protocols from Section 4 use Shamir's secret sharing. One option would be to compute something like $[M(a)]_{\mathbb{G}} = \prod_{i=1}^{t+1} [M(a_i)]_{\mathbb{G}}$, $[M(b)]_{\mathbb{G}} = \prod_{i=1}^{t+1} [M(b_i)]_{\mathbb{G}}$ and $[M(c)]_{\mathbb{G}} = J_1 \cdot [M(b)]_{\mathbb{G}} \cdot J_2 \cdot [M(a)]_{\mathbb{G}} \cdot J_3 \cdot [M(b)]_{\mathbb{G}} \cdot J_4 \cdot [M(a)]_{\mathbb{G}} \cdot J_5$, as well as generating "double shares" of the form $[M(r)]_{\mathbb{G}}, [r]$ to e.g. extract $(c+r)$ from $\mathsf{PublicOpen}([M(c)]_{\mathbb{G}} \cdot [M(r)]_{\mathbb{G}})$ and then compute $[c] = (c+r) - [r]$.

Alternatively, we can employ the following, more direct approach. Let $A = \{0, \alpha_1, \ldots, \alpha_n\}$ be the commutative exceptional set defining the non-commutative sharing scheme $[\cdot]$. Parties compute the following circuit, where each P_i inputs random $f_j^i, g_j^i, h_j^i \in R$ and receives as output their corresponding shares of $([a], [b], [c])$, that is, $(f(\alpha_i), g(\alpha_i), h(\alpha_i))$.

$$\begin{aligned} a &= \sum_{i=1}^{t+1} f_0^i, \quad b = \sum_{i=1}^{t+1} g_0^i, \quad c = a \cdot b \\ f_j &= \sum_{i=1}^{t+1} f_j^i, \quad g_j = \sum_{i=1}^{t+1} g_j^i, \quad h_j = \sum_{i=1}^{t+1} h_j^i, && j = 1, \ldots, t \\ f(\alpha_\ell) &= a + \sum_{j=1}^{t} f_j \alpha_\ell^j, \quad g(\alpha_\ell) = b + \sum_{j=1}^{t} g_j \alpha_\ell^j, \quad h(\alpha_\ell) = c + \sum_{j=1}^{t} h_j \alpha_\ell^j, && \ell = 1, \ldots, n \end{aligned}$$

The downside of these two generic approaches is that their respective protocols inherit the $\mathsf{poly}(n)$ complexity of [CDI+13]. On the upside, any improvement to MPC over non-abelian groups would directly carry over to our blackbox preprocessing.

5.2 Concretely Efficient Preprocessing for Matrix Rings ($t < n/3$)

For our more practical construction, which works over the ring $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, we describe how to implement $\mathcal{F}_{\text{Tuples}}$ using non-black-box protocols which are more efficient than the one from the previous subsection. Even though we specialize to matrices over $\mathbb{Z}_{2^k}$, our analysis and techniques can be generalized to matrices over other commutative rings.

Remember from Section 3.1 that $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ contains a commutative exceptional set of size 2^m, which is why can only use the non-multiplicative secret sharing schemes from Theorem 5 that are linear only on one side.

In order to overcome the lack of multiplicativity, as $\mathcal{F}_{\text{Tuples}}$ requires to produce values $([A], [B], [C])$ such that $C = A \cdot B$, we will also use an MPC protocol for computation over $\mathbb{Z}_{2^k}$. Applying this protocol to the entries of the matrix ring, we can trivially emulate the whole arithmetic of R. The issue is that, by doing this, we need to work over a big enough Galois extension of $\mathbb{Z}_{2^k}$, so that we can define a multiplicative, Shamir-style linear secret sharing scheme $[\![\cdot]\!]$[7]. Once we have computed this matrix product from the entry-wise shares of the matrices A and B, we need to convert $[\![\cdot]\!]$ sharings of the entries of A, B, C to sharings $[\cdot]$ and $\langle\cdot\rangle$ over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, so that parties obtain the tuple $[A], \langle A\rangle, [B], [C], [r], \langle r\rangle$ required for the online protocols in Section 4.

In particular, we will use the $\mathsf{InnerProd}$ CAFE from [DLS20], which can compute inner products of length $\delta \simeq d/2$ over $S = \mathsf{GR}(2^k, d)$ at the cost of just 2 sharings and a single opening in S. If one wants to calculate an inner product of length $rd/2$, the cost would be $2r$ sharings and a single opening in S. The following proposition captures the properties of $\mathsf{InnerProd}$ we are interested in, without getting into details about the specific construction.

Proposition 4 ([DLS20]). *Let $S = \mathsf{GR}(2^k, d)$ be a Galois Ring defined as $\mathbb{Z}_{2^k}[\mathtt{X}]/(h(\mathtt{X}))$. Let $\tilde{d}$ denote the degree of the second-highest degree monomial in $h(\mathtt{X})$. Let $\delta \in \mathbb{N}$ be such that $\delta < (d+1)/2$, $\delta < d - \tilde{d} + 1$. There exist three $\mathbb{Z}_{2^k}$-linear homomorphisms $\mathsf{E}_L : (\mathbb{Z}_{2^k})^\delta \to S$, $\mathsf{E}_R : (\mathbb{Z}_{2^k})^\delta \to S$ and $\mathsf{E}_{out} : \mathbb{Z}_{2^k} \to S$ satisfying:*

$$\mathsf{E}_L(a_1, \ldots, a_\delta) \cdot \mathsf{E}_R(b_1, \ldots, b_\delta) + \mathsf{E}_{out}(c) = \mathsf{E}_{out}(c + \sum_{\ell=1}^{\delta} a_\ell \cdot b_\ell)$$

Furthermore, the value $\mathsf{E}_{out}(c+\sum_{\ell=1}^{\delta} a_\ell \cdot b_\ell) \in S$ does not reveal any information beyond $c+\sum_{\ell=1}^{\delta} a_\ell \cdot b_\ell \in \mathbb{Z}_{2^k}$.

Since the maps $\mathsf{E}_L, \mathsf{E}_R$ and E_{out} are homomorphisms of $\mathbb{Z}_{2^k}$-modules, the image of each of them can be seen as a $\mathbb{Z}_{2^k}$-submodule of $\mathsf{GR}(2^k, d)$. We will indistinctively refer to either these homomorphisms, or the $\mathbb{Z}_{2^k}$-modules they define as *encodings*.

By extension, we define how these encodings can be applied to matrices. Given $A' \in \mathcal{M}_{1\times\delta}(\mathbb{Z}_{2^k})$, $B' \in \mathcal{M}_{\delta\times 1}(\mathbb{Z}_{2^k})$, for which we want to compute $C' = A' \cdot B'$, where $C' \in \mathbb{Z}_{2^k}$, we simply view the entries of A', B' as elements of $(\mathbb{Z}_{2^k})^\delta$, to which we apply E_L and E_R, respectively. In order to compute the product of $A, B \in \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, we need to introduce some additional notation. Let $\Delta = \lceil m/\delta \rceil$. Let $\mathcal{A} \in \mathcal{M}_{m\times\delta\Delta}(\mathbb{Z}_{2^k})$ (resp. $\mathcal{B} \in \mathcal{M}_{\delta\Delta\times m}(\mathbb{Z}_{2^k})$) denote the matrix A padded with

[7] This was described in [ACD+19], but it is also a consequence of Theorem 5. This is why we will use the $[\![\cdot]\!]$ notation to refer to the LSSS over the Galois Ring in this section. It should not be confused with $[\cdot]$ and $\langle\cdot\rangle$, which work over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$.

$\delta\Delta - m$ columns of zeroes (resp. the matrix B padded with $\delta\Delta - m$ rows of zeroes). For $\ell \in [m\Delta]$, let $A^{(\ell)} \in \mathcal{M}_{1\times\delta}(\mathbb{Z}_{2^k}), B^{(\ell)} \in \mathcal{M}_{\delta\times 1}(\mathbb{Z}_{2^k})$ be submatrices such that

$$\mathcal{A} = \begin{pmatrix} A^{(1)} & A^{(2)} & \dots & A^{(\Delta)} \\ A^{(\Delta+1)} & A^{(\Delta+2)} & \dots & A^{(2\cdot\Delta)} \\ \vdots & \vdots & \ddots & \vdots \\ A^{((m-1)\cdot\Delta+1)} & A^{((m-1)\cdot\Delta+2)} & \dots & A^{(m\cdot\Delta)} \end{pmatrix} \quad \mathcal{B} = \begin{pmatrix} B^{(1)} & B^{(\Delta+1)} & \dots & B^{((m-1)\cdot\Delta+1)} \\ B^{(2)} & B^{(\Delta+2)} & \dots & B^{((m-1)\cdot\Delta+2)} \\ \vdots & \vdots & \ddots & \vdots \\ B^{(\Delta)} & B^{(2\cdot\Delta)} & \dots & B^{(m\cdot\Delta)} \end{pmatrix}, \tag{2}$$

where $A^{(\Delta)}, A^{(2\cdot\Delta)}, \dots, A^{(m\cdot\Delta)}$ and $B^{(\Delta)}, B^{(2\cdot\Delta)}, \dots, B^{(m\cdot\Delta)}$ are the submatrices including the zero-padding. Let $\gamma \in \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ be a matrix that we will use to mask the result of $C = A \cdot B$ and let us denote the entries of $\gamma, C \in \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ as $C^{(\alpha,\beta)}, \gamma^{(\alpha,\beta)} \in \mathbb{Z}_{2^k}$, where $\alpha, \beta \in \{1, \dots, m\}$. Taking into account Definition 4, we can compute:

$$\mathsf{E}_{out}(C^{(\alpha,\beta)} + \gamma^{(\alpha,\beta)}) = \mathsf{E}_{out}(\gamma^{(\alpha,\beta)}) + \sum_{\ell=1}^{\Delta} \mathsf{E}_L(A^{((\alpha-1)\Delta+\ell)}) \cdot \mathsf{E}_R(B^{((\beta-1)\Delta+\ell)}) \tag{3}$$

Generating consistent secret-shared encodings and matrices. Towards obtaining the tuple $([A], \langle A\rangle, [B], [r], \langle r\rangle, [C])$, parties first need to generate random matrix shares $([A], \langle A\rangle, [B], [r], \langle r\rangle)$ and commutative shares $[\![\mathsf{E}_L(A^{(\ell)})]\!]$ and $[\![\mathsf{E}_R(B^{(\ell)})]\!]$ of A and B, where the latter described in Eq. (2). The parties also need shares of a random matrix $[\gamma]$, together with commutative shares $[\![\mathsf{E}_{out}(\gamma^{(\alpha,\beta)})]\!]$. These will be useful to obtain $[C]$ such that $C = AB$ later on.

The method we employ in this step is based on the following simple observation. Unlike the final preprocessing material, which includes $C = AB$, the one we just described does not have any multiplicative relation. In fact, since both the secret sharing of elements of $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$ and of $\mathsf{InnerProd}$ encodings can be seen as $\mathbb{Z}_{2^k}$-modules[8], the correlations among these random values are $\mathbb{Z}_{2^k}$-linear, which enables a very simple protocol for generating them. The simplest protocol works as follows: Every party distributes correlated shares as above, where the secrets are known to that party, and then these shares are added together to obtain final shares where the underlying secrets are random and unknown to any party. However, such method has two main issues. First, it requires has an undesirable communication complexity of $\Omega(n^2)$. Second, actively corrupt parties may not behave correctly when distributing the shares, by sharing underlying secrets that do not have the right relations among themselves or sending shares that are not consistent with a polynomial of the right degree.

The two issues above are simultaneously handled by a protocol based on Hyper-Invertible Matrices [BTH08], which are useful tool to generate and check linearly correlated randomness. Hyper-Invertible Matrices (HIMs) were introduced in the context of finite fields, and were later on generalized to the context of Galois rings in [ACD+19, DLS20], although for different types of correlations than the ones in our protocol. The remaining results we will present in this section follow from reasoning about tensor products and reducing to the works above. The reason behind having to turn to tensor products is that whereas the correlations we care about are $\mathbb{Z}_{2^k}$-linear, there are no (big enough) HIMs over $\mathbb{Z}_{2^k}$, which requires moving to a Galois Ring extension in order to find them. For completeness, we give in Section C in the Appendix a more detailed and self-contained

[8]The structure of $[\cdot]$ (resp. $\langle\cdot\rangle$), is a left (resp. right) $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$-module is compatible with its structure as a $\mathbb{Z}_{2^k}$-bimodule.

description of these techniques. A different, more explicit presentation of the protocols leading to Theorem 8 can be found in the proceedings version of this work at CRYPTO'21.

Recall that the correlations we need to generate correspond to

$$\big([A], \langle A\rangle, [B], [r], \langle r\rangle, \{[\![\mathsf{E}_L(A^{(\ell)})]\!]\}_\ell, \{[\![\mathsf{E}_R(B^{(\ell)})]\!]\}_\ell, [\gamma], \{[\![\mathsf{E}_{out}(\gamma^{(\alpha,\beta)})]\!]_{2t}\}_{\alpha,\beta}\big), \tag{4}$$

where

- All sharings are consistent with the corresponding type of polynomial evaluation, resulting on a polynomial of adequate degree. In particular, all secrets are well defined.
- The secret underlying $[A]$ (resp. $[r]$) and $\langle A\rangle$ (resp. $\langle r\rangle$) is the same matrix A (resp. r).
- $\{\mathsf{E}_L(A^{(\ell)})\}_\ell$, the secrets underlying $\{[\![\mathsf{E}_L(A^{(\ell)})]\!]\}_\ell$, are correct encodings of the matrix A, as described in the beginning of Section 5.2. Similarly for $\{[\![\mathsf{E}_R(B^{(\ell)})]\!]\}_\ell$.
- $\{\mathsf{E}_{out}(\gamma^{(\alpha,\beta)})\}_{\alpha,\beta}$, the secrets underlying $\{[\![\mathsf{E}_{out}(\gamma^{(\alpha,\beta)})]\!]_{2t}\}_{\alpha,\beta}$, are correct encodings of the matrix γ, which is the matrix underlying the shares $[\gamma]$.

Each party P_i will get a set of shares for the correlations above, which we will generically denote by $\boldsymbol{x}_i$ in the following. Some of these shares are matrices in $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, while some others are elements in $S = \mathsf{GR}(2^k, d)$. In any case, we can represent both of these types of shares as vectors of elements in $\mathbb{Z}_{2^k}$, which ultimately means that $\boldsymbol{x}_i$ can be regarded as a vector in $\mathbb{Z}_{2^k}^L$ for some L.[9]

We denote by $N \subseteq \mathbb{Z}_{2^k}^{L\cdot n}$ the set of shares $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ that constitute valid correlations under the definition above. An important property of N is that $\mathbb{Z}_{2^k}$-linear combinations of elements in N result in elements in N. In other words, N is a $\mathbb{Z}_{2^k}$-module. This allows us to consider the tensor product $N \otimes_{\mathbb{Z}_{2^k}} S$, which is isomorphic to N^d and is an S-submodule of $\mathbb{Z}_{2^k}^{L\cdot n} \otimes S \cong S^{L\cdot n}$.

If the parties obtain correlations in N^d, this corresponds to d correlations in N, which is our ultimate goal. It can be seen from the correspondences that N^d, seen as an S-submodule of $S^{L\cdot n}$, represents a linear correlation among the n parties, that is, elements in N^d can be written as $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ for $\boldsymbol{y}_i \in S^L$, where party P_i holds the "share" $\boldsymbol{y}_i$. The following proposition, which is fully proved in Section C in the Appendix, follows from the works [CCXY18, ACD+19, DLS20] and shows that there is an efficient protocol to generate this type of correlations.

Proposition 5. *Let M be an S-submodule of $S^{L\cdot n}$ representing a linear correlation among n parties. Then there exists a perfectly secure protocol for the parties to generate $t+1$ such correlations with a communication complexity of $O(L \cdot n^2)$ elements in S.*

Using the protocol above, the parties can generate $t + 1 = \Omega(n)$ correlations in N^d with a communication complexity of $O(L \cdot n^2 \cdot d)$ elements in $\mathbb{Z}_{2^k}$. Since $t+1$ correlations in N^d correspond to $d \cdot (t + 1)$ correlations in N, this leads to a communication complexity of $O(L \cdot n)$ elements in $\mathbb{Z}_{2^k}$ per correlation or, altenatively, and since $L = \Theta(dm^2)$, $O(n \log n)$ matrices over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$.

Obtaining the multiplicative correlation. Finally, we show how to leverage the shares from Eq. (4) to produce the correlated randomness we need in our preprocessing, namely $([A], \langle A\rangle, [B], [r], \langle r\rangle, [C])$, where $C = AB$. The protocol is described in Fig. 6.

Notice that the only interaction of the protocol comes from the reconstruction of m^2 shared values over S, which is done with a communication complexity of $O(n)$ elements of S per reconstructed value. This leads to a communication complexity of $O(n \cdot m^2 \cdot d)$ bits, equivalent to $O(n \cdot \log n)$ matrices over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, since $d = \Theta(\log n)$.

Protocol Π_{Tuples}

Assume correlation

$$\left([A], \langle A\rangle, [B], [r], \langle r\rangle, \{[\![\mathsf{E}_L(A^{(\ell)})]\!]\}_\ell, \{[\![\mathsf{E}_R(B^{(\ell)})]\!]\}_\ell, [\gamma], \{[\![\mathsf{E}_{out}(\gamma^{(\alpha,\beta)})]\!]_{2t}\}_{\alpha,\beta}\right)$$

1. For $\alpha, \beta \in \{1, \ldots, m\}$, the parties locally compute

$$[\![\mathsf{E}_{out}(C^{(\alpha,\beta)} + \gamma^{(\alpha,\beta)})]\!]_{2t} = [\![\mathsf{E}_{out}(\gamma^{(\alpha,\beta)}))]\!]_{2t} + \sum_{\ell=1}^{\Delta} [\![\mathsf{E}_L(A^{((\alpha-1)\Delta+\ell)})]\!]_t \cdot [\![\mathsf{E}_R(B^{((\beta-1)\Delta+\ell)})]\!]_t$$

2. Parties use the double shares of $\gamma \in R$ in order to convert $\{[\![\mathsf{E}_{out}(C^{(\alpha,\beta)} + \gamma^{(\alpha,\beta)})]\!]_{2t}\}_{\alpha,\beta}$ into $[C]$, as follows:
   (a) For $\alpha, \beta \in \{1, \ldots, m\}$ parties call Π_{PubOpen} with the values $[\![\mathsf{E}_{out}(C^{(\alpha,\beta)} + \gamma^{(\alpha,\beta)})]\!]_{2t}$, so that everyone obtains $C + \gamma \in \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$, or abort.
   (b) Parties compute
   $$[C]_t = C + \gamma - [\gamma]_t$$
3. If no abort was produced in the previous steps, the parties output the shares
   $$([A], \langle A\rangle, [B], [r], \langle r\rangle, [C]).$$

Fig. 6. Preprocessing phase for MPC over $R = \mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$.

The following theorem summarizes the security properties of the protocol Π_{Tuples}.

Theorem 8. *Assume that $t < n/3$. Then protocol Π_{Tuples} on Figure 6 implements functionality $\mathcal{F}_{\text{Tuples}}^{\text{abort}}$ in the $\mathcal{F}_{\text{BC}}$-hybrid model with perfect security.*

Proof. Let $\mathcal{A}$ be an adversary actively corrupts a subset $\{P_i\}_{i\in\mathcal{C}}$ of parties such that $|\mathcal{C}| = t < n/3$. Let $\mathcal{H}$ denote the indices of honest parties.

We describe a simulator $\mathcal{S}$, which interacts with $\mathcal{A}$ and emulates the role of honest parties and functionality $\mathcal{F}_{\text{BC}}$ towards the environment $\mathcal{Z}$. $\mathcal{S}$ emulates the generation of the prior correlation. $\mathcal{S}$ also emulates virtual honest parties, and executes the protocol *exactly* as the real honest parties would do (in case of abort $\mathcal{S}$ instructs the functionality $\mathcal{F}_{\text{Tuples}}$ to abort). Observe that the protocol has no inputs, so, trivially, the real and ideal executions before outputs are provided are perfectly indistinguishable.

To argue that the real and ideal executions are indistinguishable we only need to show that the distribution of the shares that the parties get at the end of either execution are the same. This, however, follows trivially from the fact that the first part of the output, $([A], \langle A\rangle, [B], [r], \langle r\rangle)$, has the right distribution (as it comes from the preprocessing) and the fact that the computed $[C]$ is equal to $[A \cdot B]$ by the multiplicative properties described in Eq. (3). ∎

5.3 Concretely Efficient Preprocessing for Matrix Rings ($n/3 \leq t < n/2$)

The (statistically secure) protocol for instantiating $\mathcal{F}_{\text{Tuples}}$ when $n/3 \leq t < n/2$ is very similar to the one from Section 5.2, so we limit ourselves to providing an overview only. In this new setting we cannot rely on hyperinvertible matrices anymore. Instead, as in [DN07], and with the help of a simple adaptation of the techniques from Section 5.2, we can make use of a Vandermonde matrix

[9] It will be important for our communication complexity results to note that $L = O(dm^2)$.

as a randomness extractor in order to obtain the final sharings that are intended to look random to the adversary. The main caveat in this first step is that a corrupt party may deviate from the protocol specification and may distribute inconsistent shares. This can be easily fixed with the help of Lemma 4—which is enough to prove a Schwartz-Zippel kind of result—by opening a random linear combination of the values to be checked. Alternatively, we can use standard cut-and-choose techniques, which are of a more combinatoric nature and agnostic to the underlying ring structure.

Unfortunately, even with these checks in place, the adversary can inject additive errors when opening shared values with a polynomial of degree at most $2t$, since the bound $t < n/2$ is not enough to guarantee even error detection. As a result, the functionality that we can instantiate by making use of these adaptations is not precisely $\mathcal{F}_{\text{Tuples}}$, but rather a "faulty" version of it in which the adversary is able to inject additive errors in the product part of the tuples. This is formalized as Functionality $\mathcal{F}_{\text{FaultyTuples}}$ in Fig. 7.

Functionality $\mathcal{F}_{\text{FaultyTuples}}$

The functionality runs with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{A}$ who corrupts t parties. On input $(\texttt{Tuples}, m)$, the functionality proceeds as follows:

1. Wait for a message from $\mathcal{A}$. If the message is `abort`, then the functionality notifies honest parties and aborts.
2. Otherwise, if the message is `ok` then, for $j = 1, \ldots, m$, receive from the adversary an additive error $\epsilon_j \in R$.
3. For $j = 1, \ldots, m$, sample values $a_j, b_j, r_j \xleftarrow{\$} R$.
4. Generate sharings $[a_j]_t, [b_j]_t, [a_j \cdot b_j + \epsilon_j]_t, \langle a_j \rangle_t, [r_j]_t, \langle r_j \rangle_t$ and distribute them to the parties.

Fig. 7. Preprocessing functionality.

What remains is to instantiate $\mathcal{F}_{\text{Tuples}}$ in the $\mathcal{F}_{\text{FaultyTuples}}$-hybrid model. This is achieved by generalizing the so-called *sacrificing* technique to the non-commutative domain as follows. Let $[a]_t, [b]_t, [a \cdot b + \epsilon]_t, \langle a \rangle_t, [r]_t, \langle r \rangle_t$ and $[a']_t, [b']_t, [a' \cdot b' + \epsilon']_t, \langle a' \rangle_t, [r']_t, \langle r' \rangle_t$ be two tuples produced by $\mathcal{F}_{\text{FaultyTuples}}$. To check that the first tuple (or the second) is correct, the parties run the following check. We assume the parties have access to a functionality for sampling public randomn values from a commutative exceptional set A.

1. Sample a public random value $q \in A$.
2. Call $d = \Pi_{\text{PubOpen}}([a] - q[a'])$ and $e = \Pi_{\text{PubOpen}}([b] - [b'])$.
3. Call $f = \Pi_{\text{PubOpen}}(\langle a' \rangle \cdot e + \langle r \rangle)$
4. Call $z = \Pi_{\text{PubOpen}}(d \cdot e + d \cdot [b'] + q \cdot f - q \cdot [r] + q \cdot [a' \cdot b' + \epsilon'] - [a \cdot b + \epsilon])$
5. Abort if $z \neq 0$. Else, output $[a]_t, [b]_t, [a \cdot b + \epsilon]_t, \langle a \rangle_t, [r]_t, \langle r \rangle_t$ as a valid tuple.

To analyze the correctness of this method, first observe that from the definition of z it holds that $z = q \cdot \epsilon' - \epsilon$, so the final check passes if and only if $q \cdot \epsilon' - \epsilon = 0$. If $\epsilon \neq 0$, then $g(\mathtt{X}) = \epsilon' \cdot \mathtt{X} - \epsilon$ is a non-zero polynomial such that $g^{\mathsf{L}}(q) \neq 0$. From Lemma 4 we see that this check can only be satisfied with probability $1/|A|$. If this is not small enough, this check can be repeated with more faulty tuples until the desired error probability is achieved. We see then that, with high probability, $\epsilon = 0$, which implies that the tuple $[a]_t, [b]_t, [a \cdot b + \epsilon]_t, \langle a \rangle_t, [r]_t, \langle r \rangle_t$ is correct.

Acknowledgements

Daniel Escudero was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 669255

(MPCPRO). During his time at Aarhus University, Eduardo Soria-Vazquez was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

References

ACD+19. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 471–501. Springer, Heidelberg, December 2019.

ACE+21. Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. Cryptology ePrint Archive, Report 2021/253, 2021. https://ia.cr/2021/253.

AV05. V. Arvind and T. C. Vijayaraghavan. The complexity of solving linear equations over a finite ring. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, pages 472–484, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

BBY20. Alessandro Baccarini, Marina Blanton, and Chen Yuan. Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. Cryptology ePrint Archive, Report 2020/1577, 2020. https://eprint.iacr.org/2020/1577.

BC92. Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

BNO19. Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 530–549. Springer, Heidelberg, June 2019.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426. Springer, Heidelberg, August 2018.

CDI+13. Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 185–202. Springer, Heidelberg, August 2013.

CDM00. Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer, Heidelberg, May 2000.

CFIK03. Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 596–613. Springer, Heidelberg, May 2003.

CRX19. Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. Asymptotically-good arithmetic secret sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with strong multiplication and its applications to efficient mpc. Cryptology ePrint Archive, Report 2019/832, 2019. https://eprint.iacr.org/2019/832.

DEK21. Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. *To appear at USENIX'21*, 2021.

DKH+12. Anuj Dawar, Eryk Kopczynski, Bjarki Holm, Erich Grädel, and Wied Pakusa. Definability of linear equation systems over groups and rings. *arXiv preprint arXiv:1204.3022*, 2012.

DLS20. Anders P. K. Dalskov, Eysa Lee, and Eduardo Soria-Vazquez. Circuit amortization friendly encodingsand their application to statistically secure multiparty computation. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 213–243. Springer, Heidelberg, December 2020.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.

DPS+12. Yvo Desmedt, Josef Pieprzyk, Ron Steinfeld, Xiaoming Sun, Christophe Tartary, Huaxiong Wang, and Andrew Chi-Chih Yao. Graph coloring applied to secure computation in non-abelian groups. *Journal of Cryptology*, 25(4):557–600, October 2012.

ED20. Daniel Escudero and Anders Dalskov. Honest majority mpc with abort with minimal online communication. Cryptology ePrint Archive, Report 2020/1556, 2020. https://eprint.iacr.org/2020/1556.

Fru76. Mikhail Aleksandrovich Frumkin. An application of modular arithmetic to the construction of algorithms for solving systems of linear equations. In *Doklady Akademii Nauk*, volume 229, pages 1067–1070. Russian Academy of Sciences, 1976.

LSP82. LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

Ore31. Oystein Ore. Linear equations in non-commutative fields. *Annals of Mathematics*, pages 463–477, 1931.

QBC13. Guillaume Quintin, Morgan Barbier, and Christophe Chabot. On generalized reed–solomon codes over commutative and noncommutative rings. *IEEE transactions on information theory*, 59(9):5882–5897, 2013.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Son75. Eduardo D Sontag. On linear systems and noncommutative rings. *Mathematical systems theory*, 9(4):327–344, 1975.

vzGS76. Joachim von zur Gathen and Malte Sieveking. Iv. weitere zum erfüllungsproblem polynomial äquivalente kombinatorische aufgaben. In *Komplexität von Entscheidungsproblemen Ein Seminar*, pages 49–71. Springer, 1976.

A Deferred Proofs from Section 2.2

For the proofs presented in this Appendix, it is important to first understand the notion of polynomial ring and evaluation maps described in Section 2.2 (even if the statement appeared before introducing these two concepts in that Section).

Lemma 6 (Lemma 1, restated).

1. *$a \in R^*$ if and only if a is both left-invertible and right-invertible.*
2. *If a has a right inverse, then a is not a right zero divisor.*
3. *If R is finite, then every element which has a right inverse is a unit.*

Proof. (3) Let $a \in R$ have a right inverse b, i.e. $a \cdot b = 1$ and $f_a : R \to R$ be the polynomial given by $f_a(x) = a \cdot x$. We show that $f_a(x)$ is injective by evaluation on the left. Let $b_1, b_2 \in R$ such that ${f_a}^{\mathsf{L}}(b_1) = {f_a}^{\mathsf{L}}(b_2)$, i.e. $b_1 \cdot a = b_2 \cdot a$. This is equivalent to $b_1 \cdot a \cdot b = b_2 \cdot a \cdot b \Leftrightarrow b_1 \cdot 1 = b_2 \cdot 1$.

Since f_a is injective by evaluation on the left, then it has to be surjective, because R is finite. Hence, there exists $c \in R$ such that $1 = {f_a}^{\mathsf{L}}(c) = c \cdot a$. We conclude from (1) that $a \in R^*$. ∎

Lemma 7 (Lemma 2, restated). *Let R be a finite ring. Then all non-zero elements of R are either a unit or a zero divisor.*

Proof. For every $a \in R \setminus \{0\}$, let $f_a, g_a : R \to R$ be the polynomials given by $f_a(x) = a \cdot x$ and $g_a(x) = x \cdot a$. If f_a is injective, then it has to be surjective, because R is finite. Therefore, in such a case there must exist $b \in R$ verifying $f_a(b) = a \cdot b = 1$. Since a has an inverse on the right, by Lemma 1 we conclude that a is a unit.

Assume that $f_a(x)$ is not injective on the right. Then there exist $b_1, b_2 \in R, b_1 \neq b_2$, such that $a \cdot b_1 = a \cdot b_2$, and thus $a \cdot (b_1 - b_2) = 0$. In other words, a is a left zero divisor. Assume now by

contradiction that a is *not* a right zero divisor too. In such a case, we would have that $g_a(x)$ is injective, since if we have $c_1, c_2 \in R$ such that $f_a(c_1) = f_a(c_2)$, it follows that $(c_1 - c_2) \cdot a = 0$ and we made the assumption of a not being a right zero divisor. Since R is finite and $g_a(x)$ is injective, then it is also surjective and there exists $d \in R$ s.t. $d \cdot a = 1$. Remember that a is a left zero divisor, so there also exists $b \in R \setminus \{0\}$ s.t. $a \cdot b = 0$. We are now ready to show a contradiction:

$$b = 1 \cdot b = d \cdot a \cdot b = d \cdot 0 = 0$$

So a has to be a right zero divisor too and, hence, it is a two-sided zero divisor. ■

B Appendix to Section 3

B.1 Solving Linear Systems of Equations over $\mathcal{M}_{m\times m}(\mathbb{Z}_{2^k})$

Begin by noticing that a linear system of r equations in s variables over $\mathcal{M}_{m\times m}(\mathbb{Z}_{p^k})$ translates into a system of $r \cdot m^2$ equations into $s \cdot m^2$ variables over $\mathbb{Z}_{p^k}$, so the task reduces to solving a system of linear equations over $\mathbb{Z}_{p^k}$. Although this task is considerably simpler than solving linear equations over *non-commutative* rings, the fact that $\mathbb{Z}_{p^k}$ is in general not a field makes the problem still hard to approach. Fortunately, there are existing tools in the literature to approach this problem. We begin with the following simple observation:

Proposition 6 (Proposition 1 in [AV05]). *Let A be an $m \times n$ integer matrix, b be an m integer column vector, and p be a prime and k a positive integer. The system of linear equations $Ax = b \pmod{p^k}$ is feasible (in the finite ring $\mathbb{Z}_{p^k}$) if and only if $Ax + p^k y = b$ has a solution in $\mathbb{Z}$.*

Observe that the equation $Ax + p^k y = b$ can be written as $[\ A \mid p^k\mathbb{I}\ ](\ x \mid y\ )^\intercal = b$, so the task of solving a linear system over $\mathbb{Z}_{p^k}$ is reduced to the task of solving a linear system over $\mathbb{Z}$. It has been shown in [Fru76, vzGS76] that such task admits a polynomial time algorithm, which can be used together with the proposition above to solve the linear system of equations that appears when solving for the BW-condition.

Finally, we end with a more efficient alrogithm for the case in which the number of variables equal the number of equations, and matrix defining the linear equation is invertible modulo p. This is captured in the proof of the following result, which is inspired by the proof of Hensel's lemma.

Theorem 9. *Let $A \in \mathcal{M}_{r\times r}(\mathbb{Z}_{p^k})$. For every $0 < e \leq k$, if A is invertible modulo p^e, then A is invertible modulo p^{e+1}. Furthermore, the inverse of A modulo p^{e+1} is efficiently computable from the inverse modulo p^e.*

Proof. Assume that A is invertible modulo p^e, so there exists $A' \in \mathcal{M}_{r\times r}(\mathbb{Z})$ such that $A \cdot A' \equiv \mathbb{I} \pmod{p^e}$. Let us write $A \cdot A' = \mathbb{I} + p \cdot Q$ for some $Q \in \mathcal{M}_{r\times r}(\mathbb{Z})$. Define $P \in \mathcal{M}_{r\times r}(\mathbb{Z})$ as $P = -A' \cdot Q$. It holds then that $A \cdot P = -A \cdot A' \cdot Q$, so $A \cdot P \equiv -Q \pmod{p^e}$, and in particular $p \cdot A \cdot P \equiv -p \cdot Q \pmod{p^{e+1}}$.

Now let $A'' \in \mathcal{M}_{r\times r}(\mathbb{Z})$ be defined as $A'' = A' + p \cdot P$. We have that $A \cdot A'' = A \cdot A' + p \cdot A \cdot P = \mathbb{I} + p \cdot Q + p \cdot A \cdot P$, which modulo p^{e+1} becomes $A \cdot A'' \equiv \mathbb{I} + p \cdot Q - p \cdot Q \equiv \mathbb{I} \pmod{p^{e+1}}$, which shows that A is invertible modulo p^{e+1}. We also see that the inverse of A modulo p^{e+1}, A'', is efficiently computable from A'. ■

From this theorem we get the following corollary.

Corollary 3. *Let $A \in \mathcal{M}_{r\times r}(\mathbb{Z}_{p^k})$. If A is invertible modulo p, then it is invertible modulo p^k and its inverse is efficiently computable.*

Unfortunately, this approach does not naturally extend to the case in which A is not invertible. As an example, take $A = p^k\mathbb{I}$ and let $\boldsymbol{y} = p^{k-1}\mathbf{1}$: modulo p both A and $\boldsymbol{y}$ become zero, so any $\boldsymbol{x}$ satisfies the equation $A\boldsymbol{x} = \boldsymbol{y} \pmod p$. However, not all solutions mod p can be extended to mod p^k since all the solutions modulo p^k satisfy that their reduction modulo p is $\mathbf{1}$.

B.2 Reconstructing Secret-Shared Values Efficiently

We begin by presenting a simple protocol for reconstructing a secret-shared value $[s]_d$ to a single party P_i. The protocol, presented in Fig. 2, works by simply letting each party send its own share to P_i, who locally reconstructs the secret s. Depending on the relation between the degree of the polynomial and the total number of parties, the reconstruction may either include errors, result in abort or it may guarantee the correct reconstruction of the secret. More precisely, let n be the number of shares P_r receives (counting its own), and let d be the degree used for the sharing $[s]$. Protocol Π_{PrivOpen} satisfies the following properties, of which the last two are a direct consequence of Section 3.2.

- If $0 < n - d \leq t$, then P_r outputs $s' = s + \delta$, where $\delta \in R$ is known by the adversary.
- If $t < n - d \leq 2t$, then either P_r aborts, or it outputs the correct s defined by the $\geq d+1$ shares sent by the honest parties.
- If $2t < n - d$, then P_r outputs the correct s defined by the $\geq d + 1$ shares sent by the honest parties.

Protocol $\Pi_{\text{PubOpen}}([s_0]_d, \ldots, [s_d]_d)$

Input: Sharings $[s_0]_d, \ldots, [s_d]_d$. A commutative exceptional set $A = \{\alpha_1, \ldots, \alpha_n\}$.
Output: All the parties learn $s_0, \ldots, s_d$.
Protocol: The parties proceed as follows

1. Let $f \in R[\mathtt{X}]$ be the polynomial given by $f(\mathtt{X}) = \sum_{j=0}^{d} s_j \mathtt{X}^j$. Parties locally compute $[f^{\mathsf{L}}(\alpha_i)]_d = \sum_{j=0}^{d} \alpha_i^j [s_j]_d$ for $i = 1, \ldots, n$.
2. For $i = 1, \ldots, n$, parties call $\mathsf{PrivOpen}([f^{\mathsf{L}}(\alpha_i)]_d, P_i)$.
3. If no abort was produced in the previous step, each P_i for $i = 1, \ldots, n$ sends the reconstructed $q_i = f^{\mathsf{L}}(\alpha_i)$ to each other party P_j for $j = 1, \ldots, n$.
4. After receiving the values $\{q_i\}_{i=1}^n$, where an unreceived value is interpreted as $q_i = 0$, each party P_j proceeds as follows:
   - If $t < n-d \leq 2t$: Interpolate the unique polynomial $g \in R[\mathtt{X}]_{\leq d}$ such that $g^{\mathsf{L}}(\alpha_i) = q_i$ for $i = 1, \ldots, d+1$. Check if $g^{\mathsf{L}}(\alpha_i) = q_i$ for $i = d+2, \ldots, d+t+1$. If this is the case, output $y_0, \ldots, y_d$, where $g(\mathtt{X}) = \sum_{i=0}^{d} y_i \mathtt{X}^i$. Else abort.
   - If $2t < n-d$: Apply error correction (Section 3.2) on the values $(q_1, \ldots, q_{d+2t+1})$ to recover a polynomial $g \in R[\mathtt{X}]_{\leq d}$ such that $g^{\mathsf{L}}(\alpha_i) = q_i$ for at least $d + t + 1$ points, and output $y_0, \ldots, y_d$, where $g(\mathtt{X}) = \sum_{i=0}^{d} y_i \mathtt{X}^i$.

Fig. 8. Reconstructing secret-shared values efficiently to all parties.

Now, in several scenarios it is the case that a secret-shared value $[s]_d$ must be reconstructed so that not only a single party P_r learns it, but rather it must be learned by all parties. A simple

approach for achieving this could consist of the parties calling $\Pi_{\text{PrivOpen}}([s]_d, P_r)$ for $r = 1, \ldots, n$,[10] but this would incur in a communication complexity of $\Omega(n^2)$.

To achieve a protocol with an *amortized* communication complexity of $O(n)$, we adapt the "king" idea from [DN07]. It turns out that this approach relies on error detection/correction properties of Reed-Solomon codes, which also apply in our (non-commutative) ring setting. Our protocol, Π_{PubOpen}, is described in Fig. 8. We remark that this construction is *exactly* the same as the one from [DN07], and our contribution lies in simply noting that it can also be used in the rings we work with. Protocol Π_{PubOpen} satisfies the following properties:

- If $t < n - d \leq 2t$, then either the parties abort, or they output the correct secrets $s_0, \ldots, s_{d+1}$ defined by the $\geq d + 1$ shares sent by the honest parties.
- If $2t < n - d$, then all parties output the correct secrets $s_0, \ldots, s_{d+1}$ defined by the $\geq d + 1$ shares sent by the honest parties.

To see this, observe that the protocol can be interpreted as the parties reconstructing the polynomial $f(\mathtt{X}) \in R[\mathtt{X}]_{\leq d}$ from the "shares" $(f(\alpha_1), \ldots, f(\alpha_n))$, which are sent to each party $P_1, \ldots, P_n$ on the first round of the protocol. As in the case of private reconstruction, if $t < n-d \leq 2t$ then the parties will be able to error-detect and either learn the correct $f(\mathtt{X})$ or abort, and if $t < n - d \leq 2t$ then the parties are guaranteed to learn the correct $f(\mathtt{X})$ due to the properties of error correction.

Regarding the communication involve, we observe that protocol Π_{PubOpen} first requires m parties to send shares to m parties, followed by m parties sending shares to n parties. Overall, this incurs in a communication complexity of $m(m + n)$ elements in R. Since d secrets are reconstructed per execution, we see that the amortized communication complexity per secret is $\frac{m}{d}(m + n)$. In our protocols we will have $m = \Theta(t)$ and $d = \Theta(t)$, so this amortized communication is $\Theta(n)$.

We remark that protocol Π_{PubOpen} does not deal with the case in which $0 < n - d \leq t$. This is because error detection/correction techniques do not work in this setting. Since this will be needed later on, we extend $\Pi_{\text{PubOpen}}([s])$ to this case by letting the parties call $\Pi_{\text{PrivOpen}}([s], P_1)$ and then, after P_1 gets s', it sends this value to all the other parties. Notice that no amortization is needed in this case.

C Generating Random, Linearly Correlated Values

Let $S = \mathsf{GR}(2^k, d)$. Assume that $3t + 1 < n$. The goal of this section is to prove the following proposition.

Proposition 7 (Proposition 5 restated). *Let M be an S-submodule of $S^{L \cdot n}$ representing a S-linear correlation among n parties. Then there exists a perfectly secure protocol for the parties to generate $t + 1$ such correlations with a communication complexity of $O(L \cdot n^2)$ elements in S.*

Such protocol is obtained by a simple adaptation/generalization of the protocols in [ACD+19, DLS20]. A similar approach was taken in [CCXY18] for the case of finite fields. Our protocol makes use of Hyper-Invertible Matrices (HIMs), which were introduced in [BTH08]. Their original description was limited to matrices over finite fields, but they naturally generalize to rings having

[10] Care would have to be taken for the case $0 < n - d \leq t$ so that the adversary does not cause different parties to reconstruct different values, which is fixed by sending shares through the broadcast channel. However, this is not the approach we will ultimately take so this is irrelevant.

big enough exceptional sets, as shown in [ACD+19, DLS20]. For our protocol we will use hyper-invertible matrices $H \in \mathcal{M}_{n\times n}(S)$, which are defined next.

Definition 10. *Let H be a r-by-c matrix. We say that H is* hyper-invertible *if, for all $A \subseteq [r]$, $B \subseteq [c]$ with $|A| = |B| > 0$, the sub-matrix H_A^B is invertible, where H_A denotes the matrix consisting of the rows $i \in A$ of H, H^B denotes the matrix consisting of the columns $j \in B$ of H, and $H_A^B = (H_A)^B$.*

For constructions of hyper-invertible matrices, we refer the reader to [BTH08] and [ACD+19]. The main reason why hyper-invertible matrices are a powerful tool in perfectly secure MPC is the following lemma, originally from [BTH08] and generalized to rings in [ACD+19]. Informally, it states that any combination of m inputs/outputs of the S-linear isomorphism induced by a square hyper-invertible matrix are uniquely determined by the remaining m inputs/outputs.

Lemma 8. *Let $H \in \mathcal{M}_{m\times m}(S)$ be a hyper-invertible matrix, and let $\boldsymbol{y} = H\boldsymbol{x}$. Then, for all $A, B \subseteq [m]$ with $|A| + |B| = m$, there exists an S-linear isomorphism $\phi : S^m \to S^m$ such that $\phi(\boldsymbol{x}_A, \boldsymbol{y}_B) = (\boldsymbol{x}_{\bar{A}}, \boldsymbol{y}_{\bar{B}})$, where $\bar{A} = [m] \setminus A$ and $\bar{B} = [m] \setminus B$.*

Our protocol for generating an arbitrary S-linear correlation $M \subseteq \mathbb{Z}_{2^k}^{L\cdot n}$ is stated in Fig. 9.

Protocol Π_{Corr}

Let $H \in \mathcal{M}_{n\times n}(S)$ be a hyper-invertible matrix.

1. Each party P_i for $i = 1, \ldots, n$ samples $(\boldsymbol{s}_{i,1}, \ldots, \boldsymbol{s}_{i,n}) \in_R M$ and sends $\boldsymbol{s}_{i,j} \in \mathbb{Z}_{2^k}^L$ to each party P_j.
2. Each P_j does the following. Let $S_j \in \mathcal{M}_{n\times L}(S)$ be the matrix whose rows are $\boldsymbol{s}_{1,j}, \ldots, \boldsymbol{s}_{n,j}$. P_j computes $R_j = H \cdot S_j$. Let $\boldsymbol{r}_{1,j}, \ldots, \boldsymbol{r}_{n,j}$ be the rows of R_j.
3. For every $i = 1, \ldots, 2t$, the parties send $\{\boldsymbol{r}_{i,j}\}_j$ to P_i, who checks that this vector belongs to M. If it does not, P_i broadcasts abort to the other parties.
4. If no party sent abort in the previous step, then the parties output the correlations $(\boldsymbol{r}_{i,j})_j$, for $i = 2t+1, \ldots, n$.

Fig. 9. Generating $\mathsf{GR}(2^k, d)$-linear correlations.

Theorem 10. *If there is no abort, then the protocol Π_{Corr} securely generates $n - 2t$ correct correlations $(\boldsymbol{r}_{i,1}, \ldots, \boldsymbol{r}_{i,n}) \in_R M$, for $i = 2t + 1, \ldots, n$.*

Proof. Let us assume for simplicity that the set of corrupted parties is $P_1, \ldots, P_t$, and assume there was no abort in the protocol execution. Let $\boldsymbol{s}_i = (\boldsymbol{s}_{i,1}, \ldots, \boldsymbol{s}_{i,n})$, and similarly for $\boldsymbol{r}_i$. From Lemma 8, there is a 1-1 S-linear correspondence between $\{\boldsymbol{r}_i\}_{i\in\{t+1,\ldots,2t\}} \cup \{\boldsymbol{s}_i\}_{i\notin\{1,\ldots,t\}}$ and $\{\boldsymbol{s}_i\}_{i\in\{t+1,\ldots,2t\}} \cup \{\boldsymbol{r}_i\}_{i\notin\{1,\ldots,t\}}$. Since step 3 did not result in abort, then the check performed by the honest parties $P_{t+1}, \ldots, P_{2t}$ passes, that is, $\boldsymbol{r}_i \in M$ for $i = t+1, \ldots, 2t$. Additionally, $\{\boldsymbol{s}_i\}_{i\notin\{1,\ldots,t\}} \subseteq M$ since these are distributed by honest parties. Hence, due to the correspondence, we see that $\{\boldsymbol{s}_i\}_{i\in\{1,\ldots,t\}} \cup \{\boldsymbol{r}_i\}_{i\notin\{1,\ldots,t\}} \subseteq M$, so in particular the output correlations $\boldsymbol{r}_i$ for $i \in \{2t + 1, \ldots, n\}$ lie in M.

In a similar way, using the properties of the hyper-invertible matrix H, it is shown that the output correlations look uniformly random in M to the adversary. ∎

D Perfectly Secure MPC from Strongly Multiplicative Secret Sharing

Given a strongly multiplicative, Shamir-style secret sharing scheme as the one guaranteed by Theorem 4, we can adapt the perfectly secure protocol by Beerliova and Hirt [BTH08] to non-commutative rings without many changes.

The most important matter is that of defining hyper-invertible matrices with entries in a non-commutative ring R.

Proposition 8. *Let R be a ring such that $Z(R)$ contains an exceptional set $A = \{\alpha_1, \ldots, \alpha_{2n}\}$. Let $f : R^n \to R^n$ be the function mapping $(x_1, \ldots, x_n)$ to $(y_1, \ldots, y_n)$ such that the points $(\alpha_{n+1}, y_1), \ldots, (\alpha_{2n}, y_n)$ lie on the polynomial $g \in R[\mathtt{X}]_{\leq n-1}$ uniquely defined[11] by the points $(\alpha_1, x_1), \ldots, (\alpha_n, x_n)$. Express f as the matrix $M \in \mathcal{M}_{m\times m}(R)$ with entries $m_{i,j} = \prod_{k=1,k\neq j}^{n}(\alpha_{n+i} - \alpha_k) \cdot (\alpha_j - \alpha_k)^{-1}$. The matrix M is hyper-invertible.*

Proof. This follows the same blueprint as [BTH08, Lemma 1]. The only additional step is observing that, since $Z(R)$ is a ring itself, each $m_{i,j} \in Z(R)$. ∎

Assuming an efficient error correction protocol for Reed-Solomon codes defined over the specific choice of R (see Section 3.2), the construction in [BTH08] carries over to our setting, as described by the following statement.

Corollary 4 (Corollary 2, restated). *Let R be a ring such that $Z(R)$ contains an exceptional set of size at least $2n$. Let $\mathcal{A}$ be an active adversary corrupting $t < n/3$ parties. There exists a perfectly secure, black-box MPC protocol with an amortized communication complexity of $O(n)$ ring elements per gate.*

[11] Note that, since $A \subset Z(R)$, we obtain the same polynomial whether we choose to interpolate according to "evaluation on the right" or "evaluation on the left". See Proposition 1.