More Efficient Dishonest Majority Secure Computation over $\mathbb{Z}_{2^k}$ via Galois Rings

Daniel Escudero[1], Chaoping Xing[2] and Chen Yuan[2]

[1] J.P. Morgan AI Research, New York, USA.
[2] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

Abstract. In this work we present a novel actively secure multiparty computation protocol in the dishonest majority setting, where the computation domain is a ring of the type $\mathbb{Z}_{2^k}$. Instead of considering an "extension ring" of the form $\mathbb{Z}_{2^{k+\kappa}}$ as in SPD$\mathbb{Z}_{2^k}$ (Cramer et al, CRYPTO 2018) and its derivatives, we make use of an actual ring extension, or more precisely, a Galois ring extension $\mathbb{Z}_{p^k}[\mathtt{X}]/(h(\mathtt{X}))$ of large enough degree, in order to ensure that the adversary cannot cheat except with negligible probability. These techniques have been used already in the context of honest majority MPC over $\mathbb{Z}_{p^k}$, and to the best of our knowledge, our work constitutes the first study of the benefits of these tools in the dishonest majority setting.
Making use of Galois ring extensions requires great care in order to avoid paying an extra overhead due to the use of larger rings. To address this, reverse multiplication-friendly embeddings (RMFEs) have been used in the honest majority setting (e.g. Cascudo et al, CRYPTO 2018), and more recently in the dishonest majority setting for computation over $\mathbb{Z}_2$ (Cascudo and Gundersen, TCC 2020). We make use of the recent RMFEs over $\mathbb{Z}_{p^k}$ from (Cramer et al, CRYPTO 2021), together with adaptations of some RMFE optimizations introduced in (Abspoel et al, ASIACRYPT 2021) in the honest majority setting, to achieve an efficient protocol that only requires in its online phase $12.4k(n-1)$ bits of amortized communication complexity and one round of communication for each multiplication gate. We also instantiate the necessary offline phase using Oblivious Linear Evaluation (OLE) by generalizing the approach based on Oblivious Transfer (OT) proposed in MASCOT (Keller et al, CCS 2016). To this end, and as an additional contribution of potential independent interest, we present a novel technique using Multiplication-Friendly Embeddings (MFEs) to achieve OLE over Galois ring extensions using black-box access to an OLE protocol over the base ring $\mathbb{Z}_{p^k}$ without paying a quadratic cost in terms of the extension degree. This generalizes the approach in MASCOT based on Correlated OT Extension. Finally, along the way we also identify a bug in a central proof in MASCOT, and we implicitly present a fix in our generalized proof.

1 Introduction

Secure multiparty computation is a set of tools and techniques that enables a group of parties, each having a private input, to jointly compute a given function

while only revealing its output. Since its introduction in the late 80s by Yao in [Yao86], several techniques for evaluating functionalities securely have been designed. These typically depend on the exact security setting, namely on how many parties are corrupted by an adversary, and whether they behave as an honest party (semi-honest/passive security) or if they operate in an arbitrary manner (active/malicious security).

One common aspect across all different constructions, however, is that they model the desired computation as an arithmetic circuit where gates are comprised of additions and multiplications over certain finite ring. Most attention has been devoted to the case in which the given arithmetic circuit is defined over a finite field, which is a natural choice due to its nice algebraic structure. However, there are other finite rings that are suitable for a wide range of highly relevant computations, which include, in particular, the ring $\mathbb{Z}_{p^k}$ of integers modulo p^k. For example, as shown in [DEF+19], computation over rings like $\mathbb{Z}_{2^k}$ with $k = 64$ or $k = 128$ may come with a series of performance benefits with respect to computation over prime fields of approximately the same size. Also, computation over arbitrary $\mathbb{Z}_{p^k}$ easily leads to efficient computation over arbitrary $\mathbb{Z}_m$ via the Chinese Remainder Theorem, leading to interesting results on the necessary assumptions to achieve efficient and "direct" MPC protocols.

Several such protocols have been proposed in the literature [CDE+18,OSV20,ACD+19,ACD+20]. In the honest majority setting, where the adversary corrupts at most a minority of the parties, Shamir secret-sharing is the most widely used building block to design MPC protocols. Unfortunately, such construction cannot be instantiated over $\mathbb{Z}_{p^k}$, but recent works have successfully made use of the so-called *Galois ring extensions* in order to enable Shamir secret-sharing over this ring which, together with some care, leads to MPC over $\mathbb{Z}_{p^k}$.

On the other hand, if the adversary is not assumed to corrupt a minority of the parties—a setting which is also referred to as dishonest majority—a different tool, in contrast to Shamir secret-sharing, is typically used. In this case, the main building block is additive secret-sharing, another form of secret-sharing that does not provide the redundancy features of Shamir secret-sharing, although it is considerably much simpler. To deal with active adversaries, extra redundancy comes in the form of message authentication codes, or MACs, which enable parties to determine if certain reconstructed secret is correct, or if it was tampered with.

Most constructions of dishonest majority MPC [DPSZ12,KOS16,KPR18] are designed to support arithmetic circuits defined over fields, mostly because of the limitations of their corresponding MACs, which are only secure as long as an adversary cannot design a polynomial of "low" degree with many roots. This is indeed the case if the given ring is a field, since a polynomial of degree d has at most $d - 1$ roots. However, when considering $\mathbb{Z}_{p^k}$ this does not longer hold, since there are polynomials of degree 1 such as $p^{k-1}\mathtt{X}$ that have a large amount of roots (p^{k-1} in this case).

To deal with this, a novel MAC that is compatible with arithmetic modulo 2^k was proposed in [CDE+18]. This construction has inspired several other works for MPC over $\mathbb{Z}_{2^k}$ in the dishonest majority setting [OSV20,RSS19,CKL21], and

even some in the honest majority setting [ACD+19,ACD+20]. However, these techniques comes at the expense of increasing the ring size by an additive factor of κ, the statistical security parameter. Although follow-up works that introduce somewhat homomorphic encryption (SHE) improves the performance of the prepocessing phase [OSV20,RSS19,CKL21], their online phases still follow the original protocol in [CDE+18]. The instantiation of the online phase incurs in a communication complexity of $4(k+\kappa)(n-1)$ for each multiplication gate as the parties have to open two values in $\mathbb{Z}_{2^{k+\kappa}}$.

From the discussion above, we see that it remains open to explore the benefits of using Galois ring extensions to achieve secure computation over $\mathbb{Z}_{p^k}$ in the dishonest majority setting.

1.1 Our Contribution

Computation over $\mathbb{Z}_{p^k}$. In this work, we design a highly efficient MPC protocol over $\mathbb{Z}_{p^k}$, for any prime p and integer $k \geq 1$,[3] that has a amortized communication complexity of $19.68k(n-1)$ for each multiplication gate in the online phase. Such communication complexity can be further reduced to $12.4k(n-1)$ if the security parameter is $\kappa = 64$. Furthermore, the offline phase of our protocol requires an amortized communication complexity of $5142.5kn(n-1)$ to prepare the shares for each multiplication gate in the online phase. We also note that we allow for a small k (possibly even $k = 1$), while the offline phase presented in SPD$\mathbb{Z}_{2^k}$ [CDE+18], which makes use of Oblivious Transfer (OT) as in [KOS16], requires k to be as large as the security parameter.

Computation over $\mathbb{Z}_2$. For the case $p = 2$ and $k = 1$, that is, when computation is over $\mathbb{Z}_2$, the best known protocol of [CG20] requires $10.2\ell(n-1)$ bits of communication in implementing ℓ instances of multiplication simultaneously on the online phase while our protocol requires $12.4\ell(n-1)$ bits of communication. However, their protocol needs 2 rounds of communication for each multiplication layer while our protocol only needs one round of communication. Furthermore, as the ratio of the best known RMFEs constructions improve, our construction can become more efficient. However, it is possible to bring down this cost to $8.2\ell(n-1)$ with a more tricky technique. We will briefly review such improvement in the Remark 1. Since the binary field is not the main focus of our protocol, we do not include this technique to optimize our online protocol.

Novel techniques for OLE over Galois ring extensions. As an additional contribution of potential independent interest, as part of the preprocessing phase of our protocol we present a novel method to enable Oblivious Linear Evaluation (OLE) over a Galois ring extension $R = \mathbb{Z}_{p^k}/(f(x))$ of degree d based on *any* OLE protocol over $\mathbb{Z}_{p^k}$, while only paying a factor of $O(d)$. This makes novel use of Multiplication Friendly Embeddings (MFEs) [PCCX09], which converts an

[3] Even though our title includes $\mathbb{Z}_{2^k}$, our results are presented for the more general $\mathbb{Z}_{p^k}$.

asymptotically good multiplicative secret sharing over extension field $\mathbb{F}_{2^d}$ into an asymptotically good multiplicative secret sharing scheme over binary field $\mathbb{F}_2$. This must be compared to the naive approach to achieve OLE over a Galois ring extension which would consist of representing each factor in terms of a $\mathbb{Z}_{p^k}$-basis, and then calling the underlying OLE over $\mathbb{Z}_{p^k}$ a total of $O(d^2)$ times to handle all the resulting cross products. We elaborate on this in Section 1.2.

Fixing bug in [KOS16] (and [CG20]). Our protocol shares certain similarities with the ones from [KOS16] (which is for binary extension *fields*) and [CG20] (which is for computation over $\mathbb{Z}_2$). In particular, the way we authenticate elements in the preprocessing phase, which requires OLE and makes use of MFEs as mentioned above, shares certain resemblance with the corresponding authentication methods in [KOS16,CG20] which make use of OT, or more specifically, Correlated OT Extension (COT), in order to avoid a quadratic blow-up in terms of the extension degree.

In [KOS16,CG20] the use of COT enables the adversary to introduce certain errors when authenticating inputs, which is dealt with in these works by performing certain checks in the input phase that ensure that, in spite of the adversary being able to inject these errors, the adversary is still "committed" to certain unique inputs that can be extracted by the simulator. The proof appears in [KOS16, Appendix B] ([CG20] does not present a self-contained proof, and instead simply points to the changes that should be done to the corresponding proof in [KOS16]), and it is highly non-trivial. Similarly to these works, our MFE-based preprocessing also enables the adversary to introduce certain errors, which, in spite of being of an entirely different nature to these in [KOS16,CG20], still share certain resemblance.

Due to the rough similarity between our preprocessing and the one from [KOS16,CG20], we are able to produce a proof of authentication along the lines of the one in [KOS16]. However, in the process of doing so, we identified a bug in the proof from [KOS16] (which affects [CG20] as well), which invalidates the last part of their argument where it is shown that the values extracted by the simulator are unique. We discuss this bug, together with its fix, in Section E in the Supplementary Material.

1.2 Overview of our Techniques

We present an overview of the main ideas behind the protocol introduced in this work, focusing on the high level ideas.

To get an idea of how our protocol works, consider the SPDZ-family of protocols over a field $\mathbb{F}_p$, which operates by additively secret-sharing each intermediate value $x \in \mathbb{F}_p$ as $[\![x]\!]$, together with shares of a global random key $[\![\alpha]\!]$, and shares of the Message Authentication Code (MAC) $[\![\alpha \cdot x]\!]$. Addition gates are handled locally, and multiplication gates make use of multiplication triples, which ultimately require opening some values. These openings are done without checking correctness, which is postponed to the final stage of the protocol where an aggregated check is performed.

The probability of the adversary cheating in the above protocol is $1/p$ so, if p is too small, we have to consider an extension field $\mathbb{F}_{p^d}$ so as to ensure that the adversary can not succeed with non-negligible probability. When it comes to the ring $\mathbb{Z}_{p^k}$, the failure probability of the adversary is still comparable to p^{-1} instead of p^{-k}. This is because there is only a $(1-\frac{1}{p})$-fraction of invertible elements in $\mathbb{Z}_{p^k}$. To decrease the failure probability, we consider the Galois ring $R = \mathbb{Z}_{p^k}/(f(x))$, which is a degree-$d$ extension of the ring $\mathbb{Z}_{p^k}$, where $f(x)$ is a degree-d irreducible polynomial over $\mathbb{Z}_{p^k}$. This means that, if we run the SPDZ protocol over the Galois ring R by treating the input of each parties in $\mathbb{Z}_{p^k}$ as an element in R, we can obtain a SPDZ protocol with security parameter p^{-d}. However, the communication complexity of this protocol is d times bigger than the original one. To mitigate the blow-up of communication complexity, we resort to RMFEs, which can implement multiple instances of computation by embedding multiple inputs in $\mathbb{Z}_{p^k}$ into a single element in R, while keeping the security parameter of each instance to be d.

This technique which was defined over field was introduced in [CCXY18] to amortize the communication complexity in the honest majority setting, where a minimum size on the underlying field is needed in order to enable Shamir secret-sharing. Block et al. [BMN18] independently proposed this technique to study two-party protocol over small fields. Then, this was used in [CG20] to amortize the communication complexity in the dishonest majority setting. This technique was restricted to the finite field $\mathbb{F}_2$ due to the fact that RMFEs were only known to exist over fields until very recently, when it was provided in [CRX21] a construction of RMFEs over an arbitrary ring $\mathbb{Z}_{p^k}$, which enables us to amortize the communication complexity over this ring. The previous works that employ this technique require an extra round for multiplication gate so as to re-encode the secret. We save this extra round protocol by introducing a quintuple for the multiplication gate. This technique was first presented in [ACE+21] for the honest majority setting. As the ring is a generalization of field, our protocol can also be carried out over the field. This allows us to compare our protocol with those over fields $\mathbb{F}_p$ with small p. We defer the comparison to Section 1.3.

As we have mentioned before, we also introduce, as a potential contribution of independent interest, a method to obtain OLE protocols over the Galois ring extension R of degree d, having only black-box access to an OLE functionality over the base ring $\mathbb{Z}_{p^k}$, with a communication complexity that is linear in d. This is achieved by making novel use of MFEs, which enable us to represent a product over R as roughly d-many products over $\mathbb{Z}_{p^k}$. This way, and by exploiting the $\mathbb{Z}_{p^k}$-linearity of the MFEs, we can obtain the desired OLE over R by evaluating these many smaller OLEs over the base ring $\mathbb{Z}_{p^k}$. We note that this generalizes the approach introduced in [KOS16], which uses COT in the setting of $p = 2$ and $k = 1$ in order to avoid a quadratic penalty in terms of the extension degree d.[4] Besides, our Galois ring is of size kd which is much bigger than other SPDZ

[4] Interestingly, our techniques do not constitute a strict generalization of the ones in [KOS16], since they are of a different nature. We leave it as future work to analyze

protocols. Their protocol requires either $k = 1$ or $d = 1$ which is suitable for the use of COT. However, we may face the situation that both k, d are comparable to the security parameter κ. The direct use of COT will cause the quadratic penalty in κ. This MFE technique can break the Galois Ring into a direct sum of small integer ring $\mathbb{Z}_{p^k}$ and allow us to do the oblivious product evaluation over each $\mathbb{Z}_{p^k}$ separately. This will save us at least the penalty of quadratic d even with the COT-based approach.

We also introduce the quintuples instead of Beaver triple to save one round of communication for each multiplication gate. Note that the previous works applying RMFE such as [CG20], [CCXY18] have to "re-encode" the secret. Basically speaking, all inputs $\boldsymbol{x}_i \in \mathbb{Z}_{p^k}^m$ are encoded as $\phi(\boldsymbol{x}_i)$ via the RMFE map. When two inputs $\phi(\boldsymbol{x}), \phi(\boldsymbol{y})$ enter the multiplication gate, the output should have the form $\phi(\boldsymbol{x} \star \boldsymbol{y})$ where $\star$ is the component-wise product. If we use the Beaver triple to securely compute the multiplication gate, we have to re-encode the secret to meet the desired form. In this work, we resort to the quintuple to save one round of communication which was first presented in [ACD+20]. One can find the details in the Theorem 3 and online protocol.

From amortized execution to single-circuit. Making use of Galois rings and RMFEs imposes the restriction that the computation must occur in batches, that is, multiplications and additions occurs on vectors rather than individual values. This is perfect for secure computation over SIMD circuits, which carry out the exact same computation to several inputs simultaneously, but in general there is a wide range of practical circuits that do not exhibit this structure. The general case can be easily addressed in the exact same way as in [CG20] by preprocessing certain permutation tuples, that serve as a way to re-route data throughout the circuit evaluation. We do not include this in our work, and refer the reader to [CG20, Section 4] for an explanation of how this works, which adapts seamlessly to our case with little effort.

1.3 Related Work

Dishonest majority MPC over $\mathbb{Z}_p$. The most standard case in the literature is when $k = 1$ and p is a large prime. In this case $\mathbb{Z}_p$ is a field, and there are multiple protocols designed to work in this setting, with the most notable being BeDOZa [BDOZ11], SPDZ [DPSZ12,DKL+13], MASCOT [KOS16], Overdrive [KPR18] and the more recent TopGear [BCS19]. For the case in which p is a large prime, our protocol does not need to make use of any Galois ring extension, and in fact, our online phase becomes exactly the one from [DKL+13] (which is the same as in [KOS16,KPR18,BCS19]).

In terms of the preprocessing all of the protocols above, except for MASCOT, are based on Somewhat Homomorphic Encryption (SHE), which was shown in [KPR18] to perform better than OT-based approaches like MASCOT. We leave

the potential benefits of our MFE-based techniques when $p = 2$ and $k = 1$ with respect to their COT-based approach.

it as an interesting open problem to explore the benefits of basing the offline phase of our protocol in SHE, instead of OLE as done in our work.

Finally, MASCOT makes use of OT to instantiate the necessary preprocesing over $\mathbb{Z}_p$ by interpreting elements in this field as integers and representing them in base 2. Instead, in our case, our preprocessing would be based directly on an OLE primitive over $\mathbb{Z}_p$, and the concrete efficiency would depend on the concrete instantiation for the OLE.

Dishonest majority MPC over $\mathbb{Z}_{2^k}$. In terms of computation over $\mathbb{Z}_{p^k}$ for a small p and $k > 1$, existing works focus on $p = 2$ and relatively large k.[5] The first such protocols was SPD$\mathbb{Z}_{2^k}$, which introduced a novel technique of performing MAC checks over a larger ring $\mathbb{Z}_{2^{k+\kappa}}$ to achieve authentication over $\mathbb{Z}_{2^k}$ with error probability $2^{-\kappa}$ For each multiplication gate, SPD$\mathbb{Z}_{2^k}$ requires $4(k+\kappa)(n-1)$ bits of communication since the shares are defined over $\mathbb{Z}_{2^{k+\kappa}}$.[6] Subsequent works that build on top of the same idea, most notably [CRFG20,OSV20], suffer from the same overhead. In contrast, our online phase requires $4km(n-1)$ bits of communication for simultaneously computing ℓ instances for each multiplication gate. The amortized communication complexity is $\frac{4km}{\ell}(n-1) = 19.68k(n-1)$. This complexity does not grow with the security parameter m. This means if the security parameter in [CDE+18] is 4 times bigger than k, our online protocol is more efficient. One can cut this communication complexity to $12.4k(n-1)$ if the security parameter is 64.

In terms of the offline phase, SPD$\mathbb{Z}_{2^k}$ extends the OT-based approach proposed in MASCOT [KOS16] to the $\mathbb{Z}_{2^{k+\kappa}}$ setting. However, due to the lack of invertibility in this ring, the protocol in [CDE+18] ends up adding quite some noticeable overhead so as to generate the Beaver triple. In their protocol, they claim that the parties communicate $2(k+2\kappa)(9\kappa+4k)n(n-1)$ bits to securely generate a Beaver triple for multiplication gate. In our protocol, the amortized complexity of generating the quintuple for a multiplication gate is $5142.5kn(n-1)$ for $\kappa = 64$. In Section 6, we show that our prepocessing phase is more efficient than theirs if $k \leq 29$ for $\kappa = 64$ and $k \leq 114$ for $\kappa = 128$. Moreover, our communication complexity does not grow with the security parameter as we can amortize it away by computing more instances simultaneously. This implies that our protocol should be more competitive for high security parameter range.

Finally, the approaches in [CRFG20,OSV20,RSS19] make use of homomorphic encryption (either Additively or Somewhat HE) in order to create the necessary correlations among the parties for SPD$\mathbb{Z}_{2^k}$'s online phase. It is not clear how these techniques can be used in our current context where the correlations are

[5] However, we remark that we are not aware of any limitation that would enable these works to be ported to the setting of $\mathbb{Z}_{p^k}$ for a more general prime p, and, furthermore, some of them already mention explicitly their ability to be generalized.

[6] An optimization in [CDE+18] seems to reduce this to $4k(n-1)$ since the online phase can be modified so that only elements of $\mathbb{Z}_{2^k}$ are transmitted, while full elements over $\mathbb{Z}_{2^{k+\kappa}}$ only appear in the final check phase. However, a bug in this approach leads to this cost still being present in the offline phase (personal communication).

over Galois rings, and as we have mentioned we leave it as an interesting future work to explore these potential relations.

To end, we remark that none of the protocols we have cited so far require multiple executions of the same circuit, unlike our case. As we have mentioned in Section 1.2, this can be easily overcome, as shown in [CG20], but nevertheless this adds a little overhead and an extra layer of complication.

Dishonest majority MPC over $\mathbb{Z}_2$. Finally, we consider the relevant case of $p = 2$ and $k = 1$, which corresponds to the case of computation over $\mathbb{Z}_2 = \{0, 1\}$. In this case, relevant protocols include [FPY18,LOS14,DZ13,CG20]. These works share, at a high level, the general idea of making use of an extension field of $\mathbb{Z}_2$ of large enough degree as to guarantee small cheating probability, which is a pattern that our work also employs. However, our work is more closely related to that of [CG20], which on top of using field extensions to lower failure probability, also makes use of RMFEs to reduce the overhead cause by such extensions. By doing this, as shown in [CG20], their work constitutes the state-of-the-art in terms of communication complexity in dishonest majority MPC over $\mathbb{Z}_2$.

Online phase. Our protocol is very competitive with respect to that of [CG20]. In terms of the online phase, the communication complexity of our protocol, although not better than that of [CG20], is only worse by a small multiplicative factor 0.2. However, this is the only downside of our protocol with respect to that of [CG20]. Improvements of our protocol, which stem mostly from the different type of encoding we make use of, include the following:

- Our protocol is considerably simpler as it has less necessary "pieces". As a concrete example, the reader can compare the $\mathcal{F}_{\mathsf{MPC}}$ functionality we make use of in this work with respect to the corresponding functionality defined in [CG20]: here we only need to store vectors over $\mathbb{Z}_2$ (over $\mathbb{Z}_{p^k}$ in general), while the functionality in [CG20] needs to keep two dictionaries, one to store vectors over $\mathbb{Z}_2$ and another to store elements over certain field extension of $\mathbb{Z}_2$. In addition, among several other simplifications, our protocol does not make use of the input encoding mechanisms needed in [CG20], nor it requires re-encoding secret-shared values after each multiplication.
- Our online protocol, in spite of involving only the communication complexity overhead of a small factor 1.2 with respect to that of [CG20], requires half the amount of rounds than the protocol in [CG20]. This stems from the fact that, as mentioned above, we do not require the extra round needed in [CG20] to re-encode secret-shared values. Our input phase is also more efficient as we do not need to check that secret-shared values lie in certain subspace.

Offline phase. Now, when comparing the offline phase of our protocol with respect to that of [CG20], we have to set $p^k = 2$. If we omits the cost of calls of OLE, our protocol is more efficient. However, we admit that it is not a fair comparison. We also want to emphasize that this OLE approach does save the communication cost for large k. If we replace the OLE with COT used by previous works like [CG20],

the communication cost is quite close as we follow almost the same approach to generate the triples. The deviation is that our shares and MAC shares are defined over R while they divide them into two cases.

1.4 Organization of the Paper

This work is organized as follows. In Section 2 we present the necessary preliminaries, and then in Section 3 we present the online phase of our protocol. In Section 4 we present our protocol for authenticating secrets, which in particular includes our novel approach to OLE over Galois ring extensions based on OLE over $\mathbb{Z}_{p^k}$ using MFEs, and also the updated proof that shows that, in spite of the adversary being able to introduce errors in this protocol, there will be a unique set of extractable inputs the adversary is committed to. In Section 5 we present the full-fledged offline phase of our protocol, which includes the generation of the modified triples we use in our work. Finally, in Section 6 we analyze concretely the communication complexity of our resulting protocol.

2 Preliminaries

2.1 Basic Notation

We use bold letters (e.g. $\boldsymbol{x}$, $\boldsymbol{y}$) to denote vectors, and we use the star operator ($\star$) to denote component-wise product of vectors. Also, in some cases we use the notation $\boldsymbol{x}[i]$ to denote the i-entry of the vector $\boldsymbol{x}$. Finally, we use $[N]$ to denote the set of integers $\{1, \ldots, N\}$. We denote by n the number of parties, and the set of parties is $\{P_1, \ldots, P_n\}$.

In this work we make use of the authenticated and homomorphic secret-sharing construction from [DKL$^+$13], where a value $x \in R$ is secret-shared as $\langle x \rangle = (\llbracket x \rrbracket, \llbracket x \cdot \alpha \rrbracket, \llbracket \alpha \rrbracket)$, where $\alpha \xleftarrow{\$} R$ is a global uniformly random key. More precisely, this sharing contains three parts, $\llbracket x \rrbracket = (x^{(1)}, \ldots, x^{(n)})$, $\llbracket x \cdot \alpha \rrbracket = (m^{(1)}, \ldots, m^{(n)})$ and $\llbracket \alpha \rrbracket = (\alpha^{(1)}, \ldots, \alpha^{(n)})$, where party P_i holds the random share $x^{(i)}$ of the secret x, the MAC share $m^{(i)}$ and the key share $\alpha^{(i)}$. These satisfy $\sum_{i=1}^{n} m^{(i)} = (\sum_{i=1}^{n} x^{(i)})(\sum_{i=1}^{n} \alpha^{(i)})$.

2.2 Algebraic Preliminaries

Galois rings. We denote by $\mathsf{GR}(p^k, d)$ the Galois ring over $\mathbb{Z}_{p^k}$ of degree d, which is a ring extension $\mathbb{Z}_{p^k}/(f(\mathtt{X}))$ of $\mathbb{Z}_{p^k}$, where $f(\mathtt{X}) \in \mathbb{Z}_{p^k}[\mathtt{X}]$ is a monic polynomial of degree d over $\mathbb{Z}_{p^k}$ whose reduction modulo p is irreducible over $\mathbb{Z}_p$. For details on Galois rings we refer the reader to the text [Wan03], and also to Section F in the Supplementary Material.

Multiplication-friendly embeddings. We begin by considering the crucial notions of Multiplication-Friendly Embeddings (MFEs) and Reverse Multiplication-Friendly Embeddings (RMFEs), which act as an interface between Galois ring extensions and vectors over $\mathbb{Z}_{p^k}$, making the products defined in each of these structures (component-wise products for the vectors) somewhat "compatible". The asymptotically good multiplicative secret sharing schemes over field were already known in [CC06,CCdHP08,PCCX09,PCX11]. However, the similar results for asymptotically good multiplicative secret sharing scheme over ring were not known until very recently [CRX21]. Basically speaking, they manage to show that the asymptotically good multiplicative secret sharing scheme over ring $\mathbb{Z}_{p^k}$ can achieve the same performance as the one over field $\mathbb{F}_p$. There results provide a machinery for explicitly constructing multiplication friendly embedding and reverse multiplication friendly embedding over ring. We start with MFEs below.

Definition 1. *Let $m, t \in \mathbb{N}$. A pair of $\mathbb{Z}_{p^k}$-module homomorphisms $\rho : \mathbb{Z}_{p^k}^t \to \mathsf{GR}(p^k, m)$ and $\mu : \mathsf{GR}(p^k, m) \to \mathbb{Z}_{p^k}^t$ is a* multiplication-friendly embedding*, or MFE for short, if, for all $x, y \in \mathsf{GR}(p^k, m)$ it holds that $xy = \rho(\mu(x) \star \mu(y))$.*

It is easy to see from the definition that ρ must be surjective and μ must be injective. Indeed, given $x \in \mathsf{GR}(p^k, m)$, we have that $x = x \cdot 1 = \rho(\mu(x) \star \mu(1))$, and if $\mu(x) = \mathbf{0}$ then $x = \rho(\mu(x) \star \mu(1)) = \rho(\mathbf{0} \star \mu(1)) = \rho(\mathbf{0}) = 0$. In particular, $t \geq m$.[7]

For the convenience of comparison with other works, we only list the results about $p = 2$ in the following theorem. If $p > 2$, the ratio will be smaller. This also holds for RMFEs.

Theorem 1 ([CRX21]). *There exists an explicit MFE family*

$$(\rho_m : \mathbb{Z}_{2^k}^{t(m)} \to \mathsf{GR}(2^k, m),\ \mu_m : \mathsf{GR}(2^k, m) \to \mathbb{Z}_{2^k}^{t(m)})_{m \in \mathbb{N}}$$

with $t(m)/m \to 5.12$ as $m \to \infty$.

Reverse multiplication-friendly embeddings. Now we define the notion of an RMFE.

Definition 2. *Let $m, \ell \in \mathbb{N}$. A pair of $\mathbb{Z}_{p^k}$-module homomorphisms (ϕ, ψ) with $\phi : \mathbb{Z}_{p^k}^{\ell} \to \mathsf{GR}(p^k, m)$ and $\psi : \mathsf{GR}(p^k, m) \to \mathbb{Z}_{p^k}^{\ell}$ is a* reverse multiplication-friendly embedding*, or RMFE for short, if, for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_{p^k}^{\ell}$, it holds that $\boldsymbol{x} \star \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$.*

Note that, if (ϕ, ψ) is an RMFE, then necessarily ϕ is injective and ψ is surjective, so in particular $\ell \leq m$. The following theorem shows the existence of RMFEs over $\mathbb{Z}_{2^k}$. The existence of RMFEs over $\mathbb{Z}_{p^k}$ can be found in [CRX21].

[7] In fact, one can reasonably easy prove that $t \geq 2m$.

Theorem 2 ([CRX21]). *There exists an explicit RMFE family*

$$(\phi_m : \mathbb{Z}_{2^k}^{\ell(m)} \to \mathsf{GR}(2^k, m),\ \psi_m : \mathsf{GR}(2^k, m) \to \mathbb{Z}_{2^k}^{\ell(m)})_{m\in\mathbb{N}}$$

with $m/\ell(m) \to 4.92$ *as* $m \to \infty$. *For small value, we can optimize this ratio by choosing* $(m, \ell(m)) = (65, 21)$ *or* $(m, \ell(m)) = (135, 42)$.

Without loss of generality we can assume that $\phi(\mathbf{1}) = 1$. This implies that, given $\boldsymbol{x} \in \mathbb{Z}_{p^k}^{\ell}$, a preimage of $\boldsymbol{x}$ under ψ is $x = \phi(\boldsymbol{x})$. Indeed, this holds since

$$\psi(x) = \psi(\phi(\boldsymbol{x})) = \psi(\phi(\boldsymbol{x}) \cdot 1) = \psi(\phi(\boldsymbol{x}) \cdot \phi(\mathbf{1})) = \boldsymbol{x} \star \mathbf{1} = \boldsymbol{x}.$$

2.3 Security Model

We prove the security of our protocol under the Universal Composability (UC) framework by Canetti [Can01]. We let n be the number of parties, among which at most $n-1$ can be actively corrupted. We let $\mathcal{C}, \mathcal{H} \subseteq [n]$ denote the index set of corrupted and honest parties, respectively. The adversary is static and malicious, which means that the corruption may only happen before the start of the protocols, and corrupted parties may behave arbitrarily. We say a protocol Π securely implement a functionality $\mathcal{F}$ with statistical security parameter κ,[8] if there is a simulator that interacts with the adversary (or more formally, the *environment*) so that he can distinguish the ideal/simulated world and the real worlds with probability at most $O(2^{-\kappa})$.

The composability of the UC framework enables us to build our protocol in a modular fashion by defining small protocols together with the functionalities they are intended to implement, and proving the security of each of these pieces separately. Finally, we asusme for simplicity and without loss of generality that the outputs to be computed are intended to be learned by all parties, i.e. there are no private outputs.

About security with abort. The adversary may first learn the output and then abort based on this. This is unavoidable, since it is impossible to achieve fairness in the dishonest majority setting. Thus, in the ideal world, we must allow the same thing to happen. Our functionalities enable the adversary/simulator to provide an abort signal, that causes all honest parties to abort. In the real world, whenever we say that some party aborts, we assume this party sends an abort signal through the broadcast channel so that all honest parties abort.

2.4 Communication Model

We assume private and authenticated channels between every pair of parties, as well as a broadcast channel. In addition, we assume the existence of what we

[8] We consider only statistical security since, even though dishonest majority MPC is known to be generally impossibly to achieve without computational assumptions, we rely in this work on an OLE functionality, and do not provide any instantiation of it. This allows us to design protocols in the statistical setting.

call a *simultaneous message channel*, which allows the parties, each P_i holding a value x_i, to send these to all the other parties while guaranteeing that the corrupt parties cannot modify their values based on the messages from other parties. This is ultimately used to enable reconstruction of an additively shared secret while disallowing a rushing adversary from modifying the secret at will, restricting him to additive errors only. Such channel is implemented in practice by following the standard "commit-and-open" approach in which the parties first broadcast to each other a commitment of their messages, and then, only once this is done, they open via broadcast the commitments to the values they wanted to send in a first place.[9]

More formally, we model communication as an ideal functionality $\mathcal{F}_{\mathsf{Channels}}$. This is presented in detail as Functionality 5 in Section B in the Supplementary Material. Note that all of our protocols make use of $\mathcal{F}_{\mathsf{Channels}}$, but we do not write this explicitly in their descriptions or in their associated theorems.

3 Online Phase

We set $m = \lceil \kappa \log_p(2) \rceil$ so $p^m \geq 2^\kappa$, where κ is the statistical security parameter, and denote $\phi := \phi_m : \mathbb{Z}_{p^k}^{\ell(m)} \to R$ and $\psi := \psi_m : R \to \mathbb{Z}_{p^k}^{\ell(m)}$, the mappings whose existence is guaranteed by Theorem 2. Now that m is fixed, we write ℓ instead of $\ell(m)$. We also let $R = \mathsf{GR}(p^k, m)$ and $\overline{R} = \mathsf{GF}(p^m) = \{\overline{r} : r \in R\}$. Finally, we consider the $\mathbb{Z}_{p^k}$-linear map $\tau : R \to R$ given by $\tau = \phi \circ \psi$.

We begin by describing how the online phase of our protocol works. In more detail, we describe a protocol Π_{Online} that securely implements the MPC functionality $\mathcal{F}_{\mathsf{MPC}}$, that models general purpose secure computation over vectors $\mathbb{Z}_{p^k}^{\ell}$, in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model, where, as we will see, $\mathcal{F}_{\mathsf{Prep}}$ is a functionality that provides certain correlated randomness. Then, in following Sections we discuss how to instantiate $\mathcal{F}_{\mathsf{Prep}}$.

3.1 Required Functionalities

First, we present some of the essential functionalities we will need in this section. We follow a similar approach to that in [CG20]. We let $\mathcal{F}_{\mathsf{MPC}}$ be a functionality that enables the parties to input secret *vectors* in $\mathbb{Z}_{p^k}^{\ell}$, perform arbitrary SIMD affine combinations and multiplications on these, and open results. This is ultimately the functionality that we wish to instantiate. To achieve this, we consider a restricted functionality $\mathcal{F}_{\mathsf{Prep}}$ that acts like $\mathcal{F}_{\mathsf{MPC}}$, except it does not allow for multiplications. Instead, it can store certain correlated vectors upon request by the parties, which can be used to instantiate multiplications. Finally, we define $\mathcal{F}_{\mathsf{Auth}}$, which is a more restricted version of both $\mathcal{F}_{\mathsf{MPC}}$ and $\mathcal{F}_{\mathsf{Prep}}$ that acts exactly as these two except that, with respect to $\mathcal{F}_{\mathsf{MPC}}$, it does not allow for multiplication, and with respect to $\mathcal{F}_{\mathsf{Prep}}$ it does not generate correlated randomness.

[9] This is modeled in other works with a functionality (typically denoted by $\mathcal{F}_{\mathsf{Comm}}$), but we decided to incorporate this as part of the communication channel for simplicity.

We remark that these functionalities are essentially the same as the ones presented in [CG20]. However, we note that in our case they can be fully defined over one single ring (either $\mathbb{Z}_{p^k}^{\ell}$ or R), while in [CG20], due to the type of encoding they use, both rings appear simultaneously in these functionalities.

Authentication functionality $\mathcal{F}_{\mathsf{Auth}}$. This functionality is the basic building block that allows parties to store secrets and perform affine combination on these. Intuitively, it corresponds to Homomorphic Authenticated Secret-Sharing. We remark that we will not make direct use of $\mathcal{F}_{\mathsf{Auth}}$ in this Section, but we include it since it is instructive to define this functionality first and describe both $\mathcal{F}_{\mathsf{MPC}}$ and $\mathcal{F}_{\mathsf{Prep}}$ in terms of $\mathcal{F}_{\mathsf{Auth}}$ later on. $\mathcal{F}_{\mathsf{Auth}}$is defined as Functionality 1 below.

Functionality 1: $\mathcal{F}_{\mathsf{Auth}}$

The functionality maintains a dictionary Val, which it uses to keep track of authenticated elements of R. We use $\langle x \rangle$ to denote the situation in which the functionality stores $\mathsf{Val}[\mathtt{id}] = x$ for some identifier $\mathtt{id}$.

- **Input:** On input $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), (x_1, x_2, \dots, x_L), P_i)$ from P_i and $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), P_i)$ from all other parties, set $\mathsf{Val}[\mathtt{id}_j] = x_j$ for $j = 1, 2, \dots, L$.
- **Affine combination:** On input $(\mathsf{AffComb}, \mathtt{id}, (\mathtt{id}_1, \dots, \mathtt{id}_L), (a_1, \dots, a_L), a)$ from all parties, the functionality computes $z = a + \sum_{j=1}^{L} a_j \cdot \mathsf{Val}[\mathtt{id}_j]$ and stores $\mathsf{Val}[\mathtt{id}] = z$. We denote this by $\langle z \rangle \leftarrow a + \sum_{j=1}^{L} a_j \langle x_j \rangle$, where $x_j = \mathsf{Val}[\mathtt{id}_j]$.
- **Partial openings:** On input $(\mathsf{Open}, \mathtt{id})$ from all parties, if $\mathsf{Val}[\mathtt{id}] \neq \perp$, send $x = \mathsf{Val}[\mathtt{id}]$ to the adversary and wait for an x' back. Then send x' to the honest parties.
- **Check openings:** On input $(\mathsf{Check}, (\mathtt{id}_1, \mathtt{id}_2, \dots, \mathtt{id}_L), (x_1, x_2, \dots, x_L))$ from every party wait for an input from the adversary. If he inputs OK, and if $\mathsf{Val}[\mathtt{id}_j] = x_j$ for $j = 1, 2, \dots, L$, return OK to all parties. Otherwise abort.

Preprocessing functionality $\mathcal{F}_{\mathsf{Prep}}$. This functionality extends $\mathcal{F}_{\mathsf{Auth}}$ by letting the parties obtain shares $(\langle a \rangle, \langle b \rangle, \langle \tau(a) \rangle, \langle \tau(b) \rangle, \langle \tau(a)\tau(b) \rangle)$, where $a, b \xleftarrow{\$} R$ are uniformly random and unknown to any party. It also allows the parties to obtain $\langle r \rangle$ where $r \xleftarrow{\$} \psi^{-1}(\mathbf{0})$. $\mathcal{F}_{\mathsf{Prep}}$ is defined as Functionality 2 below.

Functionality 2: $\mathcal{F}_{\mathsf{Prep}}$

This functionality behaves exactly like $\mathcal{F}_{\mathsf{Auth}}$, but in addition it supports the following commands:

- **Correlated randomness**: On input $(\mathsf{CorrRand}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}_3, \mathtt{id}_4, \mathtt{id}_5)$ from all parties, sample $a, b \in R$ uniformly at random and store $\mathsf{Val}[\mathtt{id}_1] = a$, $\mathsf{Val}[\mathtt{id}_2] = b$, $\mathsf{Val}[\mathtt{id}_3] = \tau(a)$, $\mathsf{Val}[\mathtt{id}_4] = \tau(b)$ and $\mathsf{Val}[\mathtt{id}_5] = \tau(a) \cdot \tau(b)$.

– **Input**: On Input $(\mathsf{InputPrep}, P_i, \mathtt{id})$ from all parties, samples $\mathsf{Val}[\mathtt{id}] \xleftarrow{\$} R$ and output it to P_i.
– **Kernel element**: On input $(\mathsf{Ker}, \mathtt{id})$ from all parties, sample $r \in R$ uniformly at random subject to $\psi(r) = \mathbf{0}$, and store $\mathsf{Val}[\mathtt{id}] = r$.

Parallel MPC functionality $\mathcal{F}_{\mathsf{MPC}}$. Finally, we describe the functionality that we aim at implementing in this section. It takes $\mathcal{F}_{\mathsf{Auth}}$ as a starting point, and implement the following changes/additions:

– It replaces R by $\mathbb{Z}_{p^k}^{\ell}$, so, instead of storing elements of R, it stores vectors over $\mathbb{Z}_{p^k}$ of dimension ℓ.
– Affine combinations now take coefficients over $\mathbb{Z}_{p^k}$.
– It implements a multiplication command that, on input $(\mathsf{Mult}, \mathtt{id}, (\mathtt{id}_1, \mathtt{id}_2))$ from all parties, computes $\boldsymbol{z} = \mathsf{Val}[\mathtt{id}_1] \star \mathsf{Val}[\mathtt{id}_2]$ and stores $\mathsf{Val}[\mathtt{id}] = \boldsymbol{z}$.

$\mathcal{F}_{\mathsf{MPC}}$ is defined in full detail as Functionality 6 in Section B in the Supplementary Material.

3.2 Instantiating $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-Hybrid Model

The protocol Π_{Online}, described as Protocol 1 later in the section, instantiates $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model, with statistical security parameter κ. Intuitively, the protocol consists of the parties storing vectors $\boldsymbol{x} \in \mathbb{Z}_{p^k}^{\ell}$ by storing with $\mathcal{F}_{\mathsf{Prep}}$ an element $x \in R$ with $\psi(x) = \boldsymbol{x}$. The corresponding commands in $\mathcal{F}_{\mathsf{Prep}}$ are used to instantiate Input, AffComb, Open and Check, and the correlated randomness is used to handle the Mult command.

We remark that the Input command can be instantiated more efficiently instead of relying on the corresponding command from $\mathcal{F}_{\mathsf{Prep}}$, by using the standard approach of letting each party P_i broadcast its input masked with a random value of which the parties have (preprocessed) shares. A crucial observation is that we can allow this since *any* possible input in R is a valid input, while in other works like [CG20], a special "subspace check" is needed to ensure that this input lies in a special subset of valid inputs.

Protocol 1: Π_{Online}

– **Input:** The parties, upon receiving input $(\mathsf{Input}, \mathtt{id}, P_i)$, and P_i receiving input $(\mathsf{Input}, \mathtt{id}, \boldsymbol{x}, P_i)$, execute the following:
  1. Call $\mathcal{F}_{\mathsf{Prep}}$with the command InputPrep. P_i takes the mask value $(r, \langle r \rangle)$.
  2. P_i broadcasts $\phi(\boldsymbol{x}) - r$ and each party locally computes $\phi(\boldsymbol{x}) - r + \langle r \rangle$.

  As a result, the parties obtain $\langle x \rangle$, where $x = \phi(\boldsymbol{x}) \in \psi^{-1}(\boldsymbol{x})$.
– **Affine combination:** Upon receiving $(\mathsf{AffComb}, \mathtt{id}, (\mathtt{id}_1, \ldots, \mathtt{id}_L), (a_1, \ldots, a_L), \boldsymbol{a})$, the parties send $(\mathsf{AffComb}, \mathtt{id}, (\mathtt{id}_1, \ldots, \mathtt{id}_L), (a_1, \ldots, a_L), \phi(\boldsymbol{a}))$ to $\mathcal{F}_{\mathsf{Prep}}$.

– **Multiplication:** Upon receiving input $(\mathsf{Mult}, \mathtt{id}, (\mathtt{id}_1, \mathtt{id}_2))$, the parties proceed as described below. Let $\langle x \rangle$ and $\langle y \rangle$ be the values stored by $\mathcal{F}_{\mathsf{Prep}}$ in $\mathtt{id}_1$ and $\mathtt{id}_2$ respectively. Below, when not written explicitly, the identifiers needed for the given commands are assumed to be fresh and unique, and are only used ephemerally for the purpose of handling the multiplication command.
  1. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command $\mathsf{CorrRand}$ to obtain $(\langle a \rangle, \langle b \rangle, \langle \tau(a) \rangle, \langle \tau(b) \rangle, \langle \tau(a)\tau(b) \rangle)$.
  2. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command $\mathsf{AffComb}$ to obtain $\langle d \rangle \leftarrow \langle x \rangle - \langle a \rangle$ and $\langle e \rangle \leftarrow \langle y \rangle - \langle b \rangle$.
  3. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command Open to obtain $d \leftarrow \langle d \rangle$ and $e \leftarrow \langle e \rangle$.
  4. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command $\mathsf{AffComb}$ to obtain $\langle z \rangle \leftarrow \tau(d) \langle \tau(b) \rangle + \tau(e) \langle \tau(a) \rangle + \langle \tau(a)\tau(b) \rangle + \tau(d)\tau(e)$, indicating $\mathcal{F}_{\mathsf{Prep}}$ to store this value at the identifier $\mathtt{id}$.
– **Partial openings:** Upon receiving input $(\mathsf{Open}, \mathtt{id})$, the parties execute the following. Let $\langle x \rangle$ be the value stored by $\mathcal{F}_{\mathsf{Prep}}$ at $\mathtt{id}$.
  1. Call $\mathcal{F}_{\mathsf{Prep}}$to with the command Ker to get $\langle r \rangle$ with $r \in \psi^{-1}(\mathbf{0})$.
  2. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command $\mathsf{AffComb}$ to get $\langle z \rangle \leftarrow \langle x \rangle + \langle r \rangle$, storing this value at $\mathtt{id}$ (hence overloading $\langle x \rangle$ with $\langle z \rangle$).
  3. Call $\mathcal{F}_{\mathsf{Prep}}$ with the command Open so that the honest parties in $\mathbb{Z}_{p^k}$ learn z', for a value $z' \in R$ provided by the adversary to $\mathcal{F}_{\mathsf{Prep}}$. The parties store internally the pair $(\mathtt{id}, z')$.
  4. The parties output $\boldsymbol{z}' = \psi(z')$.
– **Check openings:** Upon receiving input $(\mathsf{Check}, (\mathtt{id}_1, \mathtt{id}_2, \ldots, \mathtt{id}_L), (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L))$, the parties fetch the internally stored pairs $(\mathtt{id}_j, x'_j)$ for $j = 1, \ldots, L$ and call $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{Check}, (\mathtt{id}_1, \mathtt{id}_2, \ldots, \mathtt{id}_L), (x'_1, x'_2, \ldots, x'_L))$. If $\mathcal{F}_{\mathsf{Prep}}$ aborts, then the parties abort.

Theorem 3. *Protocol Π_{Online} implements $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model*

Proof (Sketch). A full-fledged simulation-based proof is presented in Section C in the Supplementary Material. Here we restrict ourselves to the core idea of the proof. First, notice that for every input $\boldsymbol{x} \in \mathbb{Z}_{p^k}^{\ell}$, the value stored by $\mathcal{F}_{\mathsf{Prep}}$ is $x := \phi(\boldsymbol{x}) - r + r = \phi(\boldsymbol{x}) \in R$, which satisfies $\psi(x) = \boldsymbol{x}$. We see then that, for the input phase, the values stored by $\mathcal{F}_{\mathsf{Prep}}$ are a preimage under ψ of the corresponding vectors that Π_{Online} stores. We claim that this invariant is preserved through the interaction with the $\mathsf{AffComb}$ and Mult commands.

The case of $\mathsf{AffComb}$ is easy since ψ is $\mathbb{Z}_{p^k}$-linear. To analyze Mult, consider two stored values $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_{p^k}^{\ell}$ in Π_{Online}, and assume that the invariant holds, so the underlying stored values $x, y \in R$ in $\mathcal{F}_{\mathsf{Prep}}$ satisfy $\psi(x) = \boldsymbol{x}$ and $\psi(y) = \boldsymbol{y}$. After the command Mult is issued to Π_{Online}, the parties get a tuple $(\langle a \rangle, \langle b \rangle, \langle \tau(a) \rangle, \langle \tau(b) \rangle, \langle \tau(a)\tau(b) \rangle)$, then open $d = x - a$ and $e = y - b$, and compute locally $\langle z \rangle \leftarrow \tau(d) \langle \tau(b) \rangle + \tau(e) \langle \tau(a) \rangle + \langle \tau(a)\tau(b) \rangle + \tau(d)\tau(e)$. We can verify that this is equal to $z = \tau(x)\tau(y)$, which preserves the invariant since $\psi(z) = \psi(\phi(\psi(x)) \cdot \phi(\psi(y))) = \psi(x) \star \psi(y) = \boldsymbol{x} \star \boldsymbol{y}$, using the definition of τ

together with Def. 2. The above assumes that d and e are opened correctly, but this can be assumed to be the case since, if this does not hold, this will be detected in when the Check command is issued, and the adversary does not learn sensitive information before then since the values a and b perfectly mask the stored values x and y.

Finally, for the Open command, we see that, due to the invariant, the value returned by $\mathcal{F}_{\mathsf{Prep}}$ is indeed a preimage under ψ of the value stored by Π_{Online}. However, one small technicality is that this preimage may contain "noise" from previous operations, or more precisely, which preimage is this may depend on previous data which is not intended to be revealed. This is fixed by adding a random element $r \in \psi^{-1}(\mathbf{0})$ before opening, which preserves the invariant, but guarantees that the preimage is uniformly random among all possible preimages. In formal terms, in the actual proof, this enables the simulator to simulate this value by simply sampling a uniformly random preimage of the output obtained from the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Once again, we refer the reader to Section C in the Supplementary Material for a more detailed and self-contained simulation-based proof. □

Remark 1. One idea to bring these costs down at the expense of making the final MAC check more costly. When the parties partially open d and e, they open instead $\tau(d)$ and $\tau(e)$, by applying locally τ to each of their additive shares. The image of τ has the size of $|S^\ell|$. Now, to check these openings, the parties take linear combinations with coefficients over S, not over R, open the respective elements over R, and check that they map to the correct elements after applying τ. To get good soundness we need to repeat this several times, which makes the final check more expensive. Depending on the size of the circuit, this tradeoff, specifically, if the circuit is large enough, this approach will pay up.

4 Authentication

In this section we aim at instantiating $\mathcal{F}_{\mathsf{Auth}}$. This makes use of the authenticated secret-sharing scheme briefly introduced in Section 1.2. However, for enabling the parties to create authenticated values, that is, for instantiating the Input command in $\mathcal{F}_{\mathsf{Auth}}$, we need to rely on certain functionalities that, ultimately, are used to enable two-party secure multiplication. This building block will be also used to produce the necessary preprocessing material in Section 5. We begin by introducing the required functionalities below. However, first we introduce some notation. Recall that $m = \lceil \kappa \log_p(2) \rceil$ and $R = \mathsf{GR}(p^k, m)$. Let $\rho := \rho_m : \mathbb{Z}_{p^k}^{t(m)} \to R$ and $\mu := \mu_m : R \to \mathbb{Z}_{p^k}^{t(m)}$ be the mappings from Theorem 1. We write t instead of $t(m)$.

4.1 Required Functionalities

Oblivious linear evaluation $\mathcal{F}_{\mathsf{OLE}}$. We first define the core primitive that our work relies on, which is Oblivious Linear Evaluation, or OLE for short. This

can be seen a generalization of Oblivious Transfer (OT) from arithmetic modulo 2, to arithmetic modulo p^k. Traditionally, OLE is regarded as a functionality between two parties P_A and P_B that receives $(a, b) \in \mathbb{Z}_{p^k}^2$ from P_A and $x \in \mathbb{Z}_{p^k}$ from P_B, and returns $a \cdot x + b$ to P_B (which is the evaluation of P_B's input in P_A's linear function $a \cdot \mathtt{X} + b$, hence the name), while P_A learns nothing about x. However, we can consider multiple variants of OLE depending on whether the inputs and/or outputs are uniformly random, and also if some of the inputs is intended to be fixed accross multiple executions. In the variant we consider in our work, P_A only inputs $a \in \mathbb{Z}_{p^k}$, while P_B still provides $x \in \mathbb{Z}_{p^k}$ and obtains $y = a \cdot x + b$. The difference is that $b \in \mathbb{Z}_{p^k}$ is sampled uniformly at random by the functionality, and P_A receives $-b$. This way, the parties have received uniformly random additive shares of $a \cdot x$: indeed, $y + (-b) = a \cdot x$. The functionality $\mathcal{F}_{\mathsf{OLE}}$ is described in detail below as Functionality 3. We remark that, even though OLE can be regarded as a generalization of OT, the functionality we are defining here is not a strict generalization of the functionality $\mathcal{F}_{\mathsf{ROT}}$ in [KOS16,CG20]. In their case, where $p^k = 2$, the authors can use a short OT to generate certain correlated *keys*, which then can be extended via *OT extension* (with the help of a PRF) to an arbitrary amount of OTs, which suits very well the COPE application. Such approach, unfortunately, does not work for general p^k, although it does so partially for $p^k = 2^k$. As a matter of fact, we could have actually based our Π_{COPEe} protocol on a more elaborate version of OLE that is more aimed towards its use in COPE.

Functionality 3: $\mathcal{F}_{\mathsf{OLE}}$

Upon receiving $(\mathsf{OLE}, a, P_A, P_B)$ from P_A and $(\mathsf{OLE}, x, P_A, P_B)$ from P_B where $x, a \in \mathbb{Z}_{p^k}$, the functionality samples $b \xleftarrow{\$} \mathbb{Z}_{p^k}$, sets $y = ax + b$, and sends y to P_B and $-b$ to P_A.

Public coins $\mathcal{F}_{\mathsf{Coin}}$. This functionality samples a uniformly random element in R and sends this to the parties. The detailed functionality is presented in Section B in the Supplementary Material.

Remark 2. This functionality is called $\mathcal{F}_{\mathsf{Rand}}$ in [CG20,KOS16], but we believe that $\mathcal{F}_{\mathsf{Coin}}$ is a more appropriate name since $\mathcal{F}_{\mathsf{Rand}}$ is usually reserved to denote random *secret-shared* values.

4.2 Correlated Oblivious Product Evaluation

As in [CG20] and [KOS16], the general idea to instantiate the Input command is to ask each party P_i to first sample their share $\alpha^{(i)}$ of the key α. Then, when P_j wants to input a value x, each party P_i interacts with P_j so that they obtain additive sharings $u^{(i,j)}$ and $v^{(j,i)}$ (held by P_i and P_j respectively) of $\alpha^{(i)} \cdot x$, i.e. $u^{(i,j)} + v^{(j,i)} = \alpha^{(i)} \cdot x$. Once this is done, each party P_i for $i \neq j$ can

define $m^{(i)} = u^{(i,j)}$, while P_j sets $\alpha^{(j)}x + \sum_{i \neq j} v^{(j,i)}$. This way, it holds that $\sum_{i=1}^{n} m^{(i)} = \alpha \cdot x$.[10]

We refer to the required two-party interaction above by *correlated oblivious product evaluation*, or COPE for short. Our main idea consists of instantiating a COPE between P_i and P_j by letting the parties run t OLE instances, where P_i inputs $\mu(\alpha^i)[l]$ and P_j inputs $\mu(x)[l]$ for $l \in [t]$ (recall that $\boldsymbol{z}[l]$ denotes the l-th coordinate of the vector $\boldsymbol{z}$). This way, P_i and P_j get shares $\boldsymbol{u}^{(i,j)}$ and $\boldsymbol{v}^{(j,i)}$ such that $\boldsymbol{u}^{(i,j)} + \boldsymbol{v}^{(j,i)} = \mu(\alpha^{(i)}) \star \mu(x)$, and, by using Def. 1, they can locally apply ρ to each share to obtain $u^{(i,j)} + v^{(j,i)} = \rho(\mu(\alpha^{(i)}) \star \mu(x)) = \alpha^{(i)} \cdot x$.

In [CG20] and [KOS16], COPE is instantiated by running several instances of the underlying OT functionality. However, in their setting, the input of one of the parties across all these instances must be constant, and an actively corrupt party may not follow this. As a result, they consider an extended version of COPE (called COPEe in these works) that models the effect that an actively corrupt party can have in the output after cheating by sending different inputs across different instances, and then, in [CG20,KOS16], a check in the input phase of the authentication protocol checks that such cheating did not happen (or that, if it happened, then it did so in a "controlled" way).

In our setting we encounter a similar situation. Consider the COPE instantiation sketched in previous paragraphs. We see that P_j is free to provide as input any vector $\boldsymbol{x}^{(j,i)} \in \mathbb{Z}_{p^k}^t$ to the t OLE calls, but the protocol actually requires this vector to be $\boldsymbol{x}^{(j,i)} = \mu(x)$, that is, it must lie in the image of μ.[11] Recall from Section 2.2 that μ is injective but not surjective, so there is no way in which we can enforce this. Hence, in the case of an actively corrupt P_j, the parties would not obtain additive shares of $\alpha^{(i)} \cdot x$, but instead $\rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}^{(j,i)})$ for some vector $\boldsymbol{x}^{(j,i)}$ provided as input by P_j. Ultimately, this leads to the parties obtaining MAC shares $m^{(i)}$ where $\sum_{i=1}^{n} m^{(i)} = \alpha^{(j)}x + \sum_{i \neq j} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}^{(j,i)})$.

Similarly to [CG20,KOS16], we do not attempt at removing the possibility of this attack at this stage, and instead model it as permissible behavior in the functionality $\mathcal{F}_{\mathsf{COPEe}}$ below.[12] Then, in Π_{Auth} we introduce a check that handles these inconsistencies.

Functionality 4: $\mathcal{F}_{\mathsf{COPEe}}$

This functionality runs with two parties P_i and P_j and the adversary $\mathcal{A}$. The **Initialize** phase is run once first. The **Multiply** phase can be run an arbitrary number of times.

[10] Notice that, since P_j knows x, the parties already hold trivial additive shares of x, namely all parties set their share to 0, and P_j sets it to x. However, in the actual protocol, P_j must also distribute actual random shares of x, since otherwise leakage may occur, for example, when adding and reconstructing shared values inputted by different parties.

[11] Similarly, P_i's input must lie in the image of μ, but as we will see this deviation is not that harmful.

[12] Even though this functionality is named the same as its counterpart in [CG20,KOS16], we remark that the errors the adversary can introduce in our setting are different.

– **Initialize:** On input $\alpha^{(i)} \in R$ from P_i, store this value.
– **Multiply:** On input $x \in R$ from P_j:
  - If P_j is corrupt then receive $v^{(j,i)} \in R$ and a vector $\boldsymbol{x}^{(j,i)} \in \mathbb{Z}_{p^k}^t$ from $\mathcal{A}$ and compute $u^{(i,j)} = \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}^{(j,i)}) - v^{(j,i)}$.
  - If P_i is corrupt then receive $\boldsymbol{\alpha}^{(i,j)} \in \mathbb{Z}_{p^k}^t$ and $u^{(i,j)}$ from $\mathcal{A}$ and compute $v^{(j,i)} = \rho(\boldsymbol{\alpha}^{(i,j)} \star \mu(x)) - u^{(i,j)}$
  - If both P_i and P_j are honest then sample $u^{(i,j)}$ and $v^{(j,i)}$ uniformly at random subject to $u^{(i,j)} + v^{(j,i)} = \alpha^{(i)} \cdot x$.

  The functionality sends $u^{(i,j)}$ to P_i and $v^{(j,i)}$ to P_j.

The functionality $\mathcal{F}_{\mathsf{COPEe}}$ can be instantiated as we have sketched above with the help of $\mathcal{F}_{\mathsf{OLE}}$. The corresponding protocol Π_{COPEe} is described in full detail as Protocol 8 in Section B in the Supplementary Material. We state the following theorem whose proof can be found in Section C in the Supplementary Material.

Theorem 4. *Protocol Π_{COPEe} implements $\mathcal{F}_{\mathsf{COPEe}}$ in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model.*

4.3 Authenticated Secret-Sharing

Local operations. The scheme $\langle\cdot\rangle$ is $\mathbb{Z}_{p^k}$-module homomorphic, so additions, subtractions, and in general $\mathbb{Z}_{p^k}$-affine combinations of $\langle\cdot\rangle$-shared values can be computed locally by the parties. These operations are standard and can be found for instance in [DKL+13]. However, for completeness, these are described in detail in the *procedure*[13] π_{Aff} given as Procedure 5 in Section B in the Supplementary Material. This operation is denoted by $\langle y\rangle \leftarrow a + \sum_{h=1}^{L} a_h \langle x_h\rangle$.

Opening and checking. To partially reconstruct a shared value $\langle y\rangle = (\llbracket y\rrbracket, \llbracket \alpha \cdot y\rrbracket, \llbracket\alpha\rrbracket)$, the parties can all send their share of $\llbracket y\rrbracket$ to P_1, who can reconstruct and send the (possibly modified) result y' to the parties. To check the correctness of this opening, the parties locally compute $\llbracket \alpha \cdot y\rrbracket - y' \llbracket\alpha\rrbracket$, and send the shares of this value to each other using the simultaneous message channel. The parties abort if the reconstructed value is not 0. These operations are represented by the procedures $\pi_{\mathsf{Open}}(\langle y\rangle)$ and $\pi_{\mathsf{Check}}(\langle y\rangle, y')$, which are described in detail in Section B in the Supplementary Material as Procedures 6 and 7.

4.4 Authentication Protocol

We describe our protocol Π_{Auth} implementing $\mathcal{F}_{\mathsf{Auth}}$ as Protocol 2 below. At a high level, the protocol is very similar to the corresponding one proposed in

[13] In this work we distinguish between *procedures* (denoted by smallcase π) and *protocols* (denoted by capital Π). Protocols are associated to ideal functionalities and have simulation-based proofs, whereas procedures, even though they also specify steps the parties must follow, are used as helpers within actual protocols and do not have functionalities or simulation-based proofs associated to them. This can be thought of being somewhat analogous to the difference between macros and actual fuctions in programming languages such as `C`/`C++`.

[KOS16,CG20]: allow the parties to obtain authenticated shares of their inputs by using $\mathcal{F}_{\mathsf{COPEe}}$ followed by a check, process affine combinations locally using the homomorphic properties of the secret-sharing scheme, partially open shared values by using π_{Open} and check their correctness by using π_{Check}.

Protocol 2: Π_{Auth}

The parties collectively maintain a dictionary Val of shared values.

- **Initialize:** First the parties perform an initialization step that consists of each party P_i calling the **Initialize** step in $\mathcal{F}_{\mathsf{COPEe}}$.
- **Input:** Once P_j receives $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), (x_1, x_2, \dots, x_L), P_j)$ and the other parties receive $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), P_j)$, the parties execute the following.
  1. P_j samples $x_0 \xleftarrow{\$} R$.
  2. For $h = 0, \dots, L$ and $i \in [n]$, P_j samples $x_h^{(i)} \in R$ uniformly at random subject to $\sum_{i=1}^{n} x_h^{(i)} = x_h$, and sends $\{x_h^{(i)}\}_{h=0}^{L}$ to each P_i.
  3. For every $i \in [n] \setminus \{j\}$, P_i and P_j execute the **Multiply** step of $\mathcal{F}_{\mathsf{COPEe}}$ $L+1$ times, where P_j inputs $x_0, \dots, x_L$. For $h = 0, \dots, L$, P_i receives $u_h^{(i,j)}$ and P_j receives $v_h^{(j,i)}$.
  4. For $i \in [n] \setminus \{j\}$, P_i defines $m^{(i)}(x_h) = u_h^{(i,j)}$ while P_j sets $m^{(j)}(x_h) = \alpha^{(j)} \cdot x_h + \sum_{i \neq j} v_h^{(j,i)}$. By setting $x_h^{(i)} = 0$ for $i \neq j$, the parties have defined $\langle x_h \rangle$ for $h = 0, \dots, L$.
  5. Parties call $\mathcal{F}_{\mathsf{Coin}}$ to get $r_1 \dots, r_L \xleftarrow{\$} R$. Set $r_0 := 1$.
  6. Parties compute locally $\langle y \rangle \leftarrow \sum_{h=0}^{L} r_h \langle x_h \rangle$ and call $y' \leftarrow \pi_{\mathsf{Open}}(\langle y \rangle)$ followed by $\pi_{\mathsf{Check}}(\langle y \rangle, y')$.
  7. If the previous call did not result in abort, the parties store $\mathsf{Val}[\mathtt{id}_h] = \langle x_h \rangle$ for $h \in [L]$.

- **Affine combination:** If all parties receive input $(\mathsf{AffComb}, \mathtt{id}, (\mathtt{id}_1, \dots, \mathtt{id}_L), (a_1, \dots, a_L), a)$, they fetch $\langle x_h \rangle = \mathsf{Val}[\mathtt{id}_h]$ for $h \in [L]$, compute locally $\langle z \rangle \leftarrow a + \sum_{j=1}^{L} a_j \langle x_j \rangle$, and let $\mathsf{Val}[\mathtt{id}] = \langle z \rangle$.

- **Partial openings:** Once all parties receive input $(\mathsf{Open}, \mathtt{id})$, they recover $\langle x \rangle = \mathsf{Val}[\mathtt{id}]$ and call $x' \leftarrow \pi_{\mathsf{Open}}(\langle x \rangle)$.
- **Check openings:** If all the parties receive $(\mathsf{Check}, (\mathtt{id}_1, \mathtt{id}_2, \dots, \mathtt{id}_L), (x'_1, x'_2, \dots, x'_L))$, they set $\langle x_h \rangle = \mathsf{Val}[\mathtt{id}_h]$ for $h \in [L]$ and execute the following:
  1. Call $\mathcal{F}_{\mathsf{Coin}}$ to get $r_1, \dots, r_L \xleftarrow{\$} R$.
  2. Compute locally $\langle y \rangle \leftarrow \sum_{h=1}^{L} r_h \langle x_h \rangle$ and $y' = \sum_{h=1}^{L} r_h \cdot x'_h$, and call $\pi_{\mathsf{Check}}(\langle y \rangle, y')$

Theorem 5. *Protocol Π_{Auth} implements $\mathcal{F}_{\mathsf{Auth}}$ in the ($\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{COPEe}}$)-hybrid model.*

Proof. Part of the proof is quite standard: the simulator extracts the inputs from the corrupt parties and sends this to the ideal functionality $\mathcal{F}_{\mathsf{Auth}}$, and it also emulates the functionalities $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{COPEe}}$, and emulates virtual honest parties that behave as the real honest parties, except they do not know the real inputs. At the end of the execution the simulator adjusts[14] the honest parties' shares to be compatible with the output and the corrupt parties' shares.

The most complex and non-standard step is the extraction of the corrupt parties' inputs. It turns out that the check performed in the input phase may actually pass with non-negligible probability, even if the adversary cheats in the $\mathcal{F}_{\mathsf{COPEe}}$ calls. Our goal is to show that, in spite of this, the corrupt parties are still committed to a unique set of inputs—that is, these are the only values the adversary could open these sharings to in a posterior output phase—and these inputs are extractable by the simulator. We point out that a full-fledged proof is given in Section D in the Supplementary Material. Here we only provide the intuition of how the inputs from the corrupt parties are extracted, and why they are unique. Furthermore, we assume that $k = 1$, so $R = \mathbb{F}_{p^m}$ and $\mathbb{Z}_{p^k} = \mathbb{F}_p$. The general case of a Galois ring with zero divisors in handled in Section D in the Supplementary Material, and requires a few extra technical considerations.

Let P_j be a corrupt party who is intended to provide some inputs $\{x_h\}_{h=0}^{L}$. For each P_i, in the $L+1$ calls to $\mathcal{F}_{\mathsf{COPEe}}$, P_j may use as input a vector $\boldsymbol{x}_h^{(j,i)}$, which leads to P_i and P_j obtaining $u_h^{(i,j)}$ and $v_h^{(j,i)}$ respectively, where these values add up to $\rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)})$. If P_j acts honestly, it would hold that $\boldsymbol{x}_h^{(j,i)} = \mu(x_h)$, but in the general case this may not hold. Then, the parties compute sharings $\langle y \rangle$, open this to a possibly incorrect value y, and finally each party P_i reveals $\sigma^{(i)}$.

The check passes if and only if $\sum_{i=1}^{n} \sigma^{(i)} = 0$. By computing an explicit expression of the $\sigma^{(i)}$ held by honest parties, we can verify that this check passes if and only if the key shares $\{\alpha^{(i)}\}_{i \in \mathcal{H}}$ held by honest parties satisfy

$$
\begin{aligned}
\sum_{i \in \mathcal{C}} \sigma^{(i)} &= \sum_{i \in \mathcal{H}} \alpha^{(i)} \cdot y - m^{(i)}(y) \\
&= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} r_h \cdot m^{(i)}(x_h) \right) \\
&= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} r_h \cdot u_h^{(i,j)} \right) \\
&= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} \left(r_h \cdot \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)}) - r_h \cdot v_h^{(j,i)} \right) \right).
\end{aligned}
$$

We assume, for the sake of easing the notation, that there is only one honest party $\mathcal{H} = \{P_l\}$, and we denote $\alpha := \alpha^{(l)}$ and $\boldsymbol{x}_h := \boldsymbol{x}_h^{(j,l)}$. This way, we can

[14] As we discuss in Section D in the Supplementary Material there is a subtlety with this "adjustment" that originates from the fact that R has zero-divisors.

write the above equation as

$$\underbrace{\sum_{i\in\mathcal{C}} \sigma^{(i)} - \sum_{h=0}^{L} r_h v_h^{(j,l)}}_{=:z} = \alpha \cdot y - \sum_{h=0}^{L} r_h \cdot \rho(\mu(\alpha) \star \boldsymbol{x}_h). \tag{1}$$

Notice that z above is a value provided by the adversary, as well as y and the vectors $\boldsymbol{x}_h$. Furthermore, the coefficients r_h are public, so the only unknown (from the adversary point of view) is the key α. Unfortunately, it can be the case that this equation holds with non-neglible probability even though P_j did not provide valid inputs. However, as we will see, in the event in which this equation holds, when the parties at a later point want to open the sharings produced by the input phase, there will only be one possible set of values the adversary can open these sharings to successfully.

Let us denote by $K_z \subseteq \mathbb{F}_{p^m}$ the set of all α satisfying Eq. (1). Some important observations about K_z:

- K_0 is an $\mathbb{F}_p$-vector space.
- Either $K_z = \emptyset$, or $K_z = K_0 + \xi$, for any $\xi \in K_z$.
- In particular, $|K_z| = 0$ or $|K_z| = |K_0|$.

Intuitively, $\mathcal{A}$ wants to set the values under its control so that $|K_z|$, which is either 0 or $|K_0|$, is as large as possible, since this way $\mathcal{A}$ increases the chances of $\alpha \in K_z$, and hence the more likely the input check passes. However, $\mathcal{A}$ does not fully control the values defining K_z: $\boldsymbol{x}_h$ must be chosen *before* $\{r_h\}_{h\in[L]}$, which are sampled at random.

Naturally, $\mathcal{A}$'s optimal strategy is to choose $z = 0$ (which is in fact what z equals to under honest behavior), since in this case there are no chances that $K_z = \emptyset$. To prove this intuition, we simply notice that the mapping $\alpha \mapsto \alpha \cdot y - \sum_{h=0}^{L} r_h \cdot \rho(\mu(\alpha) \star \boldsymbol{x}_h)$ is $\mathbb{F}_p$-linear, and $K_z \neq \emptyset$ if and only if z is in the range of this function, which we denote by Z. If $z \neq 0$, the probability that z is in this range is $|Z|/|R|$, and the conditional probability that $\alpha \in K_z$ is $|K_z|/|R| = |K_0|/|R|$. Furthermore, from the rank-nullity theorem we have that $|Z| \cdot |K_0| = |R|$ so the overall success probability, if $z \neq 0$, becomes $\frac{|Z|}{|R|} \cdot \frac{|K_0|}{|R|} = 1/|R|$, which is negligible. We see then that we can assume that $z = 0$.[15]

Suppose now that the Eq. (1) holds, that is, the input check passes. Imagine that at a later point in the actual circuit computation, a value $\langle x_h \rangle$ is intended to be opened (here x_h does not stand for a specific value yet—since the adversary did not input any concrete value—but instead it acts as an identifier for the sharings corresponding to the h-th input of P_j). The adversary can cause the partial opening to be a value x_h, and in the final check each party announces $\sigma_h^{(i)}$. As before, the check passes if and only if $\sum_{i=1}^{n} \sigma_h^{(i)} = 0$. We can compute what the MAC shares of $\langle x_h \rangle$ held by the honest parties are based on the input phase,

[15] We point out that in [KOS16], this subtlety that enables us to consider $z = 0$ is not mentioned explicitly.

and conclude, via a similar derivation to the one made above, that the check passes if and only if $z_h = \alpha \cdot x_h - \rho(\mu(\alpha) \star \boldsymbol{x}_h)$, where z_h is a value chosen by the adversary. As before, we can show that $z_h = 0$ for each h. Therefore, x_h must be equal to $(\rho(\mu(\alpha) \star \boldsymbol{x}_h))/\alpha$. It seems then that we have been able to "extract" the inputs that the corrupt party P_j has committed to, but these $\{x_h\}_h$ are defined in terms of the key α, which is unknown to the simulator. In what follows, we show that, with overwhelming probability, the same set of $\{x_h\}_h$ can be obtained from *any* possible key $\alpha \in K_0$, which enables extraction.

Putting together what we have seen above, for the adversary to pass all the checks it must be the case that the key $\alpha \in \mathbb{F}_{p^m}$ lies in K_0, that is, it must satisfy $0 = \sum_{h=0}^{L} r_h \left(\alpha \cdot x_h - \rho(\mu(\alpha) \star \boldsymbol{x}_h)\right)$, and furthermore, the adversary can only open the input shares at a later stage to the values to $x_h = \rho(\mu(\alpha) \star \boldsymbol{x}_h)/\alpha$ for $h = 0, \ldots, L$.

Now, our goal is to show that, no matter what $\alpha \in K_0$ the honest party happens to choose, the values $\{x_h\}_h$ are fixed, which amounts to showing that the functions $f_{\boldsymbol{x}_h} : \mathbb{F}_{p^m} \setminus \{0\} \to \mathbb{F}_{p^m}$ given by $f_{\boldsymbol{x}_h}(\alpha) = \rho(\mu(\alpha) \star \boldsymbol{x}_h)/\alpha$ are constant. This will enable the simulator to *extract* x_h by using any element in K_0.

To show that each $f_{\boldsymbol{x}_h}$ is constant, we consider $e_1, \ldots, e_d \in \mathbb{F}_{p^m}$ a basis of K_0 over $\mathbb{F}_p$, and make the simple but powerful observation that, if $\{f_{\boldsymbol{x}_h}(e_i)\}_h$ is proven to be constant as i ranges over $[d]$, then this will extend to any $\alpha \in K_0$. This is because, for any $(c_h) \in \mathbb{F}_p^{L+1}$, the set of $\alpha \in K_0 \setminus \{0\}$ such that $f_{\boldsymbol{x}_h}(\alpha) = c_h$ forms a subspace of K_0 (after adding 0), so if $\{f_{\boldsymbol{x}_h}(e_i)\}_h$ is equal to a constant $(c_h)_h$ as i ranges over $[d]$, then this subspace must include the $\mathbb{F}_p$-span of $\{e_1, \ldots, e_d\}$, which is equal to K_0 itself.

More precisely, we claim that the probability that there exist $i_0, i_1 \in [d]$ with $\{f_{\boldsymbol{x}_h}(e_{i_0})\}_h \neq \{f_{\boldsymbol{x}_h}(e_{i_1})\}_h$ is negligible. Indeed, since each $e_i \in K_0$, we have that $0 = e_i \cdot y - \sum_{h=0}^{L} r_h \cdot \rho(\mu(e_i) \star \boldsymbol{x}_h)$, so

$$y = \sum_{h=0}^{L} r_h \cdot \left(\frac{\rho(\mu(e_i) \star \boldsymbol{x}_h)}{e_i}\right) = \sum_{h=0}^{L} r_h \cdot f_{\boldsymbol{x}_h}(e_i).$$

In particular, it holds that $0 = \sum_{h=0}^{L} r_h \cdot (f_{\boldsymbol{x}_h}(e_{i_1}) - f_{\boldsymbol{x}_h}(e_{i_0}))$, or, in other words, $(r_0, \ldots, r_L)$ is orthogonal to the vector $(f_{\boldsymbol{x}_h}(e_{i_1}) - f_{\boldsymbol{x}_h}(e_{i_0}))_{h=0}^{L}$, which is non-zero if $\{f_{\boldsymbol{x}_h}(e_{i_0})\}_h \neq \{f_{\boldsymbol{x}_h}(e_{i_1})\}_h$. However, the latter vector is chosen by the adversary *before* the former one is sampled uniformly at random, so the probability of this happening is at most $1/p^m$.

From the above we see that the probability that the ordered pair (i_0, i_1) results in $\{f_{\boldsymbol{x}_h}(e_{i_0})\}_h \neq \{f_{\boldsymbol{x}_h}(e_{i_1})\}_h$ is at most $1/p^m$, so in particular, since there are at most $d^2 \leq m^2$ such pairs, the probability that there exists at least one such pair is upper bounded by $m^2/p^m = 2^{-(m \log_2(p) - 2\log_2(m))}$. Now, it is easy to see that if $\alpha \in K_0$ is an $\mathbb{F}_p$-multiple of e_i then $f_{\boldsymbol{x}_h}(\alpha) = f_{\boldsymbol{x}_h}(e_i)$, so the result we have just obtained enables us to conclude that, except with probability $2^{-(m \log_2(p) - 2\log_2(m))}$, the values $\{f_{\boldsymbol{x}_h}(\alpha)\}_h$ are constant for $\alpha \in K_0$.

Putting the pieces together, we see that if the adversary passes the input check, meaning that the key α chosen by the honest party lies in K_0, then there

is only one possible set of values that the adversary can later open these sharings to, namely, the values $x_h = f_{\boldsymbol{x}_h}(\alpha')$ given by *any* $\alpha' \in K_0$. Since K_0 is known by the simulator, these inputs can be efficiently extracted. □

5 Offline Phase

Recall that our the preprocessing phase requires to generate the quintuples $(\langle a\rangle, \langle b\rangle, \langle \tau(a)\rangle, \langle \tau(b)\rangle, \langle \tau(a)\tau(b)\rangle)$ with random elements $a, b \in R$. During the online phase, such quintuple will save one round of communication for each multiplication gate with respect to approaches like the one followed in [CG20]. To generate this quintuple, we begin by first generating authenticated pairs $(\langle a\rangle, \langle \tau(a)\rangle)$, a step that we carry out in a similar way as the "re-encode pair protocol" in [CG20]. We use a similar technique to process the random values $\langle r\rangle$ such that $r \in \ker \psi$. Finally, to obtain the products $\langle \tau(a)\tau(b)\rangle$, we make use of our $\mathcal{F}_{\mathsf{COPEe}}$ functionality, together with a cut-and-choose-based check that verifies no errors are introduced by the adversary.

We discuss these protocols in detail below.

Encoding pairs. The idea to obtain certain amount T of pairs $(\langle a\rangle, \langle \tau(a)\rangle)$ is to generate $T + s$ many pairs $(\langle a_i\rangle, \langle \tau(a_i)\rangle)$ for $i = 1, \ldots, T + s$, and then sacrifice the last s of them by taking a random $\mathbb{Z}_{p^k}$-linear combinations of the first T pairs to obtain s equations for correctness check. If there is at least one of the first T pairs that is corrupted, with probability at most p^{-s}, the corrupted pair will pass all the check, which we can make negligible in κ by choosing s to be large enough. This is presented in detail in the Procedure 3 below.

Procedure 3: π_{Enc}

The procedure generates a series of T pairs $(\langle a_i\rangle, \langle \tau(a_i)\rangle)$ where $a_i \in R$ is a random element. We assume the functionalities $\mathcal{F}_{\mathsf{COPEe}}$ and $\mathcal{F}_{\mathsf{Auth}}$.

- **Construct:**
  1. P_i samples $a_j^{(i)} \in R$ uniformly at random for $j = 1, \ldots, s + T$.
  2. P_i calls $\mathcal{F}_{\mathsf{Auth}}$ to obtain $\left\langle a_j^{(i)} \right\rangle$ and $\left\langle \tau(a_j^{(i)}) \right\rangle$.
  3. All parties computes

$$\langle a_j \rangle = \sum_{i=1}^{n} \left\langle a_j^{(i)} \right\rangle, \quad \langle \tau(a_j) \rangle = \sum_{i=1}^{n} \left\langle \tau(a_j^{(i)}) \right\rangle.$$

- **Sacrifice:**
  1. All parties call $\mathcal{F}_{\mathsf{Coin}}$ to generate s random vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,T}) \in \mathbb{Z}_{p^k}^T$.
  2. Compute $\langle b_i\rangle = \sum_{j=1}^{T} x_{i,j} \langle a_j\rangle + \langle a_{T+i}\rangle$ and $\langle c_i\rangle = \sum_{j=1}^{T} x_{i,j} \langle \tau(a_j)\rangle + \langle \tau(a_{T+i})\rangle$ and partially open b_i and c_i.
  3. If $\tau(b_i) \neq c_i$ for some $i \in \{1, \ldots, s\}$, then abort.
  4. Call $\mathcal{F}_{\mathsf{Auth}}$with the command Check on the opened values b_i and c_i.
- **Output:** Output $(\langle a_i\rangle, \langle \tau(a_i)\rangle)$ for $i = 1, \ldots, T$.

Kernel elements. In order to generate sharings $\langle r \rangle$, where r is uniformly random in $\ker \psi$, we proceed in a similar way as the procedure π_{Enc}, except that instead of each party P_i calling $\mathcal{F}_{\mathsf{Auth}}$ to input a pair $(a^{(i)}, \tau(a^{(i)}))$, each party inputs a value $r^{(i)} \xleftarrow{\$} \ker \psi$. Then, the same correctness check as in π_{Enc} works in this case since $\ker \psi$ is $\mathbb{Z}_{p^k}$-linear. The resulting procedure, π_{Ker}, is presented in detail in Section B in the Supplementary Material.

Multiplication. As for generating $\langle \tau(a)\tau(b) \rangle$, the technique employed in [KOS16] can not be applied to our quintuples. The reason is that to prevent the leakage of single bit of the input $\boldsymbol{a} = (a_1, \ldots, a_\gamma) \in R^\gamma$, the combine in the multiplication triple protocol in [KOS16] takes the random linear combination $a' = \langle \boldsymbol{a}, \boldsymbol{r} \rangle$ for a random vector $\boldsymbol{r} = (r_1, \ldots, r_\gamma) \in R^\gamma$. In our situation, it requires that $\tau(a') = \tau(\sum_{i=1}^{\gamma} r_i a_i) = \sum_{i=1}^{\gamma} r_i \tau(a_i)$ which is clearly not the case. The same problem also arises in [CG20]. Therefore, we follow the approach in [CG20] by the committed MPC technique [FPY18] to obtain $\langle \tau(a)\tau(b) \rangle$. There is some difference from [CG20], our share and MAC are defined over the same domain R, while the share and MAC in [CG20] lie in different spaces. This forces them to convert the sharing of a vector $\boldsymbol{x}$ into a sharing of an element x in the extension field at each opening. In our work, we do not need this extra step, which brings us closer to the original approach in [FPY18].

Procedure 4: π_{Mult}

The procedure takes as input a set of $N = \gamma_1 + \gamma_1\gamma_2^2 T$ authenticated pairs $(\langle a_i \rangle, \langle \tau(a_i) \rangle), (\langle b_i \rangle, \langle \tau(b_i) \rangle)$ where $[\![\tau(a_i)]\!] = (\tau(a_i)^{(1)}, \ldots, \tau(a_i)^{(n)})$ and $[\![\tau(b_i)]\!] = (\tau(b_i)^{(1)}, \ldots, \tau(b_i)^{(n)})$. As output, the procedure produces a multiplication quintuple $(\langle a \rangle, \langle b \rangle, \langle \tau(a) \rangle, \langle \tau(b) \rangle, \langle \tau(a)\tau(b) \rangle)$ where $a, b \in R$ are random elements. We assume the functionalities $\mathcal{F}_{\mathsf{COPEe}}$,$\mathcal{F}_{\mathsf{Auth}}$ and $\mathcal{F}_{\mathsf{Coin}}$. Let $u = 2N$.

- **Multiply:**
  1. For $h = 1, \ldots, N$
  2. For each ordered pair of parties P_i and P_j calls $\mathcal{F}_{\mathsf{COPEe}}$with P_i's input $\tau(a_h)^{(i)}$ and P_j's input $\tau(b_h)^{(j)}$.
  3. P_i receives $u_h^{(i,j)}$ and P_i receives $v_h^{(j,i)}$ such that
  $$u_h^{(i,j)} + v_h^{(j,i)} = \tau(a_h)^{(i)}\tau(b_h)^{(j)}.$$
  4. P_i sets its share $c_h^{(i)} = \tau(a_h)^{(i)}\tau(b_h)^{(i)} + \sum_{j \neq i} u_h^{(i,j)} + v_h^{(i,j)}$.
  5. P_i call $\mathcal{F}_{\mathsf{Auth}}$with the command Input to obtain $\left\langle c_h^{(i)} \right\rangle$.
  6. The parties locally compute $\langle c_h \rangle = \sum_{i=1}^{n} \left\langle c_h^{(i)} \right\rangle$.
- **Cut-and-Choose:**
  1. All parties call $\mathcal{F}_{\mathsf{Coin}}$to obtain γ_1 distinct elements $1 \leq \ell_1, \ldots, \ell_{\gamma_1} \leq N$.
  2. All parties open $\langle \tau(a_{\ell_i}) \rangle, \langle \tau(b_{\ell_i}) \rangle, \langle c_{\ell_i} \rangle$ for $i = 1, \ldots, \gamma_1$. Abort if $\tau(a_{\ell_i})\tau(b_{\ell_i}) \neq c_{\ell_i}$.
- **Sacrifice:**

1. Use $\mathcal{F}_{\mathsf{Coin}}$to random divide the rest $N-\gamma_1$ triples $(\langle\tau(a_h)\rangle, \langle\tau(b_h)\rangle, \langle c_h\rangle)$ into $\gamma_2^2 T$ buckets with γ_1 triples in each. (We still record the other two sharings $\langle a_h\rangle$ and $\langle b_h\rangle$ for later use)
2. In each bucket with γ_1 triples $(\langle\tau(a_i)\rangle, \langle\tau(b_i)\rangle, \langle c_i\rangle)_{i\in[\gamma_1]}$, all parties locally compute $\langle\alpha_h\rangle = \langle\tau(a_h)\rangle - \langle\tau(a_1)\rangle$ and $\langle\beta_h\rangle = \langle\tau(b_h)\rangle - \langle\tau(b_1)\rangle$ for $h = 2, \ldots, \gamma_1$. All parties partially open α_h and β_h.
3. Compute $\langle\rho_h\rangle = \langle c_h\rangle - \alpha_h \langle\tau(b_1)\rangle - \beta_h \langle\tau(a_1)\rangle - \alpha_h\beta_h - \langle c_1\rangle$. Open ρ_h for $h = 2, \ldots, \gamma_1$. Abort if $\rho_h \neq 0$ and otherwise call $(\langle\tau(a_1)\rangle, \langle\tau(b_1)\rangle, \langle c_1\rangle)$ a correct triple.

– **Combine:**

1. Combine on a: Use $\mathcal{F}_{\mathsf{Coin}}$to randomly divide the remaining $\gamma_2^2 T$ triples $(\langle\tau(a_h)\rangle, \langle\tau(b_h)\rangle, \langle c_h\rangle)$ into $\gamma_2 T$ buckets with γ_2 triples in each. In each bucket, denote by $(\langle\tau(a_h)\rangle, \langle\tau(b_h)\rangle, \langle c_h\rangle)$ for $h = 1, \ldots, \gamma_2$.
   (a) Locally compute
   $$\langle a'\rangle = \sum_{i=1}^{\gamma_2} \langle a_i\rangle, \quad \langle\tau(a')\rangle = \sum_{i=1}^{\gamma_2} \langle\tau(a_i)\rangle, \quad \langle\tau(b')\rangle = \langle\tau(b_1)\rangle.$$
   (b) For $h = 2, \ldots, \gamma_2$, locally compute $\langle\rho_h\rangle = \langle\tau(b_1)\rangle - \langle\tau(b_h)\rangle$ and partially open ρ_h.
   (c) Compute $\langle\sigma'\rangle = \langle c_1\rangle + \sum_{i=2}^{\gamma_2}(\rho_h \langle\tau(a_h)\rangle + \langle c_h\rangle) = \langle\tau(a')\tau(b_1)\rangle$. Record the new quintuple $(\langle a'\rangle, \langle\tau(a')\rangle, \langle b'\rangle, \langle\tau(b')\rangle, \langle\sigma'\rangle)$.
2. Combine on b: Use $\mathcal{F}_{\mathsf{Coin}}$to randomly divide the remaining $\gamma_2 T$ quintuples into $\gamma_2 T$ buckets with γ_2 triples in each. In each bucket, denote by $(\langle a_h\rangle, \langle\tau(a_h)\rangle, \langle b_h\rangle, \langle\tau(b_h)\rangle, \langle c_h\rangle)$ for $h = 1, \ldots, \gamma_2$.
   (a) Locally compute
   $$\langle b'\rangle = \sum_{i=1}^{\gamma_2} \langle b_i\rangle, \quad \langle\tau(b')\rangle = \sum_{i=1}^{\gamma_2} \langle\tau(b_i)\rangle, \quad \langle\tau(a')\rangle = \langle\tau(a_1)\rangle.$$
   (b) For $h = 2, \ldots, \gamma_2$, locally compute $\langle\rho_h\rangle = \langle\tau(a_1)\rangle - \langle\tau(a_h)\rangle$ and partially open ρ_h.
   (c) Compute $\langle\sigma'\rangle = \langle c_1\rangle + \sum_{h=2}^{\gamma_2}(\rho_h \langle\tau(b_h)\rangle + \langle c_h\rangle) = \langle\tau(a')\tau(b')\rangle$. Record the new quintuple $(\langle a'\rangle, \langle\tau(a')\rangle, \langle b'\rangle, \langle\tau(b')\rangle, \langle\sigma'\rangle)$.
   (d) Call $\mathcal{F}_{\mathsf{Auth}}$with the command Check on the opened values so far. If the check succeeds, output the T quintuples.

Putting the pieces together. With the procedures π_{Enc}, π_{Ker} and π_{Mul} from above, we can now define the protocol Π_{Prep} that instantiates the functionality $\mathcal{F}_{\mathsf{Prep}}$. The protocol simply uses π_{Ker} to instantiate the Ker command, and π_{Enc} and π_{Mul} in conjunction to instantiate $\mathsf{CorrRand}$. Π_{Prep} is described in detail as Protocol 10 in Section B in the Supplementary Material. Its security is stated in the following theorem, and since the proof follows in a similar way as the argument in [CG20,FPY18], we postpone it to Section C in the Supplementary Material.

Theorem 6. *Π_{Prep} securely implements $\mathcal{F}_{\mathsf{Prep}}$ in the ($\mathcal{F}_{\mathsf{COPEe}}$,$\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{Auth}}$)-hybrid model.*

6 Communication Complexity Analysis

Our online phase can simultaneously evaluate ℓ instances of the same arithmetic circuit over the ring $\mathbb{Z}_{p^k}$. To the best of our knowledge, all known secret-sharing-based amortized SPDZ-like protocols are defined over finite fields, e.g., MiniMAC [DZ13], Committed MPC [FPY18] and RMFE-based MPC [CG20]. We choose $R = \mathsf{GR}(2^k, m)$ to analyze the communication complexity below. When we compare our performance with other works over finite fields, we assume $k = 1$.

Communication complexity of the online phase. Our online phase only requires one round of communication for each multiplication gate. For each multiplication gate, we need to partially open two shared values $\langle r \rangle$ with $r \in R$. At each opening, all parties sent their shares to one selected player for opening and this player then broadcasts the opened value. This requires $4km(n-1)$ bits of communication.[16] Each input gate requires $km(n-1)$ bits of communication. For the output gate, the parties have to perform the MAC check on the random linear combination of previously opened values. This requires $2km(n-1) + 2kmn$ bits of communication. If we take the approach from Remark 1, the communication cost for each multiplication gate can be further cut down to $(2km + 2k\ell)(n-1)$.

Below we analyze the communication complexity of our protocol with respect to that from other approaches. These were taken from [CG20].

	MiniMac[DZ13]	Committed MPC [FPY18]	Cascudo et al. [CG20]	This work
Comm.	$(4\ell + 2\ell^*)(n-1)$	$(4\ell + 2\ell^* + r)(n-1)$	$(4\ell + 2m)(n-1)$	$4m(n-1)$
Rounds	2	1	2	1

Fig. 1. The total number of bits and rounds communicated for *one batch* of ℓ multiplications in the online phase.

Now we instantiate the concrete parameters for the RMFE construction, and compare our *amortized* communication complexity (that is, per multiplication) with respect to previous works. To make a fair comparison, we use the same parameter set (ℓ, m) as in [CG20], namely $(\ell, m) = (21, 65)$, $(\ell, m) = (42, 135)$ and $(\ell, \ell^*, r) = (210, 1695, 2047)$, $(\ell, \ell^*, r) = (338, 3293, 4096)$ (in our RMFE, ℓ is the number of instances simultaneously computed).

[16] Like in previous works, we assume the cost of broadcast-with-abort is comparable sending the messages directly.

Security	MiniMac[DZ13]	Committed MPC [FPY18]	Cascudo et al. [CG20]	This work
$s = 64$	$20.14(n-1)$	$29.89(n-1)$	$10.2(n-1)$	$12.4(n-1)$
$s = 128$	$23.48(n-1)$	$35.58(n-1)$	$10.42(n-1)$	$12.84(n-1)$

Fig. 2. The total number of bits communicated for computing each instance of a multiplication gate when the security parameter is $s = 64, 128$.

Communication complexity of the preprocessing phase. The main bottleneck of our preprocessing protocol Π_{Prep} are the π_{Enc} and the π_{Mul} procedures (only one kernel element is needed for each final output, so we ignore the cost of π_{Ker}). Since Π_{Prep} uses $\mathcal{F}_{\mathsf{Auth}}$, we also take into account the costs of Π_{Auth}. Furthermore, since Π_{Auth} itself makes use of $\mathcal{F}_{\mathsf{COPEe}}$, and our protocol Π_{COPEe}uses $\mathcal{F}_{\mathsf{OLE}}$, we measure the ultimate costs in terms of the number of calls to $\mathcal{F}_{\mathsf{OLE}}$.

The Input command realized by Π_{Auth} aims at generating the authenticated input for each party. The most expensive cost of this protocol is the call to Π_{COPEe}. The parties need to send $2kt(n-1)$ bits for each element in R to be authenticated and make $t(n-1)$ calls to the functionality $\mathcal{F}_{\mathsf{OLE}}$.

The check procedure is used to check the correctness of the shares obtained from Π_{COPEe}. The cost of the check protocol can be amortized away as it can authenticate a batch of values by sacrificing one authenticated value. For the command $\mathsf{CorrRand}$, we first need to prepare the authenticated pairs $(\langle a\rangle, \langle \tau(a)\rangle)$ with π_{Enc}. Assume that T is much larger than s, which will amortize away the cost of sacrificing s pairs. Thus, the cost for generating authenticated pairs $(\langle a\rangle, \langle \tau(a)\rangle)$ comes from the call of Π_{Auth} with the command Input. The total cost is $4ktn(n-1)$ bits for each authenticated pairs and $2t(n-1)$ calls of $\mathcal{F}_{\mathsf{OLE}}$.

After generating $N = \gamma_1 + \gamma_1\gamma_2^2 T$ authenticated pairs, which requires $8ktn(n-1)N$ bits of communication and $4t(n-1)N$ calls of $\mathcal{F}_{\mathsf{OLE}}$, we use π_{Mul} to create T authenticated quintuples $(\langle a\rangle, \langle b\rangle, \langle \tau(a)\rangle, \langle \tau(b)\rangle, \langle \tau(a)\tau(b)\rangle)$. In the following discussion, we assume T is big enough so as to amortize away the part of the communication that is independent of T. The procedure π_{Mul} calls Π_{COPEe} $n(n-1)N$ times, which requires $2ktn(n-1)N$ bits of communication and $tn(n-1)N$ calls of $\mathcal{F}_{\mathsf{OLE}}$. Each party P_i calls Π_{Auth} to obtain the authenticated share $\left\langle c_h^{(i)} \right\rangle$. This requires $2ktn(n-1)N$ bits of communication and $tn(n-1)$ calls of $\mathcal{F}_{\mathsf{OLE}}$. The total amount of bits communicated in the procedure π_{Mul} is $4ktn(n-1)N$ and $2tn(n-1)$ calls of $\mathcal{F}_{\mathsf{OLE}}$. During the rest of the protocol, the communication costs mainly come from the partial opening. There are $3\gamma_1 + 3\gamma_2^2 T(\gamma_1 - 1) + \gamma_2 T(\gamma_2 - 1) + T(\gamma_2 - 1)$ openings in total such that each opening requires $2mt(n-1)$ bits of communications. We choose $\gamma_1 = \gamma_2 = 3$ suggested in [FPY18]. The total bits of communication from the opening is $26T \times 2mt(n-1)$. This cost is relatively small if n is big enough.

From the above we see that generating one quintuple requires $12ktn(n-1)N/T = 12 \times 27ktn(n-1) = 324ktn(n-1)$ bits of communication and $6tn(n-1)N/T = 162tn(n-1)$ calls to $\mathcal{F}_{\mathsf{OLE}}$. From Theorem 1 and Theorem 2, we can set

$t = 5.12m$ and $m = 4.92\ell$. The communications now becomes $8161.6k\ell n(n-1)$. For small security parameter $s = 64$, we can set $m/\ell = 3.1$. To compare our result with that in [CG20], we set $k = 1$ and obtain $5142.5\ell n(n-1)$ bits of communication while their protocol requires $462.21\ell^2 n(n-1)$ bits. Our cost is smaller than theirs as ℓ has to be 21 or bigger to achieve 64 bits of statistical security.

Comparing to SPD$\mathbb{Z}_{2^k}$ [CDE+18]. We proceed to the comparison with SPD$\mathbb{Z}_{2^k}$ over the ring $\mathbb{Z}_{2^k}$. The prepocessing phase in [CDE+18] generates a Beaver triple by sacrificing $4k + 2\ell$ authenticated shares. The total cost of communication complexity to generate a Beaver triple is $2(k+2s)(9s+4k)n(n-1)$ where s is the security parameter. This is bigger than ours $5142.5kn(n-1)$ if $k \leq 29$. If we set the security parameter to be 128, then our prepocessing phase outperforms the one in [CDE+18] for $k \leq 114$. Our communication complexity does not grow with the security parameter. This gives us an advantage for larger security parameters.

	Online phase	$s = 64$	$s = 128$
This work	$19.68k(n-1)$	$12.4k(n-1)$	$12.84k(n-1)$
SPD$\mathbb{Z}_{2^k}$	$4(k+s)(n-1)$	$4(k+64)(n-1)$	$4(k+128)(n-1)$

Fig. 3. Comparison to SPD$\mathbb{Z}_{2^k}$ in the online phase. We note that SPD$\mathbb{Z}_{2^k}$ and its variant have the same communication complexity in the online phase

Acknowledgement

The research of C. Xing is supported in part by the National Key Research and Development Project 2021YFE0109900 and the National Natural Science Foundation of China under Grant 12031011. The research of C. Yuan is supported in part by the National Natural Science Foundation of China under Grant 12101403. This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

ACD+19. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/2^k\mathbb{Z}$ via galois rings. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 471–501. Springer, 2019.

ACD+20. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, Matthieu Rambaud, Chaoping Xing, and Chen Yuan. Asymptotically good multiplicative LSSS over galois rings and applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 151–180. Springer, 2020.

ACE+21. Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 222–244. Springer, 2021.

AM69. Michael Francis Atiyah and I. G. MacDonald. *Introduction to commutative algebra.* Addison-Wesley-Longman, 1969.

BCS19. Carsten Baum, Daniele Cozzo, and Nigel P Smart. Using topgear in overdrive: a more efficient zkpok for spdz. In *International Conference on Selected Areas in Cryptography*, pages 274–302. Springer, 2019.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

BMN18. Alexander R. Block, Hemanta K. Maji, and Hai H. Nguyen. Secure computation with constant communication overhead using multiplication embeddings. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, volume 11356 of *Lecture Notes in Computer Science*, pages 375–398. Springer, 2018.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

CC06. Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

CCdHP08. Hao Chen, Ronald Cramer, Robbert de Haan, and Ignacio Cascudo Pueyo. Strongly multiplicative ramp schemes from high degree rational points on curves. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT*

2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, volume 4965 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2008.

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, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426. Springer, 2018.

CDE+18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPDZ2k: Efficient MPC mod 2^k for dishonest majority. In *Annual International Cryptology Conference*, pages 769–798. Springer, 2018.

CG20. Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based mpc protocol for boolean circuits with good amortized complexity. In *TCC*, pages 652–682. Springer, 2020.

CKL21. Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In *Annual International Cryptology Conference*, pages 426–456. Springer, 2021.

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

CRX21. 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. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 656–686. Springer, 2021.

DEF+19. Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120. IEEE, 2019.

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

DZ13. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 621–641. Springer, 2013.

FPY18. Tore Kasper Frederiksen, Benny Pinkas, and Avishay Yanai. Committed MPC - maliciously secure multiparty computation from homomorphic commitments. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - , Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 587–619. Springer, 2018.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *CCS 2016, Vienna, Austria, October 24-28, 2016*, pages 830–842. ACM, 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, 2018.

LOS14. Enrique Larraia, Emmanuela Orsini, and Nigel P Smart. Dishonest majority multi-party computation for binary circuits. In *Annual Cryptology Conference*, pages 495–512. Springer, 2014.

OSV20. Emmanuela Orsini, Nigel P Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}/2^k\mathbb{Z}$ from somewhat homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 254–283. Springer, 2020.

PCCX09. Ignacio Cascudo Pueyo, Hao Chen, Ronald Cramer, and Chaoping Xing. Asymptotically good ideal linear secret sharing with strong multiplication over *Any* fixed finite field. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 466–486. Springer, 2009.

PCX11. Ignacio Cascudo Pueyo, Ronald Cramer, and Chaoping Xing. The torsion-limit for algebraic function fields and its application to arithmetic secret sharing. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 685–705. Springer, 2011.

RSS19. Deevashwer Rathee, Thomas Schneider, and KK Shukla. Improved multiplication triple generation over rings via rlwe-based ahe. In *International Conference on Cryptology and Network Security*, pages 347–359. Springer, 2019.

Wan03. Zhe-Xian Wan. *Lectures on finite fields and Galois rings*. World Scientific Publishing Company, 2003.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.

Supplementary Material

A Glossary of Parameters

Symbol	Description
p^k	Modulus, where p is a prime and k is a positive integer
κ	Statistical security parameter
R	Galois ring $\mathsf{GR}(p^k, m)$
$\overline{R}$	Galois field $\mathsf{GF}(p^m)$, obtained by taking modulo p to elements in R
m	Degree of R. Satisfies $p^m \geq 2^\kappa$
ℓ	Dimension of the vectors over $\mathbb{Z}_{p^k}$ used for the computation
t	Dimension of the vectors over $\mathbb{Z}_{p^k}$ used for the MFEs
$\phi,\ \psi$	RMFE functions $\phi : \mathbb{Z}_{p^k}^\ell \to R$ and $\psi : R \to \mathbb{Z}_{p^k}^\ell$. They satisfy $\boldsymbol{x} \star \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_{p^k}^\ell$
$\rho,\ \mu$	MFE functions $\rho : \mathbb{Z}_{p^k}^t \to R$ and $\mu : R \to \mathbb{Z}_{p^k}^t$. They satisfy $xy = \rho(\mu(x) \star \mu(y))$ for $x, y \in R$
τ	Mapping $\tau : R \to R$ given by $\tau = \phi \circ \psi$

B Missing Functionalities and Protocols

Communication resource. This functionality models the communication channels assumed.

Functionality 5: $\mathcal{F}_{\mathsf{Channels}}$

The functionality proceeds as follows:

- Upon receiving $(\mathsf{Message}, x, P_i, P_j)$ from P_i, send x to P_j.[a]
- Upon receiving $(\mathsf{Broadcast}, x, P_i)$ from P_i, send x to all parties.
- Upon receiving $(\mathsf{Simultaneous}, x_i, P_i)$ from each P_i, store this value. Once all parties have provided input, send $\{x_i\}_{i\in[n]}$ to each party P_j.[b]

[a] A more detailed functionality should allow the environment to choose the order in which the messages are delivered in order to model a *rushing adversary*, which is able to receive the messages from the honest parties before any other party does.

[b] We assume these calls use unique identifiers so that the functionality can keep track when everyone has sent their value.

MPC Functionality. This is the main functionality that we aim at instantiating in this work. It models an arithmetic black box that stores *vectors* in S^ℓ and enables pointwise additions and multiplications to be carried out on them.

Functionality 6: $\mathcal{F}_{\mathsf{MPC}}$

The functionality maintains a dictionary Val, which it uses to keep track of authenticated elements of S^ℓ. We use $\langle \boldsymbol{x} \rangle$ to denote the fact that the functionality stores $\mathsf{Val}[\mathtt{id}] = \boldsymbol{x}$ for some identifier $\mathtt{id}$.

- **Input:** On input $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_L), P_i)$ from P_i and $(\mathsf{Input}, (\mathtt{id}_1, \mathtt{id}_2, \dots \mathtt{id}_L), P_i)$ from all other parties, set $\mathsf{Val}[\mathtt{id}_j] = \boldsymbol{x}_j$ for $j = 1, 2, \dots, L$.
- **Affine combination:** On input $(\mathsf{AffComb}, \mathtt{id}, (\mathtt{id}_1, \dots, \mathtt{id}_L), (a_1, \dots, a_L), \boldsymbol{a})$ from all parties, where $a_j \in S$ for $j = 1, \dots, L$ and $\boldsymbol{a} \in S^\ell$, the functionality computes $\boldsymbol{z} = a + \sum_{j=1}^{L} a_j \cdot \mathsf{Val}[\mathtt{id}_j]$ and stores $\mathsf{Val}[\mathtt{id}] = \boldsymbol{z}$. We denote this by $\langle \boldsymbol{z} \rangle \leftarrow \boldsymbol{a} + \sum_{j=1}^{L} a_j \langle \boldsymbol{x}_j \rangle$, where $\boldsymbol{x}_j = \mathsf{Val}[\mathtt{id}_j]$.
- **Multiplication:** On input $(\mathsf{Mult}, \mathtt{id}, (\mathtt{id}_1, \mathtt{id}_2))$ from all parties, the functionality computes $\boldsymbol{z} = \mathsf{Val}[\mathtt{id}_1] \star \mathsf{Val}[\mathtt{id}_2]$ and stores $\mathsf{Val}[\mathtt{id}] = \boldsymbol{z}$.
- **Partial openings:** On input $(\mathsf{Open}, \mathtt{id}, \mathcal{S})$ from all parties, where $\mathcal{S}$ is a non-empty subset of parties. If $\mathsf{Val}[\mathtt{id}] \neq \perp$, wait for an $\boldsymbol{x} \in S^\ell$ from the adversary and send $\boldsymbol{x}$ to the honest parties in $\mathcal{S}$.
- **Check openings:** On input $(\mathsf{Check}, (\mathtt{id}_1, \mathtt{id}_2, \dots, \mathtt{id}_L), (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_L))$ from every party wait for an input from the adversary. If he inputs OK, and if $\mathsf{Val}[\mathtt{id}_j] = \boldsymbol{x}_j$ for $j = 1, 2, \dots, L$, return OK to all parties. Otherwise abort.

Public coins. This functionality returns uniformly random values in R to the parties.

Functionality 7: $\mathcal{F}_{\mathsf{Coin}}$

Upon receiving (Rand, R) from all parties, select a uniform element $x \xleftarrow{\$} R$ and send x to all parties.

Local operations. The parties can locally compute affine combinations on $\langle \cdot \rangle$-shared values using the π_{Aff} procedure defined below.

Procedure 5: $\pi_{\mathsf{Aff}}(\langle x_1 \rangle, \dots, \langle x_L \rangle, a_1, \dots, a_L, a)$

Given L shared values $\langle x_i \rangle = (x_i^{(j)}, m(x_i)^{(j)}, \alpha^{(i)})_{i \in [n]}$ for $i = 1, \dots, L$ and $L+1$ elements $a_1, \dots, a_L, a$, the parties proceed as follows to obtain shares of $a + \sum_{j=1}^{L} a_j x_j$:

1. For $i \in [n] \setminus \{1\}$, Each party P_i locally computes

$$y^{(i)} = \sum_{j=1}^{L} a_j x_j^{(i)}, \quad m(y)^{(i)} = a\alpha^{(i)} + \sum_{j=1}^{L} a_j m(x_j)^{(i)}.$$

2. Party P_1 locally computes

$$y^{(1)} = a + \sum_{j=1}^{L} a_j x_j^{(1)}, \quad m(y)^{(1)} = a\alpha^{(1)} + \sum_{j=1}^{L} a_j m(x_j)^{(1)}.$$

3. The parties store the new secret-shared value $\langle y \rangle = (y^{(i)}, m(y)^{(i)}, \alpha^{(i)})_{i \in [n]}$.

Opening and checking. The following procedures enable the parties to open secret-share values and to check their correctness.

Procedure 6: $\pi_{\mathsf{Open}}(\langle y \rangle)$

On input a secret-shared value $\langle y \rangle$:

1. Each party P_i sends their additive share $y^{(i)}$ to P_1
2. P_1 reconstructs $y = y^{(1)} + \cdots + y^{(n)}$ and *broadcasts* y back to the parties.

Procedure 7: $\pi_{\mathsf{Check}}(\langle y \rangle, y')$

Given a shared value $\langle y \rangle$ and a public value y', the parties proceed as follows:

1. Each P_i uses the simultaneous message channel (see Section 2.4) to send $\sigma^{(i)} = m^{(i)}(y) - \alpha^{(i)} \cdot y'$ to each other party.
2. The parties check that $\sigma^{(1)} + \cdots + \sigma^{(n)} = 0$, and abort if this is not the case.

COPE protocol. This protocol instantiates the functionality $\mathcal{F}_{\mathsf{COPEe}}$ presented in Section 4.2.

Protocol 8: Π_{COPEe}

The protocol is a two party protocol with P_i and P_j.

- **Initialize:** On input $\alpha^{(i)} \in R$ from P_i
- **Multiply:** On input $x \in R$ from P_j:
  1. P_i computes $\mu(\alpha^{(i)}) = (a_1, \ldots, a_t) \in \mathbb{Z}_{p^k}^t$ and P_j computes $\mu(x) = (x_1, \ldots, x_t) \in \mathbb{Z}_{p^k}^t$.
  2. For $h = 1, \ldots, t$, P_i and P_j call $\mathcal{F}_{\mathsf{OLE}}$ where P_i inputs a_h and P_j inputs x_h. P_i receives $-b_h = a_h x_h - y_h$.
  3. P_i sets $u^{(i,j)} = -\rho((b_1, \ldots, b_t))$ and P_j sets $v^{(j,i)} = \rho((y_1, \ldots, y_t))$.

Kernel procedure. This procedure generates shared elements $\langle r \rangle$, where $r \xleftarrow{\$} \ker \psi$.

Procedure 9: π_{Ker}

The procedure generates $\langle r_i \rangle$ for $i = 1, \ldots, u$ where $r \in \ker(\psi^{-1})$ is a random element in R and unknown to all parties. We assume the access to the functionalities $\mathcal{F}_{\mathsf{Auth}}$,$\mathcal{F}_{\mathsf{Coin}}$.

- **Construct:**
  1. P_i samples $r_j^{(i)} \in \ker(\psi^{-1})$ uniformly at random for $j = 1, \ldots, s+1$.
  2. P_i calls $\mathcal{F}_{\mathsf{Auth}}$ to obtain $\left\langle r_j^{(i)} \right\rangle$.
  3. All parties computes
$$\langle r_j \rangle = \sum_{i=1}^{n} \left\langle r_j^{(i)} \right\rangle .$$
- **Sacrifice:**
  1. All parties call $\mathcal{F}_{\mathsf{Coin}}$to generate s random vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,u}) \in \mathbb{Z}_{p^k}^u$.
  2. Compute $\langle b_i \rangle = \sum_{j=1}^{u} x_{i,j} \langle r_j \rangle + \langle r_{u+i} \rangle$ and partially open b_i.
  3. If $b_i \notin \ker(\psi^{-1})$ for some $i \in \{1, \ldots, s\}$, then abort.
  4. call $\mathcal{F}_{\mathsf{Auth}}$with the command Check on the opened values b_i.
- **Output:** Output $\langle r_i \rangle$ for $i = 1, \ldots, u$.

Preprocessing protocol. This protocol instantiates the necessary preprocessing. It assumes access to $\mathcal{F}_{\mathsf{Auth}}$,$\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{COPEe}}$.

Protocol 10: Π_{Prep}

The protocol has a dictionary Val, and it forwards the commands Input, $\mathsf{AffComb}$, Open and Check to $\mathcal{F}_{\mathsf{Auth}}$. For the other commands:

- **Correlated randomness**: On input $(\mathsf{CorrRand}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}_3, \mathtt{id}_4, \mathtt{id}_5)$, the parties run π_{Enc}, followed by π_{Mul}, to obtain sharings $(\langle a \rangle, \langle b \rangle, \langle \tau(a) \rangle, \langle \tau(b) \rangle, \langle \tau(a)\tau(b) \rangle)$, and store these under the identifiers $\mathtt{id}_1, \ldots, \mathtt{id}_5$.
- **Kernel element**: On input $(\mathsf{Ker}, \mathtt{id})$, the parties call the procedure π_{Ker} to obtain $\langle r \rangle$ with $r \in \ker \psi$, and store this value in the identifier $\mathtt{id}$.

C Missing Proofs

Theorem 7 (Theorem 3 re-stated). *Protocol Π_{Online} implements $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model*

Proof. In the input step, if P_i is honest, the simulator $\mathcal{S}$ broadcasts a random element $r \in R$. If P_i is corrupted, $\mathcal{S}$ waits for $\mathcal{A}$ to broadcast with $\epsilon \in R$. $\mathcal{S}$ sets $x_i' = \epsilon + r_i'$ and stores $(\mathsf{Input}, \mathtt{id}, x_i', P_i)$ as input in $\mathcal{F}_{\mathsf{Prep}}$. The addition is done locally which can be simulated trivially. As for the multiplication step, $\mathcal{S}$ sends

two random elements to the adversary ϵ, ρ and wait for it to input ϵ' and ρ'. If $(\epsilon, \rho) \neq (\epsilon', \rho')$, $\mathcal{S}$ will abort at the check opening step. At the check opening step, $\mathcal{S}$ receives the output z from $\mathcal{F}_{\mathsf{Prep}}$ in the check opening and sample $x \in z + \ker(\psi)$ uniformly at random. $\mathcal{S}$ sends x to $\mathcal{A}$ and wait $\mathcal{A}$ to reply with $\boldsymbol{x}'$. If $\boldsymbol{x}' \neq \psi(z)$, then $\mathcal{S}$ aborts.

As for the indistinguishability, we first argue that up to the output step, the view of the adversary is exactly the same in both ideal world and real world. In the input step, the value broadcast by the honest party is uniformly at random in both world. In the multiplication gate, the adversary receives honest parties shares of two fresh random values. These shares are uniformly at random in both of the world. In the meanwhile, the honest parties MAC shares of this opened values are the random sharings of a correct MAC with an error added by the adversary in the input step. Therefore, the MAC share and the share of the opened value hold by the honest parties are uniformly at random in both of the world. Now, we move to the check openings step. The adversary sees x and the shares from the honest parties. x is chosen uniformly at random in $\ker(\psi) + z$. It is clear that in both of the world, the view of the adversary is exactly the same. We remain to bound the abort probability. In the ideal world, if the adversary partially opened an incorrect value in the input step, multiplication step or the check openings step, the simulator will abort. In the real world, we will argue that the protocol will not abort with the negligible probability. This is due to the following game which also appears in [DPSZ12].

1. The challenger generates the secret key $\alpha \xleftarrow{\$} R$ and MACs $\gamma_i = \alpha m_i$ and sends $m_1, \ldots, m_T$ to the adversary.
2. The adversary sends back $m'_1, \ldots, m'_T$.
3. The challengers generates the random values $r_1, \ldots, r_T \in R$.
4. The adversary provides an error Δ.
5. The challenger checks that $\alpha \sum_{i=1}^{T} r_i(m_i - m'_i) = \Delta$.

The adversary wins this game if the check pass and $m_i \neq m'_i$ for some i. Note that α is kept secret to the adversary. Let r be the biggest integer such that $p^r \,|\, \gcd(m_1 - m'_1, \ldots, m_T - m'_T)$. It is clear that $r < k$ otherwise $m_i = m'_i$ for all i. The following equation should hold

$$\alpha \sum_{i=1}^{T} r_i \frac{m_i - m'_i}{p^r} = \frac{\Delta}{p^r} \bmod p$$

Then, this equation is defined over the residue field $\mathbb{F}_{p^m}$ and at least one of the $\frac{m_i - m'_i}{p^r}$ is nonzero. Therefore, this equality holds with probability at most $2p^{-m}$. This is indistinguishable. □

Theorem 8 (Theorem 4 re-stated). *Protocol Π_{COPEe} implements $\mathcal{F}_{\mathsf{COPEe}}$ in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model.*

Proof. If P_i is corrupted, the simulator receives $\boldsymbol{\alpha}^{(i,j)} = (a_1^{(i,j)}, \ldots, a_t^{(i,j)})$ from the adversary. The simulator samples $\boldsymbol{y} = (y_1, \ldots, y_t) \xleftarrow{\$} S^t$ uniformly at random and sends y to the adversary.

If P_j is corrupted, the simulator receives $\boldsymbol{x}^{(i,j)}$ from the adversary. The simulator does nothing as there is no output for P_j. The indistinguishability is clear since the output looks uniformly at random both in the real world and the ideal world. □

Theorem 9 (Theorem 6 re-stated). *Π_{Prep} securely implements $\mathcal{F}_{\mathsf{Prep}}$ in the ($\mathcal{F}_{\mathsf{COPEe}}$,$\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{Auth}}$)-hybrid model.*

Proof. The proof is similar to the argument in [CG20,FPY18]. We first prove that π_{Ker} securely implements $\mathcal{F}_{\mathsf{Prep}}$ with the command Ker. Note that the kernel space of ψ^{-1} is $\mathbb{Z}_{p^k}$-linear. This means the linear combination of elements in ψ^{-1} must still lie in this space. Let $V = \ker(\psi^{-1})$. The ring R can be partitioned into $\frac{|R|}{|V|}$ cosets $x + V$. Since R/V is an abelian group, we can denote by $\bar{x}$ the element representing $x + V$ in this group.

Assume that $r_i \in a_i + V$. Then, this problem is reduced to the following equality $\sum_{i=1}^{u} x_{i,j}\bar{a}_i + \bar{a}_{u+j} \in V$ for random element $x_i \in \mathbb{Z}_{p^k}$. It is clear that this holds with at most $\frac{1}{p}$ if $\bar{a}_i \neq \bar{0}$ for some i. Since we do this s times, and the adversary succeeds if it passes all s tests, this can only happen with probability at most p^{-s}.

Now we show that π_{Mul} securely implements the command $\mathsf{CorrRand}$ in $\mathcal{F}_{\mathsf{Prep}}$. The parameters γ_1, γ_2 are chosen the same as τ_1, τ_2 in [CG20]. Thus, we omit the security parameter analysis and focus on the general idea behind this argument. After the call to π_{Enc}, the parties obtain authenticated pairs $(\langle a_h \rangle, \langle \tau(a_h) \rangle)$. We first argue that this step securely generate authenticated pairs $(\langle a_h \rangle, \langle \tau(a_h) \rangle)$ in the ($\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{Auth}}$)-hybrid model with statistical security parameter s. Let $\epsilon_j = \tau(b_j) - c_j$ for $j = 1, \ldots, u + s$. Assume that there exists some $\epsilon_j \neq 0$ for $j = 1, \ldots, u$. Then, the equality $\sum_{i=1}^{u} x_{i,j}\epsilon_j + \epsilon_{u+h} = 0$ holds with probability at most $\frac{1}{p}$ as the adversary can not control the random element $x_{i,j} \in \mathbb{Z}_{p^k}$. Since the adversary succeeds if it passes all s tests, this probability is bounded by p^{-s}.

Now, during the Multiply step, suppose that the adversary inputs $a_h^{(i,j)}$ $b_h^{(i,j)}$ instead of $\tau(a_h^{(j)})$, $\tau(b_h^{(j)})$ for $j \in \mathcal{C}$. Denote by $e_{a_h}^{(i,j)} = \tau(a_h^{(j)}) - a_h^{(i,j)}$ and $e_{b_h}^{(i,j)} = \tau(b_h^{(j)}) - b_h^{(i,j)}$. Summing up the shares $c_h^{(i)}$ will lead to

$$c_h = \tau(a_h)\tau(b_h) - \sum_{i \in \mathcal{H}} \sum_{j \in \mathcal{C}} e_{a_h}^{(i,j)} \tau(b_h^{(i)}) + e_{b_h}^{(i,j)} \tau(a_h^{(i)}).$$

From this equation we notice that the adversary can control $e_{a_h}^{(i)} = \sum_{j \in \mathcal{C}} e_{a_h}^{(i,j)}$ and $e_{b_h}^{(i)} = \sum_{j \in \mathcal{C}} e_{b_h}^{(i,j)}$. The adversary can also introduce the authenticated error $e_{auth,h}$ so that

$$\sum_{i \in \mathcal{H}} e_{a_h}^{(i)} \tau(b_h^{(i)}) + e_{b_h}^{(i)} \tau(a_h^{(i)}) + e_{auth,h} = 0.$$

If this equality does not hold, then we call this triple *malformed*. The cut-and-choose step can ensure that either all buckets consist of correct triples, or there is at least one bucket in the sacrifice step that contains both correct and malformed triples. We are happy with the former case. As for the latter case, we may assume

that the first bucket in the sacrifice step contains both correct and malformed triples. We note that the pair of triples $(1, \ell)$ for $\ell = 2, \ldots, \gamma_1$ can not be either both correct or both malformed. This implies there exists a pair $(1, \ell)$ that one triple is malformed and another one is correct, and we assume for simplicity that the first triple is malformed. When the sacrifice step opens

$$\langle \rho_\ell \rangle = \alpha_\ell \langle \tau(b_1) \rangle + \beta_\ell \langle \tau(a_1) \rangle + \alpha_\ell \beta_\ell - \langle c_1 \rangle - \langle c_\ell \rangle ,$$

ρ_ℓ can not be 0 as the only discrepancy is caused by the malformed $\langle c_1 \rangle$. We observe that by guessing part of $\tau(a_h^{(i)})$, the adversary may be able to create a correct triple by introducing a nonzero error $e_{b_h}^{(i)}$. This can happen because R contains zero divisors. In order to prevent this from happening, we resort to the Combine step. The h-th triple is leaky if either $e_{a_h}^{(i)}$ or $e_{b_h}^{(i)}$ is nonzero and it is also correct. We note that if there are s leaky triples, the adversary must guess at least s bits correctly which happens with probability at most 2^{-s}. The analysis in [FPY18] shows that for fixed s, there exists γ_2 such that with overwhelming probability $1-2^{-s}$, there is at least one non-leaky triple in each bucket containing γ_2 triples. □

D Security Proof of Π_{Auth}

Here we provide a full-fledged proof of Theorem 5, which states that protocol Π_{Auth} implements $\mathcal{F}_{\mathsf{Auth}}$ in the ($\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{COPEe}}$)-hybrid model. The main challenge lies in proving that, if the adversary introduces errors in the $\mathcal{F}_{\mathsf{COPEe}}$ calls in the input phase, the corrupt parties are still committed to a unique set of values that can be extracted by the simulator. In Section 4.4 we provided a sketch of why this holds for the case in which R is a field. Here we provide a more complete and general proof that considers an arbitrary ring $R = \mathsf{GR}(p^k, m)$, and also defines the appropriate simulator. Some of the challenges that appear when switching to a ring with zero divisors such as R is that we cannot talk anymore about vector spaces, based, and other concepts that were used explicitly in the argument in Section 4.4, and only make sense if R is a field. This is addressed by making use of the fact that Galois rings, in spite of having zero divisors, still share many similarities with finite fields due to the fact that they are *local rings*. We recall that our proof presents crucial differences with respect to that from [KOS16][Appendix B] since our $\mathcal{F}_{\mathsf{COPEe}}$ functionality works in a substantially different way (using MFEs instead of correlated OT extension, for instance), plus, as we have mentioned (and as we analyze in Section E), the proof in [KOS16] contains a bug that can be fixed by adapting our proof.

Furthermore, even if we consider only passive security (so no check or input extractions take place), there are other subtle differences of our proof with respect to the ones from [KOS16,CG20], arising again from our more general ring R. These stem from the fact that this ring, in general, may contain a lot of zero-divisors, which creates certain complications when taking linear combinations (either for the final checks, or as part of π_{Aff}).

To illustrate where these subtleties appear in our protocol, we begin by noticing that most simulation-based proofs of secret-sharing-based protocols proceed as follows:

1. The simulator determines the shares of the input values held by the corrupt parties, and also extract the inputs of these parties and send these to the ideal functionality
2. The simulator emulates virtual honest parties and executes the protocol as intended using dummy shares for the honest parties. While doing this, the simulator keeps track of what shares the adversary should hold.
3. The simuator receives the output from the ideal functionality, and "corrects" the virtual honest parties' shares by sampling *uniformly random* shares that are consistent with the shares held by the adversary and the intended output.

Most of the time, when R is a field, this template makes sense: it will indeed hold that the honest parties shares of the output are uniformly random and unknown to the adversary. However, consider a simple function that takes the inputs $x_1, \ldots, x_n$ and returns $z = c_1x_1 + \cdots + c_nx_n$, for some known coefficients $c_1, \ldots, c_n \in R$. The honest parties' shares of z will indeed look random if $c_i \neq 0$ for at least one $i \in \mathcal{H}$, but if $c_i = 0$ for $i \in \mathcal{H}$, then this function depends only on corrupt parties' inputs, and the shares corresponding to the honest parties are known by (and in fact chosen by) the adverary, so the simulator cannot simply sample these at random: these have to be computed from the shares provided by the adversary. Fortunately, this subtlety is typically of no concern, since the case in which the function only depends on corrupt parties' inputs can only occur if the function is constant (since the function cannot depend on which parties are corrupted), which is not an interesting case.

In our setting where $R = \mathsf{GR}(p^k, m)$ for general p^k, these cases where the simulator must act with care are not that irrelevant anymore. For example, assume for simplicity that $m = 1$, and suppose we have that $c_i = p^{k-1}$ for every $i \in [n]$. Then the honest parties' shares of z will have the form $p^{k-1}a + p^{k-1}b$, where a is chosen by the adversary and b is uniformly random, which do not look uniformly random over $\mathbb{Z}_{p^k}$. To address this type of situations, the simulator in our proof needs to keep track, for each value stored in the dictionary Val, the coefficients that express this element as a public affine combination with respect to the input values.[17]. Then, when simulating the honest parties' shares of a given value to be outputted, the simulator samples the shares held by honest parties corresponding to *honest inputs* uniformly at random, subject to these being consistent with the intended output and the shares held by the corrupt parties. We remark that this is a subtle issue that appears when simulating *any* secret-sharing based protocol over rings with zero divisors, but we are not aware of it being mentioned in previous works.

[17] Notice that every value stored in $\mathcal{F}_{\mathsf{Auth}}$ is either provided as input by some party, or obtained as an affine combination of other stored values, so, inductively, every stored value can be written as an affine combination of input values

Theorem 10 (Theorem 5, re-stated). *Protocol Π_{Auth} implements $\mathcal{F}_{\mathsf{Auth}}$ in the ($\mathcal{F}_{\mathsf{Coin}}$,$\mathcal{F}_{\mathsf{COPEe}}$)-hybrid model.*

Proof. We define a simulator $\mathcal{S}$ as follows. For the proof, since P_1 has a special role in the π_{Open} procedure, we assume *for simplicity* that P_1 is corrupt. $\mathcal{S}$ emulates the functionalities $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{COPEe}}$, and in addition, $\mathcal{S}$ emulates virtual honest parties $P_i \in \mathcal{H}$. Furthermore, to handle the subtlety mentioned at the beginning of Section D, $\mathcal{S}$ maintains three internal dictionaries: CorrShr, HonShr, Affine and Opened, which are intended to be used as follows:

- HonShr stores honest parties' shares $\{(x^{(i)}, m^{(i)}(x))\}_{i\in\mathcal{H}}$ for every value provided with the **Input** command by a *corrupt sender*. These are computable by $\mathcal{S}$ since the corrupt parties send these to the virtual honest parties.
- CorrShr stores the adversary's aggregated shares $(x^{(\mathcal{C})}, m^{(\mathcal{C})}(x))$ for every value provided with the **Input** command. If the input comes from an honest party then $\mathcal{S}$—or actually, the virtual honest parties—is the one that generates these aggregated shares, but if the input comes from a corrupt party, then $\mathcal{S}$ must first extract the actual input x as we later show, and compute $x^{(\mathcal{C})} = x - x^{(\mathcal{H})}$
- Affine stores, for each value saved by $\mathcal{F}_{\mathsf{Auth}}$, the coefficients that define this value as an affine combination of all the values provided as input so far.
- Opened stores, for each value $\langle x\rangle$ partially opened to x', the shares $\{(x^{(i)}, m^{(i)}(x))\}_{i\in\mathcal{H}}$ that the honest parties hold.

$\mathcal{S}$ operates as described below. We omit the use of identifiers for the sake of clarity. We argue indistinguishability along with the description of $\mathcal{S}$, except for the Check phase, which requires more care and is postponed to Propostion 1 below.

Initialization phase. $\mathcal{S}$ receives $\alpha^{(i)}$ from each $P_i \in \mathcal{C}$ as part of the emulation of $\mathcal{F}_{\mathsf{COPEe}}$. On behalf of the emulated honest parties, $\mathcal{S}$ samples key shares $\{\beta^{(i)}\}_{i\in\mathcal{H}}$.[18]

Input for honest P_j. Here $\mathcal{S}$ receives from $\mathcal{A}$, for each of the $L+1$ $\mathcal{F}_{\mathsf{COPEe}}$ calls and for each $P_i \in \mathcal{C}$, a vector $\boldsymbol{\alpha}_h^{(i,j)} \in \mathbb{Z}_{p^k}^t$ and a value $u_h^{(i,j)} \in R$, and sets $v^{(j,i)} = \rho(\boldsymbol{\alpha}^{(i,j)} \star \mu(x)) - u^{(i,j)}$. $\mathcal{S}$ emulates P_j by sending to $\mathcal{A}$ random shares $\{x_h^{(i)}\}_{i\in\mathcal{C}}$. Notice that $\mathcal{S}$ can also compute the MAC shares the adversary should hold, namely $m^{(i)}(x_h) = u_h^{(i,j)}$ for $i \in \mathcal{C}$.

For emulating the final part of the input phase where the inputs are checked, $\mathcal{S}$ begins by emulating $\mathcal{F}_{\mathsf{Coin}}$ by sampling $r_1, \ldots, r_L \xleftarrow{\$} R$ and sending these values to $\mathcal{A}$. Then $\mathcal{S}$ emulates $\pi_{\mathsf{Open}}(\langle y\rangle)$ by sending to the corrupt P_1 uniformly random shares $\{y^{(i)}\}_{i\in\mathcal{H}}$ on behalf of the honest parties, receiving y' back from P_1 as part of the emulation of broadcast in $\mathcal{F}_{\mathsf{Channels}}$. Notice that this looks the same

[18] We reserve the terms $\{\alpha^{(i)}\}_{i\in\mathcal{H}}$ for the key shares sampled by honest parties in the *real world*.

as in the real world since there, the shares of the honest parties are uniformly random subject to being consistent with the secret $y = x_0 + \sum_{h=1}^{L} r_h x_h$,[19] which means they simply look uniformly random as x_0 is random and unknown to $\mathcal{A}$.

In the real world, the honest parties use $\sigma^{(i)} = m^{(i)}(y) - y' \cdot \alpha^{(i)}$ for the check. Furthermore, it can be easily shown that these values are uniformly random subject to $\sum_{i\in\mathcal{H}} \sigma^{(i)} = \alpha^{(\mathcal{H})}(y - y') + \sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(j,i)})$. $\mathcal{S}$ knows $\sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(j,i)})$, and it can also compute $y - y'$ as $\delta = \left(y' - \sum_{i\in\mathcal{H}} y^{(i)}\right) - \sum_{i\in\mathcal{C}} \sum_{h=0}^{L} r_h x_h^{(i)}$. As a result, $\mathcal{S}$ can sample the virtual honest parties' values $\{\sigma^{(i)}\}_{i\in\mathcal{H}}$ (using the simulated keys $\beta^{(i)}$, which leads to the same distribution since $\alpha^{(\mathcal{H})} \xleftarrow{\$} R$ is unknown to $\mathcal{A}$) to use them in the emulation of π_{Check}. Also, as part of the emulation of the simultaneous message command in $\mathcal{F}_{\mathsf{Channels}}$, $\mathcal{S}$ receives $\{\sigma^{(i)}\}_{i\in\mathcal{C}}$ from $\mathcal{A}$.

At this point $\mathcal{S}$ can determine if the check would fail in the real world: there, the check passes if and only if $\sum_{i\in\mathcal{C}} \sigma^{(i)} = -\sum_{i\in\mathcal{H}} \sigma^{(i)}$ and since the right-hand side is equal to $-\alpha^{(\mathcal{H})} \cdot \delta - \sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(j,i)})$, we see that the check passes if and only if $\alpha^{(\mathcal{H})} \cdot \delta + (\sum_{i\in\mathcal{C}} \sigma^{(i)} + \sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(j,i)})) = 0$. If $\delta \neq 0$, or if $\sum_{i\in\mathcal{C}} \sigma^{(i)} + \sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(i,j)}) \neq 0$, the above would be an evaluation of $\alpha^{(\mathcal{H})}$ in a non-zero polynomial of degree ≤ 1, which can only be zero with probability $\leq p^{-m}$, based on Lemma 1. From this, $\mathcal{S}$ can simply check if $\delta = 0$ and $\sum_{i\in\mathcal{C}} \sigma^{(i)} + \sum_{i\in\mathcal{C}}(\sum_{h=0}^{L} r_h v_h^{(i,j)}) \neq 0$, and abort if this is not the case.

If $\mathcal{S}$ did not abort, then $\mathcal{S}$ stores in $\mathsf{CorrShr}$ the pair $(x_h^{(\mathcal{C})}, m^{(\mathcal{C})}(x_h))$ for $h \in [L]$.

Input for corrupt P_j. In a nutshell, $\mathcal{S}$ emulates the honest parties (using key shares $\{\beta^{(i)}\}_{i\in\mathcal{H}}$) and the required functionalities exactly as in the real world. More precisely, $\mathcal{S}$ receives from $\mathcal{A}$, for each of the $L+1$ $\mathcal{F}_{\mathsf{COPEe}}$ calls and for each $P_i \in \mathcal{H}$, a vector $\boldsymbol{x}_h^{(j,i)}$ and a value $v_h^{(j,i)} \in R$. $\mathcal{S}$ also receives shares $\{x_h^{(i)}\}_{i\in\mathcal{H}}$ for $h \in \{0, \ldots, L\}$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ by sampling $r_1, \ldots, r_L \xleftarrow{\$} R$ and sending these to $\mathcal{A}$, and as part of the execution of $\pi_{\mathsf{Open}}(\langle y \rangle)$, $\mathcal{S}$ computes the honest parties' shares of $\langle y \rangle$ and sends these to P_1, getting y' back. Finally, the virtual honest parties compute $\sigma^{(i)}$ for $i \in \mathcal{H}$ as the real parties do, except that they use the keys $\beta^{(i)}$, and use these in the execution of $\pi_{\mathsf{Check}}(\langle y \rangle, y')$. In the process of emulating $\mathcal{F}_{\mathsf{Channels}}$ in this check, $\mathcal{S}$ gets $\{\sigma^{(i)}\}_{P_i\in\mathcal{C}}$ from $\mathcal{A}$. So far, it is clear that the real and ideal worlds are indistinguishable.

If $\sum_{i\in[n]} \sigma^{(i)} \neq 0$, then $\mathcal{S}$ aborts, which is exactly the same that would happen in the real world. However, even though this equation holds with the same probability in both worlds (and hence the parties abort with the same probability in both worlds), the problem is that this probability may not be negligible in the case that the adversary uses incorrect $\boldsymbol{x}^{(j,i)}$. Hence, it could happen that the parties do not abort in spite of these errors, and the simulator

[19] Here we use the fact that $r_0 = 1$. If we take $r_0 \xleftarrow{\$} R$ as in [KOS16,CG20], we would have to add that, with overwhelming probability, $r_0 \in R^*$.

must be able to still produce an indistinguishable execution with respect to the case in which the parties do not abort in the real world.

To address this, in case $\mathcal{S}$ does not abort, $\mathcal{S}$ *extracts* a set of inputs $\{x_h\}_h$ as follows: $\mathcal{S}$ defines, for $h \in \{0, \ldots, L\}$,

$$x_h = \frac{\sum_{i \in \mathcal{H}} \rho(\mu(\beta^{(i)}) \star \boldsymbol{x}_h^{(j,i)})}{\beta^{(\mathcal{H})}},$$

and sends these to the ideal functionality $\mathcal{F}_{\mathsf{Auth}}$ on behalf of the corrupt party P_j. Notice that this is possible since with overwhelming probability $\beta^{(\mathcal{H})}$ will be invertible. Finally, $\mathcal{S}$ stores in $\mathsf{CorrShr}$ the sums of the shares held by the adversary, which are $x_h^{(\mathcal{C})} = x_h - \sum_{i \in \mathcal{H}} x_h^{(i)}$ and $m_h^{(\mathcal{C})} = (\alpha^{(\mathcal{C})} + \beta^{(\mathcal{H})}) \cdot x_h - \sum_{i \in \mathcal{H}} (\rho(\mu(\beta^{(i)}) \star \boldsymbol{x}_h^{(j,i)}) - v_h^{(j,i)})$ (recall that $\mathcal{S}$ obtained $\alpha^{(i)}$ for $i \in \mathcal{C}$ from $\mathcal{A}$ in the initialization phase).

Affine combination. $\mathcal{S}$ updates Affine by mapping the given identifier to the coefficients that determine the desired result in terms of an affine combination of the values provided as input so far.

Partial openings. $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{Auth}}$ with the command Open on behalf of the adversary, getting a value z as a result. Let $\langle z \rangle$ be written as an affine combination of *input* values as $c + \sum_{h=1}^{m_{\mathcal{H}}} a_h \langle x_h \rangle + \sum_{h=1}^{m_{\mathcal{C}}} b_h \langle y_h \rangle$, where $\{\langle x_h \rangle\}_{h=1}^{m_{\mathcal{H}}}$ and $\{\langle y_h \rangle\}_{h=1}^{m_{\mathcal{C}}}$ are the values provided as input by honest and corrupt parties, respectively. Notice that $\mathcal{S}$ can compute these coefficients via the dictionary Affine, and also $\mathcal{S}$ knows $(x_h^{(\mathcal{C})}, m^{(\mathcal{C})}(x_h))$ for $h \in [m_{\mathcal{H}}]$ and $(y_h^{(\mathcal{C})}, m^{(\mathcal{C})}(y_h))$ for $h \in [m_{\mathcal{C}}]$ via the dictionary $\mathsf{CorrShr}$, and also $\{(y_h^{(i)}, m^{(i)}(y_h))\}_{i \in \mathcal{H}}$ for $h \in [m_{\mathcal{C}}]$ via HonShr.

$\mathcal{S}$ samples $\{x_h^{(i)}\}_{i \in \mathcal{H}, h \in [m_{\mathcal{H}}]}$ uniformly at random subject to

$$c + \left(\sum_{h=1}^{m_{\mathcal{H}}} a_h x_h^{(\mathcal{C})} + \sum_{h=1}^{m_{\mathcal{C}}} b_h y_h^{(\mathcal{C})} \right) + \sum_{i \in \mathcal{H}} \underbrace{\left(\sum_{h=1}^{m_{\mathcal{H}}} a_h x_h^{(i)} + \sum_{h=1}^{m_{\mathcal{C}}} b_h y_h^{(i)} \right)}_{z^{(i)}} = z,$$

and $\mathcal{S}$ also samples $\{m^{(i)}(x_h)\}_{i \in \mathcal{H}, h \in [m_{\mathcal{H}}]}$ uniformly at random subject to

$$c \cdot \left(\alpha^{(\mathcal{C})} + \beta^{(\mathcal{H})} \right) + \left(\sum_{h=1}^{m_{\mathcal{H}}} a_h m^{(\mathcal{C})}(x_h) + \sum_{h=1}^{m_{\mathcal{C}}} b_h m^{(\mathcal{C})}(y_h) \right) + \sum_{i \in \mathcal{H}} \underbrace{\left(\sum_{h=1}^{m_{\mathcal{H}}} a_h m^{(i)}(x_h) + \sum_{h=1}^{m_{\mathcal{C}}} b_h m^{(i)}(y_h) \right)}_{m^{(i)}(z)} = z \cdot \left(\alpha^{(\mathcal{C})} + \beta^{(\mathcal{H})} \right),$$

Then, as part of the execution of $\pi_{\mathsf{Open}}(\langle z \rangle)$, $\mathcal{S}$ sends $\{z^{(i)} := \sum_{h=1}^{m_{\mathcal{H}}} a_h x_h^{(i)} + \sum_{h=1}^{m_{\mathcal{C}}} b_h y_h^{(i)}\}_{i \in \mathcal{H}}$[20] to P_1, and receives back z'. Finally, $\mathcal{S}$ adds $\{(z^{(i)}, m^{(i)}(z))\}_{i \in \mathcal{H}}$

[20] Recall that we assume $P_1 \in \mathcal{C}$, so none of the honest parties are supposed to add c to their share.

to Opened. It is easy to see that the above execution is indistinguishable from the real world, since the honest parties' shares follow the exact same distribution.

Check openings. Let $z'_1, \ldots, z'_L$ be the opened values, and let $\langle z_1 \rangle, \ldots, \langle z_L \rangle$ be the stored values. $\mathcal{S}$ samples $r_1, \ldots, r_L \xleftarrow{\$} R$ and send these to $\mathcal{A}$ as part of the emulation of $\mathcal{F}_{\mathsf{Coin}}$. As part of the execution of π_{Check}, $\mathcal{S}$ receives $\{\sigma^{(i)}\}_{i \in \mathcal{C}}$ from $\mathcal{A}$. $\mathcal{S}$ can compute $\sigma^{(i)} = \beta^{(i)} \cdot \left(\sum_{h=1}^{L} r_h z'_h \right) - \sum_{h=1}^{L} r_h \cdot m^{(i)}(x_h)$ for $i \in \mathcal{H}$ from Opened, and $\mathcal{S}$ sends these values to $\mathcal{A}$ on behalf of the honest parties as the emulation of $\mathcal{F}_{\mathsf{Channels}}$ in π_{Check}. Then checks whether $\sum_{i \in [n]} \sigma^{(i)} = 0$, and if so $\mathcal{S}$ sends OK to $\mathcal{F}_{\mathsf{Auth}}$. Otherwise, send abort.

If we assume that the adversary behaves honestly when providing inputs for corrupt parties, it is easy to see that this simulation follows the same distribution as in the real world. The general case in which corrupt parties send incorrect messages in the input phase is handled in Proposition 1 below. □

Proposition 1. *In the context of the proof above of Theorem 5, it holds that the simulation of the **Check** phase is indistinguishable from the ideal world, in the case in which a corrupt party P_j behaves incosistently when providing input.*

Proof. We reuse the notation from the simulation *"Input for corrupt P_j"* from the proof above. Recall that the simulator extracts a set of inputs $\{x_h\}_h$ as $x_h = \frac{\sum_{i \in \mathcal{H}} \rho(\mu(\beta^{(i)}) \star \boldsymbol{x}_h^{(j,i)})}{\beta(\mathcal{H})}$ for $h \in \{0, \ldots, L\}$. What we show now is that, in the real world, if the parties want to open the shared values stored in these identifiers, the adversary can only successfully pass the check if these values are opened to the x_h's extracted by $\mathcal{S}$.[21] The challenge with this lies in that the honest parties keys $\{\beta^{(i)}\}_{i \in \mathcal{H}}$ in the ideal world may entirely differ from the $\{\alpha^{(i)}\}_{i \in \mathcal{H}}$ used in the real world.

Recall that $\mathcal{S}$ obtains from $\mathcal{A}$, for each of the $L+1$ $\mathcal{F}_{\mathsf{COPEe}}$ calls and for each $P_i \in \mathcal{H}$, a vector $\boldsymbol{x}_h^{(j,i)}$ and a value $v_h^{(j,i)} \in R$. First we analyze in detail under which conditions the check performed in the input phase passes. By computing an explicit expression of the $\sigma^{(i)}$ held by honest parties, we can verify that this check passes if and only if the key shares $\{\alpha^{(i)}\}_{i \in \mathcal{H}}$ held by honest parties satisfy

$$\begin{aligned}
\sum_{i \in \mathcal{C}} \sigma^{(i)} &= \sum_{i \in \mathcal{H}} \alpha^{(i)} \cdot y - m^{(i)}(y) \\
&= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} r_h \cdot m^{(i)}(x_h) \right) \\
&= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} r_h \cdot u_h^{(i,j)} \right)
\end{aligned}$$

[21] Note that in general it is likely not the case that the parties want to open these inputs. Instead, they probably want to *compute* on them, and only later open a result. However, our analysis here extends seamlessly to that case.

$$= \sum_{i \in \mathcal{H}} \left(\alpha^{(i)} \cdot y - \sum_{h=0}^{L} \left(r_h \cdot \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)}) - r_h \cdot v_h^{(j,i)} \right) \right).$$

We can write the above equation as

$$\underbrace{\sum_{i \in \mathcal{C}} \sigma^{(i)} - \sum_{i \in \mathcal{H}} \sum_{h=0}^{L} r_h v_h^{(j,i)}}_{=:z} = \alpha^{(\mathcal{H})} \cdot y - \sum_{h=0}^{L} r_h \cdot \left(\sum_{i \in \mathcal{H}} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)}) \right). \quad (2)$$

Notice that z above is a value provided by the adversary, as well as y and the vectors $\boldsymbol{x}_h$. Furthermore, the coefficients r_h are public, so the only unknowns (from the adversary point of view) are the keys $\boldsymbol{\alpha} := (\alpha^{(i)})_{i \in \mathcal{H}}$.

Let us denote by $K_z \subseteq R^{|\mathcal{H}|}$ the set of all $\boldsymbol{\alpha} = (\alpha^{(i)})_{i \in \mathcal{H}}$ satisfying Eq. (2). We first claim that we can assume that $z = 0$. We consider the map

$$F(\boldsymbol{\alpha}) = \alpha^{(\mathcal{H})} \cdot y - \sum_{h=0}^{L} r_h \cdot \left(\sum_{i \in \mathcal{H}} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)}) \right).$$

It is easy to show that F is a $\mathbb{Z}_{p^k}$-module homomorphism from $\mathbb{Z}_{p^k}^{m|\mathcal{H}|}$ to $\mathbb{Z}_{p^k}^{m}$. Then, the kernel space $\ker(F) = K_0$ and image space V_F of this map are $\mathbb{Z}_{p^k}$-module subject to that $V_F \cong \mathbb{Z}_{p^k}^{m|\mathcal{H}|} / \ker F$ by the isomorphism theorem for modules [AM69, Page 19]. We note that K_z is not empty if and only if $z \in V_F$. If $z \neq 0$, the probability that $z \in V_F$ is $\frac{|V_F|}{|R|}$ and the conditional probability that $\boldsymbol{\alpha} \in K_z$ is $\frac{|K_z|}{|R^{|\mathcal{H}|}|}$. $|K_z| = |K_0| = |\ker(F)|$ implies that if $z \neq 0$, $z = F(\boldsymbol{\alpha})$ with probability at most $\frac{1}{|R|} = p^{-km}$.

In what follows we assume that the check performed in the input phase passes, which means that Eq. (2) (with $z = 0$) holds, or equivalently, that the keys $\{\alpha^{(i)}\}_{i \in \mathcal{H}} \in K_0$. Now, imagine that at a later point in the actual circuit computation, a value $\langle x_h \rangle$ is intended to be opened (here x_h does not stand for a specific value yet—since the adversary did not input any concrete value—but instead it acts as an identifier for the sharings corresponding to the h-th input of P_j). The adversary can cause the partial opening to be a value x_h, and in the final check each party announces $\sigma_h^{(i)}$. As before, the check passes if and only if $\sum_{i=1}^{n} \sigma_h^{(i)} = 0$. We can compute what the MAC shares of $\langle x_h \rangle$ held by the honest parties are based on the input phase, and conclude, via a similar derivation to the one made above, that the check passes if and only if $z_h = \alpha^{(\mathcal{H})} \cdot x_h - \sum_{i \in \mathcal{H}} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)})$, where z_h is a value chosen by the adversary.

As before, we can show that $z_h = 0$ for each h. Otherwise, they must satisfy the equality $\sum_{h=0}^{L} r_h z_h = 0$ with random elements r_h. This holds with probability at most p^{-m} with some nonzero z_h. Therefore, x_h must be equal to $(\sum_{i \in \mathcal{H}} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)}))/\alpha^{(\mathcal{H})}$, which is essentially the value extracted by $\mathcal{S}$, except that $\mathcal{S}$ uses the virtual parties' keys $\{\beta^{(i)}\}_{i \in \mathcal{H}}$. In what follows we show that any choice in

K_0 leads to the same set of values $\{x_h\}_{h=0}^{L+1}$, which shows that the values that $\mathcal{S}$ provides to the ideal functionality $\mathcal{F}_{\mathsf{Auth}}$ are the only values the adversary can open these sharings to, and hence, these are the values computed over in the real world, which leads to the desired indistinguishability of the **Check** phase.

It is clear that $K_0 \subseteq R^{|\mathcal{H}|} \subseteq \mathbb{Z}_{p^k}^{m|\mathcal{H}|}$. Since K_0 is a $\mathbb{Z}_{p^k}$-module, by the fundamental decomposition theorem [AM69, Page 20], $K_0 = \sum_{i=1}^{a} S_i \boldsymbol{\beta}_i$ where $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_a \subseteq \mathbb{Z}_{p^k}$ are subrings of $\mathbb{Z}_{p^k}$ and $a \leq m|\mathcal{H}|$. Then, we can write $K_0 = \mathsf{span}_{\mathbb{Z}_{p^k}}\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_a\}$, and we can refer to the elements $\boldsymbol{\beta}_i$ as a basis of K_0.

Define the function

$$f_h(\boldsymbol{\alpha}) = \frac{\sum_{i\in\mathcal{H}} \sum_{h=0}^{L} \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}_h^{(j,i)})}{\sum_{i\in\mathcal{H}} \alpha^{(i)}},$$

where $\boldsymbol{\alpha} = (\alpha^{(i)})_{i\in\mathcal{H}}$. It is clear that for $\boldsymbol{\alpha} \in K_0$, we have $y' = \sum_{h=0}^{L} r_h f_h(\boldsymbol{\alpha})$. Recall that $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_a$ is $\mathbb{Z}_{p^k}$-linear basis of K_0 with $a \leq m|\mathcal{H}|$. For any pair $(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$, it holds that

$$\sum_{h=0}^{L} r_h(f_h(\boldsymbol{\beta}_i) - f_h(\boldsymbol{\beta}_j)) = 0. \tag{3}$$

Since r_h is uniformly random element in R, with probability at least $1-p^{-m}$, r_h is invertible in R. If r_h is invertible, then the Eq (3) holds with probability at most p^{-m} under the condition $f_h(\boldsymbol{\beta}_i) \neq f_h(\boldsymbol{\beta}_j)$. Since there are at most $(m|\mathcal{H}|)^2$ pairs, the probability that there exist at least one pair of $(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$ with $f_h(\boldsymbol{\beta}_i) \neq f_h(\boldsymbol{\beta}_j)$ to satisfy Eq (3) is at most $a^2 p^{-m}$. We note that if $f_h(\boldsymbol{\beta}_i)$ is a constant for $i = 1, \ldots, a$, then $f_h(\boldsymbol{\alpha}) = c$ is a constant for any $\boldsymbol{\alpha} \in K_0$. This follows from

$$f_h(\sum_{j=1}^{a} \lambda_j \boldsymbol{\beta}_j) = \frac{\sum_{j=1}^{a} \lambda_j \sum_{i\in\mathcal{H}} \sum_{h=0}^{L} \rho(\mu(\beta_j^{(i)}) \star \boldsymbol{x}_h^{(j,i)})}{\sum_{j=1}^{a} \lambda_j \sum_{i\in\mathcal{H}} \beta_j^{(i)}} = c$$

for any $\lambda_1, \ldots, \lambda_a \in \mathbb{Z}_{p^k}$ due to the observation

$$\frac{a_1}{b_1} = \frac{a_2}{b_2} = \cdots = \frac{a_r}{b_r} = c \quad \text{implies} \quad \frac{\sum_{i=1}^{r} a_i}{\sum_{i=1}^{r} b_i} = c.$$

Take a union bound over h, the probability that $f_h(\boldsymbol{\alpha})$ is a constant for $h = 0, \ldots, L$ is at least $1 - a^2(L+1)p^{-m} \geq 1 - (m|\mathcal{H}|)^2(L+1)p^{-m}$. If $f_h(\boldsymbol{\alpha})$ is a constant, then x_h is unique. This means, the adversary can only open the value to x_h for every $\boldsymbol{\alpha} \in K_0$. This concludes the proof □

E Bug in MASCOT

Our $\mathcal{F}_{\mathsf{COPEe}}$ functionality is closely related to the functionality with the same name in [KOS16]. Indeed, both of them take care of "turning products into sums",

and in both of them the adversary can introduce certain inconsistencies. The core difference lies in the concrete errors the adversary can add.

Recall that the $\mathcal{F}_{\mathsf{COPEe}}$ functionality involves two parties P_i and P_j with inputs $\alpha^{(i)} \in R$ and $x \in R$ respectively, and distributes $u^{(i,j)}$ and $v^{(j,i)}$ to P_i and P_j, respectively, where $u^{(i,j)} + v^{(j,i)} = \alpha^{(i)} \cdot x$. When P_j is corrupt, these values may add up instead to $u^{(i,j)} + v^{(j,i)} = \alpha^{(i)} \cdot \rho(\mu(\alpha^{(i)}) \star \boldsymbol{x}^{(j,i)})$, where $\boldsymbol{x}^{(j,i)} \in \mathbb{Z}_{p^k}^t$ is a vector of P_j's choice.

In MASCOT, the situation is similar. Our notation is different to the one they use, but trying to make it as comparable to us as possible, the expression they get looks like $u^{(i,j)} + v^{(j,i)} = \langle \boldsymbol{g} \star \boldsymbol{x}^{(j,i)}, \boldsymbol{\alpha}^{(i)} \rangle$, where $\boldsymbol{g}$ and $\boldsymbol{\alpha}^{(i)}$ are vectors such that $\langle \boldsymbol{g}, \boldsymbol{\alpha}^{(i)} \rangle = \alpha^{(i)}$. This way, if all the entries in $\boldsymbol{x}^{(j,i)}$ are equal to some x, then $u^{(i,j)} + v^{(j,i)} = \alpha^{(i)} \cdot x$, but in general the adversary may deviate from this. This is similar to the restriction we have in our case, where we require $\boldsymbol{x}^{(j,i)}$ to be in the image of μ.

Our proof of Theorem 5, provided in full in Section D, draws inspiration from the proof of Theorem 6 in [KOS16, Appendix B]. Both settings are similar as they deal with showing that the adversary, in spite of adding these errors in the Input command in Π_{Auth}, is committed to a unique value that the simulator can extract. The concrete bug with the proof in [KOS16] lies in proving the uniqueness of these extracted values.

The issue starts in Eq.(6) in [KOS16], which adapted to out notation looks like

$$\sum_{i \in \mathcal{H}} \left(\left\langle \boldsymbol{g} \cdot y + \boldsymbol{g} \star \sum_{j \in \mathcal{C}} \sum_{h=0}^{L} r_h \cdot \boldsymbol{x}_h^{(j,i)}, \boldsymbol{\alpha}^{(i)} \right\rangle \right).$$

This is analogous to what we obtain in Eq. (1). We see that in [KOS16], the coefficients r_h are placed *inside* the dot product. However, the correct expression involves placing the r_h's *outside*, just like in our case, where these coefficients appear in front of the ρ function, and not inside.

This, although it may seem minor, has serious consequences for the rest of the proof in [KOS16], more concretely with the proof of uniqueness of the inputs extracted by the simulator. The most concrete problem lies in that the derivation from the first equation at the top of page 34 in [KOS16] (eprint version), to the equation immediately below, cannot be carried out. This invalidates the rest of the analysis.

Remark 3. We also found multiple typographic errors in the proof of [KOS16], together with other minor technical bugs (e.g. the sign in the expression we wrote above should be a minus), which affected the proof. We believe our proof from Section D can be easily adapted to fit the setting in [KOS16], hence providing a clearer and correct proof of their theorem, which also affects the corresponding result in [CG20].

F Galois Rings

Let R be the Galois ring $\mathsf{GR}(p^k, d)$. Elements of R can be seen as polynomials over $\mathbb{Z}_{p^k}$ of degree at most $d-1$. However, the alternative representation in the following theorem is extremely useful.

Theorem 11 (Theorem 14.8 in [Wan03]). *There exists $\xi \in \mathsf{GR}(p^k, d) = \mathbb{Z}_{p^k}/(f(\mathtt{X}))$ of order $p^d - 1$ with $f(\xi) = 0$, such that every element in $a \in \mathsf{GR}(p^k, d)$ can be uniquely written as $a = a_0 + a_1 p + \cdots + a_{k-1}p^{k-1}$, where $a_i \in \{0, 1, \xi, \ldots, \xi^{p^d-2}\}$. Furthermore, $a \in \mathsf{GR}(p^k, d)^*$ if and only if $a_0 \neq 0$.*

In a similar way as taking modulo p of an element in $\mathbb{Z}_{p^k}$ leads to an element in $\mathbb{Z}_p$, we can take modulo p of an element $r \in \mathsf{GR}(p^k, d)$ to obtain an element in $\mathsf{GF}(p^d)$. We denote this by $\overline{r}$. From Theorem 11, we see that $r \in \mathsf{GR}(p^k, d)^*$ if and only if $\overline{r} \neq 0$.

The following lemma provides an upper bound on the number of roots for a nonzero polynomial over $\mathsf{GR}(p^k, d)$.

Lemma 1. *A nonzero degree-r polynomial over $\mathsf{GR}(p^k, d)$ has at most $rp^{(k-1)d}$ roots.*

Proof. We first note that any element α in $\mathsf{GR}(p^k, d)$ can be written as $\alpha = \sum_{i=0}^{k-1} \alpha_i p^i$ with $\alpha_i \in \mathbb{F}_{p^d}$. Let $f(x) = \sum_{i=0}^{r} a_i x^i$. Let $p^t = \gcd(a_1, \ldots, a_r)$. We divide both side by p^t and let $g(x) = \sum_{i=0}^{r} \frac{a_i}{p^t} x^i$. If $f(\alpha) = 0$, then $g(\alpha) = 0 \mod p$. There are at most r roots $\alpha_0 \in \mathbb{F}_{p^d}$ for polynomial $g(\alpha_0) = 0 \mod p$. Therefore, there are at most $rp^{(k-1)d}$ roots satisfying $f(\alpha) = 0$. □

An immediate consequence of this lemma is that for any nonzero degree-r polynomial $f(x)$ over $\mathsf{GR}(p^k, d)$, with probability at most rp^{-d}, a random element $\alpha \in \mathsf{GR}(p^k, d)$ will satisfy $f(\alpha) = 0$.