# Efficient Pseudorandom Correlation Generators from Ring-LPN

Elette Boyle[1(✉)], Geoffroy Couteau[2(✉)], Niv Gilboa[3(✉)], Yuval Ishai[4(✉)], Lisa Kohl[4(✉)], and Peter Scholl[5(✉)]

[1] IDC Herzliya, Herzliya, Israel
eboyle@alum.mit.edu
[2] IRIF, Paris, France
couteau@irif.fr
[3] Ben-Gurion University of the Negev, Beersheba, Israel
gilboan@bgu.ac.il
[4] Technion, Haifa, Israel
{yuvali,lisa.kohl}@cs.technion.ac.il
[5] Aarhus University, Aarhus, Denmark
peter.scholl@cs.au.dk

**Abstract.** Secure multiparty computation can often utilize a trusted source of correlated randomness to achieve better efficiency. A recent line of work, initiated by Boyle et al. (CCS 2018, Crypto 2019), showed how useful forms of correlated randomness can be generated using a cheap, one-time interaction, followed by only "silent" local computation. This is achieved via a *pseudorandom correlation generator* (PCG), a deterministic function that stretches short correlated seeds into long instances of a target correlation. Previous works constructed concretely efficient PCGs for simple but useful correlations, including random oblivious transfer and vector-OLE, together with efficient protocols to distribute the PCG seed generation. Most of these constructions were based on variants of the Learning Parity with Noise (LPN) assumption. PCGs for other useful correlations had poor asymptotic and concrete efficiency.

In this work, we design a new class of efficient PCGs based on different flavors of the *ring-LPN* assumption. Our new PCGs can generate OLE correlations, authenticated multiplication triples, matrix product correlations, and other types of useful correlations over large fields. These PCGs are more efficient by orders of magnitude than the previous constructions and can be used to improve the preprocessing phase of many existing MPC protocols.

## 1 Introduction

Correlated secret randomness is a commonly used resource for secure multiparty computation (MPC) protocols. Indeed, simple kinds of correlations enable lightweight MPC protocols even when there is no honest majority. For instance, an *oblivious transfer* (OT) correlation supports MPC for Boolean


D. Micciancio and T. Ristenpart (Eds.): CRYPTO 2020, LNCS 12171, pp. 387–416, 2020.
https://doi.org/10.1007/978-3-030-56880-1_14

circuits [29,35,40,46], while oblivious linear-function evaluation[1] (OLE), an arithmetic variant of OT, supports MPC for arithmetic circuits [36,45]. Other useful types of correlations include multiplication triples [4] and truth-table correlations [21,23,33]. Finally, *authenticated* multiplication triples serve as a powerful resource for achieving security against malicious parties [6,25].

A common paradigm in modern MPC protocols is to utilize the above kinds of correlations in the following way. In a preprocessing phase, before the inputs are known, the parties use an *offline protocol* to generate many instances of the correlation. These instances are then consumed by an *online protocol* to securely compute a function of the secret inputs. This approach is appealing because of the high efficiency of the online protocol. Indeed, with the above simple correlations, the online communication and computation costs are comparable to the size of the circuit being evaluated. The price one pays for the fast online protocol is a much slower and higher-bandwidth offline protocol. Even simple types of correlated randomness are expensive to generate in a secure way. This high cost becomes even higher when aiming for security against malicious parties.

Recently, a promising approach for instantiating the preprocessing phase of MPC protocols was suggested in [9,11], relying on a new primitive called a *pseudorandom correlation generator* (PCG). Consider a target two-party correlation $\mathcal{C}$, typically consisting of many independent instances of a simple correlation as above. A PCG for $\mathcal{C}$ consists of two algorithms: $\mathsf{Gen}(1^\lambda)$, which given a security parameter $\lambda$ generates a pair of short, correlated seeds $(\mathsf{k}_0, \mathsf{k}_1)$, and $\mathsf{Expand}(\mathsf{k}_\sigma)$, which deterministically stretches a seed $\mathsf{k}_\sigma$ to a long output $R_\sigma$. The intuitive security requirement is that the joint outputs $(R_0, R_1)$ of the above process cannot be distinguished from $\mathcal{C}$ not only by an outsider, but also by an insider who learns one of the two seeds. PCGs naturally lead to protocols with an appealing *silent preprocessing* feature, by breaking the offline phase into two parts:

1. Setup. The parties run a secure protocol to distribute the seed generation of $\mathsf{Gen}$. Since $\mathsf{Gen}$ has low computational cost and short outputs, this protocol only involves a small amount of communication, much smaller than the output of $\mathcal{C}$. Each party stores its own short seed $\mathsf{k}_\sigma$ for later use.
2. Silent expansion. Shortly before the online phase, the parties use $\mathsf{Expand}$ to generate long pseudorandom correlated strings $(R_0, R_1)$ to be used by the online protocol. This part is referred to as *silent*, since it involves no communication.

Beyond the potential improvement in the total offline communication and computation, this blueprint has two additional advantages. First, it can substantially reduce the *storage* cost of correlated randomness by enabling efficient compression. Indeed, the parties can afford to generate and store many correlated seeds, possibly with different sets of parties, and expand them just before they are needed. Second, the cost of protecting the offline protocol against malicious parties is "amortized away," since it is only the small setup part that needs to be protected. A malicious execution of $\mathsf{Expand}$ is harmless.

[1] An OLE correlation over a finite field $\mathbb{F}$ is a two-party correlation $(r_0, r_1)$ where $r_0 = (a, b)$ is uniform over $\mathbb{F}^2$ and $r_1 = (x, ax + b)$ for $x \in_R \mathbb{F}$.

The work of Boyle et al. [11] constructed efficient PCGs for several kinds of useful correlations based on different assumptions that include variants of Learning Parity with Noise (LPN) [8] and Learning With Errors (LWE) [47]. While the LPN-based PCG for OT from [11] has very good concrete efficiency, making it a practically appealing approach for generating many OTs [10], this is not the case for other useful correlations such as OLE or authenticated multiplication triples. For these correlations, two different constructions were proposed in [10]. Both are "practically feasible" but quite inefficient. In the first construction, based on homomorphic secret sharing from ring-LWE [12,14,16,18], the seed expansion can be at most quadratic due to the use of a pseudorandom generator with algebraic degree 2. In concrete terms, the seeds are several GBs long and can only be expanded by around 6x, giving far too much overhead for most applications. Their second construction is based directly on LPN, and has computational cost of at least $\Omega(N^2)$ for output length $N$, which is impractical for large $N$.

### 1.1 Our Contributions

In this work, we present efficient new PCG constructions for several widely used correlations for which previous techniques had poor concrete efficiency.

**Silent OLE and silent triple generation.** Our main construction gives the first concretely efficient PCG for OLE over big finite fields $\mathbb{F}$. This PCG is based on a variant of the *ring-LPN* assumption over $\mathbb{F}$, makes a black-box use of $\mathbb{F}$, and has $\mathsf{poly}(\lambda) \cdot \log N$ seed size and $\mathsf{poly}(\lambda) \cdot \tilde{O}(N)$ computational cost for expanding the seeds into $N$ instances of OLE. This PCG gives both an asymptotic and concrete improvement over the LPN-based construction from [11].

We also show how to modify our PCG for OLE to produce multiplication triples and authenticated triples, used in maliciously secure MPC protocols like SPDZ [25]. This incurs an extra overhead of only around a factor of two in seed size, seed generation, and silent expansion time. Finally, we extend the main construction to other types of useful correlations, including matrix products and circuit-dependent correlations.

Technically, one of our main innovations here is showing how to avoid the $\Omega(N^2)$ blowup from the previous LPN-based PCG for OLE from [11]. Our method of doing this requires switching from unstructured LPN to ring-LPN over a certain kind of polynomial rings. This is analogous to early fully homomorphic encryption schemes, where switching from a construction based on LWE [19] to one based on ring-LWE [20] reduced the ciphertext expansion in multiplication from quadratic to linear. A key difference between LWE-based constructions and LPN-based constructions over a big field $\mathbb{F}$ is the noise distribution: Gaussian in the former and low Hamming weight noise in the latter. With LPN-style noise distribution more care is needed, and some natural PCG candidates based on Reed-Solomon codes can be broken using algebraic decoding techniques.

*Concrete efficiency.* Our PCGs have attractive concrete efficiency features. To give a couple of data points, in the case of OLE the parties can store a pair of

seeds of size 1.25 MB each, and expand them on demand to produce over a million OLEs (of size 32 MB, 26x larger than the seeds) in $\mathbb{Z}_q$, where $q$ is the product of two 62-bit primes,[2] with 128-bit security. When running on a single core of a modern laptop, we estimate this takes under 10 s, resulting in a throughput of over 100 thousand OLEs per second. To produce authenticated triples instead of OLE, the expansion cost roughly doubles, giving 50 thousand triples per second, while the seed size increases to 2.6 MB. For comparison, estimates from [11] for their PCG for producing authenticated triples gave a throughput of up to 7 thousand per second, but this was only possible when generating an enormous batch of 17 GB worth of triples, with 3 GB seeds. See below for comparison with non-silent correlation generation techniques.

**Efficient setup protocols.** Recall that to avoid a trusted setup, one typically needs a setup protocol to securely distribute the PCG seed generation. We present concretely efficient setup protocols for OLE and authenticated triples, with both semi-honest and malicious security. The protocols make black-box use of lightweight cryptographic primitives, as well as of generic MPC protocols for performing binary and arithmetic computations on secret-shared values.

In practice, our PCGs and setup protocols can be used in a *bootstrapping mode*, where a portion of the PCG outputs are reserved to be used as correlated randomness for the setup procedure of the next PCG seeds. This means that the vast majority of the setup cost is amortized away over multiple instances. Concretely, we estimate that when bootstrapped in this way, the setup phase for a PCG of one million authenticated triples requires only around 4.2 MB of communication per party, to produce 32 MB worth of triples. The initial setup protocol for the first PCG (before bootstrapping can take place) requires around 25000 authenticated triples, plus some additional correlated randomness (OT and VOLE). This should be feasible to produce in under a minute (although with high communication cost) using standard protocols such as MASCOT [38] or Overdrive [39], and previous PCG protocols for OT and VOLE with malicious security [10].

Compared with non-silent secure correlation generation protocols, we expect the overall *computational* cost of our approach to be comparable with state-of-the-art protocols based on homomorphic encryption [31,37,39], but with much lower *communication* costs. For instance, in the case of authenticated multiplication triples, the Overdrive protocol [39] can produce around 30 thousand triples per second with malicious security. This is similar to our PCG expansion phase (modulo different hardware, environment, and so on), with the significant difference that Overdrive requires almost 2 GB of communication to produce the triples. In comparison, our amortized 4.2 MB communication complexity is over two orders of magnitude smaller, with the additional benefit that our short correlated seeds can be easily stored for on-demand silent expansion.

[2] Our construction works in any sufficiently large finite field, or modulus that is a product of primes via the CRT. We estimated costs with a product of two primes due to better software support.

**Extension to other correlations and multiple parties.** Beyond multiplication triples, it can be useful to have more general "degree-two" correlations, such as inner-product triples, matrix-multiplication triples, or circuit-dependent multiplication triples [5,17,21]. We use our PCG for OLE to obtain PCGs for these kinds of correlations, by exploiting a special "programmability" feature that enables reusing the same PCG output in multiple instances [11]. This gives us a way to produce many independent instances of any degree-two correlation (a vast generalization of OLE and multiplication triples), with seed size that grows sublinearly with the total number of instances. Useful special cases include the types of correlations mentioned above. This construction has a bigger overhead than our PCG for OLE, and in practice seems mainly suited for small correlations such as low-dimensional matrix products. However, these can still be useful in larger computations which involve a lot of linear algebra or other repeated sub-computations.

We can also use same programmability feature of our 2-party PCGs to extend them to the *multi-party setting*. This yields practical multi-party PCGs for multiplication triples that enable an online passively-secure MPC protocol for arithmetic circuits whose cost scales linearly (rather than quadratically) with the number of parties. This transformation to the multi-party case, which originates from [11], does not scale well to correlations with degree higher than 2. As a result, we do not get a multi-party PCG for authenticated triples (a degree-three correlation) with the same level of efficiency.

**Security of ring-LPN.** Our constructions rely on variants of the ring-LPN assumption [32] over non-binary fields. Binary ring-LPN is a fairly standard assumption that withstood a significant amount of cryptanalysis. However, since we also use relatively unexplored variants over different rings, we give a thorough survey of known attacks, and analyze the best strategies that apply to our setting. We find that there are only one or two additional attack possibilities from the additional structure we introduce, and these are easily countered with a small increase in the number of errors.

More precisely, settling for a PCG that generates a single OLE instance over a large ring of degree-$N$ polynomials, our construction can be based on a conservative variant of ring-LPN where the modulus is irreducible. A big ring-OLE correlation can then be converted into $N$ independent instances of standard OLE by communicating $O(N)$ field elements. For generating *silent* OLE over $\mathbb{F}_p$, we instead rely on a variant of ring-LPN where the modulus splits completely into $N$ linear factors. In practice, this requires using larger parameters and increases the cost of our protocols by around a factor of two, compared with irreducible ring-LPN.

### 1.2 Technical Overview

**Construction from [11].** Before describing our PCG for OLE, it is instructive to recall the PCG for general degree-two correlations by Boyle et al. [11], based

on LPN. The goal is to build a PCG for the correlation which gives each party a random vector $\mathbf{x}_i$, together with an additive secret share of the tensor product $\mathbf{x}_0 \otimes \mathbf{x}_1$. They used the dual form of LPN over a ring $\mathbb{Z}_p$, which states that the distribution

$$\left\{ H, H \cdot \mathbf{e} \,\middle|\, H \xleftarrow{\$} \mathbb{Z}_p^{m \times n}, \mathbf{e} \xleftarrow{\$} \mathbb{Z}_p^n \text{ s.t. } \mathrm{wt}(\mathbf{e}) = t \right\}$$

is computationally indistinguishable from uniform, where $\mathbf{e}$ is a sparse random vector with only $t$ non-zero coordinates, for some $t \ll n$, and $m < n$.

The idea of the construction is that the setup algorithm gives each party a random sparse $\mathbf{e}_0$ or $\mathbf{e}_1$, and computes the tensor product $\mathbf{e}_0 \otimes \mathbf{e}_1$, which has at most $t^2$ non-zero coordinates. This product is then distributed to the parties via function secret sharing (FSS), by generating a pair of FSS keys for the function that outputs each entry of the product on its respective inputs from 1 to $n^2$. This function can be written as a sum of $t^2$ point functions, allowing practical FSS schemes based on distributed point functions [13,15,28]. Note that unlike the case of PCGs for OT or Vector-OLE [10,48], here we cannot replace FSS by the simpler punctured PRF primitive.

Given shares of $\mathbf{e}_0 \otimes \mathbf{e}_1$ and either $\mathbf{e}_0$ or $\mathbf{e}_1$, the parties expand these using LPN, computing:

$$\mathbf{x}_0 = H \cdot \mathbf{e}_0, \quad \mathbf{x}_1 = H \cdot \mathbf{e}_1, \quad \mathbf{z} = (H \cdot \mathbf{e}_0) \otimes (H \cdot \mathbf{e}_1) = (H \otimes H) \cdot (\mathbf{e}_0 \otimes \mathbf{e}_1)$$

where $\mathbf{x}_i$ is computed by party $P_i$, while $\mathbf{z}$ is computed in secret-shared form, using the shares of $\mathbf{e}_0 \otimes \mathbf{e}_1$ and the formula on the right-hand side.

Notice that both $\mathbf{x}_0$ and $\mathbf{x}_1$ are pseudorandom under LPN, which gives the desired correlation.

*Optimizations and additional applications.* Boyle et al. state the computational complexity of the above as $O(n^4)$ operations, due to the tensor product of $H$ with itself. We observe that the value of $(H \cdot \mathbf{e}_0) \otimes (H \cdot \mathbf{e}_1)$ can be read directly from $H \cdot (\mathbf{e}_0 \cdot \mathbf{e}_1^\intercal) \cdot H^\intercal$, which requires much less computation and can be made even more efficient if $H$ is a structured matrix, reducing the computational complexity to $\tilde{O}(n^2)$. We also describe two variants of the PCG which allow producing large matrix multiplication correlations with different parameter tradeoffs. As these are much less practical than our main constructions, we refer the interested reader to the full version for details.

**An efficient PCG for OLE.** The problem with the above construction is that it produces an entire tensor product correlation, which inherently requires $\Omega(n^2)$ computation. Even if we only want to compute the diagonal entries of the tensor product output (that is, $n$ OLEs), we do not see a way to do this any more efficiently.

Instead, we propose to replace the tensor product with a *polynomial product*. Let $R_p = \mathbb{Z}_p[X]/(F(X))$ for some degree $N$ polynomial $F(X)$, and let $e, f$ be two sparse polynomials in $R_p$. For a random polynomial $a \in R_p$, the pair

$$(a, a \cdot e + f \mod F(X))$$

is pseudorandom under the *ring-LPN* assumption [32].

Now, given two pairs of sparse polynomials $(e_0, e_1)$ and $(f_0, f_1)$, each product $e_i \cdot f_j$ (without reduction modulo $F$) has degree $< 2N$ and only $t^2$ non-zero coefficients. These can again be distributed to two parties using FSS, but this time the expanded FSS outputs can be computed in *linear time* in $N$, instead of quadratic, since the domain size of the function being shared is only $2N$.

Given shares of $e_i \cdot f_j$, similarly to the LPN case, the parties compute expanded outputs by defining

$$x_0 = a \cdot e_0 + f_0, \quad x_1 = a \cdot e_1 + f_1, \quad z = ((1, a) \otimes (1, a)) \cdot ((e_0, f_0) \otimes (e_1, f_1))$$

The main difference here is that each tensor product is only of length 2, and can be computed in $\tilde{O}(N)$ time using fast polynomial multiplication algorithms.

This gives a PCG that compresses a single OLE over the ring $R_p$. To obtain a PCG for OLE over $\mathbb{Z}_p$, we again take inspiration from the fully homomorphic encryption literature, by using ciphertext-packing techniques [49]. We can carefully choose $p$ and $F(X)$ such that $F(X)$ splits into $N$ distinct, linear factors modulo $p$. Then $R_p$ is isomorphic to $N$ copies of $\mathbb{Z}_p$, and we can immediately convert a random OLE over $R_p$ into $N$ random OLEs over $\mathbb{Z}_p$. This works particularly well with cyclotomic rings as used in ring-LWE [43], where we can e.g. use $N$ a power of two and easily exploit FFTs for polynomial arithmetic.

**Extending to authenticated multiplication triples.** We show that our construction extends from OLE to authenticated multiplication triples, as used in the SPDZ protocol for maliciously secure MPC [22,25]. This follows from a simple trick, where we modify the FSS scheme to additionally multiply its outputs by a random MAC key $\alpha \in \mathbb{Z}_p$. Since this preserves sparsity of the underlying shared vector, it adds only at most a factor of two overhead on top of the basic scheme.

**Distributed setup.** We focus on the case of OLE correlations over $R_p$ (the setup for authenticated triples is very similar). Recall that the seed of the PCG for OLE consists of $t$-sparse degree-$N$ "error" polynomials $e_0, e_1$ and $f_0, f_1$, and FSS keys for secret-shares of the products $e_i \cdot f_j$, each represented as a coefficient vector via the sum of $t^2$ point functions $f_{\alpha,\beta} \colon [2N] \to \mathbb{Z}_p$. Each point function corresponds to a single monomial product from $e_i$ and $f_j$. The index $\alpha \in [2N]$ of the nonzero position is the *sum* of the corresponding nonzero indices, and the payload $\beta \in \mathbb{Z}_p$ is the *product* of the corresponding payloads in $e_i$ and $f_j$.

In the semi-honest setting, secure computation of this PCG generation procedure can be attained directly, using generic 2-PC for simple operations on the $\alpha$ and $\beta$ values, as well as black-box use of a protocol for secure computation of FSS key generation, such as the efficient protocol of Doerner and shelat [27].

For the malicious setting, we would wish to mimic the same protocol structure with underlying 2-PC components replaced with maliciously secure counterparts. The simple 2-PCs on $\alpha, \beta$ can be converted to malicious security with relatively minor overhead. The problem is the FSS key generation, for which efficient maliciously secure protocols currently do not exist. Generic 2-PC of the FSS key generation functionality would require expensive secure evaluations of crytographic pseudorandom generators (PRG). The semi-honest protocol of [27] is black-box in a PRG; but, precisely this fact makes it difficult to ensure consistency between different steps in the face of a malicious party.

Note that this is similar to the problem that Boyle et al. faced in [10] for silent OT generation, but their setting was conceptually simpler: There, one party always knew the position $\alpha$ of the non-zero value of the distributed point function (indeed, for their purpose the simpler building block of a puncturable pseudorandom function sufficed). Further, they did not have to assume any correlation between path values, whereas in our setting we require that the parties behave consistently regarding the path positions *and* payloads across several instances.

In this work we show how to extend the approach of [10] to the context of distributed point functions, further addressing the mentioned issues.

Our protocol realizes a PCG-type functionality for a scaled unit vector[3] with leakage: Given authenticated values for the location of the non-zero position $\alpha \in [0..N)$ and the non-zero payload $\beta \in \mathbb{Z}_p$, the functionality allows a corrupt party to choose its output vector $\mathbf{y} \in \mathbb{Z}_p^N$ and delivers to the honest party the correct corresponding output $\mathbf{y} - (0, \ldots, \beta, \ldots, 0)$, where $\beta$ is in the $\alpha$-th position. The leakage on $\alpha$ can be captured by allowing the adversary a predicate guess on $\alpha$.[4] In the setting of noise generation for (ring-)LPN, as is the case for our PCG constructions (and likely future constructions), such leakage is tolerable as, intuitively, this can be accounted for by slightly increasing the noise rate. Indeed, we prove that this functionality suffices to implement a protocol securely realizing PCG functionalities, such as the corruptible functionality for OLE and authenticated multiplication triples, based on a variant of the ring-LPN assumption that allows small amount of leakage (only 1 bit on average).

**Extensions.** A downside of the above construction, compared with the one from LPN, is that it is restricted to multiplication triples or OLE. It can be useful to obtain other degree-two correlations such as matrix multiplication triples, which allow multiplying two secret matrices with only $O(n^2)$ communication, instead of $O(n^3)$ from naively using individual triples. Another technique is preprocessing multiplications in a way that depends on the structure of the circuit, which allows reducing the online cost of 2-PC down to communicating just one field element

[3] Note that this corresponds to a distributed point function where we do not require the key setup on its own to be secure, but only require the protocol to securely implement the FSS functionality including expansion, as this suffices for using PCGs in the context of secure computation (see also [10]).

[4] In fact, the leakage can be characterized by predicates corresponding to bit-matching with wildcards.

per party, instead of two from multiplication triples [4,5,23]. This type of circuit-dependent preprocessing can also be expressed as a degree two correlation.

Our PCG for OLE satisfies a useful "programmability" feature, introduced by Boyle et al. [11], allowing certain parts of the PCG output to be reused across multiple instances. This is simply due to the fact that we can reuse the polynomials $e_0, e_1$ or $f_0, f_1$ in the PCG, without harming security. This allows us to extend the PCG to build more general correlations, by using multiple programmed instances to perform every multiplication in the general correlation.

We in fact present a more general construction, which, loosely speaking, achieves the following. Given a programmable PCG for some bilinear correlation $g$, let $f$ be another bilinear correlation that is computable using linear combinations of outputs of $g$ applied to its input. Then, we can construct a PCG for $f$ using several copies of the PCG for $g$, where the number of instances is given by the complexity of $f$ written as a function of $g$. This gives a general way of combining PCGs to obtain correlations of increasing complexity, while allowing for different complexity tradeoffs by varying the "base" bilinear correlation $f$.

*Multi-party PCGs.* As discussed earlier, the programmability feature also immediately allows us to extend our PCGs for OLE and degree-two correlations to the multi-party setting, using the construction from [11]. This does not apply to the PCG for authenticated multiplication triples; in the full version, we sketch a possible alternative solution based on three-party distributed point functions, but these are much less efficient than the two-party setting.

**Security analysis of ring-LPN.** We use the ring $R_p = \mathbb{Z}_p[X]/F(X)$, for some degree $N$ polynomial $F(X)$. There are two main ring-LPN variants we consider, depending on how the parameters are instantiated. The more conservative is when $F(X)$ is either *irreducible* in $R_p$ (hence, $R_p$ is isomorphic to a finite field), or at least when $F(X)$ has only very few low-degree factors, so $R_p$ has a large subring that is a field. This type of instantiation is similar to previous recommendations for ring-LPN [30,32] and post-quantum encryption schemes from quasi-cyclic codes [44]. The best known attacks are to solve the underlying syndrome decoding problem, and the additional ring structure does not seem to give much advantage. One exception is when a very large number of samples are available, when the ring structure can in some cases be exploited [7]. This does not apply to our setting, however, since our constructions only rely on ring-LPN with one sample[5].

The second variant, which is needed for silent OLE in $\mathbb{F}_p$, is when $F(X)$ splits modulo $p$ into many distinct factors of low degree. Here, the main attack vector that needs to be considered is that if $f_i$ is some degree-$d$ factor of $F(X)$, then reducing a ring-LPN instance modulo $f_i$ gives a new instance in smaller dimension $d$, albeit with a different noise distribution. The best case for the adversary is when $f_i$ is of the form $X^d + c_i$, when this reduction does not increase the Hamming weight of the noise (although, the corresponding error rate goes

[5] Or, two samples if security is based on ring-LPN with a uniform (not sparse) secret.

up). If such sparse factors exist, then, we must also ensure that the underlying ring-LPN instance in dimension $d$, with new noise weight, is hard to solve.

One way to counter this attack is to choose $F(X)$ to be a product of $N$ random linear factors, ensuring that any factors of $F$ an adversary can find are likely to be very dense. However, to improve computational efficiency, it is better to use a cyclotomic polynomial such as $F(X) = X^N + 1$ with $N$ a power of two, as is common in the ring-LWE setting. In this case, there are many sparse factors of the form $X^{2^i} + c_i$ which can be exploited, and we must take these into account when choosing parameters. The main advantage of performing this reduction is the vector operations in attacks such as information-set decoding become cheaper, since they are all in a smaller dimension. This only has a small overall effect on attack complexity, though, since these algorithms are all exponential in the noise weight. Therefore, to counter the attack, it suffices to ensure there are enough noisy coordinates in a reduced instance, which requires only a small increase in noise weight.

Note that for $p = 2$, this strategy was also considered in Lapin [32], and it was later shown that an optimized version of this over $\mathbb{F}_2$ reduces security of some Lapin parameter sets by $\approx 10$ bits [30]. Our analysis over $\mathbb{F}_p$ is roughly consistent with this.

## 2 Preliminaries

**Notation.** We let $\lambda$ denote a security parameter, and use the standard definitions of negligible functions, computational indistinguishability (with respect to nonuniform distinguishers), and pseudorandom generators. We use $[0..n)$ to denote the index set $\{0, \cdots, n-1\}$, as well as $[0..n] = \{0, \ldots, n\}$ and $[n] = \{1, \ldots, n\}$.

**Vectors, outer sum and outer product.** We use column vectors by default. For two vectors $\mathbf{u} = (u_1, \ldots, u_t), \mathbf{v} = (v_1, \ldots, v_t) \in R^t$, for some ring $R$, we write $\mathbf{u} \boxplus \mathbf{v}$ to mean the *outer sum* given by the length $t^2$ vector $(u_i + v_j)_{i \in [t], j \in [t]}$. Similarly, we define the flattened outer product (or tensor product) to be $\mathbf{u} \otimes \mathbf{v} = (u_i \cdot v_j)_{i \in [t], j \in [t]}$, that is, the vector $(v_1 \cdot \mathbf{u}, \ldots, v_n \cdot \mathbf{u})$. We denote the inner product of two vectors by $\langle \mathbf{u}, \mathbf{v} \rangle$.

**PCG and FSS.** We refer to the full version for formal definitions of a *pseudorandom correlation generator* (PCG), *function secret sharing* (FSS) and the special case of *distributed point function*. Here we will use an FSS scheme *SPFSS* for secret-sharing a *sum of point functions* $f_{S,\mathbf{y}}$ defined as follows. For a sequence of inputs $S = (s_1, \ldots, s_t) \in [n]^t$ and outputs $\mathbf{y} = (y_1, \ldots, y_t)$, the function $f_{S,\mathbf{y}}(x)$ returns the sum of the point functions $f_{s_i,y_i}(x)$, where the latter returns $y_i$ if $x = s_i$ and 0 otherwise. For better readability, when generating keys for a scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}, \mathsf{SPFSS.Eval})$, we write $\mathsf{SPFSS.Gen}(1^\lambda, S, \mathbf{y})$, instead of explicitly writing $f_{S,\mathbf{y}}$. To construct SPFSS for a sum of $t$ point functions, we can

simply take $t$ distributed point functions and sum up their outputs. Alternatively, more efficient constructions with optimized full-domain evaluation can be obtained using (randomized) batch codes [9,48].

## 3 Ring-LPN Assumption

In this section, we recall the ring-LPN assumption, which was first introduced in [32] to build efficient authentication protocols. Since then, it has received some attention from the cryptography community [7,24,30,42], due to its appealing combination of LPN-like structure, compact parameters, and short runtimes. Below, we also provide a definition of module-LPN, which generalizes ring-LPN in the same way that the more well-known module-LWE generalizes ring-LWE.

### 3.1 Ring-LPN

**Definition 1 (Ring-LPN).** *Let $R = \mathbb{Z}_p[X]/(F(X))$ for some prime $p$ and degree-$N$ polynomial $F(X) \in \mathbb{Z}[X]$, and let $m, t \in \mathbb{N}$. Let $\mathcal{HW}_t$ be the distribution over $R_p$ that is obtained via sampling $t$ noise positions $A \leftarrow [0..N)^t$ as well as $t$ payloads $\mathbf{b} \leftarrow \mathbb{Z}_p^t$ uniformly at random, and outputting $e(X) := \sum_{j=0}^{t-1} \mathbf{b}[j] \cdot X^{A[j]}$. The $R$-$\mathsf{LPN}_{p,m,t}$ problem is hard if for any PPT adversary $\mathcal{A}$, it holds that*

$$\left|\Pr[\mathcal{A}((a^{(i)}, a^{(i)} \cdot e + f^{(i)})_{i=1}^m) = 1] - \Pr[\mathcal{A}((a^{(i)}, u^{(i)})_{i=1}^m) = 1]\right| \leq \mathsf{negl}(\lambda)$$

*where the probabilities are taken over $a^{(1)}, \ldots, a^{(m)}, u^{(1)}, \ldots, u^{(m)} \leftarrow R_p$, $e$, $f^{(1)}, \ldots, f^{(m)} \leftarrow \mathcal{HW}_t$ and the randomness of $\mathcal{A}$.*

*Remark 2.* Note that sampling $t$ noise positions individually can lead to collisions, and thus negatively affect the entropy introduced by the payloads. The reason we decided in favor of this definition is that this entropy loss is minor in the regime of parameters we care about—as for $t \ll N$ the probability of collisions is very small—and this choice helps to simplify the analysis.

Note that our restriction to $\mathbb{Z}_p$ with $p$ prime as the underlying field is only for simplicity; in fact, $R$-$\mathsf{LPN}$ can be defined equivalently over any other field, or even over rings (e.g. $\mathbb{Z}_{2^k}$ or $\mathbb{Z}_{pq}$ for primes $p, q$); these alternative choices are not known to introduce any significant weakness or structural difference compared to the version over $\mathbb{Z}_p$.

In this work, we will also use a natural generalization of $R$-$\mathsf{LPN}$, where we replace $a^{(i)} \cdot e$ by the inner product $\langle \mathbf{a}^{(i)}, \mathbf{e} \rangle$ between length-$(c-1)$ vectors over $R$, for some constant $c$. We call this *module-LPN*, analagously to module-LWE. This will allow for more efficient instantiations, as according to our security analysis it will be enough to choose the total number of noise positions $w = c \cdot t = O(\lambda)$, and therefore increasing $c$ allows to choose a smaller $t$.

**Definition 3 (Module-LPN).** *Let $R = \mathbb{Z}_p[X]/(F(X))$ for some prime $p$ and degree-$N$ polynomial $F(X) \in \mathbb{Z}[X]$, and let $c, m, t \in \mathbb{N}$ with $c \geq 2$. Let $\mathcal{HW}_t$ be the distribution of uniformly random polynomials in $R_p$ with exactly $t$ non-zero coefficients. The $R^c$-$\mathsf{LPN}_{p,m,t}$ problem is hard if for any PPT adversary $\mathcal{A}$, it holds that*

$$\left|\Pr[\mathcal{A}((\mathbf{a}^{(i)}, \langle \mathbf{a}^{(i)}, \mathbf{e}\rangle + f^{(i)})_{i=1}^m) = 1] - \Pr[\mathcal{A}((\mathbf{a}^{(i)}, u^{(i)})_{i=1}^m) = 1]\right| \leq \mathsf{negl}(\lambda)$$

*where the probabilities are taken over $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)} \leftarrow R_p^{c-1}$, $u^{(1)}, \cdots, u^{(m)} \leftarrow R_p$ $\mathbf{e} \leftarrow \mathcal{HW}_t^{c-1}$, $f^{(1)}, \ldots, f^{(m)} \leftarrow \mathcal{HW}_t$, and the randomness of $\mathcal{A}$.*

**Equivalence to module-LPN with uniform secret.** We observe that, by the same argument as for standard LWE [2], the $R$-$\mathsf{LPN}$ (resp. module-LPN) problem with a secret chosen from the error distribution is at least as hard as the corresponding $R$-$\mathsf{LPN}$ (resp. module-LPN) problem where the secret is chosen uniformly at random, if the adversary is given one additional sample. For a proof we refer to the full version.

**Lemma 4.** *For any $c \geq 2$, let $R^c$-$\mathsf{uLPN}_{p,m,t}$ denote the variant of $R^c$-$\mathsf{LPN}$ where the secret $e$ is sampled uniformly at random. Then, for $m \geq c$, $R^c$-$\mathsf{uLPN}_{p,m,t}$ is at least as hard as $R^c$-$\mathsf{LPN}_{p,m-c,t}$.*

**Relation to syndrome decoding.** In our constructions, we typically consider module-LPN with a single sample ($m = 1$). To simplify notation and emphasize that the secret comes from the error distribution, we often combine the two into a single vector, writing

$$\left\{(\mathbf{a}, \langle \mathbf{a}, \mathbf{e}\rangle) \,\middle|\, \mathbf{a} = (1, \mathbf{a}'), \mathbf{a}' \xleftarrow{\$} R_p^{c-1}, \mathbf{e} \xleftarrow{\$} \mathcal{HW}_t^c\right\}.$$

This formulation of module-LPN is equivalent to a variant of the syndrome decoding problem in random polynomial codes. To see this, let $M_i$ be the $N \times N$ matrix over $\mathbb{Z}_p$ representing multiplication with the fixed element $a'_i \in R_p$, for $i = 1$ to $c - 1$. Define the matrix

$$H = [\mathsf{Id}_N || M_1 || \cdots || M_{c-1}].$$

$H$ is a parity-check matrix in systematic form for a polynomial code defined by the random elements $a'_i \in R_p$. Module-LPN can be seen as a decisional version of syndrome decoding for this code, where we assume that $H\mathbf{e}$ is pseudorandom for an error vector $\mathbf{e} = (e_1, \ldots, e_c)$ with a regular structure, where each of the length-$N$ blocks $e_i$ have exactly $t$ non-zero entries. The code has length $N \cdot c$ and dimension $N \cdot (c - 1)$; the rate of the code is therefore $(c - 1)/c$. With this formulation, $c$ can be viewed as the compression factor of the linear map $\mathbf{e} \to H \cdot \mathbf{e}$. Therefore, we generally refer to $c$ as the *syndrome compression factor*.

### 3.2 Choice of the Polynomial $F$

The ring-LPN assumption (and more generally, the module-LPN assumption) is dependent of the choice of the underlying polynomial $F$. We discuss possible choices for the polynomial $F$, and their implications for the security of ring-LPN/module-LPN over the corresponding ring $R$.

**Irreducible $F(X)$.** The most conservative instantiation is when $F(X)$ is irreducible over $\mathbb{Z}_p$, and so $R_p$ is a field. In this setting, no attacks are known that perform significantly better than for standard LPN.

**Reducible $F(X)$.** We also consider when $F(X)$ is reducible over $\mathbb{Z}_p$, and splits into several distinct factors. Here we have a few different instantiations.

*Cyclotomic $F(X)$.* Let $F(X)$ be the $m$-th cyclotomic polynomial of degree $N = \phi(m)$. Then, $F(X)$ splits modulo $p$ into $N/d$ distinct factors $f_i$, where each $f_i$ is of degree $d$, and $d$ is the smallest integer satisfying $p^d = 1 \bmod m$. We are particularly interested in the following cases.

- *Two-power $N$.* Let $N$ be a power of two and $p$ a large prime such that $p = 1 \bmod (2N)$ (here, $m = 2N$). Then $F(X)$ splits completely into $N$ linear factors modulo $p$, so $R_p$ is isomomorphic to $\mathbb{Z}_p^N$.
- $p = 2$. Here, each degree-$d$ subring $\mathbb{Z}_p[X]/(f_i(X))$ is isomorphic to the finite field $\mathbb{F}_{2^d}$, hence $R_p \cong \mathbb{F}_{2^d}^{N/d}$.

*Random factors.* A more conservative option is to choose an $F(X)$ that splits completely into $d$ distinct, *random* factors. For instance, for a large prime $p$ we can pick (distinct) random elements $\alpha_1, \ldots, \alpha_N \leftarrow \mathbb{Z}_p$ and let

$$F(X) = \prod_{i=1}^{N} (X - \alpha_i)$$

Just as with the two-power cyclotomic case, $R_p$ is isomorphic to $\mathbb{Z}_p^N$. Now, however, the problem may be harder since we are avoiding the structure given by roots of unity. On the other hand, the isomorphism is more expensive to compute as we can no longer use the FFT, and polynomial interpolation algorithms cost $O(N \log^2 N)$ instead of $O(N \log N)$.

## 4 PCGs for OLE and Authenticated Triples

In this section, we construct PCGs for OLE and authenticated multiplication triples, based on the $R^c$-LPN assumption. The constructions can achieve an arbitrary (a priori bounded) polynomial stretch, that is, the seed size scales logarithmically with the output length.

**Construction $G_{\mathsf{OLE}}$**

PARAMETERS: Security parameter $1^\lambda$, noise weight $t$, compression factor $c$, modulus $p$, and the ring $R = \mathbb{Z}[X]/(F(X))$, where $F(X)$ has degree $N$.
An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of $t^2$ point functions, with domain $[0..2N-1)$ and range $\mathbb{Z}_p$.
PUBLIC INPUT: random polynomials $a_1, \ldots, a_{c-1} \in R_p$, used for $R^c$-$\mathsf{LPN}$
CORRELATION: After expansion, outputs $(x_0, z_0) \in R_p^2$ and $(x_1, z_1) \in R_p^2$, where $z_0 + z_1 = x_0 \cdot x_1$.

**Gen:** On input $1^\lambda$:

1. For $\sigma \in \{0,1\}$ and $i \in [0..c)$, sample random vectors $A_\sigma^i \leftarrow [0..N)^t$ and $\mathbf{b}_\sigma^i \leftarrow \mathbb{Z}_p^t$.
2. For each $i, j \in [0..c)$, sample FSS keys $(K_0^{i,j}, K_1^{i,j}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, A_0^i \boxplus A_1^j, \mathbf{b}_0^i \otimes \mathbf{b}_1^j)$.
3. Let $\mathsf{k}_\sigma = \big((K_\sigma^{i,j})_{i,j\in[0..c)}, (A_\sigma^i, \mathbf{b}_\sigma^i)_{i\in[0..c)}\big)$.
4. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

**Expand:** On input $(\sigma, \mathsf{k}_\sigma)$:

1. Parse $\mathsf{k}_\sigma$ as $\big((K_\sigma^{i,j})_{i,j\in[0..c)}, (A_\sigma^i, \mathbf{b}_\sigma^i)_{i\in[0..c)}\big)$.
2. Define (over $\mathbb{Z}_p$) the degree $< N$ polynomials, for $i \in [0..c)$

$$e_\sigma^i(X) = \sum_{j\in[0..t)} \mathbf{b}_\sigma^i[j] \cdot X^{A_\sigma^i[j]}$$

3. Compute $x_\sigma = \langle \mathbf{a}, \mathbf{e}_\sigma \rangle \mod F(X)$, where $\mathbf{a} = (1, a_1, \ldots, a_{c-1})$, $\mathbf{e}_\sigma = (e_\sigma^0, \ldots, e_\sigma^{c-1})$.
4. For $i, j \in [0..c)$, compute $u_{\sigma,i+cj} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{i,j})$ and view this as a degree $< 2N$ polynomial, defining the length-$c^2$ vector $\mathbf{u}_\sigma$
5. Compute $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_\sigma \rangle \mod F(X)$.
6. Output $(x_\sigma, z_\sigma)$

**Fig. 1.** PCG for OLE over the ring $R_p$, based on ring-LPN

### 4.1 PCG for OLE over $R_p$

We build a PCG for producing a single OLE over the ring $R_p$. When $R_p$ splits appropriately, as described in Sect. 3, this can be locally transformed into a PCG for a large batch of OLEs or authenticated triples over a finite field $\mathbb{F}_{p^d}$ or $\mathbb{F}_p$. The OLE correlation over $R_p$ outputs a single sample from the distribution

$$\left\{ (x_0, z_0), (x_1, z_1) \,\middle|\, x_0, x_1, z_0 \xleftarrow{\$} R_p, z_1 = x_0 \cdot x_1 - z_0 \right\}$$

This is equivalent to a simple bilinear correlation for multiplication in $R_p$.

Below is an informal presentation of the construction, which is described formally in Fig. 1.

The high-level idea is to first give each of the two parties a random vector $\mathbf{e}_0$ or $\mathbf{e}_1 \in R_p^c$, consisting of sparse polynomials, together with a random, additive secret sharing of the tensor product $\mathbf{e}_0 \otimes \mathbf{e}_1$ over $R_p$.

We view $\mathbf{e}_0, \mathbf{e}_1$ as $R^c$-LPN error vectors (whose first entry is implicitly the $R^c$-LPN secret), which will be expanded to produce outputs $x_\sigma = \langle \mathbf{a}, \mathbf{e}_\sigma \rangle$ by each party $P_\sigma$ for a random, public $\mathbf{a} = (1, \hat{\mathbf{a}})$. This defines two $R^c$-LPN instances with independent secrets but the same $\mathbf{a}$ value, which are pseudorandom by a standard reduction to $R^c$-LPN with a single sample. To obtain shares of $x_0 \cdot x_1$, observe that when $\mathbf{a}$ is fixed, this is a degree 2 function in $(\mathbf{e}_0, \mathbf{e}_1)$, so can be computed locally by the parties given their shares of $\mathbf{e}_0 \otimes \mathbf{e}_1$.

The only part that remains, then, is to distribute shares of this tensor product. Recall that each entry of $\mathbf{e}_\sigma$ is a polynomial of degree less than $N$ with at most $t$ non-zero coordinates. We write these coefficients as a set of indices $A \in [0..N)^t$ and corresponding non-zero values $\mathbf{b} \in \mathbb{Z}_p^t$. Taking two such sparse polynomials $(A, \mathbf{b})$ and $(A', \mathbf{b}')$, notice that the product of the two polynomials is given by

$$\left( \sum_{i \in [0..t)} \mathbf{b}[i] \cdot X^{A[i]} \right) \cdot \left( \sum_{j \in [0..t)} \mathbf{b}'[j] \cdot X^{A'[j]} \right) = \sum_{i,j \in [0..t)} \mathbf{b}[i] \cdot \mathbf{b}'[j] \cdot X^{A[i]+A'[j]}$$

We can therefore express the coefficient vector of the product as a *sum of $t^2$ point functions*, where the $(i, j)$-th point function evaluates to $\mathbf{b}[i] \cdot \mathbf{b}'[j]$ at input $A[i] + A'[j]$, and zero elsewhere. This means the parties can distribute this product using a function secret sharing scheme SPFSS for sums of point functions. Each SPFSS takes a set of $t^2$ points and associated vector of values, and produces two keys that represent shares of the underlying sum of point functions. If each party locally evaluates its key at every point in the domain, then it obtains a pseudorandom secret-sharing of the entire sparse polynomial.

There are $c^2$ polynomials in the tensor product, so overall we need $c^2$ instances of SPFSS, where each SPFSS uses $t^2$ point functions. Instantiating this naively using $t^2$ distributed point functions, we get a seed size of $\tilde{O}(\lambda(ct)^2 \log N)$ bits. Note that to achieve exponential security against the best known attacks on $R^c$-LPN, it is enough to choose $ct = O(\lambda)$. By increasing $N$, we can therefore obtain an arbitrary polynomial stretch for the PCG, where the stretch is defined as the ratio of its output length to the seed size.

More concretely, we have the following theorem. For a proof we refer to the full version.

**Theorem 5.** *Suppose that* SPFSS *is a secure FSS scheme, and the $R^c$-LPN$_{p,1,t}$ assumption (Definition 3) holds. Then the construction in Fig. 1 is a secure PCG for OLE over $R_p$.*

*When instantiating* SPFSS *using DPFs from a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, *we have:*

- *Each party's seed has size at most* $(ct)^2 \cdot ((\log N + 1) \cdot (\lambda + 2) + \lambda + \log p) + ct(\log N + \log p)$ *bits.*
- *The computation of* Expand *can be done with at most* $(4 + 2\lfloor (\log p)/\lambda \rfloor) N (ct)^2$ *PRG operations, and* $O(c^2 N \log N)$ *operations in* $\mathbb{Z}_p$.

We remark that assuming $R^c$-LPN holds for *regular error distributions*, the seed size can be reduced to $(ct)^2 \cdot ((\log N - \log t + 1) \cdot (\lambda + 2) + \lambda + \log p) + ct(\log N + \log p)$ bits, and the number of PRG calls in Expand down to $(4 + 2\lfloor(\log p)/\lambda\rfloor)Nc^2t$. Furthermore, implementing SPFSS using batch codes reduces the number of PRG calls to $O(Nc^2)$.

**Obtaining OLEs over $\mathbb{F}_p$.** As mentioned previously, when $R$ and $p$ are chosen appropriately, an OLE over $R_p$ is locally equivalent to $N$ OLEs over $\mathbb{F}_p$ or $\mathbb{F}_{p^d}$. Hence, this PCG immediately implies PCGs over $\mathbb{F}_{p^d}$, with the same seed size and complexity.

If we want to rely on the (apparently) more conservative version of $R^c$-LPN, where $F(X)$ is *irreducible* in $\mathbb{Z}_p[X]$, the parties can still use our PCG over $R_p$ to obtain OLEs over $\mathbb{Z}_p$, but this requires $O(N)$ interaction. To do this, the parties each sample random polynomials $a, b \in \mathbb{Z}_p[X]$, each of degree $< N/2$. They then use the OLE over $R_p$ to multiply $a$ and $b$, which can be done by sending $N$ elements of $\mathbb{Z}_p$.[6] This gives shares of $c = ab$ in $R_p$, which equals $ab$ over $\mathbb{Z}_p[X]$, since no overflow occurs modulo $F(X)$ (which has degree $N$). Each party then locally computes evaluations of its shares of $a, b$ and $c$ at $N/2$ fixed, distinct, non-zero points, which gives $N/2$ secret-shared products over $\mathbb{Z}_p$ (this can be done as long as $p > N/2$).

**Optimizations.** We now discuss a few optimizations which apply to the basic scheme.

*Optimizing the MPFSS evaluation.* Naively, the computational cost of the FSS full-domain evaluation is $O((ct)^2N)$ PRG operations. Using a regular error distribution, we can bring this down to $O(c^2tN)$ (see below). With batch codes [34] or randomized batch codes [48], the full evaluation cost can be brought down to $O(c^2N)$ operations. However, if the seed generation phase has to be created by a secure distributed protocol, this may introduce further complexity.

*Using regular errors.* Suppose two sparse polynomials $e_0, e_1 \in Z_p^N$ are regular, that is $e_b = (e_{b,1}, \ldots, e_{b,t})$, where each $e_{b,j} \in \mathbb{Z}_p^{N/t}$ has weight 1, and defines a coefficient in the range $[(j-1) \cdot (N/t), j \cdot (N/t) - 1]$. Each pair $(e_{0,i}, e_{1,j})$ gives rise to an index in $[(i + j - 2) \cdot (N/t), (i + j) \cdot (N/t) - 2]$, so the product of two regular error polynomials can be represented by a $t^2$-point SPFSS of domain size $2N/t$. This leads to a total expansion cost of $O(c^2tN)$ PRG operations.

**Extension to higher degree correlations.** We can naturally extend this construction from OLE over $R_p$ to general degree-$D$ correlations (over $R_p$), for any constant $D$, by sharing $D$-way products of sparse polynomials instead of just pairwise products. However, this comes at a high cost: the seed size increases to $O((ct)^D \log N\lambda)$, and the computational cost becomes $\tilde{O}((ct)^D \cdot N)$.

[6] This can be reduced to $N/2$, by defining $a, b$ to be the first $N/2$ coefficients of the polynomials $x_0, x_1$ produced by the OLE, so that only the second half of the coefficients need to be sent in the multiplication protocol.

### 4.2 Authenticated Multiplication Triples

We now show how to modify the PCG for OLE to produce *authenticated multiplication triples*, which are often used in maliciously secure MPC protocols such as the BDOZ [6] and SPDZ [22,25] line of work. Note that although OLE can be used to build authenticated triples in a black-box way, doing this requires several OLEs and some interaction, for every triple. Our PCG avoids this interaction, with only a small overhead on top of the previous construction: the seeds are less than 2x larger, while the expansion phase has around twice the computational cost.

*Secret-sharing with MACs.* We use authenticated secret-sharing based on SPDZ MACs between $n$ parties, where a secret-sharing of $x \in \mathbb{Z}_p$ is defined as:

$$[\![x]\!] = (\alpha_i, x_i, m_{x,i})_{i=1}^n \quad \text{such that } \sum_i x_i = x, \sum_i m_{x,i} = x \cdot \sum_i \alpha_i$$

Note that the MAC key shares $\alpha_i$ are fixed for every shared $x$. The MAC shares $m_{x,i}$ are used to prevent a sharing from being opened incorrectly, via a MAC check procedure from [22]. An *authenticated multiplication triple* is a tuple of random sharings $([\![x]\!], [\![y]\!], [\![z]\!])$, where $x, y \xleftarrow{\$} \mathbb{Z}_p$ and $z = x \cdot y$. Our PCG outputs a single multiplication triple over the ring $R_p$, for $n = 2$ parties, together with additive shares of the MAC key $\alpha \in \mathbb{Z}_p$. When using the fully-reducible variant of ring-LPN, this is equivalent to $N$ triples over $\mathbb{F}_{p^d}$ (where for suitably chosen $p$ we can have $d = 1$).

*PCG construction.* The construction is remarkably simple. Recall that our previous construction for OLE uses FSS keys which are expanded into shares of sparse polynomials $u_{i,j} = e_i \cdot e_j \in \mathbb{Z}_p[X]$. The FSS payload was defined by some (column) vector $\mathbf{v} \in \mathbb{Z}_p^{t^2}$, which defines the $t^2$ values of the non-zero coefficients in $u_{i,j}$. We can modify this to produce *authenticated OLE* by extending the FSS range from $\mathbb{Z}_p$ to $\mathbb{Z}_p^2$, and letting the payload be $\mathbf{v} \cdot (1, \alpha) \in \mathbb{Z}_p^{2N \times 2}$, for a random $\alpha \in \mathbb{Z}_p$. Evaluating the FSS keys at some input $k$ now produces shares of $(\mathbf{v}[k], \alpha \cdot \mathbf{v}[k])$. Hence, these can be used to obtain authenticated shares of $x_0 \cdot x_1$, as well as the OLE.

To extend the above to authenticated triples, the seed generation phase will now produce three sets of FSS keys. The first two sets, $(K^i_{x,0}, K^i_{x,1})$ and $(K^i_{y,0}, K^i_{y,1})$, are used to compress shares of the $2c$ sparse polynomials defined by $(A^i_0, \mathbf{b}^i_0)$ and $(A^i_1, \mathbf{b}^i_1)$. These have sparsity $t$, so can be compressed using $t$-point SPFSS, and are later expanded to produce shares and MAC shares for $R_p$ elements $x$ and $y$. The third set, $(K^i_{z,0}, K^i_{z,1})$, compresses pairwise products of the previous sparse polynomials, so each of these can be defined using $t^2$-point SPFSS, as in the previous construction. This gives the shares and MAC shares for the product term $z = x \cdot y$. For the full protocol description we refer to the full version of this paper.

We omit the proof of the following theorem, which is very similar to that of Theorem 5. Recall that to achieve exponential security against the best known

attacks on $R^c$-LPN, it is enough to choose $ct = O(\lambda)$, therefore choosing a larger $c$ allows to decrease the size of $t$. For more details on concrete parameter choices we refer to Sect. 7.

**Theorem 6.** *Suppose that the $R^c$-$\mathsf{LPN}_{p,1,t}$ assumption holds and given a PRG $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$. Then there exists a secure PCG for two-party authenticated multiplication triples over $R_p$ with the following complexities:*

- *Each party's seed has size at most $2(2ct + (ct)^2) \cdot ((\log N + 1) \cdot (\lambda + 2) + \lambda + \log p) + \log p$ bits.*
- *The computation of* Expand *can be done with at most $(8 + 4\lfloor(\log p)/\lambda\rfloor)$ $N(2ct + (ct)^2)$ PRG operations, and $O(c^2 N \log N)$ operations in $\mathbb{Z}_p$.*

## 5 Distributed Setup Protocols

Up to this point, exposition has focused on how to obtain and use pseudorandom correlation generators (PCG), abstracted in an idealized model where the short PCG seeds are sampled by a third-party trusted dealer. In this section, we address solutions for parties to *jointly* generate the desired PCG correlations, via secure distributed setup protocols.

In Sect. 5.1, we show how to securely realize (against a semi-honest adversary) the randomized functionality $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ that executes the seed generation for our PCG construction $\mathsf{PCG}_{\mathsf{OLE}}$ constructed in Sect. 4, and outputs the corresponding PCG seeds to each party. This can in turn be used to realize a functionality for the secure generation of OLE correlations, by having the parties simply expand their received PCG seeds locally. Our protocol to implement $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ makes black-box use of sub-protocols for simple secure computations over $\mathbb{Z}_p$ and $\{0,1\}^\ell$, and for secure computation of DPF key generation where the position and payload values are held secret shared across the two parties. In particular, the latter can be implemented with the efficient DPF key generation protocol of Doerner and shelat [27].

In the full version, for the malicious case we present a protocol securely realizing the randomized functionality $\mathcal{F}_{\mathsf{mal\text{-}OLE}}$, in which a corrupt adversary can choose his output $(x_\sigma, z_\sigma) \in R_p^2$ and the honest party receives a random consistent value $(x_{1-\sigma}, z_{1-\sigma})$, i.e. for which $z_0 + z_1 = x_0 \cdot x_1$ (or the parties receive a random sample from the correlation given honest behavior; see the full version for details). As discussed in [11], such a protocol can *directly* serve as a substitute for ideal OLE correlations in a wide range of higher-level applications, already proven to remain secure given this functionality. Further, we give a protocol realizing an analogous functionality for *authenticated multiplication triples* in the malicious model at little extra cost. Achieving security in the malicious

setting poses further challenges, including managing potential leakage on the secret noise positions from the [27] protocol in the face of malicious behavior, while simultaneously enforcing consistency.

For a detailed listing of efficiency of our protocols we refer to the full version. For an overview of concrete efficiency we refer to Table 2 in Sect. 7.

### 5.1 Semi-honest Distributed Setup

We present a protocol for securely executing the seed-generation functionality respective to our PCG construction $\mathsf{PCG}_{\mathsf{OLE}}$ from Sect. 4. For a description of the functionality $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ we refer to the full version.

Recall that in the $\mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}$ procedure (see Fig. 1), each party receives a succinct description $(A^i_\sigma, \mathbf{b}^i_\sigma)_{i\in[0..c)}$ of $t$-sparse "noise vectors" $e^0_\sigma, \ldots, e^{c-1}_\sigma$ each of length $N$, as well as a collection of $(ct)^2$ distributed point function (DPF) keys as a compact representation of all possible products $e^i_0 \cdot e^j_1$. A secure distributed realization of this procedure can then be achieved given access to two secure sub-protocols:

- Secure computation of DPF key generation for a "path" $\alpha$ and "payload" $\beta$ held secret shared across the parties. Concretely, this can be instantiated by the DPF-generation protocol of Doerner and shelat [27], given *bitwise* additive shares of the path $\alpha \in \mathbb{Z}_{2N}$ and $\mathbb{Z}_p$-additive shares of the nonzero payload $\beta$. For the functionality $\mathcal{F}_{\mathsf{DPF}}$ and a high-level description of the protocol by Doerner and shelat [27], we refer to the full version of this paper.
- Generic secure computation of simple computations over $\mathbb{Z}_2$ or $\mathbb{Z}_p$, used to securely compute secret shares of the products $(\mathbf{b}^i_0[k] \cdot \mathbf{b}^j_1[l])$ over $\mathbb{Z}_p$, and shares of the $\mathbb{Z}_p$-sums $(A^i_0[k] + A^j_1[l]) \in [0..2N-1)$. Note that the latter computation is nontrivial, as the parties must hold *bitwise* additive shares of $(A^i_0[k] + A^j_1[l])$ for the [27] protocol, but the sum itself is with respect to $\mathbb{Z}_N$. This "grade school addition" over bits can be implemented via a binary circuit for integer addition with $\log N$ AND gates, similar to previous (e.g., garbled circuit based [41]) protocols. For more details on the functionality $\mathcal{F}_{\mathsf{2\text{-}PC}}$, an implementation of $\mathcal{F}_{\mathsf{2\text{-}PC}}$ and efficiency considerations we refer to the full version.

The protocol $\Pi_{\mathsf{OLE}}$ for securely realizing $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ in the $(\mathcal{F}_{\mathsf{2\text{-}PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model is given in Fig. 2.

**Theorem 7.** *Assuming hardness of $R^c$-$\mathsf{LPN}_{p,1,t}$, the protocol $\Pi_{\mathsf{OLE\text{-}Setup}}$ (Fig. 2) securely realizes the OLE generation functionality $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ with security against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{2\text{-}PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model.*

*Proof.* Observe that the protocol $\Pi_{\mathsf{OLE\text{-}Setup}}$ is directly a secure evaluation of the computation steps of $\mathcal{F}_{\mathsf{OLE\text{-}Setup}}$ (see description of $\mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}$ as given in Fig. 1 within Sect. 4), with the exception that the FSS key generation

**Protocol $\Pi_{\mathsf{OLE\text{-}Setup}}$**

PARAMETERS: Security parameter $1^\lambda$, natural number $N = 2^k$, prime $p$, distributed point function $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain $[0..2N)$ and range $\mathbb{Z}_p$.

We assume access to a functionality $\mathcal{F}_{\mathsf{2\text{-}PC}}$ as follows: $\mathbf{Input}(P_\sigma, x)$ receives a value $x \in \{0,1\}^\ell$ or $x \in \mathbb{Z}_p$ from party $P_\sigma$ and stores an identifier $[\![x]\!]$, $\mathbf{BitAdd}([\![x]\!]_2, [\![y]\!]_2)$ (for $x, y \in \{0,1\}^\ell$) computes $z = x + y \in \{0,1\}^{\ell+1}$ via arithmetic addition and stores $[\![z]\!]_2$, and $\mathbf{Mult}([\![x]\!]_p, [\![y]\!]_p)$ (for $x, y \in \mathbb{Z}_p$) computes $z = x \cdot y \mod p$ and stores $[\![z]\!]_p$.

PROTOCOL:

1. For $\sigma \in \{0,1\}, i \in [0..c)$, party $P_\sigma$ samples random vectors $A^i_\sigma \leftarrow [0..N)^t$ (where each entry of $A^i_\sigma$ is viewed as a length-$\log N$ bit-string) and $\mathbf{b}^i_\sigma \leftarrow \mathbb{Z}^t_p$. Note that as outlined in Figure 1, each pair $A^i_\sigma, \mathbf{b}^i_\sigma$ defines a $t$-sparse polynomial $e^i_\sigma \in R_p$.
2. For $\sigma \in \{0,1\}, i \in [0..c)$ and $k \in [0..t)$:
   //$P_\sigma$ inputs the $k$-th non-zero position and corresponding payload of $e^i_\sigma$.
   $$[\![A^i_\sigma[k]]\!]_2 \leftarrow \mathbf{Input}(P_\sigma, A^i_\sigma[k]) \text{ and } [\![\mathbf{b}^i_\sigma[k]]\!]_p \leftarrow \mathbf{Input}(P_\sigma, \mathbf{b}^i_\sigma[k])$$
3. For every $i, j \in [0..c)$ and $k, l \in [0..t)$ (in parallel) the parties do the following:
   (a) $[\![\alpha^{i,j}_{k,l}]\!]_2 \leftarrow \mathbf{BitAdd}([\![A^i_0[k]]\!]_2, [\![A^j_1[l]]\!]_2)$ //Compute the $k + tl$-th position of $e^i_0 \cdot e^j_1$.
   (b) $[\![\beta^{i,j}_{k,l}]\!]_p \leftarrow \mathbf{Mult}([\![\mathbf{b}^i_0[k]]\!]_p, [\![\mathbf{b}^j_1[l]]\!]_p)$ //Compute the $k + tl$-th payload of $e^i_0 \cdot e^j_1$.
   (c) For each $i, j \in [0..c)$ and $k, l \in [0..t)$, call $\mathcal{F}_{\mathsf{DPF}}$ with domain size $[0..2N)$ on input $[\![\alpha^{i,j}_{k,l}]\!]_2$ and $[\![\beta^{i,j}_{k,l}]\!]_p$, and let $K^{i,j}_{\sigma,k,l}$ denote the output to party $P_\sigma$.
   //Compute compressed additive secret shares of $e^i_0 \cdot e^j_1$.
4. Party $P_\sigma$ outputs $\mathsf{k}_\sigma = \left((K^{i,j}_{\sigma,k,l})_{i,j\in[0..c),k,l\in[0..t)}, (A^i_\sigma, \mathbf{b}^i_\sigma)_{i\in[0..c)}\right)$.

**Fig. 2.** Distributed setup of OLE seeds in $(\mathcal{F}_{\mathsf{2\text{-}PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model, against semi-honest adversaries. $[\![x]\!]_2, [\![x]\!]_p$ denote additive shares of bit-strings or $\mathbb{Z}_p$ elements.

for the sum of point functions $\mathsf{SPFSS.Gen}(1^\lambda, A^i_0 \boxplus A^j_1, \mathbf{b}^i_0 \otimes \mathbf{b}^j_1)$ is instantiated directly by the DPF key generation for every individual nonzero component. As this is a valid instantiation of $\mathsf{SPFSS}$, the claim holds.

## 6 Extensions and Applications

In this section we extend our PCG for OLE in several directions. First, we build PCGs for inner product correlations from OLE over $R_p$, with the advantage that we do not need to rely on the full reducibility of $F(X)$, and can also obtain correlations over $\mathbb{F}_2$. Secondly, we present a method for building PCGs for general bilinear correlations, such as matrix multiplication, in a black-box way from our previous PCG. Finally, we show that all of these PCGs for degree two correlations can be extended in a natural way to the multi-party setting.

### 6.1 Bilinear Correlations

The class of bilinear correlations we consider is as follows.

**Definition 8 (Simple Bilinear Correlation).** *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be Abelian groups and $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map. We define the* simple bilinear correlation *for $e$ by the distribution $\mathcal{C}_e$ over $(\mathbb{G}_1 \times \mathbb{G}_T) \times (\mathbb{G}_2 \times \mathbb{G}_T)$ of the form*

$$\mathcal{C}_e = \{((r_0, s_0), (r_1, s_1)) \mid r_0 \leftarrow \mathbb{G}_1, r_1 \leftarrow \mathbb{G}_2, s_0 \leftarrow \mathbb{G}_T, s_1 = e(r_0, r_1) - s_0\}\,.$$

*We denote by $\mathcal{C}_e^n$ the correlation that outputs $n$ independent samples from $\mathcal{C}_e$.*

This covers several common correlations like OT and OLE, for example, OLE over a ring $R$ can be obtained with $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_T = (R, +)$ and $e(x, y) = x \cdot y$. Also, note that two independent bilinear correlations can be locally converted to produce an additively secret-shared instance of the correlation—for example, two OLEs are locally equivalent to one multiplication triple.

### 6.2 Inner Product Correlations

An inner product correlation is a simple bilinear correlation with the inner product map over $\mathbb{F}_p$. These can be used to compute inner products in an MPC online phase, in a similar way to using multiplication triples. Inner products are common in tasks involving linear algebra, like privately evaluating or training machine learning models such as SVMs and neural networks, and a single inner product can also be used to measure the similarity between two input vectors.

We remark that given $n$ random OLEs in $\mathbb{F}_p$, it is easy to locally convert these into an length-$n$ inner product correlation, so we can build a PCG for inner products of any length using a PCG for OLE. However, the constructions in this section *do not* rely on the fully-reducible ring-LPN assumption that is needed for OLE in $\mathbb{F}_p$; instead, we use the ring-OLE construction from Fig. 1 over more conservative rings, which do not split completely into linear factors.

For the proof of the following lemma, we refer to our full version.

**Lemma 9.** *Let $R_p = \mathbb{Z}_p[X]/(F(X))$, where $F(X)$ is a degree-$N$ polynomial with non-zero constant coefficient. Then, a single OLE over $R_p$ can be locally converted into an inner product correlation over $\mathbb{F}_p^N$.*

Note that, for the special case of $F(X) = X^n + 1$, the vector $M_a[0]$ can be computed as $(a_0, -a_{n-1}, \ldots, -a_1)$, without any modular reductions.

**Corollary 10 (Large inner product from irreducible ring-LPN).** *Suppose the $R$-$\mathsf{LPN}_{p,1,t}$ assumption holds for $R = \mathbb{Z}_p[X]/(F(X))$, where $F(X)$ is degree $N$ and irreducible over $\mathbb{Z}_p$. Then there is a PCG for the length-$N$ inner product correlation, where the seeds have size $O(\lambda t^2 \log N)$ bits, and the computational complexity of the* Expand *operation is $\tilde{O}(N)$ operations in $\mathbb{Z}_p$, plus $O(t^2 N)$ PRG operations.*

**Corollary 11 (Small inner products from reducible ring-LPN).** *Suppose the $R$-$\mathsf{LPN}_{p,1,t}$ assumption holds for $R = \mathbb{Z}_p[X]/(F(X))$, where $F(X)$ is degree $N$ and splits into $N/d$ distinct factors of degree $d$. Then there is a PCG for producing $N/d$ instances of length-$d$ inner product correlations, with the same seed size and complexity as above.*

The latter construction has two benefits over naively using OLE over $\mathbb{F}_p$ to generate an inner product. Firstly, OLE in $\mathbb{F}_p$ requires that $R$ splits fully into *linear factors*, whereas for inner products the factors can be degree-$d$ (and irreducible), which is a much more conservative assumption; in particular, the dimension-reduction attack we consider in the full version is less effective. Secondly, we can also use this to generate inner products over small fields such as $\mathbb{F}_2$, whereas we cannot efficiently obtain OLEs over $\mathbb{F}_2$ with our present constructions.

### 6.3 Bilinear Correlations from Programmable PCG for OLE

We can build a PCG to create a large batch of samples from *any* simple bilinear correlation, using the PCG for OLE from Sect. 4. To do this, we exploit the fact that this PCG is *programmable*, which, roughly speaking, means that one party can "reuse" its input $a$ or $b$ in several instances of the PCG, while maintaining security. Boyle et al. [11] previously used this property to construct multi-party PCGs from several instances of programmable two-party PCGs; unlike their work, we exploit the property for a different purpose in the two-party setting.

In the full version, we recall the definition of programmability, and show that our PCG for OLE satisfies this definition.

Below we describe the main result, and some applications.

**Decomposition of bilinear maps.** Let $f : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and $g : \mathbb{G}_1^u \times \mathbb{G}_2^v \to \mathbb{G}_T^w$ be bilinear maps. We will consider ways of computing $g$ that are restricted to a fixed number of calls to $f$ on the components of the inputs to $g$, followed by linear combinations in $\mathbb{G}_T$ of the results of the $f$ evaluations.

**Definition 12 (Simple $f$-decomposition).** *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be additive abelian groups, viewed as $\mathbb{Z}$-modules. Let $f : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and $g : \mathbb{G}_1^u \times \mathbb{G}_2^v \to \mathbb{G}_T^w$ be non-degenerate bilinear maps. We say that $g$ has a* simple *$f$-decomposition if there exist $\gamma \in \mathbb{N}$, $W \in \mathbb{Z}^{w \times \gamma}$ and $\alpha_i \in [u], \beta_i \in [v]$, for $i \in [\gamma]$, such that for all $x = (x_1, \ldots, x_u) \in \mathbb{G}_1^u$ and $y = (y_1, \ldots, y_v) \in \mathbb{G}_2^v$, it holds that*

$$g(x, y) = W \cdot \begin{pmatrix} f(x_{\alpha_1}, y_{\beta_1}) \\ \vdots \\ f(x_{\alpha_\gamma}, y_{\beta_\gamma}) \end{pmatrix}$$

*We say that the $f$-*complexity *of this decomposition of $g$ is given by $n_f(g) := \gamma$.*

Note that if $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are all a (commutative) ring $R$ and $f$ is multiplication in $R$, then any $g$ has a simple $f$-decomposition of complexity $u \cdot v$. However, it can still be useful to find a different $f$ that achieves lower complexity.

We now show that any map $g$ with a simple $f$-decomposition can be used to construct a PCG for the simple bilinear correlation $\mathcal{C}_g$, given a programmable PCG for $\mathcal{C}_f$.

For the construction and its security proof we refer to the full version. There, we further show that the new PCG also satisfies the programmability property.

**Theorem 13.** *Let $f$ and $g$ be bilinear maps as above, and suppose that $g$ has a simple $f$-decomposition with $f$-complexity $n_f(g)$. Furthermore, let $\mathsf{PCG}_f = (\mathsf{PCG}_f.\mathsf{Gen}, \mathsf{PCG}_f.\mathsf{Expand})$ be a programmable PCG for $\mathcal{C}_f^n$. Then there exists a PCG $\mathsf{PCG}_g = (\mathsf{PCG}_g.\mathsf{Gen}, \mathsf{PCG}_g.\mathsf{Expand})$ for $\mathcal{C}_g^n$, with the following properties:*

- $\mathsf{PCG}_g.\mathsf{Gen}$ *runs $n_f(g)$ executions of $\mathsf{PCG}_f.\mathsf{Gen}$, and its key sizes are $n_f(g)$ times that of $\mathsf{PCG}_f$.*
- $\mathsf{PCG}_g.\mathsf{Expand}$ *runs $n_f(g)$ executions of $\mathsf{PCG}_f.\mathsf{Expand}$, and $n$ evaluations of the linear map $W$ from the $f$-decomposition of $g$.*

### 6.4 Applications

In the following we will give a brief overview of applications of the general bilinear construction. For a more detailed discussion we refer to the full version.

**Matrix multiplication triples.** Multiplication of $n_1 \times n_2$ and $n_2 \times n_3$ matrices is easily decomposed as a sequence of $n_1 \cdot n_3$ inner products of length $n_2$, where each inner product is taken from a consecutive portion of the two inputs. Therefore, the matrix multiplication map $g$ has a linear $f$-decomposition with $f$-complexity $n_1 \cdot n_3$.

This results in a matrix multiplication triple with seed size around $n_1 \cdot n_3$ times larger than the PCG seed for OLE, which will likely be practical for small-to-medium matrices. Note that using a programmable PCG for OLE directly to build matrix multiplications would require $n_1 \cdot n_2 \cdot n_3$ instances of the base PCG, giving a much worse expansion factor.

**Circuit-dependent MPC preprocessing.** Circuit-dependent preprocessing is a variation on the standard multiplication triples technique, which is based on Beaver's circuit randomization technique [4] and extended in more recent works [5,23]. The idea is to preprocess multiplications in a way that depends on the structure of the circuit, and leads to an online phase that requires just *one opening per multiplication gate*, instead of two when using multiplication triples.

With the PCG for general bilinear correlations, we can generate circuit-dependent preprocessing for a large batch of identical circuits. This can be useful, for instance, when executing the same function many times on different inputs, or when a larger computation contains many small, repeated instances of a particular sub-circuit. For more details we refer to the full version.

**Multi-party PCGs for bilinear correlations.** In [11] [Theorem 41], Boyle et al. showed that any two-party, programmable PCG for a simple bilinear correlation can be used to build a multi-party PCG for an additively secret-shared version of the same correlation.

Using that the PCG for bilinear correlations from Theorem 13 is programmable and the results of the previous section, we obtain $N$-party PCGs for (unauthenticated) multiplication triples, matrix triples and circuit-dependent preprocessing over $\mathbb{Z}_p$ based on ring-LPN, for any polynomial number of parties $N$. Each party's seed contains $2(N-1)$ seeds of the underlying two-party PCG, plus $N-1$ seeds for a PRG. The expansion procedure of the PCG consists of expanding the $2(N-1)$ PCG seeds, as well as the PRG seeds.

## 7 Efficiency Analysis

In the full version, we provide a detailed security analysis of the flavors of the $R$-LPN assumption which we use. In this section, based on our security analysis, we discuss concrete choices of parameters for which the corresponding ring-LPN problems are secure against the attacks we considered, and analyse the concrete efficiency of our PCGs. In all instances, we assume the noise vectors to have a regular structure (as was done e.g. in [9–11]), since this does not introduce any known weakness, but significantly increases the efficiency of Expand. For large fields, we focus on the statistical decoding and information set decoding (ISD) families of attacks (the latter being always at least as efficient as the Gaussian elimination attack), combined with the speedup obtained with the DOOM attack against quasi-cyclic codes. For statistical decoding, we compute our estimations with a conservative lower bound of $n \cdot (cN/(N-1))^w$ arithmetic operations. For ISD, we used LEDA's public domain software implementation of an automated procedure for the design of tight and optimal sets of parameters, developed in the context of the LEDA candidate [3] for the NIST post-quantum competition[7]. This software takes as input the parameters (dimension, number of sample, number of noisy coordinates, block-size of the quasi-cyclic matrices) of the instance, and outputs the complexity of attacking the instance with several ISD variants.

**Theoretical Analysis for Reducible Ring-LPN.** We consider a field $\mathbb{F} = \mathbb{Z}_p$ of size $|\mathbb{F}| \approx 2^{128}$. As in our applications, we focus on the case where there is a factor $f_i$ of degree $n = N/k$, for some $k$, which has sparsity 1, such as when $N$ is a power of two and $F(X) = X^N + 1$ splits completely into $N$ linear factors modulo $p$. From the analysis in the full version, we can reduce an instance modulo a 1-sparse factor $f_i$ of degree $n = 2^i$, reducing the expected number of noisy coordinates to

$$w_i = w - cn + (c(n-1) + w) \cdot \left(1 - \frac{1}{n}\right)^{w/c-1},$$

[7] https://github.com/LEDAcrypt/LEDAtools.

the dimension to $n_i = (c-1) \cdot 2^i$, and the number of samples to $q_i = c \cdot 2^i$. In our experiments, we found that the optimal behavior for the adversary was always to pick the smallest $i$ such that the new weight $w_i$ of the noise is not higher than the dimension $n_i$ (such that the reduced instance is still uniquely decodable and is not statistically close to random). For security parameter $\lambda = 80$ (resp. $\lambda = 128$), the smallest such $i$ is $i = 6$ (resp. $i = 7$). In Table 1, we provide various choices of parameters $(\lambda, N, c, w)$ such that the best attack on any reduced instance requires at least $2^\lambda$ multiplications over a field $\mathbb{F}$ of size $|\mathbb{F}| \approx 2^{128}$. The table also presents the concrete seed sizes and computational requirements for our PCG for OLE, based on Theorem 5 (and with optimizations due to the regular error distribution).

Our conservative estimates of the running time of the statistical decoding attack have better asymptotic complexity than the ISD attacks, according to our analysis in the full version; and indeed, we found statistical decoding to always give the best available attack. Given that statistical decoding should not generally perform better than ISD [26], this suggests that our estimation of the cost of statistical decoding might be overly conservative, meaning that our parameters are slightly pessimistic.

**Estimated Runtimes for OLE and Triple Generation.** We estimate the computational cost of expanding a PCG seed to produce $N = 2^{20}$ OLEs over $\mathbb{Z}_p$, using our PCG over a ring $R_p$ which splits fully into linear factors. The main costs in the expansion step are the DPF full-domain evaluations, and polynomial operations over $R_p$. We separately benchmarked these using the DPF code from [10], and NFLLib [1] for polynomial arithmetic with a 124-bit modulus $p$, which is a product of two 62-bit primes (such that $R_p$ splits completely into linear factors, by the CRT). We also estimated the communication complexity required to distribute the PCG seeds with active security, based on the analysis provided in the full version.

The results are in Table 2 for OLE, and Table 3 for authenticated triples. Note that compared with Table 1, the noise weight $w$ has been rounded so it is divisible by $c$, so that $t = w/c$ is an integer. We see that as the module-LPN compression parameter $c$ increases, the polynomial arithmetic gets more expensive, while the DPF cost first decreases at $c = 4$, and then goes back up at $c = 8$. This is because the DPF complexity scales with $c^2 t$, so doubling $c$ only reduces its cost if $t$ can be reduced by more than a factor of 4. The best choice for speed seems to be $c = 4$, where we are able to silently expand over 100 thousand OLEs per second at the 128-bit security level, with a seed size of around 1 MB. When generating authenticated triples instead of OLEs, the seed size and runtimes increase by roughly a factor of two, while the setup communication cost is only slightly larger.

**Table 1.** Concrete parameters and seed size (per party, counted as equivalent number of field elements) for our PCG for OLE over $R_p = \mathbb{Z}_p[X]/F(X)$, where $p = 1 \bmod 2N$, $\log p \approx 128$ and $F = X^N + 1$ is the $2N$-th cyclotomic polynomial which fully splits over $\mathbb{Z}_p$, for various $\lambda$, $N$, syndrome compression factor $c$, and number of noisy coordinates $w$. 'Stretch', computed as $2N/(\text{seed size})$, is the ratio between storing a full random OLE (i.e., $2N$ field elements) and the smaller PCG seed. Parameters are chosen such that the best-known distinguishing attacks, over any instance reduced modulo a sparse factor of $F$, require $T$ field multiplications over $\mathbb{Z}_p$ to have distinguishing advantage $T/2^\lambda$. This setting is useful for generating batches of $N$ OLE correlations or authenticated triples over $\mathbb{Z}_p$, or small inner-product correlations (Sect. 6.2). When using a smaller field $\mathbb{F}'$, the bit-length of the seed stays roughly the same but increases by a factor of $\approx \log_2 |\mathbb{F}| / \log_2 |\mathbb{F}'|$ in field elements, the stretch decreases by the same factor, (e.g. about a factor 2 when using $\mathbb{F}$ with $\log_2 |\mathbb{F}'| \approx 64$), and all other entries remain the same.

| $\lambda$ | $N$ | $c$ | $w$ | $(i, w_i)$ | Seed size | Stretch | # $R$-mults | #PRG calls |
|---|---|---|---|---|---|---|---|---|
| 80 | $2^{20}$ | 2 | 97 | $(6, 74)$ | $2^{17.4}$ | 12 | 4 | $2^{29.6}$ |
| 80 | $2^{20}$ | 4 | 40 | $(6, 37)$ | $2^{15.0}$ | 65 | 16 | $2^{29.3}$ |
| 80 | $2^{20}$ | 8 | 26 | $(6, 25)$ | $2^{13.9}$ | 139 | 64 | $2^{29.7}$ |
| 128 | $2^{20}$ | 2 | 152 | $(7, 121)$ | $2^{18.6}$ | 5 | 4 | $2^{30.2}$ |
| 128 | $2^{20}$ | 4 | 64 | $(7, 60)$ | $2^{16.3}$ | 27 | 16 | $2^{30.0}$ |
| 128 | $2^{20}$ | 8 | 41 | $(7, 40)$ | $2^{15.1}$ | 59 | 64 | $2^{30.4}$ |
| 80 | $2^{25}$ | 2 | 97 | $(6, 74)$ | $2^{17.7}$ | 306 | 4 | $2^{34.6}$ |
| 80 | $2^{25}$ | 4 | 40 | $(6, 37)$ | $2^{15.3}$ | 1654 | 16 | $2^{34.3}$ |
| 80 | $2^{25}$ | 8 | 26 | $(6, 25)$ | $2^{14.2}$ | 3623 | 64 | $2^{34.7}$ |
| 128 | $2^{25}$ | 2 | 152 | $(7, 121)$ | $2^{19.0}$ | 130 | 4 | $2^{35.2}$ |
| 128 | $2^{25}$ | 4 | 64 | $(7, 60)$ | $2^{16.6}$ | 673 | 16 | $2^{35.0}$ |
| 128 | $2^{25}$ | 8 | 41 | $(7, 40)$ | $2^{15.4}$ | 1513 | 64 | $2^{35.4}$ |

**Table 2.** Estimated costs for our PCG for producing $N = 2^{20}$ OLEs in $\mathbb{Z}_p$, with $\log p \approx 124$. Setup comm. measures the per-party communication required to setup the PCG seeds with active security (ignoring costs for correlated randomness that can come from a previous PCG).

| $\lambda$ | $c$ | $w$ | Seed size (MB) | Setup comm. (MB) | Runtimes for Expand (s) | | |
|---|---|---|---|---|---|---|---|
| | | | | | $R$-mult (s) | DPF eval. (s) | Total (s) |
| 80 | 2 | 96 | 2.69 | 7.43 | 0.4 | 9.8 | 10.2 |
| 80 | 4 | 40 | 0.52 | 1.41 | 1.4 | 7.5 | 8.9 |
| 80 | 8 | 32 | 0.35 | 0.94 | 5.3 | 9.3 | 14.6 |
| 128 | 2 | 152 | 6.37 | 17.77 | 0.4 | 12.6 | 13.0 |
| 128 | 4 | 64 | 1.26 | 3.45 | 1.4 | 8.6 | 10.0 |
| 128 | 8 | 40 | 0.55 | 1.47 | 5.3 | 14.4 | 19.7 |

**Table 3.** Estimated costs for our PCG for producing $N = 2^{20}$ authenticated triples in $\mathbb{Z}_p$, with $\log p \approx 124$. Setup comm. measures the per-party communication required to setup the PCG seeds with active security (ignoring costs for correlated randomness that can come from a previous PCG).

| $\lambda$ | $c$ | $w$ | Seed size (MB) | Setup comm. (MB) | Runtimes for Expand (s) | | |
|---|---|---|---|---|---|---|---|
| | | | | | $R$-mult (s) | DPF eval. (s) | Total (s) |
| 80 | 2 | 96 | 5.49 | 9.07 | 0.8 | 19.6 | 20.4 |
| 80 | 4 | 40 | 1.09 | 1.74 | 2.8 | 15.0 | 17.8 |
| 80 | 8 | 32 | 0.74 | 1.16 | 10.6 | 18.6 | 29.2 |
| 128 | 2 | 152 | 12.91 | 21.73 | 0.8 | 25.2 | 26.0 |
| 128 | 4 | 64 | 2.60 | 4.22 | 2.8 | 17.2 | 20.0 |
| 128 | 8 | 40 | 1.14 | 1.80 | 10.6 | 28.8 | 39.4 |

**Acknowledgements.** We would like to thank Vadim Lyubashevsky, Chris Peikert, Ronny Roth and Jean-Pierre Tillich for helpful discussions and pointers.

E. Boyle, N. Gilboa, and Y. Ishai, and L. Kohl supported by ERC Project NTSC (742754). E. Boyle additionally supported by ISF grant 1861/16 and AFOSR Award FA9550-17-1-0069. G. Couteau supported by ERC Project PREP-CRYPTO (724307). N. Gilboa additionally supported by ISF grant 1638/15, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai additionally supported by NSF-BSF grant 2015782, BSF grant 2018393, and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India. Work of L. Kohl was done in part while at Karlsruhe Institute of Technology, supported by ERC Project PREP-CRYPTO (724307) and DFG grant HO 4534/2-2. P. Scholl supported by the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC) and an Aarhus University Research Foundation starting grant.

## References

1. Aguilar-Melchor, C., Barrier, J., Guelton, S., Guinet, A., Killijian, M.-O., Lepoint, T.: NFLlib: NTT-based fast lattice library. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 341–356. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_20
2. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_35
3. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: Design of LEDAkem and LEDApkc instances with tight parameters and bounded decryption failure rate (2019). https://www.ledacrypt.org/archives/official_comment.pdf
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34

5. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: double your online SPDZ! improving SPDZ using function dependent preprocessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 530–549. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_26
6. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_11
7. Bernstein, D.J., Lange, T.: Never trust a bunny. In: Hoepman, J.-H., Verbauwhede, I. (eds.) RFIDSec 2012. LNCS, vol. 7739, pp. 137–148. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36140-1_10
8. Blum, A., Furst, M., Kearns, M., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_24
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: ACM CCS 2018, pp. 896–912. ACM Press (2018)
10. Boyle, E., et al.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCS 2019, pp. 291–308. ACM Press (2019)
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_16
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: optimizations and applications. In: ACM CCS 2017, pp. 2105–2122. ACM Press (2017)
13. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
14. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19
15. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: ACM CCS 2016, pp. 1292–1303. ACM Press (2016)
16. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
17. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 341–371. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_14
18. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1
19. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press, October 2011

20. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
21. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 473–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_17
22. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1
23. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The TinyTable protocol for 2-party secure computation, or: gate-scrambling revisited. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 167–187. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_6
24. Damgård, I., Park, S.: How practical is public-key encryption based on LPN and ring-LPN? Cryptology ePrint Archive, Report 2012/699 (2012). http://eprint.iacr.org/2012/699
25. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
26. Debris-Alazard, T., Tillich, J.P.: Statistical decoding. In: 2017 IEEE International Symposium on Information Theory (ISIT), pp. 1798–1802. IEEE (2017)
27. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: ACM CCS 2017, pp. 523–535. ACM Press (2017)
28. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
29. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987
30. Guo, Q., Johansson, T., Löndahl, C.: A new algorithm for solving ring-LPN with a reducible polynomial. IEEE Trans. Inf. Theory **61**(11), 6204–6212 (2015)
31. Hazay, C., Ishai, Y., Marcedone, A., Venkitasubramaniam, M.: LevioSA: lightweight secure arithmetic computation. In: ACM CCS 2019, pp. 327–344. ACM Press (2019)
32. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: an efficient authentication protocol based on ring-LPN. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 346–365. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_20
33. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 600–620. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_34
34. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: Babai, L. (ed.) 36th ACM STOC, pp. 262–271. ACM Press, June 2004
35. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32

36. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 294–314. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_18
37. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: a low latency framework for secure neural network inference. In: USENIX 2018, pp. 1651–1669 (2018)
38. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: ACM CCS 2016, pp. 830–842. ACM Press (2016)
39. Keller, M., Pastro, V., Rotaru, D.: Overdrive: making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 158–189. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_6
40. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC, pp. 20–31. ACM Press, May 1988
41. Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_1
42. Lipmaa, H., Pavlyk, K.: Analysis and implementation of an efficient ring-LPN based commitment scheme. In: Reiter, M., Naccache, D. (eds.) CANS 2015. LNCS, vol. 9476, pp. 160–175. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26823-1_12
43. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
44. Melchor, C.A., Blazy, O., Deneuville, J., Gaborit, P., Zémor, G.: Efficient encryption from random quasi-cyclic codes. IEEE Trans. Inf. Theory **64**(5), 3927–3943 (2018). https://doi.org/10.1109/TIT.2018.2804444
45. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: 31st ACM STOC, pp. 245–254. ACM Press, May 1999
46. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_40
47. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press, May 2005
48. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: improved constructions and implementation. In: ACM CCS 2019, pp. 1055–1072. ACM Press (2019)
49. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Des. Codes Cryptogr. **71**(1), 57–81 (2014)