Correlated Pseudorandomness from Expand-Accumulate Codes

Elette Boyle[1], Geoffroy Couteau[2][0000−0002−6645−0106], Niv Gilboa[3][0000−0001−7209−3494], Yuval Ishai[4], Lisa Kohl[5], Nicolas Resch[5], and Peter Scholl[6][0000−0002−7937−8422]

[1] IDC Herzliya and NTT Research
[2] IRIF
[3] Ben-Gurion University
[4] Technion
[5] Cryptology Group, CWI Amsterdam
[6] Aarhus University

Abstract. A pseudorandom correlation generator (PCG) is a recent tool for securely generating useful sources of correlated randomness, such as random oblivious transfers (OT) and vector oblivious linear evaluations (VOLE), with low communication cost.

We introduce a simple new design for PCGs based on so-called *expand-accumulate* codes, which first apply a sparse random expander graph to replicate each message entry, and then accumulate the entries by computing the sum of each prefix. Our design offers the following advantages compared to state-of-the-art PCG constructions:

- Competitive concrete efficiency backed by provable security against relevant classes of attacks;
- An offline-online mode that combines near-optimal cache-friendliness with simple parallelization;
- Concretely efficient extensions to pseudorandom correlation *functions*, which enable incremental generation of new correlation instances on demand, and to new kinds of correlated randomness that include circuit-dependent correlations.

To further improve the concrete computational cost, we propose a method for speeding up a full-domain evaluation of a puncturable pseudorandom function (PPRF). This is independently motivated by other cryptographic applications of PPRFs.

1 Introduction

Correlated secret randomness is a powerful and ubiquitous resource for cryptographic applications. In the context of secure multiparty computation (MPC) with a dishonest majority, simple sources of correlated randomness can serve as a "one-time pad" for lightweight, concretely efficient protocols [Bea91]. As a classical example, consider the case of a random *oblivious transfer* (OT) correlation, in which Alice and Bob receive (s_0, s_1) and (b, s_b) respectively, where s_0, s_1, b are

random bits. Given $2n$ independent instances of this simple OT correlation, Alice and Bob can evaluate any Boolean circuit with n gates (excluding XOR and NOT gates) on their inputs, with perfect semi-honest security, by each sending 2 bits and performing a small constant number of Boolean operations per gate.

The usefulness of correlated randomness for MPC gave rise to the following popular two-phase approach. First, the parties run an input-independent *preprocessing protocol* for secure distributed generation of correlated randomness. This correlated randomness is then consumed by an *online protocol* that performs a secure computation on the inputs. Traditional approaches for implementing the preprocessing protocol (e.g., [IKNP03, DPSZ12, KPR18]) have an $\Omega(n)$ communication cost that usually forms the main efficiency bottleneck of the entire protocol.

This situation changed in a recent line of work, initiated in [BCG$^+$17, BCGI18, BCG$^+$19b], that suggested a new approach. At the heart of the new approach is the following simple observation: by settling for generating a *pseudorandom* correlation, which is indistinguishable from the ideal target correlation even from the point of view of insiders, the offline communication can be sublinear in n while retaining the asymptotic and concrete efficiency advantages of the online protocol.

This approach was implemented through the notion of a *pseudorandom correlation generator* (PCG) [BCGI18, BCG$^+$19b]. A PCG enables two or more parties to *locally* stretch short correlated seeds into long pseudorandom strings that emulate a specified ideal target correlation, such as n instances of the OT correlation. This was recently extended to the notion of a *pseudorandom correlation function* (PCF) [BCG$^+$20b], which essentially emulates random access to exponentially many PCG outputs, analogously to the way a standard pseudorandom function (PRF) extends a standard pseudorandom generator (PRG).

Generating pseudorandom correlations: a template. To construct these primitives, a general template was put forth in [BCGI18], and further refined in subsequent works. At a high level, the template combines two key ingredients: a method to generate a *sparse* version of the target correlation, and a carefully chosen linear code where the syndrome decoding problem is conjectured to be intractable. To give a concrete example, let us focus on the *vector oblivious linear evaluation* (VOLE) correlation, which is in a sense a minimal step above simple linear correlations. The correlation distributes $(\vec{u}, \vec{v})$ to Alice and $(\Delta, \vec{w})$ to Bob; here, $\vec{u}, \vec{v}, \vec{w}$ are length-n vectors over a finite field $\mathbb{F}$ and $\Delta \in \mathbb{F}$ is a scalar, all chosen at random subject to satisfying the correlation $\vec{w} = \Delta \cdot \vec{u} + \vec{v}$. Among other applications [DIO20, BMRS21, YSWW21, RS21b], VOLE is an appealing target correlation because (a simple variant of) VOLE can be locally converted into n pseudorandom instances of OT correlation using a suitable hash function [IKNP03, BCG$^+$19b].

For the first ingredient, there is a simple construction that allows generating (from short seeds) pairs $(\vec{u}, \vec{v})$ and $(\Delta, \vec{w})$ as above, but where $\vec{u}$ is a random *unit* vector. This uses a *puncturable pseudorandom function* (PPRF), a type of PRF where some keys can be restricted to hide the PRF value at a fixed point.

A bit more concretely, $\vec{v}$ and $\vec{w}$ will be generated by evaluating the PRF on its entire domain; the missing value will be at the only position i where $u_i \neq 0$, and the party with the punctured key will fill it using a share of $\mathsf{PRF}_K(i) + \Delta \cdot u_i$. Such a PPRF can be efficiently constructed from any length-doubling PRG [GGM86, KPTZ13, BW13, BGI14]. With a t-fold repetition of this process (keeping Δ the same across all instances), after locally summing their expanded vectors, the parties obtain the target correlation, where $\vec{u}$ is t-sparse. As long as t remains small, the seed size is small as well.

The aim of the second ingredient is to transform this sparse correlation into a *pseudorandom* correlation. To this end, the parties multiply their vectors with a public compressing matrix H, obtaining $(H \cdot \vec{u}, H \cdot \vec{v})$ and $(\Delta, H \cdot \vec{w})$. When H is random, $H \cdot \vec{u}$ is pseudorandom: this is exactly the dual variant of the *learning parity with noise* (LPN) assumption [BFKL94, IPS09]. However, computing $H \cdot \vec{v}$ (or $H \cdot \vec{w}$) takes time $\Omega(n^2)$. When n is in the millions, as in typical MPC applications, this is clearly infeasible. A better approach is to sample H from a distribution such that (1) $H \cdot \vec{u}$ is still plausibly pseudorandom, and yet (2) the mapping $\vec{v} \mapsto H \cdot \vec{v}$ can be computed efficiently, ideally in time $\tilde{O}(n)$ or even $O(n)$.

The quest for the right code. In essence, all previous works in this area [BCGI18, BCG+19b, BCG+19a, SGRR19, BCG+20b, YWL+20, BCG+20a, CRR21] have built upon this template, sometimes for more general classes of correlations [BCG+20b], sometimes to achieve the more flexible notion of PCF [BCG+20a], or trying to strike the best balance between security and efficiency [BCGI18, BCG+19a, CRR21]. At the heart of all these works is, every time, a careful choice of which linear code to use. In [BCGI18, BCG+19a], it is suggested that relying on LDPC codes or on quasi-cyclic codes provides a reasonable balance between security (since the underlying LPN assumptions are well studied [Ale03, ABB+20]) and efficiency. In contrast, [CRR21] advocates a more aggressive choice, building a new concrete linear code, highly optimized for correlated randomness generation, guided by heuristic considerations and extensive computer simulations. Taking a different route, [BCG+20a] shows how a newly defined family of *variable density* linear codes allows generating a virtually unbounded amount of correlated randomness on demand, and [BCG+20b] generates more general correlations using an LPN variant over polynomial rings.

These works demonstrate that with a careful choice of code, silent preprocessing can have an extremely high throughput [CRR21] (as fast as generating tens of millions of pseudorandom oblivious transfers per second on one core of a standard laptop with low communication costs), broad expressiveness [BCG+20b] (handling richer correlations which are crucial in some advanced MPC protocols [ANO+21, RS21a]), and advantageous flexibility [BCG+20a] (generating any amount of correlated randomness on demand). Nonetheless, on most aspects, this area of research is in its infancy. Some important correlations remain frustratingly out of reach, such as circuit-dependent correlations (used e.g. in [DNNR17, HOSS18, WRK17b, Cou19, BGI19]), or authenticated multiplication triples over $\mathbb{F}_2$ (used in [HSS17, WRK17a]). The current fastest construction [CRR21] lacks any clear theoretical security analysis, but constructions built on firmer grounds

are an order of magnitude slower. Finally, the PCFs of [BCG+20a] are only realistically usable in a regime of parameters where they lack any security analysis.

The quest for constructions with clear, rigorous security arguments *and* very high concrete efficiency remains largely open; its fulfilment, we believe, is a promising path towards making MPC truly efficient on a large scale.

1.1 Our Contributions

In this work, we push forward the study of efficient generation of correlated randomness, significantly improving over the state of the art on several fronts. Our main contributions are threefold.

Expand-accumulate codes. We put forth a new simple family of linear codes, called *expand-accumulate codes* (EA codes), which are related to the well-studied class of repeat-accumulate codes [DJM98]. To encode a message with an EA code, a sparse degree-ℓ expander is first applied to the input, effectively replicating each message entry a small number of times; the result is then *accumulated* by computing the sum of all prefixes. We demonstrate that such an EA code is a particularly appealing choice of linear code in the context of generating correlated pseudorandomness, which uniquely combines multiple attractive features: firm security foundations, extremely high concrete efficiency, and a high level of *parallelization* and *cache-friendliness*. Furthermore, the special structure of EA codes allows us to obtain several advanced constructions, including PCFs (with better efficiency and security foundations compared to [BCG+20a]), and the first practical PCGs for useful correlations such as circuit-dependent correlations. In more detail:

1. We formally prove that the (dual-)LPN assumption for EA codes, denoted EA-LPN, cannot be broken by a large class of attacks, which captures in particular all relevant known attacks on LPN. Our analysis comes with concrete, usable security bounds for realistic parameters. In contrast, previous works either only achieved provable bounds in a purely asymptotic sense [BCG+20a] (with poor concrete efficiency), or heuristically extrapolated plausible parameters through computer simulations on small instances [CRR21].
2. We also derive sets of more aggressive parameters through heuristics and simulations to obtain apple-to-apple efficiency comparisons with the work of [CRR21]. We show that EA codes are highly competitive with the code of [CRR21], while having a much simpler structure (hence simpler to implement and more amenable to analysis).
3. When implemented in an "offline-online" mode, PCGs built from EA-LPN are *highly parallelizable*, allowing for simultaneously achieving low latency and high throughput. This stands in stark contrast with essentially *all* previous constructions, including the recent high-throughput construction from [CRR21].[1]

[1] A notable exception is the "primal" PCG construction of [BCGI18], which is also parallelizable. However, this PCG is limited to quadratic stretch; in practice, this makes it less efficient than other alternatives, even when using the bootstrapping approach from [YWL+20].

Hence, over multicore architectures, we expect our new PCG to outperform all alternatives by a large margin.

4. We obtain the first practical PCG constructions for different kinds of useful correlations including circuit-dependent correlations (which show up in communication-efficient MPC protocols [DNNR17, HOSS18, WRK17b, Cou19, BGI19] and in constant-round MPC protocols based on garbled circuits [WRK17b]). Generating n bits of correlations with our construction requires $O(n \log^2 n)$ work. In contrast, the only known previous approaches either use LPN but incur a prohibitive $\Omega(n^2)$ cost [BCG+19b] or require expensive high-end cryptographic primitives, such as multi-key threshold FHE [DHRW16, BCG+19b].
5. Finally, we construct a pseudorandom correlation *function* from the EA-LPN assumption, the first such construction to be both concretely efficient and standing on firm security arguments. The only other practically feasible constructions of PCFs are the variable-density construction of [BCG+20a] (which is much slower, even for aggressive parameters, and only has asymptotic security guarantees) and the recent construction of [OSY21] (which relies on the standard DCR assumption, but is also slower, is restricted to OT and VOLE correlations – our construction can handle other useful correlations – and is not post-quantum – our construction plausibly is).

Offline-online pseudorandom correlation generators. PCGs allow one to expand, in a "silent" fashion (*i.e.* without any communication), short seeds into long sources of correlated pseudorandomness. This silent expansion largely dominates the overall computational cost of the entire protocol: in the online phase, the computation amounts to a few cheap `xor` operations per gate, and the limiting factor is communication. Even with a very high bandwidth, the latency of multi-round protocols can form a bottleneck. This implies that, in many settings, some idle computation time is wasted during the online phase. We put forth a new notion of PCG, called *offline-online* PCG, which seeks to push the vast majority of the offline work back to the online phase, but in an incremental fashion that minimizes latency.

In more detail, most of the computational slowdown in the silent expansion of modern PCGs is incurred by cache misses. Indeed, most of the efficiency improvements of the PCG of [CRR21] come precisely from heuristically building a cache-friendly linear code. However, constructing such cache-friendly codes with firm security foundations remains elusive. Moreover, the cache-friendliness of the construction from [CRR21] comes at the expense of a fully sequential silent expansion. Instead, we suggest a new approach: using EA codes, cache misses are bound to occur because of their expander-based structure; however, it is relatively easy to push all these cache misses to the online phase, where they will happen during idle moments (caused by bandwidth limitations or latency). Concretely, EA codes achieve the following:

– In the offline phase, the *sparse version* of the correlation is generated using a PPRF; this amounts to computing a few hundred binary trees of hashes (*a*

la GGM), which is highly parallelizable and cache-friendly. In the literature, this is typically referred to as the *full evaluation* part, because it amounts to evaluating several PRRFs on their entire domain.
- Still in the offline phase, an *accumulation step* is performed, which converts a vector $(x_1, x_2, \cdots, x_N)$ in an accumulated vector $(x_1, x_1 \oplus x_2, \cdots, \bigoplus_{i=1}^{N} x_i)$. This can be done with $N-1$ `xor`s of short strings in one pass, which is extremely fast and cache-friendly; furthermore, this accumulation is easy to parallelize with a simple two-pass algorithm while still retaining cache-friendliness.
- At the end of the offline phase, a length-N vector $\vec{y}$ of short strings is stored, where N is a small constant factor times the target amount n of correlations (concretely, $N \approx 5 \cdot n$ in our instantiations). Eventually, to generate an instance of the target correlation, one must retrieve ℓ random entries of $\vec{y}$, and `xor` them, where the *output locality* parameter ℓ corresponds to the degree of the graph defining the EA code. This is where cache misses can occur; however, this step is still highly parallelizable, and these random accesses can easily be arranged to fill exactly the idle computation time of the online phase. Concretely, for conservative parameters fully within the bound of our theoretical analysis, ℓ can be set to about 40 when producing $n \geq 2^{20}$ correlations; using more aggressive parameters, setting ℓ as low as 7 seems to nonetheless provide a sufficient security level according to our experiments.

Our estimates suggest that relying on offline-online PCGs instead of standard PCGs will likely lead to significant improvements in MPC protocols. The offline part of our EA-based offline-online PCGs is insanely fast – we estimate of the order of 100ms to generate the offline material for 10 million random OTs on a single core of a standard laptop, a runtime which can be sped up by almost a factor of k when k processors are available, even with a few dozen processors.

Further speedups in the offline phase. Up to this point, we discussed the application of a new family of linear codes to speed up PCGs and achieve new advanced constructions. We now turn our attention to the other main component of a PCG: the *full evaluation* procedure, which boils down to evaluating several PPRFs on their entire domain. Concretely, using the GGM PPRF [GGM86, KPTZ13, BW13, BGI14], generating a length-N vector with this procedure requires $2N$ calls to a hash function along the leaves of a full binary tree. We obtain new PPRF constructions that aim to reduce the total number of calls to the underlying hash function. Our main construction reduces the number of calls to $1.5N$; we prove its security in the random oracle model. We also put forth a candidate construction with the same $1.5N$ cost in the ideal cipher model (supporting an implementation based on standard block ciphers such as AES), but leave its security analysis open. We describe several additional optimizations; in particular:

- We show that, by "flattening the GGM tree," the number of calls can be further reduced, at the cost of sightly increasing the seed size (and seed distribution cost). Concretely, we can reduce the total number of calls to $1.17N$, only increasing the seed size and seed distribution time by a factor

of 1.5 (this is a desirable tradeoff, since these costs vanish when N increases, and are typically marginal with standard parameters).
– We show that, in the specific context of generating OT correlations, the cost can be further reduced to N (without the flattening optimization) or $0.67N$ (with flattening) calls to the hash function.

We note that these contributions are of a very different nature compared to our previous constructions, and add to the growing body of work on the analysis in idealized models of symmetric primitives for MPC applications [GKWY20, CT21]. Since full evaluations of PPRFs have many applications beyond PCGs, to problems such as zero-knowledge proofs [KKW18, CDG+20, KZ20, FS21], circuit garbling [HK21], secure shuffling [CGP20] and private information retrieval [MZR+13], these results are also of independent interest.

1.2 Technical Overview

We now survey the technical tools that we use to achieve our results.

EA *codes.* A generator matrix H for an EA code is of the form $H = BA$, where $B \in \mathbb{F}_2^{n \times N}$ is a matrix with sparse rows, and $A \in \mathbb{F}_2^{N \times N}$ is the accumulator matrix, that is, $\vec{x}^\intercal A = (x_1, x_1 + x_2, \ldots, x_1 + \cdots + x_N)$. We propose the EA-LPN-assumption which states that samples of the form $H\vec{e}$, where $\vec{e} \in \mathbb{F}_2^N$ is a random sparse vector, are computationally indistinguishable from uniform.

In order to provide evidence for the EA-LPN-assumption, we show that it is not susceptible to *linear tests*. While this class of tests is very large, they all boil down to the same general strategy: given the vector $\vec{b}$ (which is either uniformly random or $H\vec{e}$ with $\vec{e}$ sparse), one looks at the matrix H, chooses some nonzero vector $\vec{x} \in \mathbb{F}_2^n$, and then checks if the dot product $\vec{x}^\intercal \cdot \vec{b}$ is biased towards 0. If $\vec{x}^\intercal H$ and $\vec{e}$ are both sufficiently dense then we can rule out the possibility that $\vec{x}^\intercal \cdot (H\vec{e}) = (\vec{x}^\intercal H) \cdot \vec{e}$ has noticeable bias. As we would like to keep $\vec{e}$ as sparse as possible, we need to show that for every nonzero vector $\vec{x} \in \mathbb{F}_2^n$, $\vec{x}^\intercal H$ has large weight. In other words, we need to show that the code generated by H has good minimum distance.

We now briefly outline how we show that a random EA code has good minimum distance. It is convenient for us to assume that the coordinates of B are all sampled independently as Bernoulli random variables with probability p. Writing $(y_1, \ldots, y_N) := \vec{x}^\intercal H = \vec{x}^\intercal (BA)$ we can view the sequence of $y_1, \ldots, y_N$ as an N-step random walk (over the randomness of B) on a Markov chain with state space $\{0, 1\}$, where the transition probabilities are governed by the Hamming weight $\mathcal{HW}(\vec{x})$. Furthermore the *spectral gap* of this Markov chain is easily computable, allowing us to apply an *expander Hoeffding bound* which tells us that the random walk $y_1, \ldots, y_N$ is unlikely to spend too much time on the 0 state; equivalently, it is unlikely that $\mathcal{HW}(\vec{y}^\intercal) = \mathcal{HW}(\vec{x}^\intercal H)$ is too small. By taking a union bound over all nonzero vectors $\vec{x} \in \mathbb{F}_2^n$ and doing a case-analysis based on $\mathcal{HW}(\vec{x})$, we can show that so long as $p = \Omega(\log N/N)$, except with probability $1 - 1/\mathsf{poly}(N)$ the code has minimum distance $\Omega(N)$. If one desires

negligible in N failure probability this can also be obtained by slightly increasing p: e.g., $p = \Omega(\log^2 N/N)$ suffices to guarantee $n^{-O(\log N)}$ failure probability. Further, we can show that this this analysis is (asymptotically) tight.

Offline-online PCGs from EA *codes.* We introduce the notion of offline-online PCGs, where an offline and online key are generated. Each party σ uses its offline key to generate a local offline string Y_σ from which it can later use its online key to generate a (vector of) samples from the target correlation. We call the length of Y_σ the *storage cost*, and the number of entries that must be read from Y_σ to generate a single sample the *output locality*.

Recall that the goal of VOLE is to obtain correlations $((\vec{u}, \vec{v}), (\Delta, \vec{w}))$, where $\vec{u}, \vec{v}, \vec{w}$ are length-n vectors over a finite field $\mathbb{F}$ and $\Delta \in \mathbb{F}$ is a scalar, all chosen at random subject to satisfying the correlation $\vec{w} = \Delta \cdot \vec{u} + \vec{v}$. Using PPRFs, during the offline phase the parties expand their keys to obtain strings $\vec{w}', \vec{u}', \vec{v}' \in \mathbb{F}^N$, where $\vec{u}'$ is a sparse, $\mathsf{EA\text{-}LPN}$ noise vector and $\vec{w}', \vec{v}'$ are pseudorandom conditioned on satisfying $\vec{w}' = \Delta \cdot \vec{u}' + \vec{v}'$. Further, the parties already perform the accumulation step and output $\vec{u}^{\mathsf{off}} = A \cdot \vec{u}'$ and $\vec{v}^{\mathsf{off}} = A \cdot \vec{v}'$, and $\vec{w}^{\mathsf{out}} = A \cdot \vec{w}'$ and Δ, respectively. In the online phase, the parties can then recover a tuple $((u_i, v_i), (\Delta, w_i))$ by checking only an expected number of $p \cdot N$ of the offline strings, resulting in an online locality of $p \cdot N$. We can thereby obtain offline-online PCGs with highly parallelizable and cache-friendly offline phase, and online phase with low locality (recall that we can choose p as low as $p = c \cdot \log N/N$, thereby resulting in $\ell = c \cdot \log N$).

PCFs from EA *codes.* We have already described a general recipe for using compressing matrices H for which the LPN assumption plausibly holds to construct PCGs; indeed, we even sketched an offline-online PCG. However, in order to use EA codes to obtain PCFs, more care is required. Recall that a PCF must behave in an incremental fashion, using the short correlated seeds to provide as many pseudorandom instances of the target correlation as required. The main challenge is that to obtain a PCF we need to set N to be superpolynomial in the security parameter, and thus computing matrix-vector products of the form $A \cdot \vec{e}$ is too expensive. Fortunately, we can avoid the need to explicitly compute $A \cdot \vec{e}$ by appealing to *distributed comparison functions* (DCFs). DCFs, which can be constructed with PRGs as is the case for distributed point functions [BGI16, BCG$^+$21], allow one to efficiently share a comparison function $f^{\beta}_{<\alpha} : [N] \to \mathbb{F}$ which maps every $x < \alpha$ to β and every $x \geq \alpha$ to 0. When the noise $\vec{e}$ has a *regular* structure (i.e., it consists of N/t unit vectors concatenated together) one can naturally view $A \cdot \vec{e}$ (after permuting the coordinates) as a concatenation of comparison functions. We furthermore observe that for constructing PCFs for VOLE and OT we can use a *relaxed* version of a DCF, denoted RDCF, as we only require α to be hidden from one of the two parties.

In the following we give a high-level overview of our RDCF construction. For simplicity we assume we want to share a comparison function with range $(\{0,1\}^\lambda, \oplus)$, although our construction generalizes to arbitrary abelian groups $(\mathbb{G}, +)$. Our construction follows the spirit of the DCF construction of [BCG$^+$21],

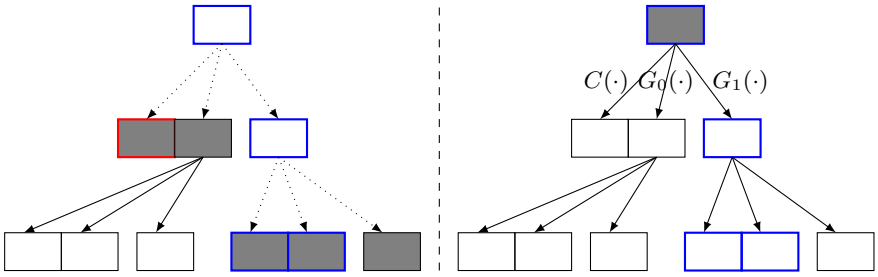

Fig. 1: Pictorial representation of our relaxed DCF construction. In our example the path $\alpha = 10 \in \{0,1\}^2$ is marked in blue, the box marked in red corresponds to box where β is added, and the boxes filled in gray correspond to the key of P_0 (in knowledge of α) and P_1, respectively.

but one party knowing α allows for significant savings. We build on PRGs $G_0, G_1, C\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ such that the concatenation of the three is a secure PRG. In Figure 1 we give a pictorial representation of the relaxed DCF construction, which we explain in the following.

To evaluate the RDCF on an input x, one traverses the tree and adds up all "C" values on the path from the root to the corresponding leaf and finally adds the "G" value of this leaf. The idea is that P_0 will add β (blinded by a "C" value) to the output, if and only if it leaves the path defined by α to the *left* (which happens if and only if $x < \alpha$). For concreteness, say one wants to evaluate the RDCF in Figure 1 on input $x = 00$. Then, both parties add the first box on the second level (the first "C" value, marked in red), the first box on the third level (the second "C" value) and the second box on the third level (the "G" value of the leaf), which P_1 can both derive from its key. The corresponding output shares add up to β as required, since β is added to the first "C" value held by P_0. Further, β remains hidden from P_0 by the pseudorandomness of the PRG C.

One of our improvements compared to the DCF construction of [BCG+21] is that we observe that we only need "C" values on the left children, since only there the β value has to be hidden potentially. This leads to shorter keys and savings on the number of PRG evaluations.

Overall, comparing with the standard DCF construction of [BCG+21] where each key is of size $2 \log N(\lambda + \log |\mathbb{F}|)$, in our RDCF one of the party's keys is only of size λ, and the other is roughly half the size of [BCG+21]. Further, our construction reduces the number of calls to AES (when using this to implement a PRG) by 25% on average.

Additionally, we show that in the setting where a full evaluation is feasible (i.e., where one is interested in an iterative PCG rather than a full-fledged PCF), the keys of our RDCF can be distributed in 2-PC with a simple, 2-round protocol based on 2-round OT following the techniques of [Ds17, BCG+19a], whereas the corresponding distributed setup protocol for the DCF construction of [BCG+21] would require $\log N$ rounds.

PCF Constructions. Given our relaxed DCF, we readily obtain a PCF for the subfield VOLE correlation, which also implies a PCF for oblivious transfer when combined with a suitable hash function [BCG+20a]. We also show how to build a PCF for general degree-2 correlations: in particular, we get a two-party PCF for authenticated multiplication triples over any ring R, and can also support general, circuit-dependent correlations. For this, instead of comparison functions, we need a way to secret share the *product* of comparison functions. Fortunately, this can be done using function secret sharing for 2-dimensional interval functions [BGI16], based on any PRG.

We show that our $\mathsf{EA\text{-}LPN}$-based PCFs can obtain good concrete efficiency. With conservative parameter choices, which our simulated experiments show resist linear attacks, our PCF for VOLE has comparable key size to the most aggressive variant of the PCF from [BCG+20a] (which did not have any provable security analysis), while we need around an order of magnitude less computation. For our degree-2 PCFs, to get good concrete efficiency we need to rely on more aggressive $\mathsf{EA\text{-}LPN}$ parameters with a lower noise weight. With this, our PCFs for VOLE/OT have key sizes of under 1MB, and takes only a few thousand PRG evaluations to compute each output. Our PCF for general degree-2 correlations (including multiplication triples, matrix triples and circuit-dependent correlations) has key sizes of around 200MB, and requires 2–3M PRG evaluations per output. The degree-2 PCF from [BCG+20a] does not come close to this level of efficiency, since it is not compatible with the most efficient variant of LPN they use.

Speeding up the offline phase. The final task that we set for ourselves is to improve the runtime of the offline phase for PCGs, where the offline phase requires evaluating several punctured PRFs (PPRFs) on their entire domain, a functionality called $\mathsf{FullEval}$. As alluded to earlier we apply the GGM construction to obtain a PPRF from a hash function. The standard way to do this is as follows: given hash functions H_0, H_1 (which can be modeled as random oracles (RO)) one generates the GGM tree corresponding to secret key k, that is, the depth m binary tree where the root is labeled by k and the left and right child of a node labeled by x are labeled by $\mathsf{H}_0(x)$ and $\mathsf{H}_1(x)$, respectively.To puncture a key at a point $\alpha \in \{0,1\}^m$ one gives the values of the nodes of the *co-path*, *i.e.*, the siblings of each node appearing on the path indexed by α.

To save on the calls to the hash function we consider the following definition: given an RO H we define $\mathsf{H}_0(x) = \mathsf{H}(x)$ and $\mathsf{H}_1(x) = \mathsf{H}(x) \oplus x$. Note that this clearly fails to give a PPRF, as given $\mathsf{H}(x)$ and $\mathsf{H}(x) \oplus x$ one can recover x and thereby distinguish the value at the punctured point from random. Nonetheless, we can show that the resulting construction yields a weaker primitive that we call a *strong unpredictable punctured function* (strong UPF), which informally means that given a key punctured at a point α one essentially cannot predict the value at α any better than by randomly guessing. While this primitive is weaker, we note that it already suffices for some applications (such as PCGs for OT), reducing the number of necessary calls to the random oracle for a full evaluation by half. If one subsequently hashes the right child at the leaves, we can further show that this does yield a genuine PPRF. In this way, we require

only $1.5N$ calls to the hash function for a $\mathsf{FullEval}$, whereas the standard GGM approach requires $2N$ calls, providing us with a 25% cost reduction.

To prove the construction yields a strong UPF, we observe that the punctured key can be equivalently sampled by choosing random values for the co-path and then programming the random oracle so as to be consistent with these choices. Assuming there are no collisions, such a punctured key is then independent of the value of the function at α, so the only way for an adversary to learn the value at α is if it happens to query H at one of m values on the path that it does not see.

To increase our savings, we consider k-ary trees for $k > 2$, which informally corresponds to "flattening" the GGM tree. This does incur a $(k-1)\log_2 k$ factor increase in the size of the punctured key; however, with the standard GGM construction of a PPRF the number of calls to the hash function in an invocation of $\mathsf{FullEval}$ now drops to $(1 + 1/(k-1))N$. By combining this with the first optimization, when $k = 3$ we can decrease the number of calls to H to $1.33N$, and when $k = 4$ to $1.17N$.

Lastly, given the current hardware support of AES, we also put forward a candidate construction of a weaker notion of UPF given an ideal invertible permutation. Recall that the standard strategy to construct a hard-to-invert function from an invertible permutation is via the Davies-Meyer construction, where H is defined as $\mathsf{H}(k) := P(k) \oplus k$ for an invertible permutation P. Unfortunately, instantiating H this way clearly breaks down with our previous construction, as $\mathsf{H}_1(k)$ would become equal to $P(k)$, and hence be invertible. Instead, the idea of the construction is to set $\mathsf{H}_0(k) := \mathsf{H}(k) \oplus k$ and $\mathsf{H}_1(k) := \mathsf{H}(k) + k \bmod 2^\lambda$. While on first glance one might seem easy to predict given the other, we show that this is not the case, thereby giving some evidence that the corresponding candidate indeed achieves unpredictability. We cannot hope to achieve the same strong notion of unpredictability as we do with our random oracle construction though, since $\mathsf{H}(k) \oplus k$ does in general leak some information about $\mathsf{H}_1(k) := \mathsf{H}(k) + k \bmod 2^\lambda$. Still, by subsequently hashing the right child at all leaves standard unpredictability would be sufficient to obtain a true PPRF, thereby yielding a 25% cost reduction for PPRF constructions implemented with fixed-key AES. We leave the full analysis of the construction to future work.

1.3 Roadmap

We start by giving preliminaries in Section 2. In Section 3 we present EA codes and provide a security analysis of $\mathsf{EA\text{-}LPN}$. In Section 4 we provide new constructions of PCFs based on the $\mathsf{EA\text{-}LPN}$ assumption. Finally, in Section , we give a brief overview of optimizations for the offline costs of PCG constructions. For a formal definition of offline-online PCGs and a construction of offline-online PCGs for subfield VOLE from EA codes, we refer to the full version of this paper.

2 Preliminaries

For preliminaries on bias and Markov chains we refer to the full version of this paper.

2.1 Learning Parity With Noise and LPN-Friendly Codes

We define the LPN assumption over a ring $\mathcal{R}$ with dimension n, number of samples N, w.r.t. a code generation algorithm $\mathbf{C}$, and a noise distribution $\mathcal{D}$:

Definition 1 (Dual LPN). *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{n,N}(\mathcal{R})\}_{n,N\in\mathbb{N}}$ denote a family of efficiently sampleable distributions over a ring $\mathcal{R}$, such that for any $n, N \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{n,N}(\mathcal{R})) \subseteq \mathcal{R}^N$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(n, N, \mathcal{R})$ outputs a matrix $H \in \mathcal{R}^{n\times N}$. For dimension $n = n(\lambda)$, number of samples (or block length) $N = N(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the (dual) $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(n, N)$ assumption states that*

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}(n, N, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}_{n,N}(\mathcal{R}), \vec{b} \leftarrow H \cdot \vec{s}\}$$
$$\overset{\mathrm{c}}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}(n, N, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^N\}.$$

Note that the generator matrix H sampled from $\mathbf{C}$ is used in the reverse direction compared to encoding: a codeword is a vector $\vec{x} \cdot H$, where $\vec{x} \in \mathcal{R}^{1\times n}$, while the assumption is about vectors of the form $H \cdot \vec{e}$ for $\vec{e} \in \mathcal{R}^N$. The dual LPN assumption is also called the *syndrome decoding assumption* in the code-based cryptography literature; in this case, H is typically seen as the *parity-check matrix* of a code generated by a matrix G such that $H \cdot G = 0$. The dual LPN assumption as written above is equivalent to the (perhaps more common) *primal* LPN assumption with respect to G (a matrix $G \in \mathcal{R}^{N\times N-n}$ such that $H \cdot G = 0$), which states that $G \cdot \vec{s} + \vec{e}$ is indistinguishable from random, where $\vec{s} \xleftarrow{\$} \mathcal{R}^{N-n}$ and $\vec{e} \xleftarrow{\$} \mathcal{D}_{n,N}(\mathcal{R})$; the equivalence follows from the fact that $H \cdot (G \cdot \vec{s} + \vec{e}) = H \cdot \vec{e}$.

We say that a family of codes sampled by a code generation algorithm $\mathbf{C}$ is *LPN-friendly* when instantiating the general LPN assumption with these codes leads to a secure flavor of the assumption for standard noise distributions. Of course, when we call a code "LPN-friendly", this implicitly means "plausibly LPN-friendly in light of known cryptanalysis of LPN".

Examples of noise distributions. Several choices of noise distribution are common in the literature. Fix for example $\mathcal{R} = \mathbb{F}_2$ (all the distributions below generalize to other structures) and a parameter t which governs the average density of nonzero entry in a random noise vector. Then the following choices are standard:

- Bernoulli noise: the noise vector $\vec{e}$ is sampled from $\mathsf{Ber}^N_{t/N}(\mathbb{F}_2)$. This is the most common choice in theory papers.
- Exact noise: the noise vector $\vec{e}$ is a uniformly random weight-t vector from $\mathbb{F}_2^N$; let us denote $\mathsf{HW}^N_t(\mathbb{F}_2)$ this distribution. This is the most common choice in concrete LPN-based constructions.
- Regular noise: the noise vector $\vec{e}$ is a concatenation of t random unit vectors from $\mathbb{F}_2^{N/t}$; let us denote $\mathsf{Reg}^N_t(\mathbb{F}_2)$ this distribution. This is a very natural choice in the construction of pseudorandom correlation generators as it significantly improves efficiency [BCGI18, BCG+19b, BCG+19a] without harming security.

Examples of LPN-friendly codes. Over the years, many codes have been conjectured to be LPN friendly. Common choices include setting H to be a uniformly random matrix over $\mathbb{F}_2$ (this is the standard LPN assumption), the generating matrix of an LDPC code [Ale03] (often called the "Alekhnovich assumption"), a quasi-cyclic code (used in several recent submissions to the NIST post-quantum competition [ABB+17, AMBD+18, MAB+18] and in previous works on pseudorandom correlation generators, such as [BCG+19a]), Toeplitz matrices [GRS08, LM13] and many more. All these variants of LPN generalize naturally to larger fields (and LPN is typically believed to be at least as hard, if not harder, over larger fields).

When designing new LPN-based primitives, different choices of code lead to different performance profiles. Established codes, such as those listed above, have the advantage of having been analyzed by experts for years or decades; however, it might happen in some applications that all established codes lead to poor performance. Plausibly secure but yet-unstudied codes could yield considerable performance improvements. In light of this, we require a heuristic to select plausibly LPN-friendly codes. Such a heuristic has been implicit in the literature for some time, and was put forth explicitly in recent works [BCG+20a, CRR21].

From large minimum distance to LPN-friendliness. The core observation is that essentially all known attacks (attacks based on Gaussian elimination and the BKW algorithm [BKW00, Lyu05, LF06, EKM17] and variants based on covering codes [ZJW16, BV16, BTV16, GJL20], information set decoding attacks [Pra62, Ste88, FS09, BLP11, MMT11, BJMM12, MO15, EKM17, BM18], statistical decoding attacks [AJ01, FKI06, Ove06, DAT17], generalized birthday attacks [Wag02, Kir11], linearization attacks [BM97, Saa07], attacks based on finding low weight code vectors [Zic17], or on finding correlations with low-degree polynomials [ABG+14, BR17]) fit in a common framework of *linear tests* which corresponds, roughly, to attacks where an adversary tries to detect a bias in the LPN samples by computing a linear function of these samples. (The choice of the linear function itself can depend arbitrarily on the code matrix.) Then, it is relatively easy to show that for any noise distribution $\mathcal{D}$ whose nonzero entries "hit any large subset" with high enough probability, the LPN assumption with respect to a code generator $\mathbf{C}$ and $\mathcal{D}$ provably resists (exponentially) all linear tests as long as a random code from $\mathbf{C}$ has high minimum distance with good probability. This is formalized below.

Definition 2 (Security against Linear Tests). *Let $\mathcal{R}$ be a ring, and let $\mathcal{D} = \{\mathcal{D}_{n,N}\}_{n,N\in\mathbb{N}}$ denote a family of noise distributions over $\mathcal{R}^N$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(n, N)$ outputs a matrix $H \in \mathcal{R}^{n\times N}$. Let $\varepsilon, \eta : \mathbb{N} \mapsto [0, 1]$ be two functions. We say that the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(n, N)$ is* (ε, η)-secure against linear tests *if for any (possibly inefficient) adversary $\mathcal{A}$ which, on input H outputs a nonzero $\vec{v} \in \mathcal{R}^n$, it holds that*

$$\Pr[H \xleftarrow{\$} \mathbf{C}(n, N), \vec{v} \xleftarrow{\$} \mathcal{A}(H) \ : \ \mathsf{bias}_{\vec{v}}(\mathcal{D}_H) \geq \varepsilon(\lambda)] \leq \eta(\lambda),$$

where $\mathcal{D}_H$ denotes the distribution induced by sampling $\vec{e} \leftarrow \mathcal{D}_{n,N}$, and outputting the LPN samples $H \cdot \vec{e}$.

The *minimum distance* of a matrix H, denoted $\mathsf{d}(H)$, is the minimum weight of a vector in its row-span. Then, we have the following straightforward lemma:

Lemma 3. *Let $\mathcal{D} = \{\mathcal{D}_{n,N}\}_{n,N\in\mathbb{N}}$ denote a family of noise distributions over $\mathcal{R}^N$. Let $\mathbf{C}$ be a probabilistic code generation algorithm. Then for any $d \in \mathbb{N}$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(n, N)$ assumption is (ε_d, η_d)-secure against linear tests, where*

$$\varepsilon_d = \max_{\mathsf{HW}(\vec{v})>d} \mathsf{bias}_{\vec{v}}(\mathcal{D}_{n,N}), \qquad \textit{and} \qquad \eta_d = \Pr_{H\xleftarrow{\$}\mathbf{C}(n,N)}[\mathsf{d}(H) \geq d].$$

For example, using a Bernoulli noise distribution of error rate t/N, for any $\vec{v}$ of weight at least d, it holds that $\mathsf{bias}_{\vec{v}}(\mathsf{Ber}^n_{t/N}(\mathbb{F}_2)) = (1-2t/N)^d/2 < e^{-2td/N}$; that is, if the relative distance d/N of the code is a constant (*i.e.* the code is a good code), the bias will decrease exponentially with t. Similar calculations show that for any $\vec{v}$ of weight at least d, $\mathsf{bias}_{\vec{v}}(\mathsf{Reg}^N_t) \leq (1-2(d/t)/(N/t))^t < e^{-2td/N}$.

When the minimum distance heuristic fails. From the above, one can be tempted to conjecture that any good code, say, together with Bernoulli noise, is LPN-friendly. However, this is known to fail in at least three situations:

1. When the code is strongly algebraic. For example, Reed-Solomon codes, which have a strong algebraic structure, have high minimum distance, but can be decoded efficiently with the Berlekamp-Massey algorithm, hence they do not lead to a secure LPN instance (and indeed, Berlekamp-Massey does not fit in the linear test framework).
2. When the noise is structured (which is the case e.g. for regular noise) and the adversary can see enough samples. This opens the door to algebraic attacks such as the Arora-Ge attack [AG11]. However, this typically requires a large number of samples: for example, using regular noise, one needs $N = \Omega((N-n)^2)$ for the attack to apply. In contrast, all our instances will have $N = O(N-n)$.
3. When $\mathcal{R}$ has a subring, one can always project onto the subring before performing a linear attack; this technically does not directly fit in the linear test framework. When analyzing security against linear test, one must therefore account for all subrings the attacker could first project the problem onto. In polynomial rings, the reducible case of cyclotomics is discussed in [BCG+20b]. In integer rings like $\mathbb{Z}_{2^k}$, one weakness is that projecting onto $\mathbb{Z}_2$ can make error values become zero with probability $1/2$ [LWYY22], reducing the effective noise rate. To fix this, [LWYY22] propose an alternative noise distribution that is provably as secure as LPN over $\mathbb{F}_2$, but with k times the noise rate. Alternatively, a plausible fix without increasing the noise rate is to choose error values to be invertible, which ensures they are non-zero in all subrings.

The above three scenarios are the only exceptions we are aware of. Hence the following natural rule of thumb: if a code is combinatorial in nature (it is not a strongly algebraic code, such as Reed-Solomon or Reed-Müller), and if the code rate is not too close to 1 (e.g. code rate $1/2$, i.e. $n = N/2$), then being a good code makes it a plausible LPN-friendly candidate.

2.2 Puncturable Pseudorandom Functions

Pseudorandom functions (PRF), introduced in [GGM86], are keyed functions which are indistinguishable from truly random functions. A *puncturable pseudorandom function* (PPRF) is a PRF F such that given an input x, and a PRF key k, one can generate a *punctured* key, denoted $k\{x\}$, which allows evaluating F at every point except for x, and does not reveal any information about the value $F.\mathsf{Eval}(k, x)$. PPRFs have been introduced in [KPTZ13, BW13, BGI14].

Definition 4 (t-Puncturable Pseudorandom Function). *A puncturable pseudorandom function (PPRF) with key space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$, is a pseudorandom function F with an additional punctured key space $\mathcal{K}_p$ and three probabilistic polynomial-time algorithms $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ such that*

- *$F.\mathsf{KeyGen}(1^\lambda)$ outputs a random key $K \in \mathcal{K}$,*
- *$F.\mathsf{Puncture}(K, \{S\})$, on input a ley $K \in \mathcal{K}$, and a subset $S \subset \mathcal{X}$ of size t, outputs a punctured key $K\{S\} \in \mathcal{K}_p$,*
- *$F.\mathsf{Eval}(K\{S\}, x)$, on input a key $K\{S\}$ punctured at all points in S, and a point x, outputs $F(K, x)$ if $x \notin S$, and $\perp$ otherwise,*

The security requirement is that for any set S, given a punctured key $K\{S\}$, the values $(F(K, x))_{x \in S}$ are pseudorandom.

In the full version of this paper, we recall the PPRF construction based on any length-doubling pseudorandom generator from [KPTZ13, BW13, BGI14].

2.3 Pseudorandom Correlation Generators and Functions

For a full definition of pseudorandom correlation generators and function, we refer to [BCG+19b] and [BCG+20a], or to the full version of this paper. In the following we only provide a sketch of the definitions.

Correlation. We say a PPT algorithm $\mathcal{Y}$ is a *correlation*, if $\mathcal{Y}$ on input 1^λ outputs a pair of strings $(y_0, y_1) \in \{0,1\}^{\tau_0} \times \{0,1\}^{\tau_1}$ where $\tau_0(\lambda), \tau_1(\lambda) \in \mathsf{poly}(\lambda)$ describe the output lengths.

The security definition of PCGs requires the target correlation to satisfy a technical requirement, which roughly says that it is possible to efficiently sample from the conditional distribution of y_0 given y_1 and vice versa. More precisely, we require the existence of a PPT algorithm $\mathsf{RSample}$ that on input (σ, y_σ) outputs $y_{1-\sigma}$, such that the distributions $\{(y_\sigma, y_{1-\sigma}) \mid (y_0, y_1) \leftarrow \mathcal{Y}(1^\lambda)\}$ and $\{(y_\sigma, y'_{1-\sigma}) \mid (y_0, y_1) \leftarrow \mathcal{Y}(1^\lambda), y'_{1-\sigma} \leftarrow \mathsf{RSample}(\sigma, y_\sigma)\}$ are statistically close. We call such a correlation generator *reverse-sampleable*.

By $\mathcal{Y}^n$ we define the algorithm outputting n instance according to $\mathcal{Y}$. We refer to such an algorithm also as *correlation generator*. Further, we extend $\mathsf{RSample}$ to input vectors R_σ of the form $R_\sigma = (y_\sigma^1, \ldots, y_\sigma^n)$ by applying $\mathsf{RSample}$ componentwise.

Pseudorandom correlation generator. If $\mathcal{Y}$ is a reverse-sampleable correlation generator, then a *pseudorandom correlation generator (PCG)* for $\mathcal{Y}$ with stretch n is a tuple of PPT algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$, such that the following holds

- $\mathsf{PCG.Gen}(1^\lambda)$ outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ on input of $\sigma \in \{0,1\}$ and a seed k_σ, deterministically outputs a bit string $R_\sigma \in (\{0,1\}^{\tau_\sigma})^n$.
- **Correctness.** The correlation obtained via:

$$\{(R_0, R_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), (R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma))_{\sigma=0,1}\}$$

  is computationally indistinguishable from $\mathcal{Y}^n(1^\lambda)$.
- **Security.** For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:

$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and}$$
$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}), R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\}$$

  where $\mathsf{RSample}$ is the reverse sampling algorithm for correlation $\mathcal{C}$.

Examples of correlations. A random OT correlation is a pair $(y_0, y_1) \in \{0,1\}^2 \times \{0,1\}^2$, where $y_0 = (u, v)$ for two random bits u, v, and $y_1 = (b, u \cdot b \oplus v)$ for a random bit b. OT correlations is perhaps the most common and fundamental type of correlation in secure computation (though many others – such as Beaver triples, authenticated Beaver triples, or function-dependent correlations – are also standard).

It is known that, to generate n pseudorandom OT correlations, it suffices to generate the following simpler correlation: Alice gets a (pseudo)random pair of length-n vectors $(\vec{u}, \vec{v})$, where $\vec{u} \xleftarrow{\$} \mathbb{F}_2^n$ and $\vec{v} \in \mathbb{F}_{2^\lambda}^n$, and Bob gets $x \xleftarrow{\$} \mathbb{F}_{2^\lambda}$ and $\vec{w} \leftarrow x \cdot \vec{u} + \vec{v}$. This correlation (known as the *subfield vector-OLE* correlation) can be locally converted by Alice and Bob into n pseudorandom OT correlations using a correlation-robust hash function; see [BCG+19b] for details.

For a general template to construct PCGs for VOLE from PPRFs and LPN-friendly codes, we refer to [BCG+19b], or the full version of this paper.

Pseudorandom correlation function. If $\mathcal{Y}$ is a reverse-sampleable correlation generator, then a *pseudorandom correlation function (PCF)* for $\mathcal{Y}$ with input length $\nu = \nu(\lambda) \in \mathbb{N}$ is a tuple of PPT algorithms $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ with the following syntax:

- $\mathsf{PCF.Gen}(1^\lambda)$ outputs a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$;
- $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ on input of $\sigma \in \{0,1\}$, a key k_σ and an input $x \in \{0,1\}^\nu$, deterministically outputs a tuple $(y_0, y_1) \in \{0,1\}^{\tau_0} \times \{0,1\}^{\tau_1}$.

Correctness and security are defined similarly to a PCG, except that instead of obtaining the complete (potentially exponential-sized) output the adversary gets to query the output string an arbitary polynomial number of times, to obtain a tuple $(x, \mathsf{Eval}(\sigma, \mathsf{k}_\sigma, x))$ (or the according reverse-sampled correlation) for x sampled uniformly at random.

3 Expand-Accumulate Codes

In this section we introduce expand-accumulate codes, which are defined by the product $H = BA$ for a sparse expanding matrix B and the accumulator matrix A. We conjecture that the LPN problem is hard to solve for this matrix ensemble and provide theoretical evidence for this conjecture by demonstrating that it resists linear attacks.

3.1 Expand-Accumulate Codes, and the EA-LPN Assumption

First, we formally define the accumulator matrix.

Definition 5 (Accumulator Matrix). *For a positive integer N and ring $\mathcal{R}$, the* accumulator matrix $A \in \mathcal{R}^{N \times N}$ *is the matrix with* 1*'s on and below the main diagonal, and* 0*'s elsewhere.*

In particular, if $A\vec{x} = \vec{y}$ with $\vec{x}, \vec{y} \in \mathcal{R}^N$, we have the following relations:

$$y_i = \sum_{j=1}^{i} x_j \ \forall i \in [N] \qquad\qquad y_i := x_i + y_{i-1} \ \forall 2 \leq i \leq N \ . \tag{1}$$

Note in particular that (1) guarantees that the vector-matrix product $A\vec{x}$ can be computed with only $N-1$ (sequential) ring addition operations. In particular, when $\mathcal{R}$ is the binary field $\mathbb{F}_2$, this requires just $N-1$ `xor` operations. Furthermore, this can be computed even more efficiently in *parallel*, which is a major benefit of our construction; please see the full version for more details. We now formally introduce *expand-accumulate* (EA) codes, which underline our main constructions of offline-online PCGs.[2]

Definition 6 (Expand-Accumulate (EA) codes). *Let $n, N \in \mathbb{N}$ with $n \leq N$ and let $\mathcal{R}$ be a ring. For a desired density $p \in (0,1)$, a generator matrix for an* expand-accumulate (EA) code *is sampled as follows:*

[2] These codes are heavily inspired by *repeat-accumulate* codes; the full version elaborates further on this point.

– *Sample row vectors* $\vec{r}_1^\intercal, \vec{r}_2^\intercal, \ldots, \vec{r}_n^\intercal \xleftarrow{\$} \mathsf{Ber}_p^N(\mathcal{R})$ *independently and put*

$$B = \begin{bmatrix} \text{———} & \vec{r}_1^\intercal & \text{———} \\ \text{———} & \vec{r}_2^\intercal & \text{———} \\ \vdots & \vdots & \vdots \\ \text{———} & \vec{r}_n^\intercal & \text{———} \end{bmatrix} .$$

– *Output the matrix-matrix product* BA*, where* $A \in \mathcal{R}^{N\times N}$ *is the accumulator matrix.*

We use $\mathsf{EA}(n, N, p, \mathcal{R})$ *to denote a code sampled from this distribution, and the sampling of the corresponding generator matrix is denoted* $H \xleftarrow{\$} \mathsf{EAGen}(n, N, p, \mathcal{R})$*. When the ring* $\mathcal{R}$ *is omitted it is assumed* $\mathcal{R} = \mathbb{F}_2$*.*

Remark 7. While it is more standard in the coding-theoretic literature to use G for a generator matrix of a code, as we are interested in the *dual* LPN assumption connected to a code, we actually view H as the parity-check matrix for the code for which the EA code is the dual. Thus, as H is the standard notation for a parity-check matrix, we have chosen to use this notation for the generator matrix of an EA code.

3.2 The EA-LPN Assumption and Security Analysis

In this work, we provide a new (dual) LPN-type assumption connected to EA codes which we term $\mathsf{EA\text{-}LPN}$. It is obtained by specializing Definition 1 to the case where the code generation algorithm samples $H \xleftarrow{\$} \mathsf{EAGen}$. For the noise distribution $\mathcal{D}(\mathcal{R})$, we can consider Bernoulli noise $\mathsf{Ber}_{t/N}^N(\mathcal{R})$, exact noise $\mathsf{HW}_t^N(\mathcal{R})$, and regular noise $\mathsf{Reg}_t^N(\mathcal{R})$.

Definition 8 ($\mathsf{EA\text{-}LPN}$ Assumption). *Let* $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_N(\mathcal{R})\}_{N\in\mathbb{N}}$ *denote a family of efficiently sampleable distributions over* $\mathcal{R}$*, such that for any* $N \in \mathbb{N}$*,* $\mathsf{Im}(\mathcal{D}_N(\mathcal{R})) \subseteq \mathcal{R}^N$*. For a dimension* $n = n(\lambda)$*, number of samples* $N = N(\lambda)$*, ring* $\mathcal{R} = \mathcal{R}(\lambda)$ *and parameter* $p = p(\lambda) \in (0, 1)$ *the* $(\mathcal{D}, \mathcal{R})$-$\mathsf{EA\text{-}LPN}(n(\lambda), N(\lambda), p(\lambda))$ *assumption states that*

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}_N(\mathcal{R}), \vec{b} \leftarrow H \cdot \vec{e}\}$$
$$\overset{c}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^N\}.$$

In order to provide evidence for the $\mathsf{EA\text{-}LPN}$-assumption, we will show that it is secure against linear tests (Definition 2), at least when $\mathcal{R} = \mathbb{F}_2$. To do this, recalling Lemma 3, it suffices to show that $\mathsf{d}(H)$ is large (with high probability). The technical core of our proof is the following bound on the probability that a message vector $\vec{x} \in \mathbb{F}_2^n$ of weight r is mapped to a codeword of weight $\leq \delta N$.

Lemma 9. *Let* $n, N \in \mathbb{N}$ *with* $n \leq N$ *and put* $R = \frac{n}{N}$*. Fix* $p \in (0, 1/2)$ *and* $\delta > 0$*, and put* $\beta = 1/2 - \delta$*. Let* $r \in \mathbb{N}$ *and let* $\vec{x} \in \mathbb{F}_2^n$ *be a vector of weight* r*. Define* $\xi_r = (1 - 2p)^r$*. Then,*

$$\Pr\left[\mathcal{HW}(\vec{x}^\intercal H) \leq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)\right] \leq 2\exp\left(-2\frac{1-\xi_r}{1+\xi_r}N\beta^2\right) .$$

To prove this lemma, we imagine revealing the coordinates of the random vector $\vec{x}^{\intercal}H$ one at a time, and observe that this can be viewed as a random walk on a Markov chain with state space $\{0,1\}$ and second eigenvalue ξ_r. We can then apply an Expander Hoeffding bound to guarantee that such a random walk is unlikely to spend too much time on the 0 state, which is equivalent to saying that the random vector $\vec{x}^{\intercal}H$ does not have too small weight. For space reasons, the proof is deferred to the full version of this paper.

We now state the main theorem of this section.

Theorem 10. *Let $n, N \in \mathbb{N}$ with $n \leq N$ and put $R = \frac{n}{N}$, which we assume to be a constant. Let $C > 0$ and set $p = \frac{C \ln N}{N} \in (0, 1/2)$. Fix $\delta \in (0, 1/2)$ and put $\beta = 1/2 - \delta$. Assume the following relation holds:*

$$R < \min\left\{\frac{2}{\ln 2} \cdot \frac{1 - e^{-1}}{1 + e^{-1}} \cdot \beta^2, \frac{2}{e}\right\} \tag{2}$$

Then, assuming N is sufficiently large we have

$$\begin{aligned}\Pr\left[\mathsf{d}(H) \geq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)\right] &\geq 1 - 2\sum_{r=1}^{n}\binom{n}{r}\exp\left(-2\frac{1-\xi_r}{1+\xi_r}N\beta^2\right) \\ &\geq 1 - 2RN^{-2\beta^2 C + 2}.\end{aligned} \tag{3}$$

Informally, the conclusion is that when $p = \Theta(\log N/N)$ a constant rate EA code will have distance $\Omega(N)$ with probability $1 - 1/\mathsf{poly}(N)$. If one would like the failure probability to be negligible in N this can still be achieved by increasing p: for example, if $p = \Theta(\log^2 N/N)$ the failure probability is $N^{-\tilde{O}(\log N)}$. The proof is again deferred to the full version.

3.3 Discussion

In our investigation of EA codes we considered many variants and studied plausibly secure concrete parameter choices. For space reasons, many of these details are necessarily deferred to the full version. In this section, we summarize our main findings.

Different expanding matrices. Rather than sampling each row of B according to the Bernoulli distribution, one could naturally try the exact distribution $\mathsf{HW}_\ell^N(\mathbb{F}_2)$, or even the regular distribution $\mathsf{Reg}_\ell^N(\mathbb{F}_2)$. Unfortunately, these matrices are not as amenable to analysis. Nonetheless, after running some computer simulations we are willing to conjecture that they should behave relatively similarly: once $\ell = \Omega(\log N)$ we can hope to have constant rate and relative distance.

Arbitrary rings. We also consider the minimum distance of EA codes over arbitrary rings. Guided by computer simulations, we are willing to conjecture that the minimum distance should only increase as the ring size increases, thereby implying that its resilience to linear attacks only increases. However, we caution that for rings like $\mathbb{F}_{2^k}$ "modular reduction/projection" attacks allow one to work modulo 2 and, e.g., recover the coordinates bit-by-bit (this attack is outside the scope of the linear tests framework). Thus, the general conclusion is that the security one obtains for the EA-LPN assumption should only increase as the characteristic of the ring increases.

Pseudodistance. By showing that H has large distance, we can rule out any linear test. However, we are are only concerned with *efficient* linear tests, *i.e.*, tests that can efficiently find the attack vector from the matrix H. So long as δ and R satisfy $\delta < (1-R)/2$ (this rules out the standard *information set decoding* attack) we conjecture that it is infeasible to find $\vec{x}$ for which $\mathcal{HW}(\vec{x}^\intercal H) \leq \delta N$ when $p = \Omega(\log N/N)$ is sufficiently large.

Rejection sampling. Our analysis suggests that when an EA code fails to have good minimum distance it is often for the simple reason that the generator matrix H already has a low weight row. Thus, we propose testing the matrix after it is sampled to verify that indeed all the rows have large weight, and we heuristically argue that this leads to significant savings in the failure probability.

Density of B. From a theoretical standpoint, we can unfortunately show that the condition that $p = \Omega(\log N/N)$ is necessary. However, from a concrete standpoint we believe it is reasonable to choose the density of B much smaller. We elaborate upon this further in the following section.

3.4 Concrete Parameter Choices

Conservative parameters. In this section, we consider relatively conservative parameter choices, and compute the failure probability as given by (3). That is, instead of computing the probability that $\mathsf{d}(H) \leq \delta N$ for $H \leftarrow \mathsf{EA}(n, N, p)$ as $2RN^{-2C\beta^2+2}$ (where, as in the theorem statement, $\beta = 1/2 - \delta$, $R = n/N$ and $p = \frac{C \ln N}{N}$) we endeavour to numerically compute the bound

$$2\sum_{r=1}^{n} \binom{n}{r} \exp\left(-2\frac{1-\xi_r}{1+\xi_r} N\beta^2\right),$$

where as before $\xi_r = (1-2p)^r$. For our applications we would like $n = 2^{20}$, 2^{25} and 2^{30}. It is reasonable to choose $R = 0.2$, implying $N = 5 \cdot n$. Our results are summarized in Figure 2.

For context, we recall Lemma 3 which translates minimum distance into security against linear tests. It says that if $H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)$ has minimum

	$C = 3$			$C = 2.5$			$C = 2.3$		
δ \ n	2^{20}	2^{25}	2^{30}	2^{20}	2^{25}	2^{30}	2^{20}	2^{25}	2^{30}
0.005	0.000317	0.0000686	0.0000148	0.0133	0.00645	0.00312	0.0599	0.0401	0.0268
0.02	0.00120	0.000347	0.000100	0.0410	0.0253	0.0156	0.174	0.147	0.124
0.05	0.0157	0.00794	0.00402						

Fig. 2: In this table, we list the (extrapolated) analytical upper bounds on the failure probabilities for various parameter choices. The rate is set to 1/5, i.e., $N = 5n$. If the cell is empty it is because the extrapolated value exceeds 1.

distance δN with probability at least $\eta(\lambda)$ and the error vector $\vec{e} \in \mathbb{F}_2^N$ has (expected) weight t (e.g., $\vec{e} \xleftarrow{\$} \mathsf{Ber}_{t/N}^N(\mathbb{F}_2)$, $\mathsf{HW}_{t/N}^N(\mathbb{F}_2)$ or $\mathsf{Reg}_t^N(\mathbb{F}_2)$), then if we want $(2^{-\lambda}, \eta(\lambda))$-security against linear tests we require $e^{-2t\delta} \leq 2^{-\lambda}$, i.e., $t \geq \frac{(\ln 2)\cdot\lambda}{2\delta}$.

Looking at Figure 2, for $n = 2^{30}$ and $N = 5n$, if $t = 664$ then we have $t > \frac{(\ln 2)\cdot 98}{2\cdot 0.05}$, which implies that $H \xleftarrow{\$} \mathsf{EAGen}\left(n, N, \frac{3\ln N}{N}\right)$ is $(2^{-98-\log_2 5}, 0.00402)$-secure[3] against linear tests. Decreasing C, if $H \xleftarrow{\$} \mathsf{EAGen}\left(n, N, \frac{2.3\ln N}{N}\right)$ then so long as $t \geq 1658$ it is $(2^{-98-\log_2 5}, 0.124)$-secure against linear tests.

For all of the extrapolated values in Figure 2, we provide the necessary number of noisy coordinates for 128 bits of security against linear tests.

Density of B, concretely. However, when it comes to concrete parameter choices, it is reasonable to be more aggressive. Our intuition, which is guided by the proof of Theorem 10, tells us that if there is to be a low-weight vector in an EA code, then it is likely obtained as the encoding of a low-weight message. In particular, if an EA code has distance d it is probably because $H \xleftarrow{\$} \mathsf{EAGen}$ has a row of weight d. Furthermore, while there could very well be lower weight vectors in the EA codes, we do not see an easy means to *find* these vectors. Recalling the discussion of the notion of *pseudodistance*, this already implies that the construction could be secure against *efficient* linear attacks.

Being aggressive, we consider $\ell = 7, 9, 11$, and then empirically estimate the minimum (relative) weight of a row of an EA matrix $H \xleftarrow{\$} \mathsf{EAGenReg}(n, 5\cdot n, \ell)$[4] for $n = 2^{20}, 2^{25}, 2^{30}$. For $\ell = 7, 9, 11$, we endeavour to empirically estimate the minimum row weight of a matrix $H \xleftarrow{\$} \mathsf{EAGenReg}(n, 5\cdot n, \ell)$.[5]

These experiments embolden us to make the following sort of conjecture: given a matrix $H \xleftarrow{\$} \mathsf{EAGenReg}(2^{30}, 5\cdot 2^{30}, 7)$, we expect it to be hard to find a vector in the row-span of H with relative weight less than 0.02, even though

[3] Note that as computing a dot-product requires $5\cdot 2^{30}$ time, this is sufficient for 128 bits of security.

[4] The regular distribution appears to us to be the most reasonable in practice; however, other distributions appear to behave similarly.

[5] The regular distribution appears to us to be the most reasonable in practice; however, other distributions appear to behave similarly.

δ \ n	2^{20}	2^{25}	2^{30}
0.005	7326	6979	6632
0.02	1832	1745	1658
0.05	732	698	664

(a) Value of t required for $128-\log_2 N$-bit security against linear tests. The failure probability for different values of C is found in Figure 2.

n \ ℓ	7	9	11
2^{20}	0.0613	0.0923	0.121
2^{25}	0.0370	0.0624	0.0879
2^{30}	0.0223	0.0422	0.06391

(b) The (extrapolated, empirical) average minimum row-weight for H sampled as $H \leftarrow \mathsf{EAGenReg}(n, 5n, \ell)$.

we expect there to exist (many) such vectors. We leave testing of the validity of this assumption as an interesting challenge for future work.

Relation to Silver. [CRR21] also introduced a new code family, called Silver. However, their security analysis is purely based on computer simulation. Concretely, the authors of [CRR21] sampled random choices of code parameters, and sampled many instances of the code for small value of n. Then, they approximated the minimum distance for each sample by encoding low-weight vectors, and estimated the variance from the distance distribution. From that, they extrapolated a lower bound on the minimum distance for multiple small values of n, which they further extrapolated, from the curve of these lower bounds, to larger values of n. In the end, they picked the parameters that led to the best extrapolations.

We applied a relatively similar heuristic, by using computer simulations to approximate the minimum distance of our code for small values of n, and extrapolating its behavior for large values of n, with the purpose of enabling an apple-to-apple comparison with Silver. We observed that, already when setting the number of ones per row of B to only 7 and using $t \approx 5000$ noisy coordinates, we achieve heuristic security guarantees roughly on par with Silver. Note that the choice of increasing t to lower the row-weight of B is well motivated, since it vanishes when n grows and only influences the seed size, which is $\Omega(t\lambda \log n)$.

4 Pseudorandom Correlated Functions from Expand-Accumulate Codes

In this section, we give a high-level summary of the ideas behind the PCF for subfield-VOLE. For more details on how to obtain PCFs for subfield VOLE, OT and general degree-two correlations over a ring under (variants of) the $\mathsf{EA\text{-}LPN}$ assumption, we refer to the full version of this paper.

Fix an extension field $\mathbb{F}$ of $\mathbb{F}_2$; we target PCF for the (subfield) VOLE correlation over $\mathbb{F}$. That is, $\mathsf{PCF.Gen}$ outputs a pair $(\mathsf{k}_0, \mathsf{k}_1)$ of correlated keys such that for any input x, writing $(u, v) \leftarrow \mathsf{PCF.Eval}(0, \mathsf{k}_0, x)$ and $w \leftarrow \mathsf{PCF.Eval}(1, \mathsf{k}_1, x)$, it always holds that $w = \Delta \cdot u + v$, where $\Delta \in \mathbb{F}$ is the same accross all evaluations, $(v, w) \in \mathbb{F}^2$, and $u \in \mathbb{F}_2$. We refer to the full version for a reminder of the formal definition of the security properties of a PCF. As for PCGs, one can

use a correlation-robust hash function to turn a PCF for subfield-VOLE over $\mathbb{F} = \mathbb{F}_{2^\lambda}$ (where λ is a security parameter) into a PCF for the (one out of two) oblivious transfer correlation over $\mathbb{F}$.

The main difference between an offline-online PCG and a PCF is that the latter must operate in a fully incremental fashion: given the short correlated keys, the parties should be able to obtain pseudorandom instances of the target correlation on demand, without having to stretch the entire pseudorandom correlation. At a high level, our PCF construction proceeds as follows: given the two keys $\mathsf{k}_0, \mathsf{k}_1$, the parties will be able to locally retrieve (in time logarithmic in N) additive shares (over $\mathbb{F}$) of any given position in the vector $\Delta \cdot A \cdot \vec{e}$, where $\Delta \in \mathbb{F}$ is a scalar known to P_1, $\vec{e}$ is a sparse noise vector (over $\mathbb{F}_2$) known to P_0, and A is the accumulator matrix of Definition 5.

Suppose we manage to achieve the above. Then, a random input x to the PCF is parsed by both players as defining a random row B_x of the sparse matrix B; that is, x is the randomness used to sample a row B_x from $\mathsf{Ber}_p^N(\mathbb{F}_2)$. Let ℓ be the number of ones in the sampled row; with the parameters of our analysis, $\ell = O(\log N)$ with overwhelming probability. Let $\vec{e}' = A \cdot \vec{e}$ denote the accumulated noise vector. To evaluate $\mathsf{PCF.Eval}$ on x, the parties compute shares of $\Delta \cdot e'_i$ for all ℓ positions i corresponding to non-zero entries in B_x, and locally sum their shares. This procedure takes total time $O(\ell \log N) = O(\log^2 N)$, polylogarithmic in N: we can therefore set N to be exponential in the security parameter λ to allow for an exponential stretch. Defining $u_i \leftarrow B \cdot e'_i$ and $(-v_i, w_i)$ to be the shares computed this way, it is easy to check that the relation $\Delta \cdot u_i + v_i = w_i$ holds, and that u_i is indeed pseudorandom under the $\mathsf{EA\text{-}LPN}$ assumption.

It remains to find a way to locally construct these shares of $\Delta \cdot e'_i$. Here, observe that we cannot use anymore a puncturable pseudorandom function as in our construction of offline-online PCG: the accumulation step, while very efficient and parallelizable, runs in time *linear in* N. For a PCF, however, N is necessarily superpolynomial, since a PCF allows to stretch (on demand) an arbitrary polynomial amount of correlated pseudorandomness. Fortunately, we can sidestep this unaffordable accumulation step by relying on a primitive known as a *distributed comparison function* (DCF), of which very efficient instantiations (from any one-way function) were recently proposed in [BGI19, BCG+21]. For subfield VOLE, it's enough to use a weaker form of distributed comparison function, where one party knows part of the function, which we show can be constructed more efficiently.

5 Optimizing Offline Cost

Up to now, focus has been placed on optimizing the *online* portion of the offline-online PCG constructions, corresponding to the choice and analysis of advantageous linear codes. In this section, we turn attention to the *offline* portion of our construction, consisting of two primary components:

1. Evaluating several punctured PRFs (PPRFs) on their entire domain (a functionality called $\mathsf{FullEval}$), and

2. Performing an *accumulation* step, which converts a vector $(x_1, \cdots, x_N)$ to an accumulated vector $(x_1, x_1 \oplus x_2, \cdots, \bigoplus_{i=1}^{N} x_i)$.

Recall that with respect to the general template of PCG construction, the combination of the accumulation step and the online process in our construction jointly play the role of applying a compressing linear map $\vec{x} \mapsto H \cdot \vec{x}$ as dictated by the selected linear code.

We remark that all previous works in this line (of constructing PCGs from the linear code plus PPRF template) focused almost exclusively on optimizing this $\vec{x} \mapsto H \cdot \vec{x}$ step, which was for a long time the dominant cost of the construction. We now instead focus on reducing the cost of the FullEval (and accumulation) component. Our motivations are threefold:

1. First, in the recent work of [CRR21], the cost of the mapping is reduced so significantly that, according to their evaluation, the cost of FullEval now accounts for about half of the total computation. Reducing the cost of FullEval has therefore an important impact on the total runtime.
2. Second, using our new notion of offline-online PCGs and instantiating them with expand-accumulate codes, the offline part boils down solely to a FullEval computation and an accumulation. The cost of accumulate is exceptionally small, and dominated by the cost of FullEval (by several orders of magnitude). Hence, reducing the cost of FullEval directly translate to reducing the cost of the offline PCG expansion, by the same factor.
3. Eventually, PCGs are not the sole target: other cryptographic primitives also sometimes rely on the FullEval algorithm of a PPRF. Reducing the cost of FullEval directly translates to improvements for these primitives.

The high-level intuition of our main results in this section correspond to the observation that for PCG construction, in fact a PPRF is a *stronger* tool than necessary. In doing so, we put forth and explore a weaker notion with the aim of improved efficiency.

In the following we give an overview of our results. For details, we refer to the full version of this paper.

Overview of the results. First, we give high-level optimizations for the offline operations. This includes procedures for parallelizing the accumulation step, as well as methods for improving the computation cost of FullEval for GGM-type constructions such as PPRF in exchange for increased key size, by "flattening" the depth of the GGM tree.

Next, we introduce a relaxed version of PPRF, a *(strong) unpredictable punctured function* (UPF). We provide constructions of (strong) UPFs in the random oracle (RO) model (ROM) that require half the number of RO calls for FullEval as compared with the standard RO-based PPRF construction. Given the current existence of hardware support for AES, we additionally provide a conjectured construction given access to the Random Invertible Permutation Model (RIPM).

We further explore conversions from UPF to the (stronger) standard notion of PPRF in the random oracle model, beginning with a generic compiler that

simply applies the random oracle to each UPF output. For our specific RO-based UPF construction of the previous subsection, we show that this same goal can be achieved by applying the RO to only half of the UPF outputs. In turn, this provides a construction of standard PPRF in the RO model in which FullEval on a domain of size N requires only $1.5N$ calls to the random oracle.

Finally, we prove that for some PCG constructions, strong UPFs already suffice in the place of PPRFs. In particular, this holds for the PCG constructions of subfield VOLE and Silent OT. In these applications, we can thus replace the PPRF by our RO-based strong UPF, in which FullEval on a domain of size N requires only N calls to the random oracle, in comparison to $2N$ when based on PPRF.

Applications and bottom line. Using the baseline GGM PPRF with domain size N, the cost of FullEval (i.e., evaluating the entire binary tree with N leaves) boils down to $2N$ calls to the underlying primitives (in concrete instantiations, this can translate to $2N$ evaluations of fixed-key AES). To reduce this cost, we suggest to replace the GGM PPRF by our proposed PPRF construction. Concretely, computing all leaves of the UPF requires exactly N calls to the underlying primitive (modeled either as a random oracle or as a random invertible permutation) in each of our two constructions. Converting the UPF to a PPRF requires further hashing half of the leaves, leading to a total cost of $1.5N$ calls to the underlying primitive. This is a 25% cost reduction compared to the GGM PPRF approach.

The "tree-flattening" optimizations translate to a 41.5% reduction of the FullEval time, hence of the entire offline time of our offline-online PCG construction. Since FullEval also amounts to roughly 50% of the cost of the full PCG expansion in [CRR21], plugging our new constructions should directly translate to a reduction of the total cost by about 20% (which is quite significant given how fast the construction already is).

As mentioned, for certain PCG constructions, such as Silent OT, these numbers jump already to 50% cost reduction of FullEval, corresponding to roughly 25% reduction in the overall cost of full PCG expansion.

These results also have further implications beyond PCGs. The FullEval algorithm of PPRFs and related primitives is also used in some zero-knowledge applications, typically in the MPC-in-the-head paradigm. Some examples include Picnic [KKW18, CDG+20] and its variants [KZ20], the signature schemes of [Beu20], or the zero-knowledge proof of [FS21]. FullEval is also used in some constructions of private information retrieval, such as [MZR+13]; the list is not exhaustive. In all these applications, replacing FullEval by our improved variant leads to computational savings (the amount of which depends on how dominant the cost of FullEval is in each application).

6 Acknowledgements

E. Boyle supported by AFOSR Award FA9550-21-1-0046, a Google Research Award, and ERC Project HSS (852952). G. Couteau supported by the ANR

SCENE. N. Gilboa supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai supported by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20. L. Kohl is funded by NWO Gravitation project QSC. N. Resch is supported by ERC H2020 grant No.74079 (ALGSTRONGCRYPTO). P. Scholl is supported by the Danish Independent Research Council under project number 0165-00107B (C3PO) and an Aarhus University Research Foundation starting grant.

References

ABB+17. Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike: Bit flipping key encapsulation. 2017.

ABB+20. Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, and Santosh Ghosh. BIKE. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

ABG+14. Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. Candidate weak pseudorandom functions in AC^0 *o* MOD_2. In *ITCS 2014*, January 2014.

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *International Colloquium on Automata, Languages, and Programming*, pages 403–415. Springer, 2011.

AJ01. Abdulrahman Al Jabri. A statistical decoding algorithm for general linear block codes. In *IMA International Conference on Cryptography and Coding*, pages 1–8. Springer, 2001.

Ale03. Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, October 2003.

AMBD+18. Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018.

ANO+21. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. Cryptology ePrint Archive, Report 2021/1587, 2021. `https://eprint.iacr.org/2021/1587`.

BCG+17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, October / November 2017.

BCG+19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, November 2019.

BCG+19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, August 2019.

BCG+20a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, November 2020.

BCG+20b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, August 2020.

BCG+21. Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT 2021, Part II*, October 2021.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM CCS 2018*, October 2018.

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

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In *EUROCRYPT 2020, Part III*, May 2020.

BFKL94. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO'93*, August 1994.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, March 2014.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, October 2016.

BGI19. Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *TCC 2019, Part I*, December 2019.

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT 2012*, April 2012.

BKW00. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, May 2000.

BLP11. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO 2011*, August 2011.

BM97. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT'97*, May 1997.

BM18. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, 2018.

BMRS21. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *CRYPTO 2021, Part IV*, August 2021.

BR17. Andrej Bogdanov and Alon Rosen. Pseudorandom functions: Three decades later. Cryptology ePrint Archive, Report 2017/652, 2017. `https://eprint.iacr.org/2017/652`.

BTV16. Sonia Bogos, Florian Tramer, and Serge Vaudenay. On solving lpn using bkw and variants. *Cryptography and Communications*, 8(3):331–369, 2016.

BV16. Sonia Bogos and Serge Vaudenay. Optimization of LPN solving algorithms. In *ASIACRYPT 2016, Part I*, December 2016.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013, Part II*, December 2013.

CDG+20. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. The picnic signature scheme, 2020.

CGP20. Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *ASIACRYPT 2020, Part III*, December 2020.

Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019, Part II*, May 2019.

CRR21. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021, Part III*, August 2021.

CT21. Yu Long Chen and Stefano Tessaro. Better security-efficiency trade-offs in permutation-based two-party computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 275–304. Springer, 2021.

DAT17. Thomas Debris-Alazard and Jean-Pierre Tillich. Statistical decoding. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1798–1802. IEEE, 2017.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, August 2016.

DIO20. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446, 2020. `https://eprint.iacr.org/2020/1446`.

DJM98. Dariush Divsalar, Hui Jin, and Robert J McEliece. Coding theorems for" turbo-like" codes. In *Proceedings of the annual Allerton Conference on Communication control and Computing*, volume 36, pages 201–210. University Of Illinois, 1998.

DNNR17. Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO 2017, Part I*, August 2017.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, August 2012.

Ds17. Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In *ACM CCS 2017*, October / November 2017.

EKM17. Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In *CRYPTO 2017, Part II*, August 2017.

FKI06. Marc PC Fossorier, Kazukuni Kobara, and Hideki Imai. Modeling bit flipping decoding based on nonorthogonal check sums with application to iterative decoding attack of mceliece cryptosystem. *IEEE Transactions on Information Theory*, 53(1):402–411, 2006.

FS09. Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In *ASIACRYPT 2009*, December 2009.

FS21. Nils Fleischhacker and Mark Simkin. On publicly-accountable zero-knowledge and small shuffle arguments. In *PKC 2021, Part II*, May 2021.

GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, (4), October 1986.

GJL20. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. *Journal of Cryptology*, (1), January 2020.

GKWY20. Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, May 2020.

GRS08. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. Good variants of HB+ are hard to find. In *FC 2008*, January 2008.

HK21. David Heath and Vladimir Kolesnikov. One hot garbling. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 574–593. ACM, 2021.

HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In *ASIACRYPT 2018, Part III*, December 2018.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *ASIACRYPT 2017, Part I*, December 2017.

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, August 2003.

IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC 2009*, March 2009.

Kir11. Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `https://eprint.iacr.org/2011/377`.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*, October 2018.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT 2018, Part III*, April / May 2018.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 2013*, November 2013.

KZ20. Daniel Kales and Greg Zaverucha. Improving the performance of the picnic signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 154–188, 2020.

LF06. Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *SCN 06*, September 2006.

LM13. Vadim Lyubashevsky and Daniel Masny. Man-in-the-middle secure authentication schemes from LPN and weak PRFs. In *CRYPTO 2013, Part II*, August 2013.

LWYY22. Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of lpn over any integer ring and field for pcg applications. Cryptology ePrint Archive, Paper 2022/712, 2022. `https://eprint.iacr.org/2022/712`.

Lyu05. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, randomization and combinatorial optimization. Algorithms and techniques*, pages 378–389. Springer, 2005.

MAB+18. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2:4–13, 2018.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT 2011*, December 2011.

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT 2015, Part I*, April 2015.

MZR+13. Yiping Ma, Ke Zhong, Tal Rabin, Sebastian Angel, Andrew J Blumberg, Eleftherios Ioannidis, Jess Woods, Qizhen Zhang, Xinyi Chen, Sidharth Sankhe, et al. Incremental offline/online pir. *Journal of Clinical Investigation*, 123(1), 2013.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, October 2021.

Ove06. Raphael Overbeck. Statistical decoding revisited. In *ACISP 06*, July 2006.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

RS21a. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. Cryptology ePrint Archive, Report 2021/1579, 2021. `https://eprint.iacr.org/2021/1579`.

RS21b. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In *EUROCRYPT 2021, Part II*, October 2021.

Saa07. Markku-Juhani Olavi Saarinen. Linearization attacks against syndrome based hashes. In *INDOCRYPT 2007*, December 2007.

SGRR19. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM CCS 2019*, November 2019.

Ste88. Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988.

Wag02. David Wagner. A generalized birthday problem. In *CRYPTO 2002*, August 2002.

WRK17a. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM CCS 2017*, October / November 2017.

WRK17b. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *ACM CCS 2017*, October / November 2017.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. Cryptology ePrint Archive, Report 2021/076, 2021. `https://eprint.iacr.org/2021/076`.

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 2020*, November 2020.

Zic17. Lior Zichron. Locally computable arithmetic pseudorandom generators. Master's thesis, School of Electrical Engineering, Tel Aviv University, 2017.

ZJW16. Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In *EUROCRYPT 2016, Part I*, May 2016.