Correlated Pseudorandomness from Expand-Accumulate Codes

Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, Peter Scholl

Correlated Pseudorandomness from Expand-Accumulate Codes

Elette Boyle [*] Geoffroy Couteau [†] Niv Gilboa [‡] Yuval Ishai [§] Lisa Kohl [¶]
Nicolas Resch [‖] Peter Scholl [**]

August 5, 2022

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 simple parallelization with a cache-friendly offline phase;
- 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.

[*] IDC Herzliya and NTT Research, eboyle@alum.mit.edu
[†] IRIF, couteau@irif.fr
[‡] Ben-Gurion University, niv.gilboa@gmail.com
[§] Technion, yuvali@cs.technion.ac.il
[¶] Cryptology Group, CWI Amsterdam, lisa.kohl@cwi.nl
[‖] Cryptology Group, CWI Amsterdam, nicolas.resch@cwi.nl
[**] Aarhus University, peter.scholl@cs.au.dk

The above 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). PCGs and PCFs naturally give rise to the paradigm of MPC with *silent preprocessing*. In MPC protocols following this paradigm, the parties invoke a one-time interactive setup with low communication and computation costs for securely generating the short, correlated seeds. These seeds can later be used, upon demand, for "silently" generating large quantities of the correlated randomness consumed by fast online MPC protocols.

Generating pseudorandom correlations: a template. To construct the PCG and PCF 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 [WYKW21, DIO21, BMRS21, 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].

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 over $\mathbb{F}$ [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 $O(n)$. 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 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 which is highly optimized for correlated randomness generation and is 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 many MPC protocols [Bea91, BDOZ11, DPSZ12, ANO+21, RS21a]), and optimal flexibility [BCG+20a] (generating any amount of correlated

randomness on demand). Nonetheless, in several aspects this research area is in its infancy: some important correlations remain frustratingly out of reach (such as circuit-dependent correlations, used e.g. in [DPSZ12, DNNR17, HOSS18, WRK17b, Cou19, BGI19], or authenticated multiplication triples over $\mathbb{F}_2$, used in [HSS17, WRK17a]); the current fastest construction [CRR21] is not parallelizable and lacks any clear theoretical security analysis whereas 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 one of the most promising paths 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, as it uniquely combines multiple attractive features: firm security foundations, simplicity, high concrete efficiency, and easy *parallelization*. When an EA-based PCG is implemented in an offline-online mode, the offline phase is both parallelizable and *cache-friendly*, whereas the online phase requires accessing a small number of memory locations per correlation instance. Finally, 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 apples-to-apples 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. The 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] Hence, over multicore architectures, we expect our new PCG to outperform alternative approaches.

4. We obtain the first practical PCG constructions for different kinds of useful correlations. This includes 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]

[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].

(which is much slower, even for aggressive parameters) 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 within the secure computation application). 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 $(x_1, x_2, \cdots, x_N)$ to an accumulated vector $(x_1, x_1 \oplus x_2, \cdots, \bigoplus_{i=1}^{N} x_i)$. This "prefix sum" computation can be done with $N-1$ `xor`s of short strings in one pass, which is extremely fast and cache-friendly; furthermore, it 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). Finally, 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 extremely 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 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 [CK20, MZRA21], 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 $N^{-\Omega(\log N)}$ failure probability. Further, we can show that 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, EA-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

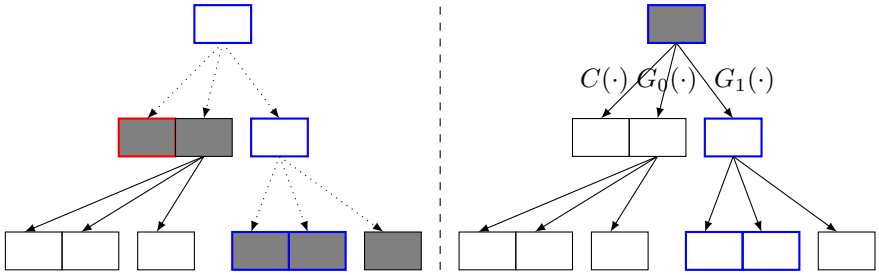

Figure 1: Pictorial representation of our relaxed DCF construction. In our example the path $\alpha = 10 \in \{0,1\}^2$ is marked in blue, the box marked in red corresponds to box where β is added, and the boxes filled in gray correspond to the key of P_0 (in knowledge of α) and P_1, respectively.

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 similarly to 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], 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 securely distributed using a simple, 2-round 2-party OT-based protocol 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 PCFs for VOLE and subfield VOLE correlations, which also imply 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 EA-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 guarantees), while we need around an order of magnitude less computation. For our degree-2 PCFs, to get reasonable concrete efficiency we need to rely on more aggressive EA-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] do 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 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 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 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 the necessary 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 introduce the notion of offline-online PCGs and present a construction of offline-online PCGs for subfield VOLE from EA codes. In Section 5 we provide new constructions of PCFs based on the $\mathsf{EA\text{-}LPN}$ assumption. Finally, in Section 6.1, we describe our optimizations for the offline costs of PCG constructions.

2 Preliminaries

2.1 Preliminaries on Bias

Definition 2.1 (Bias of a Distribution) *Given a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a vector $\vec{u} \in \mathbb{F}^n$, the bias of $\mathcal{D}$ with respect to $\vec{u}$, denoted $\mathsf{bias}_{\vec{u}}(\mathcal{D})$, is equal to*

$$\mathsf{bias}_{\vec{u}}(\mathcal{D}) = |\mathbb{E}_{\vec{x}\sim\mathcal{D}}[\vec{u}^\intercal \cdot \vec{x}] - \mathbb{E}_{\vec{x}\sim\mathcal{U}_n}[\vec{u}^\intercal \cdot \vec{x}]| = \left|\mathbb{E}_{\vec{x}\sim\mathcal{D}}[\vec{u}^\intercal \cdot \vec{x}] - \frac{1}{|\mathbb{F}|}\right|,$$

where $\mathcal{U}_n$ denotes the uniform distribution over $\mathbb{F}^n$. The bias of $\mathcal{D}$, denoted $\mathsf{bias}(\mathcal{D})$, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\vec{u}$.

Given t distributions $(\mathcal{D}_1, \cdots, \mathcal{D}_t)$ over $\mathbb{F}_2^n$, we denote by $\bigoplus_{i\le t} \mathcal{D}_i$ the distribution obtained by *independently* sampling $\vec{v}_i \xleftarrow{\$} \mathcal{D}_i$ for $i = 1$ to t and outputting $\vec{v} \leftarrow \vec{v}_1 \oplus \cdots \oplus \vec{v}_t$. We will use the following bias of the exclusive-or (cf. [Shp09]).

Lemma 2.2 *Let $t \in \mathbb{N}$ be an integer, and let $(\mathcal{D}_1, \cdots, \mathcal{D}_t)$ be t independent distributions over $\mathbb{F}_2^n$. Then $\mathsf{bias}(\bigoplus_{i\le t} \mathcal{D}_i) \le 2^{t-1} \cdot \prod_{i=1}^t \mathsf{bias}(\mathcal{D}_i) \le \min_{i\le t} \mathsf{bias}(\mathcal{D}_i)$.*

Finally, let $\mathsf{Ber}_r(\mathbb{F}_2)$ denote the Bernoulli distribution that outputs 1 with probability r, and 0 otherwise. More generally, we denote by $\mathsf{Ber}_r(\mathcal{R})$ the distribution that outputs a uniformly random nonzero element of a ring $\mathcal{R}$ with probability r, and 0 otherwise.

We will use a standard simple lemma for computing the bias of a XOR of Bernoulli samples:

Lemma 2.3 (Piling-up lemma) *For any $0 < r < 1/2$ and any integer n, given n random variables $X_1, \cdots, X_n$ i.i.d. to $\mathsf{Ber}_r(\mathbb{F}_2)$, it holds that $\Pr[\bigoplus_{i=1}^n X_i = 0] = 1/2 + (1-2r)^n/2$.*

2.2 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 2.4 (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 2.5 (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 2.6 *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.

2.3 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].

Experiment Exp-s-pPRF

Setup Phase. The adversary $\mathcal{A}$ sends a size-t subset $S^* \in \mathcal{X}$ to the challenger. When it receives S^*, the challenger picks $K \xleftarrow{\$} F.\mathsf{KeyGen}(1^\lambda)$ and a random bit $b \xleftarrow{\$} \{0,1\}$.

Challenge Phase. The challenger sends $K\{S^*\} \leftarrow F.\mathsf{Puncture}(K, S^*)$ to $\mathcal{A}$. If $b = 0$, the challenger additionally sends $(F(K,x))_{x \in S^*}$ to $\mathcal{A}$; otherwise, if $b = 1$, the challenger picks t random values ($y_x \xleftarrow{\$} \mathcal{Y}$ for every $x \in S^*$) and sends them to $\mathcal{A}$.

Figure 2: Selective security game for puncturable pseudorandom functions. At the end of the experiment, $\mathcal{A}$ sends a guess b' and wins if $b' = b$.

Definition 2.7 (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,*

such that no probabilistic polynomial-time adversary wins the experiment Exp-s-pPRF *represented on Figure 2 with non-negligible advantage over the random guess.*

2.4 Puncturable PRFs from Length-Doubling PRGs

Here, we recall how a PPRF can be constructed from any length-doubling pseudorandom generator G, using the GGM tree-based construction [GGM86, KPTZ13, BW13, BGI14]. The construction proceeds as follows: On input a key K and a point x, set $K^{(0)} \leftarrow K$ and perform the following iterative evaluation procedure: for $i = 1$ to $\ell \leftarrow \log |x|$, compute $(K_0^{(i)}, K_1^{(i)}) \leftarrow G(K^{(i-1)})$, and set $K^{(i)} \leftarrow K_{x_i}^{(i)}$. Output $K^{(\ell)}$. This procedure creates a complete binary tree with edges labeled by keys; the output of the PRF on an input x is the key labeling the leaf at the end of the path defined by x from the root of the tree.

- $F.\mathsf{KeyGen}(1^\lambda)$: output a random seed for G.
- $F.\mathsf{Puncture}(K, z)$: on input a key $K \in \{0,1\}^k$ and a point x, apply the above procedure and return $K\{x\} = (K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$.
- $F.\mathsf{Eval}(K\{x\}, x')$, on input a punctured key $K\{x\}$ and a point x, if $x = x'$, output $\perp$. Otherwise, parse $K\{x\}$ as $(K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$ and start the iterative evaluation procedure from the first $K_{1-x_i}^{(i)}$ such that $x'_i = 1 - x_i$.

To obtain a t-puncturable PRF with input domain $[n]$, one can simply run t instances of the above puncturable PRF and set the output of the PRF to be the bitwise xor of the output of each instance. With this construction, the length of a key punctured at t points is $t\lambda \log n$, where λ is the seed size of the PRG.

2.5 Pseudorandom Correlation Generators

We recall the notion of pseudorandom correlation generator (PCG) from [BCG+19b]. At a high level, a PCG for some target ideal correlation takes as input a pair of short, correlated seeds and outputs long correlated pseudorandom strings, where the expansion procedure is deterministic and can be applied locally.

Definition 2.8 (Correlation Generator) *A PPT algorithm $\mathcal{C}$ is called a* correlation generator*, if $\mathcal{C}$ on input 1^λ outputs a pair of strings in $\{0,1\}^n \times \{0,1\}^m$ for $n(\lambda), m(\lambda) \in \mathsf{poly}(\lambda)$.*

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 R_0 given $R_1 = r_1$ and vice versa.

Definition 2.9 (Reverse-sampleable Correlation Generator) *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* $\mathsf{RSample}$ *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \,|(R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

The following definition of pseudorandom correlation generators is taken almost verbatim from [BCG+19b]; it generalizes an earlier definition of pseudorandom VOLE generator in [BCGI18].

Definition 2.10 (Pseudorandom Correlation Generator (PCG) [BCG+19b]) *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* PCG for $\mathcal{C}$ *is a pair of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *should satisfy the following:*

- **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 $\mathcal{C}(1^\lambda)$.

- **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.

2.6 From PPRFs and LPN-Friendly Codes to PCGs

The general template for building a PCG from an LPN-friendly code was established in [BCGI18, BCG+19b]. It uses two main ingredients: an LPN-friendly code, and a method for the two parties to obtain a compressed form of (pseudo)random secret shares $\vec{s}_0, \vec{s}_1$, satisfying the following equation (1): $\vec{s}_1 = \vec{s}_0 + \vec{e} \cdot x \in \mathbb{F}_{2^\lambda}^n$, where $\vec{e} \in \{0,1\}^n$ is a random t-parse vector held by one party, and $x \in \mathbb{F}_{2^\lambda}$ is a random field element held by the other party. Given this, the parties can locally compute $H \cdot \vec{s}_0$ and $H \cdot \vec{s}_1 = H \cdot \vec{s}_0 + x \cdot (H \cdot \vec{e})$. Under the LPN assumption with respect to H, those are indeed shares of $x \cdot \vec{u}$ where $\vec{u} = H \cdot \vec{e}$ is pseudorandom. This leads to a PCG for the vector OLE correlation; the shares can also be locally converted into pseudorandom string-OTs with the hashing technique [IKNP03].

To obtain a compressed form of the shares $\vec{s}_0, \vec{s}_1$ the constructions of [BCG+19a] uses a puncturable pseudorandom function. The construction works as follows: Gen will give the sender a random key K and x, and give to the receiver t random points $\alpha_j \in \{1, \ldots, N\}$, t punctured keys $K\{\alpha_j\}$, and the values $z_j = F_K(\alpha_j) + x$. Given these seeds, the sender and receiver can now define the expanded outputs, for $i = 1, \ldots, n$:

$$\vec{s}_0[i] = F_K(i), \quad \vec{s}_1[i] = \begin{cases} F_K(i) & i \notin \{\alpha_1, \cdots, \alpha_t\} \\ z_j & \text{if } i = \alpha_j \end{cases}.$$

2.7 Pseudorandom Correlation Functions

Pseudorandom correlation generators allow to stretch short correlated seeds into an a priori bounded pair of pseudorandom correlated strings, in a one-time expansion phase. Pseudorandom correlation *functions* (PCF, [BCG+20a]), on the other hand, are much more versatile: they allow to locally generate, from a pair of short correlated keys, an arbitrary polynomial amount of pseudorandom correlations *on demand*, in an incremental way (i.e., the keys are indefinitely reusable). The definitions of this section are taken almost verbatim from [BCG+20a]. To define PCFs, we must consider an incremental notion of reverse-sampleable correlations:

Definition 2.11 (Reverse-sampleable correlation) *Let $1 \leq \tau_0(\lambda), \tau_1(\lambda) \leq \mathsf{poly}(\lambda)$ be output-length functions. Let $\mathcal{Y}$ be a probabilistic algorithm that on input 1^λ returns a pair of outputs $(y_0, y_1) \in \{0,1\}^{\tau_0(\lambda)} \times \{0,1\}^{\tau_1(\lambda)}$, defining a correlation on the outputs.*

We say that $\mathcal{Y}$ defines a reverse-sampleable correlation*, if there exists a probabilistic polynomial time algorithm $\mathsf{RSample}$ that takes as input $1^\lambda, \sigma \in \{0,1\}$ and $y_\sigma \in \{0,1\}^{\tau_\sigma(\lambda)}$, and outputs $y_{1-\sigma} \in \{0,1\}^{\tau_{1-\sigma}(\lambda)}$, such that for all $\sigma \in \{0,1\}$ the following distributions are statistically close:*

$$\{(y_0, y_1) \mid (y_0, y_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda)\} \quad \textit{and}$$
$$\{(y_0, y_1) \mid (y'_0, y'_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda), y_\sigma \leftarrow y'_\sigma, y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)\}$$

It can also be useful to consider correlations *across* different inputs, as e.g. in vector oblivious linear evaluation (VOLE). This is captured by the notion of reverse sampleable correlation with setup, which allows all algorithms to depend on a fixed global secret, ensuring consistency across different invocations; we omit this formal definition for conciseness and refer the reader to [BCG+20a] for more details.

Definition 2.12 (Pseudorandom correlation function (PCF)) *Let $\mathcal{Y}$ be a reverse-sampleable correlation with output length functions $\tau_0(\lambda), \tau_1(\lambda)$ and let $\lambda \leq n(\lambda) \leq \mathsf{poly}(\lambda)$ be an input length function. Let $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ be a pair of algorithms with the following syntax:*

- $\mathsf{PCF.Gen}(1^\lambda)$ *is a probabilistic polynomial time algorithm that on input 1^λ, outputs a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$; we assume that λ can be inferred from the keys.*
- $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ *is a deterministic polynomial-time algorithm that on input $\sigma \in \{0,1\}$, key k_σ and input value $x \in \{0,1\}^{n(\lambda)}$, outputs a value $y_\sigma \in \{0,1\}^{\tau_\sigma(\lambda)}$.*[2]

*We say $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ is a (weak) (N, B, ε)-*secure pseudorandom correlation function (PCF) for $\mathcal{Y}$*, if the following conditions hold:*

- ***Pseudorandom $\mathcal{Y}$-correlated outputs.*** *For every $\sigma \in \{0,1\}$ and non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds*
$$\left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \varepsilon(\lambda)$$
for all sufficiently large λ, where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$ for $b \in \{0,1\}$ is as defined in Figure 3. In particular, the adversary is given access to $N(\lambda)$ samples.
- ***Security.*** *For each $\sigma \in \{0,1\}$ and non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds*
$$\left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]\right| \leq \varepsilon(\lambda)$$
for all sufficiently large λ, where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}(\lambda)$ for $b \in \{0,1\}$ is as defined in Figure 4 (again, with $N(\lambda)$ samples).

[2] Note that it would be sufficient for $\mathsf{PCF.Eval}$ to take as input k_σ and x by appending σ to the key k_σ. This corresponds to the view of a PCF as a single keyed function.

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)}$:
for $i = 1$ **to** $N(\lambda)$**:**
$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
$(y_0^{(i)}, y_1^{(i)}) \leftarrow \mathcal{Y}(1^\lambda)$
$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$
return b

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)}$:
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ **to** $N(\lambda)$**:**
$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
for $\sigma \in \{0,1\}$**:**
$y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$
return b

Figure 3: Pseudorandom $\mathcal{Y}$-correlated outputs of a PCF.

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)}$:
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ **to** $N(\lambda)$**:**
$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
$y_{1-\sigma}^{(i)} \leftarrow \mathsf{PCF.Eval}(1-\sigma, \mathsf{k}_{1-\sigma}, x^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]})$
return b

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)}$:
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ **to** $N(\lambda)$**:**
$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
$y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x^{(i)})$
$y_{1-\sigma}^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]})$
return b

Figure 4: Security of a PCF. Here, $\mathsf{RSample}$ is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 2.11.

We say that $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ *is a* PCF for $\mathcal{Y}$ *if it is a* $(p, 1/p, p)$*-secure PCF for* $\mathcal{Y}$ *for every polynomial* p*. If* $B = N$*, we will write* (B, ε)*-secure PCF for short.*

The above definition captures a notion of *weak* pseudorandom function, where security is only required to hold given random adversarial queries. As for PRFs, one can also strengthen the definition to strong PCFs, which allow arbitrary adversarial queries; a formal definition is given in [BCG+20a]. As shown in [BCG+20a], any weak PCF can be turned into a strong PCF in the random oracle model, by hashing the input before feeding it to the function.

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 3.1 (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:*

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

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. For more details, see Section 6.1. We now formally introduce *expand-accumulate* (EA) codes, which underline our main constructions of offline-online PCGs.

Definition 3.2 (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:*

- *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 3.3 *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.*

Remark 3.4 (Connection to RA codes) *Our definition of expand-accumulate codes is heavily inspired by the definition of* repeat-accumulate *(*RA*) codes. For a desired rate $1 = 1/\ell$ with $\ell \in \mathbb{N}$, such a code has generator matrix $R_\ell \Pi A$, where*

- *$R_\ell \in \mathbb{F}_2^{(N/\ell) \times N}$ is the matrix which repeats each coordinate ℓ times, i.e.,*

$$\vec{x}^{\intercal} R_\ell = (\underbrace{x_1, x_1, \ldots, x_1}_{\ell \text{ times}}, \underbrace{x_2, x_2, \ldots, x_2}_{\ell \text{ times}}, \ldots, \underbrace{x_{N/\ell}, x_{N/\ell}, \ldots, x_{N/\ell}}_{\ell \text{ times}}).$$

- *$\Pi \in \mathbb{F}_2^{N \times N}$ is a uniformly random permutation matrix.*
- *A is the accumulator matrix as defined in Definition 3.1.*

Unfortunately, it is known that such codes cannot be asymptotically good: if the rate is constant, then the minimum distance is at most $N^{-\Omega(1)}$. One solution to this problem is to have multiple rounds of permuting and accumulating: for example, 2 rounds of accumulating (so called RAA codes) already achieve asymptotic goodness, and with a larger (constant) number of rounds we can approach the Gilbert-Varshamov bound. However, for our desired applications having a single accumulation step is desirable. Firstly, this is maximally cache-friendly. Secondly, for our construction of PCFs we crucially rely on the structure of the noise vector after accumulating, and this structure is destroyed if we accumulate twice.

Remark 3.5 (Connection to codes of Tillich and Zémor) *Upon completing our work we discovered that* EA *codes are in fact quite similar to a class of codes studied by Tillich and Zémor [TZ06]. They presented their codes via parity-check matrices, but careful observation shows that the codes are indeed a natural variant of a rate $1/2$* EA *code (the rows of the corresponding matrix B are not independently sampled, but they are chosen to be sparse, which is qualitatively what one wants from an* EA *code). Applying these codes for cryptographic tasks appears to be a promising avenue for future research.*

3.2 Markov Chains

In order to analyze EA codes, we will make use of an *expander Hoeffding bound*. We provide in this subsection the necessary background on the Markov chains and state the bound we use.

A *(discrete-time and finite) Markov chain* consists of a finite set $\mathcal{V}$ called the *state space* and a *transition matrix* $P \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}}$ with the property that all row sums are equal to 1 and the entries are all non-negative. For $u, v \in \mathcal{V}$, one interprets $P_{u,v}$ as the probability of transitioning from state u to v, which we denote as $P_{u,v} = \Pr[\, u \to v \,]$. P then naturally describes a random walk in the following sense. Let $\vec{\nu} \in \mathbb{R}^{\mathcal{V}}$ denote a distribution over the state space $\mathcal{V}$, so ν_v is the probability of sampling state v. Consider the following procedure:

- First, sample a state $V_0 \in \mathcal{V}$ according to ν.
- Then, sample an N step random walk $V_0, V_1, \ldots, V_N$ according to P. I.e., for $i = 1, \ldots, N$, transition from state V_{i-1} to state V_i with probability P_{V_{i-1},V_i}.

A distribution $\vec{\pi} \in \mathbb{R}^{\mathcal{V}}$ is called a *stationary distribution* if

$$\vec{\pi}^\intercal P = \vec{\pi}^\intercal \ .$$

In other words, it is a left eigenvector for P with eigenvalue 1.

In this work, all Markov chains are assumed to be *irreducible*, which means that any state can be reached from any other any state in a finite number of steps. That is, if one constructs the directed graph with vertex set $\mathcal{V}$ with edges (u, v) whenever $P_{u,v} > 0$, we require this graph to be strongly connected. In this case, it is known that the Markov chain has a *unique* stationary distribution. Furthermore, an irreducible Markov chain is *reversible* if

$$\forall u, v \in V, \quad \pi_u P_{u,v} = \pi_v P_{v,u} \ .$$

In this work, we will make use of tail-bounds for Markov chains. That is, suppose we have a function $f : \mathcal{V} \to [0, 1]$ and define $\mu := \mathbb{E}_{V \sim \vec{\pi}}[f(V)]$ to be its expected value under the distribution $\vec{\pi}$. We would like to understand the probability that the random variable $S_N = \sum_{i=1}^N f(V_i)$ deviates significantly from its expected value $N \cdot \mu$ as N grows, where $(V_1, \ldots, V_N)$ is an N-step random walk. This can be quantified via P's second eigenvalue: by the classical theory of nonnegative matrices, the eigenvalues of P are $1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_{|\mathcal{V}|} \geq -1$, where the strict inequality between λ_1 and λ_2 follows from irreducibility. We call λ_2 the second largest eigenvalue of P and, when λ_2 is small (equivalently, the *spectral gap* $1 - \lambda_2$ is large), we find that the random variable S_N is very concentrated. This is the topic of "expander Hoeffding/Chernoff bounds." The version which is most suitable for our application is due to Leon and Perron [LP04].

Theorem 3.6 (Expander Hoeffding Bound) *Let $(\mathcal{V}, P)$ denote a finite, irreducible and reversible Markov chain with stationary distribution $\vec{\pi}$ and second largest eigenvalue λ. Let $f : \mathcal{V} \to [0, 1]$ with $\mu = \mathbb{E}_{V \sim \vec{\pi}}[f(V)]$. For any integer $N \geq 1$, consider the random variable $S_N = \sum_{i=1}^N f(V_i)$, where V_0 is sampled uniformly at random from V and then $V_1, \ldots, V_N$ is a random walk starting at V_0.*

Then, for $\lambda_0 = \max(0, \lambda)$ and any $\varepsilon > 0$ with $\mu + \varepsilon < 1$, the following bound holds:

$$\Pr\left[\, S_N \geq N(\mu + \varepsilon) \,\right] \leq \exp\left(-2\frac{1 - \lambda_0}{1 + \lambda_0} N\varepsilon^2\right) \ .$$

When we apply the above theorem, it will be for 2-state Markov chain with uniform stationary distribution $\vec{\pi} = (1/2, 1/2)$, and for our function of interest $\mu = 1/2$. However, the initial state will not be uniform; it will be equal to one of the two states with probability 1. We can easily derive from the above theorem a corollary which is sufficient for our purposes, wherein the above probability estimate only degrades by a factor of 2.

Corollary 3.7 *Let $(\mathcal{V}, P)$ denote a finite, irreducible and reversible Markov chain with $\mathcal{V} = \{v_0, v_1\}$, stationary distribution $\vec{\pi} = (1/2, 1/2)$ and second largest eigenvalue λ. Let $f : \mathcal{V} \to [0, 1]$ with $1/2 = \mathbb{E}_{V \sim \vec{\pi}}[f(V)]$. For any integer $N \geq 1$, consider the random variable $\tilde{S}_N = \sum_{i=1}^N f(V_i)$, where $V_0 = v_0$ with probability 1 and then $V_1, \ldots, V_N$ is a random walk starting at v_0.*

Then, for $\lambda_0 = \max(0, \lambda)$ and any $\varepsilon > 0$ with $1/2 + \varepsilon < 1$, the following bound holds:

$$\Pr\left[\, \tilde{S}_N \geq N(1/2 + \varepsilon) \,\right] \leq 2\exp\left(-2\frac{1 - \lambda_0}{1 + \lambda_0} N\varepsilon^2\right) \ .$$

Proof. Applying the total probability rule,

$$\begin{aligned}
\Pr\left[\, S_N \geq N(1/2 + \varepsilon) \,\right] &= \Pr\left[\, S_N \geq N(1/2 + \varepsilon) | V_0 = v_0 \,\right] \cdot \Pr_{V_0 \sim \vec{\pi}}\left[\, V_0 = v_0 \,\right] \\
&\quad + \Pr\left[\, S_N \geq N(1/2 + \varepsilon) | V_0 = v_1 \,\right] \cdot \Pr_{V_0 \sim \vec{\pi}}\left[\, V_0 = v_1 \,\right] \\
&= \Pr\left[\, S_N \geq N(1/2 + \varepsilon) | V_0 = v_0 \,\right] \cdot \frac{1}{2} \\
&\quad + \Pr\left[\, S_N \geq N(1/2 + \varepsilon) | V_0 = v_1 \,\right] \cdot \frac{1}{2} \\
&\geq \Pr\left[\, \tilde{S}_N \geq N(1/2 + \varepsilon) \,\right] \cdot \frac{1}{2} \ .
\end{aligned}$$

The result now follows readily from Theorem 3.6. □

3.3 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 EA-LPN. It is obtained by specializing Definition 2.4 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 $\mathsf{Ber}^N_{t/N}(\mathcal{R})$, exact noise $\mathsf{HW}^N_t(\mathcal{R})$, and regular noise $\mathsf{Reg}^N_t(\mathcal{R})$.

Definition 3.8 (EA-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 $(\mathcal{D},\mathcal{R})$-EA-LPN$(n(\lambda), N(\lambda), p(\lambda))$ assumption states that*

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

In order to provide evidence for the EA-LPN-assumption, we will show that it is secure against linear tests (Definition 2.5), at least when $\mathcal{R} = \mathbb{F}_2$. To do this, recalling Lemma 2.6, 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 3.9 *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 Corollary 3.7 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.

Proof. Denote the columns of B as $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_N \in \mathbb{F}_2^n$, and observe that

$$H = BA = \begin{bmatrix} | & | & \cdots & | \\ \vec{(c}_1 + \cdots + \vec{c}_N) & (\vec{c}_2 + \cdots + \vec{c}_N) & \cdots & \vec{c}_N \\ | & | & \cdots & | \end{bmatrix} .$$

Hence, for a fixed vector $\vec{x} \in \mathbb{F}_2^n$, if $\vec{y}^\intercal = \vec{x}^\intercal H = \vec{x}^\intercal BA$, we have

$$(y_1, y_2, \ldots, y_N) = (\vec{x}^\intercal\cdot(\vec{c}_1+\cdots+\vec{c}_N), \vec{x}^\intercal\cdot(\vec{c}_2+\cdots+\vec{c}_N), \ldots, \vec{x}^\intercal\cdot(\vec{c}_1+\cdots+\vec{c}_N)) .$$

Note that each entry of B is sampled independently according to $\mathsf{Ber}_p(\mathbb{F}_2)$. Hence, the $\vec{c}_i$'s are independently sampled as $\vec{c}_i \leftarrow \mathsf{Ber}^n_p(\mathbb{F}_2)$. We furthermore have the following recursive relation:

$$y_N := \vec{x}^\intercal\cdot\vec{c}_N,$$
$$y_i := \vec{x}^\intercal\cdot\vec{c}_i + y_{i+1} \quad \forall 1 \leq i \leq N-1 .$$

Lastly, as $\mathcal{HW}(\vec{x}) = r$ the Piling-up Lemma (Lemma 2.3) tells us that each $\vec{x}^\intercal\cdot\vec{c}_i$ is distributed as $\mathsf{Ber}_{p_r}(\mathbb{F}_2)$ where

$$p_r := \frac{1-(1-2p)^r}{2} = \frac{1-\xi_r}{2} .$$

Defining $z_i = y_{N-i}$ for $i \in [N]$, we may therefore view the sequence $(z_1, z_2, \ldots, z_N)$ as the outcome of an N-step random walk on a Markov chain with state space $\{v_0, v_1\}$ (corresponding to 0 and 1, respectively) which starts in state $z_0 = v_0$. The transition probabilities are

$$\Pr[\, v_0 \to v_0 \,] = \Pr[\, v_1 \to v_1 \,] = \frac{1+\xi_r}{2} ,$$
$$\Pr[\, v_0 \to v_1 \,] = \Pr[\, v_1 \to v_0 \,] = \frac{1-\xi_r}{2} .$$

The transition matrix of this Markov chain is

$$\begin{bmatrix} \frac{1+\xi_r}{2} & \frac{1-\xi_r}{2} \\ \frac{1-\xi_r}{2} & \frac{1+\xi_r}{2} \end{bmatrix} = \frac{1-\xi_r}{2} J + \xi_r I \ ,$$

where J is the 2×2 all-1's matrix and I is the 2×2 identity matrix. It follows readily that the second-largest eigenvalue of this matrix is ξ_r and that its stationary distribution is uniform, *i.e.*, $(1/2, 1/2)$. It is also immediate that this Markov chain is reversible. We are thus in position to apply Corollary 3.7 with $f : V \to [0,1]$ defined via $f(v_0) = 1$ and $f(v_1) = 0$. Thus, if $V_1, \ldots, V_N$ denotes a N step random walk starting at state $V_0 = v_0$ and $\tilde{S}_N = \sum_{i=1}^{N} f(V_i)$, we find

$$\begin{aligned} \Pr\left[\mathcal{HW}(\vec{x}^{\mathsf{T}} H) \leq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p) \right] &= \Pr\left[\tilde{S}_n \geq (1/2 + \beta) N \right] \\ &\leq 2 \exp\left(-2 \frac{1-\xi_r}{1+\xi_r} N \beta^2 \right) \ . \end{aligned}$$

□

We now state the main theorem of this section.

Theorem 3.10 *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 $N^{-O(\log N)}$.

Proof. By the union bound, we may upper bound

$$\begin{aligned} &\Pr\left[\mathsf{d}(H) \leq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p) \right] \\ &\quad = \Pr\left[\exists \vec{x} \in \mathbb{F}_2^n \setminus \{\vec{0}\} \text{ s.t. } \mathcal{HW}(\vec{x}^{\mathsf{T}} H) \leq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p) \right] \\ &\quad \leq \sum_{r=1}^{n} \sum_{\substack{\vec{x} \in \mathbb{F}_2^n \\ \mathcal{HW}(\vec{x}) = r}} \Pr\left[\mathcal{HW}(\vec{x}^{\mathsf{T}} H) \leq \delta N \mid H \xleftarrow{\$} \mathsf{EAGen}(n, N, p) \right] \ . \end{aligned} \tag{4}$$

Applying Lemma 3.9, we may upper bound

$$(4) \leq \sum_{r=1}^{k} \sum_{\substack{\vec{x} \in \mathbb{F}_2^n \\ \mathcal{HW}(\vec{x}) = r}} 2 \exp\left(-2 \frac{1-\xi_r}{1+\xi_r} N \beta^2 \right) = 2 \sum_{r=1}^{n} \binom{n}{r} \exp\left(-2 \frac{1-\xi_r}{1+\xi_r} N \beta^2 \right) \ , \tag{5}$$

where we have again set $\xi_r = (1-2p)^r$. We now bound each term in the above sum. We begin by considering the case $r = 1$.

Case $r = 1$. For this case, we begin by noting that

$$\frac{1-\xi_1}{1+\xi_1} = \frac{1-(1-2p)}{1+(1-2p)} = \frac{2p}{2(1-p)} = \frac{p}{1-p} \geq p \ .$$

Thus, we may bound

$$\begin{aligned}\binom{n}{1}\exp\left(-2\frac{1-\xi_1}{1+\xi_1}N\beta^2\right) &\le n\cdot\exp(-2pN\beta^2)\\ &= RN\cdot\exp\left(-2\frac{C\ln N}{N}N\beta^2\right)\\ &= RN\cdot\exp\left(-2C\beta^2\ln N\right)\\ &= RN\cdot N^{-2C\beta^2} = RN^{-2C\beta^2+1}\ . \end{aligned} \tag{6}$$

We now turn to larger values of r. It is most convenient to look at

$$-\frac{1}{N}\ln\left(\binom{n}{r}\exp\left(-2\frac{1-\xi_r}{1+\xi_r}N\beta^2\right)\right) = 2\beta^2\frac{1-\xi_r}{1+\xi_r} - \frac{1}{N}\ln\binom{n}{r}\ . \tag{7}$$

Our target is to lower bound (7) $\ge \Omega(\log N/N)$. We consider two more cases depending on the value of r.

Case 2: $2 \le r \le \frac{N}{2C\ln N}$**.** First, as $1+\xi_r \le 2$, it suffices to lower bound

$$\beta^2(1-\xi_r) - \frac{1}{N}\ln\binom{n}{r} = \beta^2\left(1-\left(1-\frac{2C\ln N}{N}\right)^r\right) - \frac{1}{N}\ln\binom{RN}{r}. \tag{8}$$

We will use the elementary inequality $e^{-z} \le 1-\frac{z}{2}$, which is valid for all $z\in[0,1]$.[3] As $r \le \frac{N}{2C\ln N}$ we have

$$\left(1-\frac{2C\ln N}{N}\right)^r \le e^{-r\frac{2C\ln N}{N}} \le 1-\frac{Cr\ln N}{N}\ .$$

Thus, applying also the standard upper bound $\binom{a}{b}\le\left(\frac{ea}{b}\right)^b$, we may lower bound (8) via

$$\begin{aligned}&\beta^2\left(1-\left(1-\frac{2C\ln N}{N}\right)^r\right) - \frac{1}{N}\ln\binom{RN}{r}\\ &\qquad\ge \beta^2\left(1-\left(1-\frac{Cr\ln N}{N}\right)\right) - \frac{r}{N}\ln\left(\frac{eRN}{r}\right)\\ &\qquad= \beta^2\frac{Cr\ln N}{N} - \frac{r}{N}\ln\left(\frac{eRN}{r}\right)\\ &\qquad= \frac{r}{N}\ln\left(N^{C\beta^2}\cdot\frac{r}{eRN}\right)\ .\end{aligned}$$

Clearly, the above bound decreases as r decreases. Hence, we substitute in r's minimum value $r=2$:

$$\frac{2}{N}\ln\left(N^{C\beta^2}\cdot\frac{2}{eRN}\right) \ge \frac{1}{N}\ln\left(N^{2C\beta^2-1}\right) \tag{9}$$

where the last inequality uses the assumption $R \le 2/e$. This completes the argument in this case.

Case 3: $r \ge \frac{N}{2C\ln N}$**.** In this case, we may bound

$$\left(1-\frac{2C\ln N}{N}\right)^r \le e^{-r\frac{2C\ln N}{N}} \le e^{-1}$$

and

$$\frac{1}{N}\ln\binom{n}{r} = \frac{1}{N}\ln\binom{RN}{r} \le \frac{1}{N}\ln 2^{RN} = R\ln 2\ .$$

Hence, in this case we may lower bound (7) as

$$2\beta^2\frac{1-\left(1-\frac{2C\ln N}{N}\right)^r}{1+\left(1-\frac{2C\ln N}{N}\right)^r} - \frac{1}{N}\ln\binom{n}{r} \ge 2\beta^2\frac{1-e^{-1}}{1+e^{-1}} - R\ln 2 > 0\ , \tag{10}$$

[3] This inequality may be readily verified using calculus.

where the strict inequality follows from (2).

Note that, for sufficiently large N, the lower bound in (9) is smaller than that in (10). Thus, for each $2 \leq r \leq n$, we may upper bound

$$\binom{n}{r} \exp\left(-2\frac{1-\xi_r}{1+\xi_r}N\beta^2\right) \leq e^{-\frac{1}{N}\ln\left(N^{2C\beta^2-1}\right)} = N^{-2C\beta^2+1} .$$

Returning now to (4), and recalling the bound for $r = 1$ from (6), we therefore have

$$2\sum_{r=1}^{n}\binom{n}{r} \exp\left(-2\frac{1-\xi_r}{1+\xi_r}N\beta^2\right) \leq (2RN)\cdot N^{-2C\beta^2+1} = 2RN^{-2C\beta^2+2} .$$

This completes the proof. □

3.4 Other Variants

In our investigation of EA codes we considered many variants. We begin by discussing other models for sampling expand-accumulate codes.

Different expanding matrices. There are other natural ways to choose the matrix for expanding the input prior to accumulating: that is, the B matrix from Definition 3.2. In order to be useful for efficient offline-online PCGs, we need to ensure that every row of B is as sparse as possible.

The following are natural choices for the distribution of the rows of B. We conjecture that the behaviour of any of the random codes generated by BA should be comparable in terms of minimum distance (and hence, in terms of resistance to linear attacks). Let $\ell \in \mathbb{N}$ be the row-weight parameter and let $\mathcal{R}$ be a ring. We consider the following distributions for independently sampling the rows $\vec{r}_1, \ldots, \vec{r}_n$ of the matrix B.

- Sample each $\vec{r}_i$ as a uniformly random vector in $\mathcal{R}^N$ of weight ℓ, i.e., sample $\vec{r}_i \leftarrow \mathsf{HW}_\ell^N(\mathcal{R})$. We denote the sampling of the matrix $H = BA \in \mathcal{R}^{n\times N}$ by $H \xleftarrow{\$} \mathsf{EAGenExa}(n, N, \ell, \mathcal{R})$.
- Sample each $\vec{r}_i$ as the concatenation of ℓ uniformly random unit vectors from $\mathcal{R}^{N/\ell}$, i.e., sample $\vec{r}_i \leftarrow \mathsf{Reg}_\ell^N(\mathcal{R})$. We denote the sampling of the matrix $H = BA \in \mathcal{R}^{n\times N}$ by $H \xleftarrow{\$} \mathsf{EAGenReg}(n, N, \ell, \mathcal{R})$.[4]

The above distributions might be preferable in practice. For example, if the noise is exact we do not have to worry about rows of abnormally large weight, in which case the online phase would be more efficient.

We conjecture that EA codes when the rows of the matrix B are sampled according to the exact distribution or the regular distribution should still be asymptotically good. Formally:

Conjecture 3.11 *There exist constants $R, \delta, C > 0$ such that the following holds. Let $N \in \mathbb{N}$, put $n = \lfloor RN \rfloor$ and let $\ell \in \mathbb{N}$ be an integer such that $\ell \geq C \ln N$. Then*

$$\Pr\left[\mathsf{d}(H) \geq \delta N \mid H \xleftarrow{\$} \mathsf{EAGenExa}(n, N, \ell, \mathbb{F}_2)\right] \geq 1 - 1/\mathsf{poly}(N) .$$

Assume further $\ell | N$. Then also

$$\Pr\left[\mathsf{d}(H) \geq \delta N \mid H \xleftarrow{\$} \mathsf{EAGenReg}(n, N, \ell, \mathbb{F}_2)\right] \geq 1 - 1/\mathsf{poly}(N) .$$

While we have left the $1/\mathsf{poly}(N)$ failure probability in the statement unspecified, be do not see any reason that the failure probability should deviate significantly from the $O(N^{-2C\beta^2+2})$ bound provided in Theorem 3.10.

Unfortunately, the Markov chain-based analysis of Lemma 3.9 does not appear to port over naturally when the rows of B are sampled according to either of these distributions, as the columns of B then fail to be independent.

Nonetheless, we endeavour to provide further evidence for this conjecture. We do this by considering the following matrices:

[4] This distribution can be naturally extended when $\ell \nmid N$: in this case, if $a = M \mod \ell$ we sample a unit vectors in $\mathcal{R}^{\lceil N/\ell \rceil}$ and $\ell - a$ unit vectors from $\mathcal{R}^{\lfloor N/\ell \rfloor}$.

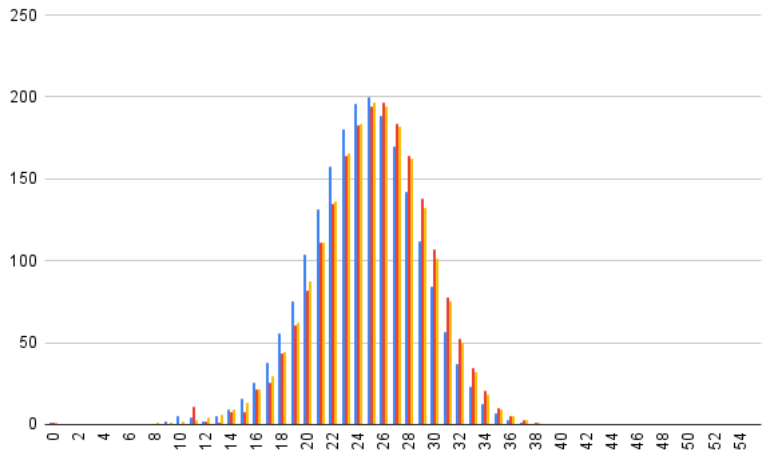

Figure 5: Plots of the average weight distributions for the 3 row sampling procedures: exact rows, regular rows, and Bernoulli rows. Blue corresponds to exact; red corresponds to regular; and orange corresponds to Bernoulli.

- $H_1 \xleftarrow{\$} \mathsf{EAGenExa}(11, 55, 11, \mathbb{F}_2)$;
- $H_2 \xleftarrow{\$} \mathsf{EAGenReg}(11, 55, 11, \mathbb{F}_2)$; and
- $H_3 \xleftarrow{\$} \mathsf{EAGen}(11, 55, 11/55 = 0.2, \mathbb{F}_2)$.

For these small parameters, we are able to explicitly compute the weight distribution of the codes generated by these (randomly sampled) matrices. That is, we can compute the vector $(a_0, a_1, \ldots, a_{55}) \in \mathbb{N}^{56}$ where a_j is the number of $\vec{x} \in \mathbb{F}_2^{11}$ for which $\mathcal{HW}(\vec{x}^\intercal H_i) = j$. For each of these sampling procedures, we ran 100 trials and computed the average weight distribution. The results are plotted in Figure 5.

While the broad conclusion is that the weight distributions all appear to be relatively similar, if anything it appears that regular noise is best in the sense that it puts the most mass on heavier weights. Providing theoretical guarantees for these other row distributions remains an open problem meriting further study.

Arbitrary rings. Theorem 3.10 applies only to the case of the binary field, and one can naturally wonder about the distance over arbitrary rings. We conjecture that the minimum distance should at the very least not degrade over larger rings; in fact, we believe that it should increase commensurately. However, the expander Hoeffding bound that we apply does not appear to work well in this case; in particular, we are forced to require the rate $R \ll \frac{1}{\ln q}$, where q is the size of the ring $\mathcal{R}$.

However, we believe that this is an artifact of the proof. To justify this suspicion, we consider the following:

- $H_1 \xleftarrow{\$} \mathsf{EAGenExa}(4, 20, 4, \mathbb{F}_{17})$;
- $H_2 \xleftarrow{\$} \mathsf{EAGenExa}(4, 20, 4, \mathbb{Z}_{16})$;

Further, we consider the sampling procedure where each row is sampled to be a uniformly random vector of weight ℓ where *all the nonzero coordinates take value* 1.[5] We denote the sampling procedure as $H \xleftarrow{\$} \mathsf{EAGenExa}^*(k, n, \ell, \mathcal{R})$. Let also

- $H_3 \xleftarrow{\$} \mathsf{EAGenExa}^*(4, 20, 4, \mathbb{F}_{17})$;
- $H_4 \xleftarrow{\$} \mathsf{EAGenExa}^*(4, 20, 4, \mathbb{Z}_{16})$.

For these small parameters, we are able to explicitly compute the weight distribution of the codes generated by these (randomly sampled) matrices. For each of these sampling procedures, we ran 100 trials and computed the average weight distribution, which we subsequently normalized by $100/q^4$ where q is the ring size. The results are provided in Figure 6.

[5] Of course, this notion is not interesting if $\mathcal{R} = \mathbb{F}_2$.

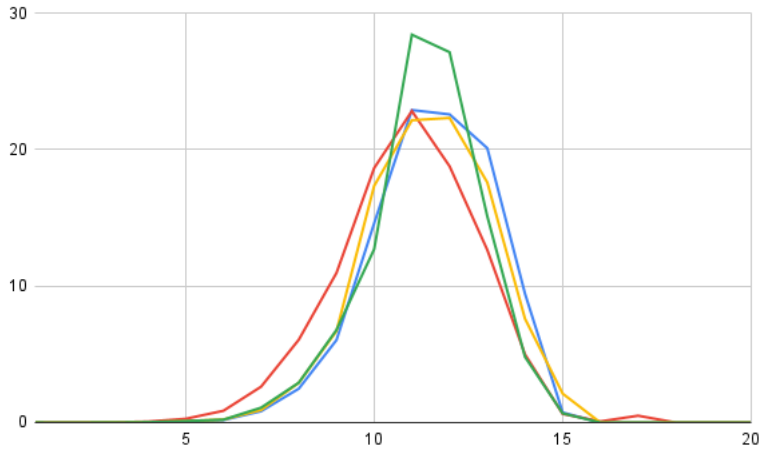

Figure 6: In this graphic we have plotted the average weight distributions for 4 of the sampling procedures outlined in this section, normalized by $\frac{100}{q^4}$ where q is the ring size. Blue corresponds to H_1; red to H_2; yellow to H_3; and green to H_4.

The conclusions we draw are the following. First, the weight distribution only appears to get better (in the sense that more mass moves towards higher weight vectors) as q increases. Secondly, we do not notice too much difference between the case where the nonzero entries are all set to 1, although the effect is more noticeable for $\mathbb{Z}_{16}$ than $\mathbb{F}_{17}$. However, we caution that for codes over $\mathbb{Z}_{2^k}$ the amount of security one gets is no greater than the amount one gets over $\mathbb{Z}_2$ as one can always perform a modular reduction/projection attack to leak one bit at a time (such an attack is not covered by the linear attacks framework).

To put it concisely, our experiments suggest that the concrete time to break the EA-LPN assumption should only increase as the characteristic of the ring increases (at least, assuming one is restricted to modular reduction followed by linear attacks). Proving this theoretically remains an interesting challenge for future work.

3.5 Tightness of the Security Analysis

As we are interested in concrete parameters, we are naturally motivated to ask whether or not the analysis of Theorem 3.10 is tight. We now explore certain avenues to potentially improve the above result, especially when dealing with concrete parameters.

Rate-Distance Tradeoff. One aspect of Theorem 3.10 that one could hope to improve is the rate-distance tradeoff (2). Inspecting the proof, we see that this assumption was used in bounding the probability that vectors $\vec{x} \in \mathbb{F}_2^n$ of weight at least $\frac{N}{C \ln N}$ have an encoding of weight at most δN. Indeed, if $n = RN$ is too large then we are unable to guarantee that it is not the case that there are exponentially many vectors $\vec{x}$ of weight $\approx \frac{n}{\log N}$ that have an encoding of weight at most δN.

However, even if such attack vectors exist, it is not clear to us that any efficient adversary could *find* one. Recall that proving that EA codes have good minimum distance shows that they are not susceptible to linear attacks, *without* the requirement that the algorithm implementing the linear attack be efficient. On the other hand, the EA-LPN assumption only implies that it is computationally infeasible to find an attack vector whose encoding has low weight.

Therefore, it is reasonable to believe that for values of $R = n/N$ and δ that violate the inequality (2), the code generated by $H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)$ could still have *pseudodistance* δ. We provide below a formal definition of pseudodistance. (For concreteness, the definition is specialized to the case of $\mathbb{F}_2$, although the definition can easily be generalized to other rings.)

Definition 3.12 (Pseudodistance) *Let* $\mathbf{C}$ *be a probabilistic code generation algorithm such that* $\mathbf{C}(n, N)$ *outputs a matrix* $H \in \mathbb{F}_2^{n \times N}$*. For a weight parameter* $\delta(\lambda)$*, we say that* $\mathbf{C}(n(\lambda), N(\lambda))$ *has* pseudodistance $\delta(\lambda)$ *if for every PPT algorithm* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathcal{A}(H) = \vec{x} \text{ s.t. } \vec{x} \neq \vec{0} \text{ and } \mathcal{HW}(\vec{x}^\intercal H) \leq \delta N \mid H \xleftarrow{\$} \mathbf{C}(n, N)\right] \leq \mathsf{negl}(\lambda) \ .$$

The above definition has appeared previously in the literature, often referred to as the *binary shortest vector problem (bSVP)* [AHI+17].[6] We consider it reasonable to suspect that, so long as $R = n/N$ and δ satisfy $\delta < (1-R)/2$,[7] $\mathsf{EAGen}(n, N, p)$ has pseudodistance δ assuming $p = \Omega(\log N/N)$.

Density of B, theoretically. As a conceptual question, one can ask how dense the matrix B must be. Note that the density of a row of B greatly influences the efficiency of the online phase of our PCG construction, so this is an important point to understand. Theorem 3.10 requires each row to have (expected) density $\Omega(\log N)$, and it is reasonable to wonder if the rows could have constant density.

If we are hoping for the code to be asymptotically good, *i.e.*, both rate and relative distance constant, this is unfortunately not possible. Indeed, in this case we can show that, in expectation, at least $\delta^\ell n$ vectors of weight 1 have encodings of weight at most δ. Note that $\vec{e}_1^{\intercal}B, \ldots, \vec{e}_n^{\intercal}B$ are a collection of n random vectors with distribution $\mathsf{Ber}^N_{\ell/N}(\mathbb{F}_2)$; thus, we analyze the probability that $\vec{x}^{\intercal}A$ has weight $\leq \delta N$ if $\vec{x} \xleftarrow{\$} \mathsf{Ber}^N_{\ell/N}(\mathbb{F}_2)$. Note that if all of $\vec{x}$'s nonzero coordinates land in the interval $\{N - \delta N + 1, \ldots, N\}$, then $\vec{x}^{\intercal}A$ will have weight at most δN, and this event occurs with probability $(1-\ell/N)^{N(1-\delta)} \approx e^{-\ell(1-\delta)} \geq \delta^\ell$. As there are n unit vectors, this implies in expectation that there will be at least $\delta^\ell n$ weight one messages with weight $\leq \delta N$ encoding. If $1/\ell$, δ, and $R = n/N$ are all $\Omega(1)$, then we find that there are linearly many codewords of weight at most δN. A similar analysis applies for other choices of row distribution: so long as their expected weight is ℓ, the weight of the vector after accumulation will be at most δN with probability roughly δ^ℓ.

The above is not too surprising in light of the known impossibility results concerning RA codes. Recalling Remark 3.4, for such a code the matrix $B \in \mathbb{F}_2^{N/\ell \times N}$ is chosen so that the rows are all uniformly random of weight ℓ, subject to the constraint that every column has weight 1. For $\ell = O(1)$, it is know that the minimum distance of such codes is at most $O(N^{1-1/(\ell-1)})$ [BMS08]. Thus, we cannot hope to have an asymptotically good code with just one round of accumulation.

Better parameters with rejection sampling. By the above discussion and, indeed, the proof of Theorem 3.10, low-weight codewords in an EA code are typically obtained as the encodings of low-weight messages. If one is willing to spend a bit more time to check the quality of the code after sampling it by checking the weight of low-weight messages, it is reasonable to expect that this would improve the failure probability.

More specifically, upon sampling the generator matrix $H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)$, one could run the following tester T: for each nonzero message vector $\vec{x} \in \mathbb{F}_2^n$ of weight at most a, compute $\vec{x}^{\intercal}H$, and verify that it has weight at least δN. When a is a small constant, such a test is feasible. Moreover, heuristically one should expect that if H passes the test then the probability that it fails to have distance δ should be at most

$$\sum_{r=a+1}^{n} \binom{n}{r} \exp\left(-2\frac{1-\xi_r}{1+\xi_r} N\beta^2\right) \tag{11}$$

which is just the bound in (5) where we have removed the terms for $r \leq a$. (Of course, this assumes that the probability $\mathcal{HW}(\vec{x}^{\intercal}H) \leq \delta N$ for non $\vec{x} \in \mathbb{F}_2^n$ of weight at least $a+1$ does not decrease if we condition on the event $\mathcal{HW}(\vec{y}^{\intercal}H) > \delta N$ for all nonzero $\vec{y} \in \mathbb{F}_2^n$ of weight at most a. But we feel that this is reasonable to suspect, even if it appears very difficult to prove formally.)

Already when $a = 1$, we can see some noticeable savings. Indeed, just with $a = 1$ we extrapolated the value of (11) for $n = 2^{20}, 2^{25}$ and 2^{30} with $N = 5n$. For $C = 2.3$ our estimate of the failure probability for the EA code having distance at least $0.005N$ or $0.02N$ is noticeably better than the bound from Figure 8. The result is summarized in Figure 7. We see that the failure probability decreases in all cases by a factor of roughly 100.

3.6 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

[6] In [AHI+17] the code is presented in terms of a parity-check matrix. In our case, presenting a code via a generator matrix is more convenient. As one can efficiently pass between the two representations the definitions are equivalent.

[7] This assumption implies that the standard *information set decoding* attack does not succeed.

$C = 2.3$			
δ \ n	2^{20}	2^{25}	2^{30}
0.005	0.000887	0.000405	0.000185
0.02	0.00720	0.00512	0.00374

Figure 7: In this table, we list the (extrapolated, heuristic) failure probabilities for an EA code of rate 1/5 with the given parameters, conditioned on all the rows having weight at least δN. We see significant savings (roughly a factor of 100) compared to Figure 8.

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

Figure 8: In this table, we list the (extrapolated) 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.

$\mathsf{d}(H) \leq \delta N$ 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 $N = 5 \cdot n$. However, computing (3) for these values of n, N is beyond our computational resources, so we instead computed the failure probability for smaller values $n = 2^9, 2^{10}, \ldots, 2^{15}$, and extrapolated the failure probability.[8]. Our results are summarized in Figure 8.

For context, we recall Lemma 2.6 which translates minimum distance into security against linear tests. It says that if $H \xleftarrow{\$} \mathsf{EAGen}(n, N, p)$ has minimum 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 8, 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[9] 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 8, 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 3.10, 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.

[8] We used exponential regression in $\log_2 n$, which fit very well: R values in excess of .99.

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

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 8.

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 $H \leftarrow \mathsf{EAGenReg}(n, 5n, \ell)$.

Being aggressive, we consider choosing smaller values of ℓ, and then empirically estimate the minimum (relative) weight of a row of an EA matrix H. For our applications we would like $n = 2^{20}$, 2^{25} and 2^{30}. It is reasonable to choose $R = 0.2$, implying $N = 5 \cdot n$. For $\ell = 7, 9, 11$, we endeavour to empirically estimate the minimum row weight of a matrix $H \xleftarrow{\$} \mathsf{EAGenReg}(n, 5 \cdot n, \ell)$.[10]

However, for this value of n this is beyond our computational resources. Instead, we computed the average minimum row weight over 100 trials for smaller values $n = 2^9, 2^{10}, \ldots, 2^{15}$, and extrapolated the failure probability. We used exponential regression in $\log_2 n$, which fit the data very well (R values all in excess of 0.99). The results are given in Figure 9b.

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

Aggressive parameters. In the previous section, we could show, e.g., that when $C = 3$ the minimum distance of the code generated by $H \xleftarrow{\$} \mathsf{EAGen}(2^{30}, 5 \cdot 2^{30}, \frac{C \ln(5 \cdot 2^{30})}{5 \cdot 2^{30}})$ is very likely to exceed $0.05 \cdot 5 \cdot 2^{30}$. But again, recalling our discussion of pseudodistance (Definition 3.12), it seems very difficult to find a codeword of small weight. In particular, assuming $\delta < (1 - R)/2$ it is plausible that it is hard to find a codeword of weight less than δ (at least, assuming C is sufficiently large to guarantee no rows have weight that small). Thus, taking $R = 0.2$ it is reasonable to suspect that it is hard to find a codeword of weight $< 0.4 \cdot N$. This implies the following values of t are sufficient for 128 bits of security: for $n = 2^{20}$, $t = 94$; for $n = 2^{25}$, $t = 90$; for $n = 2^{30}$, $t = 85$.

Further, if we make R smaller so that it is plausibly hard to find a codeword of weight less than $\approx 0.5N$, the following values of t plausibly yield 128 bits of security: for $n = 2^{20}$, $t = 75$; for $n = 2^{25}$, $t = 72$; for $n = 2^{30}$, $t = 68$.

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 Offline-Online Pseudorandom Correlation Generators from Expand-Accumulate Codes

In this section, we introduce the notion of offline-online pseudorandom correlation generators (PCGs) and show how to construct offline-online PCGs with a highly cache-friendly and parallelizable offline

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

phase from EA codes. At a high level, the aim of offline-online PCGs is to target a better distribution of the computational effort over time: when using a PCG in a secure computation protocol, the cost of the online phase is largely dominated by communication (since computation typically involves extremely cheap operations, a few XOR per gate). This means that if we can push part of the PCG seed expansion cost to the online phase, it will be absorbed by the communication cost to some extent, allowing to reduce the offline expansion cost without harming the runtime of the online phase. Below, we put forward a formal definition of offline-online PCG which captures this goal.

Definition 4.1 (Offline-Online PCG) *Let $\mathcal{C}$ be a reverse-sampleable correlation generator, where for $n = n(\lambda)$, $\tau_\sigma = \tau_\sigma(\lambda)$, for $\sigma = 0, 1$, the σ-th output of of $\mathcal{C}$ is in $(\{0,1\}^{\tau_\sigma})^n$. An* offline-online PCG for $\mathcal{C}$ *with storage cost $N(\lambda)$ and output locality $\ell(\lambda)$ is a triple of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Offline}, \mathsf{PCG.Online})$ *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of offline and online keys $((\mathsf{k}_0^{\mathsf{off}}, \mathsf{k}_0^{\mathsf{on}}), (\mathsf{k}_1^{\mathsf{off}}, \mathsf{k}_1^{\mathsf{on}}))$;*
- $\mathsf{PCG.Offline}(\sigma, \mathsf{k}_\sigma^{\mathsf{off}})$ *is a polynomial-time* offline expansion algorithm *that given party index $\sigma \in \{0,1\}$ and an offline key $\mathsf{k}_\sigma^{\mathsf{off}}$, outputs an offline string $Y_\sigma \in \{0,1\}^N$. When considering arithmetic correlations over $\mathbb{F} = \mathbb{F}(\lambda)$, the output of* PCG.Offline *is a vector $Y_\sigma \in \mathbb{F}^N$.*
- $\mathsf{PCG.Online}(\sigma, \mathsf{k}_\sigma^{\mathsf{on}}, Y_\sigma, x)$ *is a polynomial-time* online algorithm *that on input $\sigma \in \{0,1\}$, online key $\mathsf{k}_\sigma^{\mathsf{on}}$, and index $1 \le x \le n$, outputs a value $y_\sigma \in \{0,1\}^{\tau_\sigma}$, or $y_\sigma \in \mathbb{F}^{\tau_\sigma}$ in the arithmetic case. Furthermore,* PCG.Online *reads at most $\ell(\lambda)$ entries of Y_σ.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Offline}, \mathsf{PCG.Online})$ *should satisfy the following: if we define* PCG.Expand *that applies* PCG.Offline *and then concatenates the outputs of* PCG.Online *with $x = 1, 2, \ldots, n$, the resulting* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *is a PCG for $\mathcal{C}$ according to Definition 2.10.*

Definition 4.1 is purely syntactic, and allows to better discuss the concrete efficiency of existing PCG schemes in terms of how their cost is distributed between the offline and the online phase. In particular, we expect the locality parameter ℓ to be small, growing at most logarithmically with n.

4.1 Generic Offline-Online PCGs

We start by observing that any PCG gives a direct construction of an offline-online PCG with storage cost n and output locality 1, where the online phase is trivial: PCG.Offline is just PCG.Expand, and PCG.Online takes as input the full pseudorandom string Y_σ and the index x, and outputs the x-th bit of Y_σ. In a sense, in such a PCG, some idle time of the online part is wasted: in concrete MPC applications, the online runtime is dominated by communication (in essence, we have room for more computation online).

At the other end of the spectrum, a PCF also provides a direct construction of an offline-online PCG: PCG.Gen generates PCF keys $(Y_0, Y_1) = (\mathsf{k}_0, \mathsf{k}_1)$, PCG.Offline does nothing, and $\mathsf{PCG.Online}(\sigma, \bot, Y_\sigma, x)$ outputs $\mathsf{PCF.Eval}(\mathsf{k}_\sigma, x)$. There, the storage cost is small (it only amounts to storing the offline keys) and PCG.Online must access Y_σ (which is just the offline key) in full. With this cost distribution, the computation during the online phase is likely to largely overflow the "available idle time" that results from communicating during the online phase. Below, we show that our newly introduced EA codes lead to an offline-online PCG with a much better cost balance.

4.2 Offline-Online PCGs from EA Codes

In the following we give offline-online PCGs for subfield VOLE from EA Codes, building on puncturable pseudorandom functions. Let $p, q \in \mathbb{N}$, such that $q = p^r$. Recall that the goal of subfield VOLE is to obtain output correlations (w_i, Δ), (u_i, v_i), such that $w_i = \Delta \cdot u_i + v_i$, where $u_i \xleftarrow{\$} \mathbb{F}_p$, $v_i \xleftarrow{\$} \mathbb{F}_q$, $\Delta \xleftarrow{\$} \mathbb{F}_q$ is the same across all evaluations, and $w_i := \Delta \cdot u_i + v_i$. As for standard 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) OT correlation over $\mathbb{F}$. Further, note that for the purpose of offline-online PCGs for OT, the PPRF can be replaced by a strong unpredictable puncturable function, as we show in Section 6.4 (note though that this change requires a stronger assumption on the hash function used in the transformation from PCGs for subfield VOLE to OT).

Recall that $(\mathsf{HW}_t^N, \mathbb{F}_p)$-$\mathsf{EA\text{-}LPN}(n, N)$ states that for $H \xleftarrow{\$} \mathsf{EA}(n, N, \rho, \mathbb{F}_p)$ it is hard to distinguish $H \cdot \vec{e}$ from random for sparse vectors $\vec{e}$, where H is of the form $H = BA$ for an accumulator matrix

$A \in \mathbb{F}_p^{N \times N}$ and a matrix $B \in \mathbb{F}_p^{n \times N}$ with rows sampled according to Ber_ρ^N. To simplify the description of the online keys we will slightly deviate from this and sample the rows of the matrix B by choosing ℓ positions $p_i \in [N]$ and corresponding weights $r_i \in \mathbb{F}_p$, where $\ell = \rho \cdot N$ (i.e., instead of having an expected number of ℓ non-zero entries, we will obtain exactly or less than ℓ non-zero entries). Further, we assume a PRF $\tilde{F}\colon \{0,1\}^\lambda \times [n] \to ([N] \times \mathbb{F}_p)^\ell$ to sample $\{(r_i, p_i)\}_{i \in [\ell]}$ pseudorandomly, where both parties know the key. We will denote this by $\mathsf{EA\text{-}LPN}'(n, N, \ell)$ *relative to* $\tilde{F}$ in the following. Note that while this differs from the EA assumption we analysed in the previous section, the altered assumption remains plausible since the combinatorial distance properties of the generated matrix stay essentially the same. Further note that the construction generalizes to standard EA codes at the cost of storing a matrix $B \in \mathbb{F}_p^{n \times N}$ with rows sampled according to Ber_ρ^N, which can be reused across instantiations.

As noise distribution we choose HW_t^N over $\mathbb{F}_p$. Note that building on $\mathsf{EA\text{-}LPN}$ with regular weight-t noise instead one can achieve a more efficient instantiation, since the domain size of the PPRF shrinks from N to N/t.

Theorem 4.2 *Let $n, N \in \mathbb{N}$ with $n < N$. Let $p, q, \ell \in \mathbb{N}$ with $p = q^r$. Let $\tilde{F}\colon \{0,1\}^\lambda \times [n] \to ([N] \times \mathbb{F}_p)^\ell$ be a PRF and let $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ be a puncturable PRF over domain $[N]$ and range $\mathbb{F}_q$. Then, assuming the $(\mathsf{HW}_t^N, \mathbb{F}_p)$-$\mathsf{EA\text{-}LPN}'(n, N, \ell)$ assumption relative to $\tilde{F}$ (as defined above), the construction in Fig. 10 is an offline-online PCG for subfield VOLE to generate n correlations with storage cost N and output locality ℓ.*

Note that we ignore the cost for storing and accessing $\vec{u}^{\mathsf{off}}$ here, since $\vec{u}^{\mathsf{off}}$ is an accumulated unit vector which can be stored and accessed in compact representation and therefore does not affect the overall costs significantly.

Note that the proof that the given strategies is indeed a subfield VOLE is identical to the analysis in [BCG+19b, BCG+19a] up to swapping the code to the EA code. We therefore omit it here.

4.3 The Offline-Online Balance

Since the point of offline-online PCG is to push some of the offline work back to the offline phase to happen during idle times of the computation, the right choice of balance is of course application-dependent. Here, we make a few estimations to help the reader getting a high level intuition of what to expect. Concretely, there are two causes of slowdown in the online phase that create available idle time: bandwidth and latency.

Bandwidth limitations. Suppose for example that the parties use the GMW protocol on a circuit with 2^{20} `and` gates; ignoring for now all latency issues (hence the round complexity), they will have to communicate 2^{22} bits. Over a 10MB network, this $\approx 1/20$ second; for 100MB networks, this becomes $\approx 1/200$ seconds. To generate the necessary correlations, the parties have expanded $5 \cdot 2^{21}$ machine words of preprocessing material, which can all fit in the L3 cache of a modern computer. Randomly an entry of the L3 cache takes a few nanoseconds; hence, in the time used to communicate, around 10^6 (with a 100MB bandwidth) to 10^7 (with 10MB) accesses to the cache can be made. With a reasonable parameter (midway between aggressive and conservative) of 20 accesses per output and 2^{20} outputs to compute, this is already roughly in the range of the available idle time.

Latency. When latency is involved, the amount of available idle time skyrockets: cross-continent communication can easily incur $\approx$ 150ms of latency per round, and even same-continent communication can be of the order of a few dozen milliseconds. As soon as the protocol takes a few rounds, this is more than enough idle time to perform the required $20 \cdot 2^{20}$ random memory accesses.

5 Pseudorandom Correlated Functions from Expand-Accumulate Codes

In this section, we put forth new constructions of pseudorandom correlation functions (PCFs) under the $\mathsf{EA\text{-}LPN}$ assumption. We show how to obtain PCFs for the subfield VOLE, OT and general degree-two correlations over a ring. We do this by combining $\mathsf{EA\text{-}LPN}$ with a *distributed comparison function*, a special type of FSS for comparisons. Below we give a high-level summary of the ideas behind the PCF for subfield-VOLE.

PARAMETERS:

- Security parameter 1^λ, integers $N > n$, $q = p^r$, and noise weight t.
- A PPRF scheme ($F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval}$) over domain $[N]$ and range $\mathbb{F}_q$ to share the noise vector.
- A PRF $\tilde{F}\colon \{0,1\}^\lambda \times [n] \to ([N] \times \mathbb{F}_p)^\ell$ to generate a pseudorandom sparse matrix B defining the EA code.

CORRELATION: Outputs $((u_i, v_i), (w_i, \Delta))$, such that $w_i = \Delta \cdot u_i + v_i$, where $u_i \xleftarrow{\$} \mathbb{F}_p$, $v_i \xleftarrow{\$} \mathbb{F}_q$, $\Delta \xleftarrow{\$} \mathbb{F}_q$ and $w_i := \Delta \cdot u_i + v_i$.

GEN:

- Pick a random size-t set $S = \{s_1, \ldots, s_t\} \subset [N]$, sorted in increasing order.
- Pick a random vector $\vec{y} = (y_1, \ldots, y_t) \in (\mathbb{F}_p^*)^t$ and $\Delta \xleftarrow{\$} \mathbb{F}_q$.
- For $i \in [t]$ proceed as follows:
  - Draw $k \xleftarrow{\$} \{0,1\}^\lambda$.
  - Draw $k_i \leftarrow F.\mathsf{KeyGen}(1^\lambda)$.
  - Compute $k_i^* \leftarrow F.\mathsf{Puncture}(k_i, \{s_i\})$.
  - Compute $\Delta_i := F(k_i, s_i) - \Delta \cdot y_i$.
  - Set $K_0^i := (k_i^*, \Delta_i)$ and $K_1^i := k_i$.
- Let $\mathsf{k}_0^{\mathsf{off}} := (n, N, \{K_0^i\}_{i\in[t]}, S, \vec{y})$, $\mathsf{k}_1^{\mathsf{off}} := (n, N, \{K_1^i\}_{i\in[t]}, \Delta)$ and $\mathsf{k}_0^{\mathsf{on}} := \mathsf{k}_1^{\mathsf{on}} := k$.
- Output $((\mathsf{k}_0^{\mathsf{off}}, \mathsf{k}_0^{\mathsf{on}}), (\mathsf{k}_1^{\mathsf{off}}, \mathsf{k}_1^{\mathsf{on}}))$.

OFFLINE: On input $(\sigma, \mathsf{k}_\sigma^{\mathsf{off}})$:

1. If $\sigma = 0$, parse $\mathsf{k}_0^{\mathsf{off}}$ as $(n, N, \{K_0^i\}_{i\in[t]}, S, \vec{y})$ and proceed as follows:
   - Define $\vec{\mu} \in \mathbb{F}_p^N$ to be the vector with
   $$\mu_j := \begin{cases} y_i & \text{if } j = s_i \text{ for some } i \in [t] \\ 0 & \text{else} \end{cases}.$$
   - For all $i \in [t]$, define $\vec{v}_0^i \in \mathbb{F}_p^N$ to be the vector with
   $$v_{0,j}^i := \begin{cases} F.\mathsf{Eval}(k_i^*, \{s_i\}, j) & \text{if } j \neq s_i \\ \Delta_i & \text{else} \end{cases}.$$
   - Set $\vec{u}_{\mathsf{off}} := A \cdot \vec{\mu} \in \mathbb{F}_p^N$ and $\vec{v}_{\mathsf{off}} := A \cdot \left(\sum_{i=1}^t \vec{v}_0^i\right) \in \mathbb{F}_q^N$.
   - Output $Y_0 := (\vec{u}_{\mathsf{off}}, \vec{v}_{\mathsf{off}})$
2. If $\sigma = 1$, parse $\mathsf{k}_1^{\mathsf{off}}$ as $(n, N, \{K_1^i\}_{i\in[t]}, \Delta)$ and proceed as follows:
   - Define $\vec{w}_0^i \in \mathbb{F}_p^N$ to be the vector with $w_{0,j}^i := F(k_i, j)$ for all $i \in [t]$.
   - Compute $\vec{w}_{\mathsf{off}} := A \cdot \left(\sum_{i=1}^t v_1^i\right) \in \mathbb{F}_q^N$.
   - Output $Y_1 := (\vec{w}_{\mathsf{off}}, \Delta)$.

ONLINE: On input $(\sigma, k_\sigma^{\mathsf{on}}, Y_\sigma, x)$:

1. If $\sigma = 0$, parse k_0^{on} as k and Y_0 as $(\vec{u}_{\mathsf{off}}, \vec{v}_{\mathsf{off}})$ and proceed as follows:
   - For $i \in [\ell]$ set $(p_i, r_i) := \tilde{F}(k, x)_i \in [N] \times \mathbb{F}_p$. Compute
     - $u_x := \sum_{i=1}^\ell r_i \cdot u_{\mathsf{off}, p_i}$,
     - $v_x := \sum_{i=1}^\ell r_i \cdot v_{\mathsf{off}, p_i}$
   - Output (u_x, v_x).
2. If $\sigma = 1$, parse k_0^{on} as k and Y_1 as $(\vec{w}_{\mathsf{off}}, \Delta)$ and proceed as follows:
   - For $i \in [\ell]$ set $(p_i, r_i) := \tilde{F}(k, x)_i \in [N] \times \mathbb{F}_p$. Compute
     - $w_x := \sum_{i=1}^\ell r_i \cdot w_{\mathsf{off}, p_i}$
   - Output (w_x, Δ).

Figure 10: Offline-online PCG for n sets of subfield VOLE.

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 $u \in \mathbb{F}_2$. The security properties of PCF are defined in Section 2.7. 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 3.1.

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. We elaborate below.

5.1 Distributed Comparison Functions

In this section, as in previous works on PCG and PCF, we assume that the noise $\vec{e}$ has a regular structure. That is, $\vec{e}$ is a concatenation of t unit vectors of length N/t, for some noise parameter t. All our constructions easily extend to the setting where $\vec{e}$ is an arbitrary weight-t vector, albeit at the cost of a loss in efficiency.

Given a regular noise vector $\vec{e}$, the accumulated noise $\vec{e}' = A \cdot \vec{e}$ (over group $\mathbb{G}$)[11] has the following structure:

$$\vec{e}' = (\vec{e}')^{(1)} \| (\vec{e}')^2 \| \cdots = (0, \ldots, 0, \beta, \ldots, \beta) \| (\beta, \ldots, \beta, 0 \ldots, 0), \| \ldots$$

where depending on the parity of i, each $(\vec{e}')^{(i)}$ starts with a non-zero $\beta \in \mathbb{G}$ or zero.

For simplicity, consider the case that i is even (we will later explain how to handle odd cases), then $(\vec{e}')^{(i)}$ can be described as the output of a comparison function:

$$(\vec{e}')^{(i)}_j = \begin{cases} \beta & \text{if } x < \alpha \\ 0 & \text{else} \end{cases}$$

for some $\alpha \in [N/t]$.

This suggest the following approach: we use t instances of a *function secret sharing* (FSS) scheme for comparison functions, also known as a *distributed comparison function* [BGI19, BCG+21] (DCF). A DCF allows to share a comparison function $f^\beta_{<\alpha}$ into two functions (F_0, F_1) such that $F_0(j) + F_1(j) = f^\beta_{<\alpha}(j) = \beta \cdot [j < \alpha]$ for all j, yet (α, β) remain computationally hidden given F_σ (for $\sigma = 0$ or 1). Then,

[11] For concreteness, one can assume $\mathbb{G}$ is the additive group of a field, but as this will be useful later we state it in this generality.

we can simply set $\mathsf{k}_\sigma = (K_\sigma^0, \cdots, K_\sigma^{t-1})$ for $\sigma = 0, 1$, where the $K_\sigma^{(i)}$ are independent FSS keys for the comparison functions $F_{j_i,\beta}$, where each j_i denotes the position of the 1 in the i-th block of $\vec{e}$.

Our construction follows the above outline, with a twist: as in previous works on PCG [BCG+19a, SGRR19], we observe that we do not need the full power of FSS. Indeed, the latter guarantees that (α, β) remains hidden to both parties, while in our construction it is okay for P_0 to know α; we call such a primitive a *relaxed distributed comparison function* (RDCF).

Relaxed Distributed Comparison Function In the following we consider the space $\{0,1\}^m$ as ordered set, by interpreting bit-strings as numbers in the interval $[0, 2^m - 1]$ and taking the order induced by $<$ on $\mathbb{N}$.

Definition 5.1 (Relaxed distributed comparison function) *Let $\mathbb{G}$ be an additive group, $m \in \mathbb{N}$, $\alpha \in \{0,1\}^m$, $\beta \in \mathbb{G}$ and*

$$f_{<\alpha}^{\beta} \colon \{0,1\}^m \to \mathbb{G},\ f_{<\alpha}^{\beta}(x) := \begin{cases} \beta & \text{if } x < \alpha \\ 0 & \text{else} \end{cases}$$

be the comparison function defined by α and β.

We say a tuple of PPT algorithms $\mathsf{RDCF} = (\mathsf{Setup}, \mathsf{Eval})$ *is a* relaxed distributed comparison function *if:*

- $\mathsf{Setup}(1^\lambda, \alpha, \beta)$ *on input of* 1^λ, $\alpha \in \{0,1\}^m$, $\beta \in \mathbb{G}$ *outputs a pair of keys* (k_0, k_1)
- $\mathsf{Eval}(\sigma, k_\sigma, x)$ *is a deterministic function that on input* $\sigma \in \{0,1\}$, *key* k_σ *and value* $x \in \{0,1\}^m$ *outputs a value* $y \in \mathbb{G}$

such that the following holds:

Correctness: *For all $\lambda \in \mathbb{N}$, for all $\alpha \in \{0,1\}^m$, $\beta \in \mathbb{G}$, for all (k_0, k_1) in the image of* $\mathsf{Setup}(1^\lambda, \alpha, \beta)$, *and for all $x \in \{0,1\}^m$ it holds:*

$$\mathsf{Eval}(0, k_0, x) - \mathsf{Eval}(1, k_1, x) = f_{<\alpha}^{\beta}(x).$$

Security: *The security requirement, intuitively, is that the party holding k_0 learns nothing about the payload β and the party holding k_1 learns nothing about the path α* and *the payload β. More formally, we require:*

- *For all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all $\alpha \in \{0,1\}^m$, $\beta, \beta' \in \mathbb{G}$ and for all $\lambda \in \mathbb{N}$:*

$$\begin{aligned} &|\Pr[\mathcal{A}(k_0) = 1 \mid (k_0, k_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha, \beta)] \\ &- \Pr[\mathcal{A}(k_0) = 1 \mid (k_0, k_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha, \beta')] \leq \mathsf{negl}(\lambda). \end{aligned}$$

- *For all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all $\alpha, \alpha' \in \{0,1\}^m$, $\beta, \beta' \in \mathbb{G}$ and for all $\lambda \in \mathbb{N}$:*

$$\begin{aligned} &|\Pr[\mathcal{A}(k_1) = 1 \mid (k_0, k_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha, \beta)] \\ &- \Pr[\mathcal{A}(k_1) = 1 \mid (k_0, k_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha', \beta')] \leq \mathsf{negl}(\lambda). \end{aligned}$$

Pseudorandomness: $\mathsf{Eval}(0, k_0, \cdot)$ *and* $\mathsf{Eval}(1, k_1, \cdot)$ *define pseudorandom functions.*

Constructing RDCF. In Figure 11, we show our construction based on any PRG that maps a λ-bit string to an output of 2λ bits plus one element of the group $\mathbb{G}$. Compared with the standard DCF construction from [BCG+21], where each key is of size $2m(\lambda + \log|\mathbb{G}|)$, one of the party's keys in our RDCF is only of size λ, and the other is around half the size of [BCG+21]. Further, our construction saves on average $m/2$ calls to the PRG $C \colon \{0,1\}^\lambda \to \mathbb{G}$ per Eval operation, since C has only be invoked $\#\{i \mid x_i = 0\}$ times on input $x \in \{0,1\}^m$ (instead of m times for the standard DCF construction). Additionally, we show that in the setting where a full evaluation is feasible (i.e., when one is interested in constructing an incremental PCG) keys can be distributed in 2-PC with a simple, 2-round protocol based on 2-round OT.

Lemma 5.2 *If $G \colon \{0,1\}^\lambda \to \{0,1\}^{\ell+2\lambda}$ is a pseudorandom generator, then* $\mathsf{RDCF} = (\mathsf{Setup}, \mathsf{Eval})$ *as defined in Figure 11 is a relaxed distributed comparison function.*

Let $(\mathbb{G}, +)$ be an abelian group and $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element in $\mathbb{G}$.
Let $G_0, G_1 \colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ and $C \colon \{0,1\}^\lambda \to \mathbb{G}$ to be such that $G(k) := C(k) \| G_0(k) \| G_1(k)$ is a secure PRG.

- Setup on input 1^λ, $\alpha \in \{0,1\}^m$, $\beta \in \{0,1\}^\ell$ proceeds as follows:
  - Draw $k \xleftarrow{\$} \{0,1\}^\lambda$ at random.
  - For all $j \in [m]$ define
    - $\overline{k}_j := G_{\overline{\alpha}_j}(G_{\alpha_{j-1}}(\dots, G_{\alpha_1}(k)\dots))$,
    - $\gamma_j := C(G_{\alpha_{j-1}}(\dots, G_{\alpha_1}(k)\dots))$,
    - $(c_j, \overline{c}_j) := \begin{cases} (\gamma_j, 0) & \text{if } \alpha_j = 0 \\ (0, \gamma_j) & \text{else} \end{cases}$,
    - $\overline{S}_j := \overline{c}_j + \sum_{i=1}^{j-1} c_i$ and
    - $\overline{B}_j := \overline{S}_j + \alpha_j \cdot \beta$.
  - Define $\tilde{y} := G_{\alpha_m}(G_{\alpha_{m-1}}(\dots, G_{\alpha_1}(k)\dots))$ and $y := \mathsf{Convert}_{\mathbb{G}}(\tilde{y}) + \sum_{i=1}^m c_i$
  - Set $K_0 := (\alpha, \{\overline{k}_j\}_{j\in[m]}, \{\overline{B}_j\}_{j\in[m]}, y\}$ and $K_1 := k$.
  - Output (K_0, K_1).
- Eval on input index $\sigma \in \{0,1\}$, key k_σ and $x \in \{0,1\}^m$ proceeds as follows.
  - If $\sigma = 0$:
    - Parse K_0 as $K_0 =: (\alpha, \{\overline{k}_j\}_{j\in[m]}, \{\overline{B}_j\}_{j\in[m]}, y\}$.
    - If $x = \alpha$, output $y^0 := y$.
    - Else, let $j \in [m]$ be the minimal index such that $x_j \neq \alpha_j$ and proceed as follows:
      1. For all $i \in \{j+1, \dots, m\}$ set
      $$c_i^0 := \begin{cases} C(G_{x_{i-1}}(\dots, G_{x_{j+1}}(\overline{k}_j)\dots)) & \text{if } x_i = 0 \\ 0 & \text{else} \end{cases}.$$
      2. Compute $\tilde{y}^0 := G_{x_m}(\dots, G_{x_{j+1}}(\overline{k}_j)\dots))$.
      3. Output $y^0 := \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^0) + \overline{B}_j + \sum_{i=j+1}^m c_i^0$.
  - If $\sigma = 1$:
    - Set $k := K_1$.
    - For all $i \in [m]$ set $c_i^1 := \begin{cases} C(G_{x_{i-1}}(\dots, G_{x_1}(k)\dots)) & \text{if } x_i = 0 \\ 0 & \text{else} \end{cases}$.
    - Compute $\tilde{y}^1 := G_{x_m}(\dots, G_{x_1}(k)\dots))$.
    - Output $y^1 := \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^1) + \sum_{i=1}^m c_i^1$.

Figure 11: Relaxed distributed comparison function $\mathsf{RDCF} = (\mathsf{Setup}, \mathsf{Eval})$ from a PRG $G : \{0,1\}^\lambda \to \mathbb{G} \times \{0,1\}^{2\lambda}$.

Proof. *Correctness.* Let $x \in \{0,1\}^m$ and $\lambda \in \mathbb{N}$. We have to show that for all (k_0, k_1) in the image of $\mathsf{Setup}(1^\lambda)$ it holds that

$$\mathsf{Eval}(0, k_0, x) - \mathsf{Eval}(1, k_1, x) = f^{\beta}_{<\alpha}(x).$$

We distinguish the following cases:

- *Case $x < \alpha$:* Let $j \in [m]$ be the smallest index such that $x_j \neq \alpha_j$. Since $x < \alpha$, it must hold that $x_j = 0$ and $\alpha_j = 1$. Therefore, for $\overline{B}_j$ as defined in the Setup procedure on input α, we have

  $$\overline{B}_j = \overline{S}_j + \alpha_j \cdot \beta = \overline{S}_j + \beta = \overline{c}_j + \sum_{i=1}^{j-1} c_i + \beta.$$

  Further, it holds $\overline{c}_j = c^1_j$ and $c_i = c^1_i$ for all $i \in [j-1]$ (where $\overline{c}_j, \{c_i\}$ are as defined in the Setup procedure on input α and $c^1_j, \{c^1_i\}$ are as defined in the Eval procedure on input $1, k_1, x$).

  Next, observe that

  $$\overline{k}_j = G_{\overline{\alpha}_j}(G_{\alpha_{j-1}}(\dots, G_{\alpha_1}(k)\dots) = G_{x_j}(G_{x_{j-1}}(\dots, G_{x_1}(k)\dots).$$

  This implies $c^0_i = c^1_i$ for all $j+1 \leq i \leq m$, as well as $\tilde{y}^0 = \tilde{y}^1$.

  Altogether, this yields

  $$\begin{aligned} y^0 &= \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^0) + \overline{B}_j + \sum_{i=j+1}^{m} c^0_i = \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^0) + \overline{c}_j + \sum_{i=1}^{j-1} c_i + \beta + \sum_{i=j+1}^{m} c^0_i \\ &= \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^1) + c^1_j + \sum_{i=1}^{j-1} c^1_i + \beta + \sum_{i=j+1}^{m} c^1_i = y^1 + \beta, \end{aligned}$$

  as required.

- *Case $x = \alpha$:* In this case, we have $c^1_i = c_i$ for all $i \in [m]$, as well as $\tilde{y} = \tilde{y}^1$. This implies

  $$y^0 = y = \mathsf{Convert}_{\mathbb{G}}(\tilde{y}) + \sum_{i=1}^{m} c_i = \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^1) + \sum_{i=1}^{m} c^1_i = y^1.$$

- *Case $x > \alpha$:* This case is similar to the case $x < \alpha$, except that we have $x_j = 1$ and $\alpha_j = 0$ and thus

  $$\overline{B}_j = \overline{S}_j + \alpha_j \cdot \beta = \overline{S}_j = \overline{c}_j + \sum_{i=1}^{j-1} c_i.$$

  With the considerations above it follows

  $$\begin{aligned} y^0 &= \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^0) + \overline{B}_j + \sum_{i=j+1}^{m} c^0_i = \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^0) + \overline{c}_j + \sum_{i=1}^{j-1} c_i + \sum_{i=j+1}^{m} c^0_i \\ &= \mathsf{Convert}_{\mathbb{G}}(\tilde{y}^1) + c^1_j + \sum_{i=1}^{j-1} c^1_i + \sum_{i=j+1}^{m} c^1_i = y^1. \end{aligned}$$

Security. Since $k \xleftarrow{\$} \{0,1\}^\lambda$ is chosen uniformly at random, independent from α and β, the second part of the security requirement (i.e., security against the holder of the key $K_1 = k$) follows immediately. It is left to show that an adversary holding K_0 cannot distinguish between a key obtained via $\mathsf{Setup}(1^\lambda, \alpha, \beta)$ and a key obtained via $\mathsf{Setup}(1^\lambda, \alpha, \beta')$.

We show this via a sequence of hybrid games, where in the ι-th hybrid, we replace the ι-th level of the tree by random. More precisely, in hybrid H_ι we setup the key K_0 as follows:

- Draw $k \xleftarrow{\$} \{0,1\}^\lambda$ at random and define $k_0 := k$.
- For all $j \in [m]$ define

- If $j \leq \iota$:
  - $(\gamma_j, \kappa_{j,0}, \kappa_{j,1}) \xleftarrow{\$} \{0,1\}^{\ell+2\lambda}$.
- If $j > \iota$:
  - $(\gamma_j, \kappa_{j,0}, \kappa_{j,1}) := G(k_{j-1})$.
- $(k_j, \overline{k}_j) := (\kappa_{j,\alpha_j}, \kappa_{j,\overline{\alpha}_j})$,
- $(c_j, \overline{c}_j) := \begin{cases} (\gamma_j, 0) & \text{if } \alpha_j = 0 \\ (0, \gamma_j) & \text{else} \end{cases}$,
- $\overline{S}_j := \overline{c}_j + \sum_{i=1}^{j-1} c_i$ and
- $\overline{B}_j := \overline{S}_j + \alpha_j \cdot \beta$.

- Define $\tilde{y} := k_m$ and $y := \mathsf{Convert}_{\mathbb{G}}(\tilde{y}) + \sum_{i=1}^{m} c_i$.
- Set $K_0 := (\alpha, \{\overline{k}_j\}_{j\in[m]}, \{\overline{B}_j\}_{j\in[m]}, y\}$ and $K_1 := k$.
- Output (K_0, K_1).

We have the following:

- *Hybrid H_0:* Note that the hybrid H_0 corresponds to the standard setup algorithm, since in this case it holds $k_0 = k$ as well as $k_j = \kappa_{j,\alpha_j} = G_{\alpha_j}(k_{j-1})$ and thus

$$k_j = G_{\alpha_j}(G_{\alpha_{j-1}}(\ldots, G_{\alpha_1}(k)\ldots)).$$

  Together with $\overline{k}_j = \kappa_{j,\overline{\alpha}_j} = G_{\overline{\alpha}_j}(k_{j-1})$ this implies

$$\overline{k}_j = G_{\overline{\alpha}_j}(G_{\alpha_{j-1}}(\ldots, G_{\alpha_1}(k)\ldots))$$

  and

$$\gamma_j = C(G_{\alpha_{j-1}}(\ldots, G_{\alpha_1}(k)\ldots))$$

  as required.

- *Hybrid $H_{\iota-1} \rightsquigarrow H_\iota$:* We have to show that if G is a pseudorandom generator, then the hybrids $H_{\iota-1}$ and H_ι are computationally indistinguishable for all $\iota \in [m]$. This is straightforward, since an adversary attacking the pseudorandomness of G can embed its input as $(\gamma_{\iota,0}, \kappa_{\iota,0}, \kappa_{\iota,1})$ (and otherwise follow the setup algorithm of game $H_{\iota-1}$). If it obtained a pseudorandom input, then it perfectly simulates $H_{\iota-1}$. If it obtained a truly random input, then it perfectly simulates H_ι. A distinguisher between the two games could thus be used to break pseudorandomness of the underlying PRG G, yielding a contradiction as required.

- *Hybrid H_m.* In hybrid H_m the parts $\{\overline{k}_j\}_{j\in[m]}, \{\overline{B}_j\}_{j\in[m]}, y$ of K_0 are distributed independently uniformly at random and are therefore in particular independent of β. This concludes the proof.

Pseudorandomness. Note that since correctness holds, it is sufficient to show $\mathsf{Eval}(\sigma, k_\sigma, \cdot)$ for either $\sigma = 0$ *or* $\sigma = 1$, since the pseudorandomness of the other follows immediately. Since the proof for $\sigma = 1$ follows the proof of [GGM86] very closely, we omit it here. □

Distributed setup protocol. Following the strategy of [Ds17, BCG+19a] for setting up the GGM-based PPRF in 2 rounds in the semi-honest setting, we describe a setup protocol for setting up our RCDF in 2 rounds, given a 2-round OT protocol, if the domain size 2^m is polynomially bounded (i.e., a full evaluation is feasible to compute). For simplicity, we assume that party P_1 knows the offset $\beta \in \mathbb{G}$ (as is the case in our PCF constructions). Since the protocol follows the protocol of [BCG+19a] very closely, we only give a sketch here. Recall that the underlying idea is for the party P_1 to prepare the sum of all left leaves and the sum of all right leaves for each level, and P_0 will learn one of these sums via OT. Since for each level, P_0 can already compute all but one of either the left or right leaves from information about previous levels, the OT allows recovering the missing node corresponding to the key $\overline{k}_j$.

- $P_1(\beta)$ proceeds as follows:
  - Draw $k \xleftarrow{\$} \{0,1\}^\lambda$.

- For all $i \in [m]$, $x \in \{0,1\}^i$ compute:
  - $k_x := G_{x_i}(G_{x_{i-1}}(\cdots, G_{x_1}(k)\cdots))$.
  - $\gamma_x := C(G_{x_{i-1}}(G_{x_{i-2}}(\cdots, G_{x_1}(k)\cdots)))$.
- For all $i \in [m]$ compute:
  - $S_{i,0} := \sum_{x\in\{0,1\}^{i-1}} k_{x\|0}$ and $S_{i,1} := \sum_{x\in\{0,1\}^{i-1}} k_{x\|1}$.
  - $\Gamma_i := \sum_{x\in\{0,1\}^{i-1}} \gamma_{x\|0}$.

- $P_0(\alpha)$ and $P_1(\beta)$ run m OT's in parallel, where P_0 acts as receiver and P_1 acts as sender, and the two parties proceed as follows in the j-th OT instance:
  - P_0 inputs $\overline{\alpha}_j$ as choice bit.
  - If $j < m$, P_1 inputs messages $M_0^j := (S_{j,0}, \Gamma_j + \beta)$, $M_1^j := (S_{j,1}, \Gamma_j)$.
  - If $j = m$, P_1 inputs messages $M_0^j := \Gamma_m + \beta$, $M_1^j := \Gamma_m$ and sends both $S_{m,0}$ and $S_{m,1}$ to P_0.

- $P_0(\alpha)$ recovers K_1 as follows: Parse $M^j_{\overline{\alpha}_j} = (S_j^*, \Gamma_j^*)$ for $j \in [m-1]$, as well as $M^m_{\overline{\alpha}_m} = \Gamma_m^*$.
  - For $j = 1$:
    - Set $\overline{k}_1 := S_1^\star$.
    - Set $(c'_1, \overline{c}'_1) := \begin{cases} (\Gamma_1^*, 0) & \text{if } \alpha_1 = 0 \\ (0, \Gamma_1^*) & \text{else} \end{cases}$
    - For all $i \in [m]$, $x \in \{0,1\}^i$ with $x_1 = \overline{\alpha}_1$, compute:
      - $k_x := G_{x_i}(G_{x_{i-1}}(\cdots, G_{x_2}(\overline{k}_1)\cdots))$.
      - $\gamma_x^* := C(G_{x_{i-1}}(\cdots, G_{x_2}(\overline{k}_1)\cdots))$.
    - Define $\overline{B}_1 := \overline{c}'_1$.
  - For $1 < j < m$:
    - Set $\overline{k}_j := S_j^\star - \sum_{x\in\{0,1\}^{j-1}, x\neq\alpha_1\|\ldots\|\alpha_{j-1}} k_{x\|\overline{\alpha}_j}$.
    - Set $\gamma_j^* := \Gamma_j^* - \sum_{x\in\{0,1\}^{j-1}, x\neq\alpha_1\|\ldots\|\alpha_{j-1}} \gamma_{x\|0}$.
    - Set $(c'_j, \overline{c}'_j) := \begin{cases} (\gamma_j^*, 0) & \text{if } \alpha_1 = 0 \\ (0, \gamma_j^*) & \text{else} \end{cases}$.
    - For all $i \in [m]$, $x \in \{0,1\}^i$ with $x_1\| \ldots \|x_{j-1}\|x_j = \alpha_1\| \ldots \|\alpha_{j-1}\|\overline{\alpha}_j$, compute:
      - $k_x := G_{x_i}(G_{x_{i-1}}(\cdots, G_{x_{j+1}}(\overline{k}_j)\cdots))$.
      - $\gamma_x := C(G_{x_{i-1}}(\cdots, G_{x_{j+1}}(\overline{k}_j)\cdots))$.
    - Define $\overline{B}_j := \overline{c}'_j + \sum_{i=1}^{j-1} c'_i$.
  - For $j = m$:
    - Set $k_m := S_{m,\alpha_m} - \sum_{x\in\{0,1\}^{m-1}, x\neq\alpha_1\|\ldots\|\alpha_{m-1}} k_{x\|\alpha_m}$.
    - Set $\overline{k}_m := S_{m,\overline{\alpha}_m} - \sum_{x\in\{0,1\}^{m-1}, x\neq\alpha_1\|\ldots\|\alpha_{m-1}} k_{x\|\overline{\alpha}_m}$.
    - Set $\gamma_m^* := \Gamma_m^* - \sum_{x\in\{0,1\}^{m-1}, x\neq\alpha_1\|\ldots\|\alpha_{m-1}} \gamma_{x\|0}$.
    - Set $(c'_m, \overline{c}'_m) := \begin{cases} (\gamma_m^*, 0) & \text{if } \alpha_1 = 0 \\ (0, \gamma_m^*) & \text{else} \end{cases}$.
    - Define $\overline{B}_m := \overline{c}'_m + \sum_{i=1}^{m-1} c'_i$.
    - Define $\tilde{y} := k_m$ and $y := \mathsf{Convert}_{\mathbb{G}}(\tilde{y}) + \sum_{i=1}^{m} c'_i$.
  - Output $K_0 := (\alpha, \{\overline{k}_j\}_{j\in m}, \{\overline{B}_j\}_{j\in[m]}, y\}$.

In the following we give a brief sketch of correctness. Note that security against semi-honest adversaries is quite straightforward, since P_0 does not receive more information than it can derive from K_0. Since the proof of security further closely follows [Ds17, BCG+19a], we omit it here.

Correctness. First, note that for $j = 1$ we have

$$\overline{k}_1 = S_1^\star = S_{j,\overline{\alpha}_1} = k_1 = G_{\overline{\alpha}_1}(k)$$

and

$$(c'_1, \overline{c}'_1) = \begin{cases} (\Gamma^*_1, 0) & \text{if } \alpha_1 = 0 \\ (0, \Gamma^*_1) & \text{else} \end{cases} = \begin{cases} (C(k), 0) & \text{if } \alpha_1 = 0 \\ (0, C(k) + \beta) & \text{else} \end{cases}.$$

This implies

$$\overline{B}_1 = \begin{cases} 0 & \text{if } \alpha_1 = 0 \\ C(k) + \beta & \text{else} \end{cases}$$

as required. Further, note that k_x and γ_x for $x \in \{0,1\}^i$ with $x_1 = \overline{\alpha}_1$ are computed as by P_1 and $c'_1 = \begin{cases} C(k) & \text{if } \alpha_1 = 0 \\ 0 & \text{else} \end{cases}$.

For $1 < j < m$, we proceed inductively. Assume that k_x and γ_x are computed as by P_1 for all $i \in [m]$, $x \in \{0,1\}^i$ with $x_1 \| \dots \| x_{j-1} \neq \alpha_1 \| \dots \| \alpha_{j-1}$ and for all $i \in [j-1]$

$$c'_i = \begin{cases} C(G_{\alpha_{i-1}}(\cdots, G_{\alpha_1}(k)\dots)) & \text{if } \alpha_1 = 0 \\ 0 & \text{else} \end{cases}.$$

Then, we have

$$\begin{aligned} \overline{k}_j &= S^\star_j - \sum_{x \in \{0,1\}^{j-1}, x \neq \alpha_1 \| \dots \| \alpha_{j-1}} k_{x \| \overline{\alpha}_j} \\ &= \sum_{x \in \{0,1\}^{j-1}} k_{x \| \overline{\alpha}_j} - \sum_{x \in \{0,1\}^{j-1}, x \neq \alpha_1 \| \dots \| \alpha_{j-1}} k_{x \| \overline{\alpha}_j} = k_{\alpha_1 \| \dots \| \alpha_{j-1} \| \overline{\alpha}_j}. \end{aligned}$$

Further, we have

$$\begin{aligned} \gamma^*_j &= \Gamma^*_j - \sum_{x \in \{0,1\}^{j-1}, x \neq \alpha_1 \| \dots \| \alpha_{j-1}} \gamma_{x \| 0} \\ &= \sum_{x \in \{0,1\}^{j-1}} \gamma_{x \| 0} + \alpha_j \cdot \beta - \sum_{x \in \{0,1\}^{j-1}, x \neq \alpha_1 \| \dots \| \alpha_{j-1}} \gamma_{x \| 0} \\ &= \gamma_{\alpha_1 \| \dots \| \alpha_{j-1} \| 0} + \alpha_j \cdot \beta. \end{aligned}$$

This implies

$$\overline{c}'_j = \begin{cases} 0 & \text{if } \alpha_1 = \beta \\ C(G_{\alpha_{j-1}}(\cdots, G_{\alpha_1}(k)\dots)) + \beta & \text{else} \end{cases}$$

and

$$c'_j = \begin{cases} C(G_{\alpha_{j-1}}(\cdots, G_{\alpha_1}(k)\dots)) & \text{if } \alpha_1 = \beta \\ 0 & \text{else} \end{cases}.$$

We thus obtain $\overline{B}_j$ is of the required form. Further, note that the above considerations imply that k_x and γ_x computed in the j-th iteration are computed as by P_1 for all $i \in [m]$, $x \in \{0,1\}^i$ with $x_1 \| \dots \| x_j \neq \alpha_1 \| \dots \| \alpha_j$.

Finally, we have

$$\begin{aligned} k_m &= S_{m,\alpha_m} - \sum_{x \in \{0,1\}^{m-1}, x \neq \alpha_1 \| \dots \| \alpha_{m-1}} k_{x \| \alpha_m} \\ &= \sum_{x \in \{0,1\}^{m-1}} k_{x \| \alpha_m} - \sum_{x \in \{0,1\}^{m-1}, x \neq \alpha_1 \| \dots \| \alpha_{m-1}} k_{x \| \alpha_m} = k_{\alpha_1 \| \dots \| \alpha_{m-1} \| \alpha_m}. \end{aligned}$$

This shows that $\tilde{y}$ is computed as required, and since $\overline{k}_m, \gamma^*_m, (c'_m, \overline{c}'_m, \overline{B}_m$ are computed as in case $1 < j < m$, correctness follows.

5.2 PCFs for VOLE and OT from EA Codes and RDCF

We now show how to combine EA-LPN and RDCF to get a PCF for the subfield VOLE correlation. This also implies a PCF for OT, by applying a correlation-robust hash function [BCG+20a]. Recall that in subfield VOLE over a base field $\mathbb{F}_p$ and extension $\mathbb{F}_q$ with $p|q$, each output of the correlation gives (r, y_0)

to one party and (Δ, y_1) to another party, satisfying $y_0 = y_1 + r \cdot \Delta$. Here, r is uniform in $\mathbb{F}_p$, while Δ and y_0 are uniform over $\mathbb{F}_q$, but Δ is fixed for every output.

To simplify the presentation we will slightly modify the noise distribution used in EA-LPN (without changing the assumption). Recall that the accumulated noise $\vec{e}' = A \cdot \vec{e}$ has an alternating structure

$$\vec{e}' = (\vec{e}')^{(1)} \| (\vec{e}')^2 \| \cdots = (0, \ldots, 0, 1, \ldots, 1) \| (1, \ldots, 1, 0 \ldots, 0), \| \ldots$$

where depending on the parity of i, each $(\vec{e}')^{(i)}$ starts with a one or zero. We tweak this by reversing the order of the odd-indexed sub-vectors, so each $(\vec{e}')^{(i)}$ always begins with a 1 (or, with a random non-zero noise value, for the general assumption over rings). When using EA-LPN with this distribution, this is equivalent to using the original error distribution and permuting the columns of the matrix H. Since the columns are independent, the assumption with the modified error distribution is equivalent.

With this modification, we present a PCF for the subfield VOLE correlation in Fig. 12. Given the relaxed DCF, the construction is very similar to prior LPN-based PCGs and PCFs.

$\mathsf{PCF.Gen}(1^\lambda)$:

1. Sample $\Delta \xleftarrow{\$} \mathbb{F}_q$
2. For $i \in [t]$, sample $\alpha_i \xleftarrow{\$} \{0, \ldots, N/t - 1\}, \beta_i \xleftarrow{\$} \mathbb{F}_p \setminus \{0\}$
3. For $i \in [t]$, sample $(K^{\mathsf{dcf}}_{i,0}, K^{\mathsf{dcf}}_{i,1}) \leftarrow \mathsf{RDCF.Setup}(1^\lambda, \alpha_i, \beta_i \cdot \Delta)$
4. Output $\mathsf{k}_0 = \left(K^{\mathsf{dcf}}_{i,0}, \alpha_i, \beta_i\right)_{i \in [t]}$ and $\mathsf{k}_1 = \left(\Delta, (K^{\mathsf{dcf}}_{i,1})_{i \in [t]}\right)$

$\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$:

1. Let x define a row B_x of B, with ℓ non-zero entries $x_1, \ldots, x_\ell \in \mathbb{F}_p$ at indices $i_1, \ldots, i_\ell \in \{0, \ldots, N-1\}$
2. For each $j \in [\ell]$, write $i_j = \gamma_j \cdot (N/t) + \delta_j$, for $\delta_j \in \{0, \ldots, N/t - 1\}$ and $\gamma_j \in \{0, \ldots, t-1\}$.
3. Compute over $\mathbb{F}_q$:
$$y_\sigma = \sum_{j=1}^{\ell} x_j \cdot \mathsf{RDCF.Eval}(K^{\mathsf{dcf}}_{\gamma_j, \sigma}, \delta_j))$$
4. If $\sigma = 0$, compute $r = \sum_j x_j \cdot \beta_{\gamma_j} \cdot ((\delta_j \overset{?}{<} \alpha_{\gamma_j}))$ and output $(r, y_0) \in \mathbb{F}_p \times \mathbb{F}_q$
5. If $\sigma = 1$, output $(\Delta, y_1) \in \mathbb{F}_q^2$

Figure 12: PCF for subfield VOLE from EA codes and RDCF

Lemma 5.3 *If* RDCF *is a secure relaxed distributed comparison function with pseudorandom outputs, and the* EA-LPN *assumption with regular, weight-t noise is secure, then the construction in Fig. 12 is a PCF for subfield VOLE.*

Proof. (sketch) We first argue that the outputs of the PCF are pseudorandomly correlated according to the VOLE correlation. Let $\vec{e} = (\vec{e})^{(1)}, \ldots, (\vec{e})^{(t)}$ be the accumulated noise vector defined by α_i, β_i, so that $(\vec{e})^{(i)}$ begins with repeated β_i's, until entry α_i when it becomes zero.

Note that for each i_j defined by the input x, $\gamma_j + 1 \in [t]$ gives the index of the length-N/t sub-vector in B_x which contains the non-zero value x_j, while δ_j gives the index *within* this sub-vector where x_j lies. It follows that the output r satisfies $r = \langle B_x, \vec{e} \rangle$. This implies that r is pseudorandom under the EA-LPN assumption.

Also, by the correctness of RDCF, we have

$$\mathsf{RDCF.Eval}(K^{\mathsf{dcf}}_{\gamma_j+1,0}, \delta_j) - \mathsf{RDCF.Eval}(K^{\mathsf{dcf}}_{\gamma_j+1,1}, \delta_j) = (\delta_j \overset{?}{<} \alpha_{\gamma_j+1}) \cdot \beta_{\gamma_j+1} \cdot \Delta$$

Then, we have $y_0 - y_1 = \sum_j x_j \cdot (\delta_j \overset{?}{<} \alpha_{\gamma_j+1}) \cdot \beta_{\gamma_j+1} \cdot \Delta = r \cdot \Delta$, as required. Finally, from the pseudorandom outputs property of RDCF, it holds that each individual share y_σ is also pseudorandom.

To argue security, we need to show that given one of the keys k_σ, the other party's outputs are indistinguishable from an ideal VOLE output that is sampled conditioned on the VOLE outputs from k_σ.

We start by considering the case when party 0 is corrupted. In this case, the reverse-sampled outputs of party 1 are the pairs (Δ, y_1), where $\Delta \leftarrow \mathbb{F}_q$ and y_1 is computed as $y_0 - r \cdot \Delta$ (using y_0, r from k_0). The only difference between this experiment and the real outputs is that the real output of party 1 uses a random Δ which is also used to generate the RDCF keys in k_0. These two cases are indistinguishable due to security property of RDCF, which guarantees that k_0 hides Δ.

When party 1 is corrupted, the reverse-sampled outputs of party 0 are defined by sampling are random r, and computing y_0 using r and the outputs from key k_1. To argue security, we start with the experiment where the distinguisher gets the key k_1 together with the real outputs of party 0, derived from k_0. We then consider a hybrid where instead of computing y_0 from k_0, we use $y_0 = y_1 + r \cdot \Delta$; from the correctness of RDCF, this is indistinguishable from the first experiment. Next, we switch to a hybrid where the RDCF keys in k_1 are replaced with ones generated with independent α_i values; this is indistinguishable due to the RDCF security property. Finally, we replace the r values, which are computed from the EA-LPN distribution, with uniformly random ones. This is identical to the reverse-sampled distribution, and indistinguishable from the previous hybrid under EA-LPN. □

5.3 PCF for Degree-2 Correlations

In our PCF for VOLE, each random output r is obtained by taking a random row of B defined by x, and computing $r = \sum_j x_j \beta_{\gamma_j} \cdot ((\delta_j \overset{?}{<} \alpha_{\gamma_j}))$. The RDCF was used to compute shares of $((\delta_j \overset{?}{<} \alpha_{\gamma_j}))$ multiplied by Δ. To generate multiplication triples in a ring $\mathcal{R}$, we want to sample two such random values $a, b \in \mathcal{R}$, and output shares of these together with shares of $c = ab$. Computing shares of a, b can be done by defining them similarly to r, and using (non-relaxed) DCFs. Computing shares of c then boils down to being able to compute shares of values of the form

$$(x \overset{?}{<} \alpha) \cdot (y \overset{?}{<} \alpha') \cdot \beta$$

where x and y are public while α, α', β are secret. This can be seen as the product of two comparison functions, or a *two-dimensional interval function*.

On its own, a distributed comparison function is not sufficient to obtain shares of a two-dimensional interval. However, function secret-sharing for two-dimensional intervals has been constructed using the FSS for decision tree construction from [BGI16], based on any PRG. Formally, we define the function class of two-dimensional $N \times N$ intervals over a ring $\mathcal{R}$, where each function in the class is specified by a pair of indices $\alpha_1, \alpha_2 \in [N]$ and $\beta \in \mathcal{R}$, and defined by

$$f_{\alpha_1,\alpha_2,\beta} : [N] \times [N] \to \mathcal{R}$$
$$(x_1, x_2) \mapsto \begin{cases} \beta & \text{if } x_1 < \alpha_1 \text{ and } x_2 < \alpha_2 \\ 0 & \text{otherwise} \end{cases}$$

The FSS construction from [BGI16] has a key size bounded by $2\,|V|\,(\lambda + 1)$ bits, where V is the set of nodes in the binary decision tree; a $2^m \times 2^m$ interval function can be expressed as a decision tree with $2m(m+1)+1$ nodes, giving a key size of roughly $4m^2\lambda$ bits for the FSS scheme. In our construction, we use $m = \log_2(N/t)$.

PCF for Multiplication Triples. Given these tools, we present a PCF for the multiplication triple correlation in Figure 13. Correctness of the construction follows from the explanation above, and its security follows from the pseudorandomness of both DCF and FSS outputs, together with the EA-LPN assumption. Note that the noise values $\vec{\beta}^{(0)}, \vec{\beta}^{(1)}$ are sampled from R^*, the set of invertible elements, to prevent the subring projection attacked mentioned in Section 2.2. The proof of the following is similar to the previous lemma, except instead of RDCF, we rely on the security of standard DCF and FSS.

Lemma 5.4 *If* DCF *is a secure distributed comparison function with pseudorandom outputs and* FSS *is a secure function secret-sharing scheme with pseudorandom outputs, then the construction in Fig. 13 is a PCF for multiplication triples over $\mathcal{R}$ under the* EA-LPN *assumption with regular, weight-t noise.*

$\mathsf{PCF.Gen}(1^\lambda)$:

1. Sample $\vec{\alpha}^{(0)}, \vec{\alpha}^{(1)} \xleftarrow{\$} \{0, \ldots, N/t-1\}^t$ and $\vec{\beta}^{(0)}, \vec{\beta}^{(1)} \xleftarrow{\$} (\mathcal{R}^*)^t$
2. Sample the following DCF and FSS keys:
   - $(K^a_{i,j,0}, K^a_{i,j,1}) \leftarrow \mathsf{DCF.Gen}(1^\lambda, (\alpha^{(0)}_i, \beta^{(0)}_i))$, for $i \in [t]$
   - $(K^b_{i,j,0}, K^b_{i,j,1}) \leftarrow \mathsf{DCF.Gen}(1^\lambda, (\alpha^{(1)}_i, \beta^{(1)}_j))$, for $i \in [t]$
   - $(K^c_{i,j,0}, K^c_{i,j,1}) \leftarrow \mathsf{FSS.Gen}(1^\lambda, f_{\alpha^{(0)}_i, \alpha^{(1)}_j, \beta^{(0)}_i \cdot \beta^{(1)}_j})$, for $i, j \in [t]$
3. Output $\mathsf{k}_\sigma = \big((K^a_{i,\sigma}, K^b_{i,\sigma})_{i\in[t]}, (K^c_{i,j,\sigma})_{i,j\in[t]}\big)$, for $\sigma = 0, 1$

$\mathsf{PCF.Eval}(\mathsf{k}_\sigma, (x, y))$:

- Let x, y define two rows B_x, B_y of B, with ℓ non-zero entries $x_1, \ldots, x_\ell \in \mathcal{R}$ at indices $i_1, \ldots, i_\ell$ and $y_1, \ldots, y_\ell$ at $\iota_1, \ldots, \iota_\ell$ (for x, y resp.)
- For each $j \in [\ell]$, write $i_j = \gamma_j \cdot (N/t) + \delta_j$, for $\delta_j \in \{0, \ldots, n/t-1\}$ and $\gamma_j \in \{0, \ldots, t-1\}$. Similarly, write $\iota_j = \widetilde{\gamma}_j \cdot (N/t) + \widetilde{\delta}_j$.
- Compute

$$a_\sigma = (-1)^\sigma \sum_{j=1}^{\ell} x_j \cdot \mathsf{DCF.Eval}(K^a_{\gamma_j,\sigma}, \delta_j)),$$

$$b_\sigma = (-1)^\sigma \sum_{j=1}^{\ell} y_j \cdot \mathsf{DCF.Eval}(K^b_{\widetilde{\gamma}_j,\sigma}, \widetilde{\delta}_j)),$$

$$c_\sigma = (-1)^\sigma \cdot \sum_{j=1}^{\ell}\sum_{k=1}^{\ell} x_j \cdot y_k \cdot \mathsf{FSS.Eval}(K^c_{\gamma_j,\widetilde{\gamma}_k,\sigma}, (\delta_j, \widetilde{\delta}_j))$$

- Output $(a_\sigma, b_\sigma, c_\sigma)$

Figure 13: PCF for multiplication triples over $\mathcal{R}$ from EA codes, DCF and FSS for 2-dimensional intervals

Extensions: Degree-2 Correlations, Authenticated Correlations and Multi-Party. The PCF in Fig. 13 can be easily extended to obtain a PCF for any degree-two correlation. By this, we mean that on input a random nonce to Eval, the two parties obtain shares of a random vector $\vec{y}$ over $\mathcal{R}$, as well as shares of $z = P(\vec{y})$, where P is any degree-two polynomial (with polynomially many terms). In fact, the polynomial P can be chosen adaptively, such that if desired, different y_i terms can be reused in later queries.

To do this, the key generation procedure is unchanged from Fig. 13. In Eval, instead of using two public inputs (x, y), we now input a vector $(x_1, \ldots, x_m)$ which will be used to define the pseudorandom vector $\vec{y}$ using m rows of B and the $\mathsf{EA\text{-}LPN}$ assumption. Each degree two term in $P(\vec{y})$ can now be evaluated by taking a sum of ℓ^2 FSS evaluations, the same way as the shares of ab are computed in Fig. 13. Then, the final share of $P(\vec{y})$ is obtained as a linear combination of these.

The cost of each evaluation is essentially m times that of a single Eval for a multiplication triple, where m is the number of degree-2 monomials in $P(\vec{y})$.

Authenticated Correlations. The above PCFs can also easily be extended to produce *authenticated* correlations, by applying the technique used for multiplication triples in [BCG+20b]. Here, in addition to shares of the polynomial $P(\vec{y})$, the parties obtain shares of MACs $\Delta \cdot P(\vec{y})$, where $\Delta \in R$ is a random, secret-shared MAC key (fixed for each correlation). This can be done with roughly a factor 2 overhead in computation by extending the output of the FSS scheme from 1 to 2 ring elements, where the 2nd output is multiplied by the fixed value Δ. More generally, if R is small e.g. $\mathbb{F}_2$, we can pick $\Delta \in R^\lambda$ and obtain shares of each component of Δ multiplied by $P(\vec{y})$; this allows us to increase the size of the MAC key, and obtain PCFs for authenticated boolean triples, as needed for the TinyOT protocol [NNOB12, WRK17a].

Multi-Party PCFs. When considering *unauthenticated* degree-two correlations, our PCF can also be extended to the multi-party setting. This follows from a natural programmability property of the construction, meaning that with multiplication triples, for instance, the a values can be programmed to be the same across two PCF instances. This allows a multi-party degree-two correlation to be produced by splitting it up into a sum of $n(n-1)$ two-party correlations, following the approach used for PCGs in [BCG+19b].

5.4 Efficiency Analysis

VOLE and OT. The PCF for subfield VOLE has a key size dominated by t keys for an RDCF on an m-bit domain, where $m = \log_2(N/t)$, with outputs in R. With our optimized RDCF, we get a total key size of around $tm(\lambda + \log|R|)$ bits. The computational cost in a call to Eval mainly consists of a linear combination of ℓ $\mathsf{RDCF.Eval}$ outputs, which has a cost of $\ell \cdot m$ PRG evaluations.

For instance, plugging in the conservative choice (cf. Fig. 9a) of $t = 664$, $N = 5n$ and up to $n = 2^{30}$ samples over a 64-bit ring $\mathcal{R}$, we get a key size of around 370kB. This is around twice the size of the estimated key size for the more aggressive PCF candidate from [BCG+20a], however, their aggressive assumption did not have a security analysis against linear attacks. Regarding computational complexity, with row weight $\ell = 3 \ln n$ for the matrix B, we require around 1300 PRG evaluations per PCF output, which is more than 10x cheaper than the PCF of [BCG+20a]. Increasing the number of samples to 2^{48} (in line with the parameters from [BCG+20a], the number of PRG evaluations is 3900, while the key size becomes 650kB. So we see that even using our conservative parameter estimates, we see significant performance benefits from using $\mathsf{EA\text{-}LPN}$ compared with previous PCF candidates.

If we use instead aggressive parameters for our construction, we get the following costs (based on parameters from Section 3.6):

- (seed-size optimized, $t = 68$, $\ell = 62$) seed size: 65kB; evaluation time: 3700 PRG calls.
- (evaluation-time optimized, $t = 1000$, $\ell = 7$) seed size: 960kB; evaluation time: 660 PRG calls.

To give a rough runtime estimation, the PRG can be instantiated using two calls to fixed-key AES. Using the AES-NI instructions of modern CPUs, one byte of AES-128 can be computed in ~ 1.3 cycles. Concretely, using a 3GHz processor and our conservative instantiation with $n = 2^{30}$ (1300 PRG calls per evaluation), we estimate around $1.4 \cdot 10^5$ evaluations per second. The number doubles with our most aggressive, evaluation-time optimized variant. Since the calls are perfectly independent and therefore parallelizable, these numbers can be scaled up linearly with the number of processors. This shows that our PCF already provide high real world performances.

Multiplication triples and degree-2 correlations. Our degree-2 PCF needs t^2 FSS keys for a 2-dimensional interval function on a $2m$-bit domain, which gives a main storage cost of around $4(mt)^2\lambda$ bits. Meanwhile, the computation involves ℓ^2 calls to $\mathsf{FSS.Eval}$, which costs a total of $\approx (\ell m)^2$ PRG operations. Using the conservative parameters ($N = 5n, t = 664, \ell = 3\ln n$) with $n = 2^{30}$ outputs, the computation per output is 2M PRG operations, and seeds are around 15GB. However, if we instead use more aggressive parameters, for instance, with a much smaller noise weight of $t = 85$, the storage cost falls down to around 300MB, while the computation increases by less than a factor of two. So, it seems that with our aggressive parameter choices, $\mathsf{EA\text{-}LPN}$ gives the first, plausible candidate for a practical PCG and PCF for degree-2 correlations over a ring $\mathcal{R}$. (The PCF from [BCG+20a] does not have good concrete efficiency, since their degree-2 construction does not seem compatible with their most aggressive LPN candidate, and using their conservative candidate leads to much worse performance.)

Distributed setup costs. To setup the PCF keys in a distributed manner, we unfortunately cannot use the efficient, 2-round protocol for our RDCF (and similar protocols for FSS [Ds17], since they require a polynomial-sized domain. The setup therefore involves a more expensive, non-black-box protocol with higher communication. However, importantly, with a PCF this is a cost that only happens once and for all.

6 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 $\mathsf{FullEval}$ (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 $\mathsf{FullEval}$ now accounts for about half of the total computation. Reducing the cost of $\mathsf{FullEval}$ 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 $\mathsf{FullEval}$ computation and an accumulation. The cost of accumulate is exceptionally small, and dominated by the cost of $\mathsf{FullEval}$ (by several orders of magnitude). Hence, reducing the cost of $\mathsf{FullEval}$ 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 $\mathsf{FullEval}$ algorithm of a PPRF. Reducing the cost of $\mathsf{FullEval}$ 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.

The results in this section. In Section 6.1, we begin with 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 $\mathsf{FullEval}$ for GGM-type constructions such as PPRF in exchange for increased key size, by "flattening" the depth of the GGM tree.

In Section 6.2 we define 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).

In Section 6.3 we 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.

In Section 6.4 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 the PPRF of Section 6.3, instantiated with some of the UPF constructions of Section 6.2. 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 from Section 6.1 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).

6.1 High-Level Optimizations

Further reduction using a flatter tree. We observe that for FullEval in GGM tree based constructions of PPRFs and related primitives (such as our weaker unpredictable puncturable functions (UPFs) in the upcoming sections) we can get even better savings in exchange for slightly larger seed sizes (translating, in turn, to slightly larger PCG seed sizes), by "flattening" the GGM tree. Concretely, all constructions (the GGM PPRF, and our two UPF constructions of Section 6.2) compute hashes along the nodes of a binary tree. This can be naturally generalized to a k-ary tree.

When using a k-ary tree instead of a binary tree, the size of a punctured key increases by a factor $(k-1)\cdot\log_k(2)$ (because the punctured key contains all nodes on the co-path to the target point). For example, using $k=3$ leads to a factor 1.26 increase in the punctured key, and using $k=4$ leads to a factor 1.5 increase. However, in exchange for this mild key size increase (which leads to an increased runtime of the distributed seed generation by a comparable factor), the running time of FullEval drops from $2N$ calls to $(1+1/(k-1))N$ calls to the hash function (a 25% reduction for $k=3$, and a 33% reduction for $k=4$).

Looking ahead, this optimization can be combined with our UPF constructions, leading to further savings. Concretely, setting $k=3$ (hence increasing the seed size by a factor 1.26), the cost of our RIPM-based UPF drops to $1.33N$, and our ROM-based UPF drops to $1.17N$. Going to $k=4$ (a factor 1.5

increase in the seed size), the cost of the RIPM-based UPF further drops to $1.17N$.

Accumulating in Parallel. We now turn our attention to the accumulation step. Here, a single-pass algorithm can perform this step using $n-1$ `xor` in total; however, this cannot benefit from multicore architectures. When multiple processors are available, a much better strategy consists in using a simple two-pass algorithm, which we describe below.

Let c be the number of cores available. Suppose that the array size N is divisible by $c+1$. The algorithm proceeds as follows: partition the array into $c+1$ blocks of size $N/(c+1)$ and compute in parallel prefix-sums of each block except the last one; this takes $N/(c+1)$ parallel time. Then, compute the sum of each interval $[1, N']$ for $N' = c'N/(c+1)+1$, where $c' = 1, \ldots, c$, by computing the prefix-sum of the c sums computed before; this takes c parallel time. Eventually, add to the internal prefix sum of each block (except the first) the sum of all values to the left of this block, which was computed before. This again takes $N/(c+1)$ parallel time. Overall, the above algorithm takes $\approx 2N/(c+1)$ parallel time. Concretely, this translates to a 1.5x speedup with 2 cores, 8.5x with 16 cores, or a 32.5x with 64 cores.

We note that more cache-friendly variants of this two-pass algorithm have been studied in the literature, to benefit from GPU architectures. For example, the work of [ZWR] provides such an optimized variant variant. Using 48 threads, they report extremely impressive number, around $2 \cdot 10^{10}$ values processed per second. The bottom line is, implementing the accumulation with a simple parallelizable algorithm suffices to makes its cost completely negligible compared to the rest of the computation.

6.2 Unpredictable Puncturable Functions

We begin by defining a weakened version of punctured PRFs (PPRF), where the pseudorandomness requirement is relaxed to unpredictability.

Definition 6.1 (Unpredictable punctured function) *An* unpredictable punctured function (UPF) *over domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of a tuple of PPT algorithms* $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ *such that*

- Setup *on input 1^λ outputs a key k.*
- $\mathsf{Puncture}$ *on input of key k and value $\alpha \in \mathcal{X}_\lambda$ outputs a punctured key k^*.*
- Eval *on input of key k and value $x \in \mathcal{X}_\lambda$ outputs value $y \in \mathcal{Y}_\lambda$.*
- $\mathsf{PuncEval}$ *on input of punctured key k^*, restriction $\alpha \in \mathcal{X}_\lambda$ and value $x \in \mathcal{X}_\lambda$ outputs value $y \in \mathcal{Y}_\lambda$ or $\perp$.*

Further, the following properties are required to hold:

Correctness: *For all $\lambda \in \mathbb{N}$, for all keys k in the image of $\mathsf{Setup}(1^\lambda)$, for all $\alpha \in \mathcal{X}_\lambda$ and for all k^* in the image of $\mathsf{Puncture}(k, \alpha)$ it holds:*

$$\mathsf{Eval}(k, x) = \mathsf{PuncEval}(k^*, \alpha, x)$$

for all $x \in \mathcal{X}_\lambda$ with $x \neq \alpha$.

Unpredictability: *For all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to R_{\geq 0}$ such that for all $\lambda \in \mathbb{N}$:*

$$\Pr[\mathsf{Exp}^{\mathsf{unp}}_{\mathsf{UPF},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{unp}}_{\mathsf{UPF},\mathcal{A}}(\lambda)$ is as defined in Figure 14.

For the purpose of using the UPF as a plug-in replacement for a puncturable PRF in the Silent OT extension, we require a stronger definition, where we capture that except with some small probability, the adversary has *no* information about $\mathsf{Eval}(k, \alpha)$ on the punctured point α. We capture this in the following definition of *strong unpredictability*. Note that strong unpredictability in particular implies unpredictability, it therefore suffices to show that our constructions satisfy the notion of strong unpredictability.

Definition 6.2 (Strong unpredictability) *Let $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^*$ be modeled as a random oracle and let $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ be an UPF over domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ having access to H. We say UPF satisfies* strong unpredictability *relative to H, if for every PPT adversary $\mathcal{A}$ the following holds: There exists an event $\mathsf{bad} = \mathsf{bad}_\lambda$ over the random choice of the random oracle H, the random choice of $\alpha \xleftarrow{\$} \mathcal{X}_\lambda$, as well as the random coins of $k \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathcal{A}^{\mathsf{H}(\cdot)}(1^\lambda, \alpha, k^*)$, where $k^* \leftarrow \mathsf{Puncture}(\alpha, k)$, such that the following holds:*

$\underline{\mathsf{Exp}^{\mathsf{unp}}_{\mathsf{UPF},\mathcal{A}}(\lambda)}$:

$\alpha \xleftarrow{\$} \mathcal{X}_\lambda$

$k \leftarrow \mathsf{Setup}(1^\lambda)$

$k^* \leftarrow \mathsf{Puncture}(k, \alpha)$

$y \leftarrow \mathcal{A}(k^*, \alpha)$

If $y = \mathsf{Eval}(k, \alpha)$ **return** 1

Else return 0.

Figure 14: Unpredictability experiment

- *There exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to R_{\geq 0}$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr[\mathsf{bad}_\lambda] \leq \mathsf{negl}(\lambda)$.
- *For all* $\lambda \in \mathbb{N}$, $\alpha \in \mathcal{X}_\lambda$, k *in the image of* $\mathsf{Setup}(1^\lambda)$ *and* $k^* \leftarrow \mathsf{Puncture}(k, \alpha)$ *it holds that conditioned on* $\neg\mathsf{bad}_\lambda$, $\mathsf{Eval}(k, \alpha)$ strongly unpredictable *from the view of* $\mathcal{A}^{\mathsf{H}(\cdot)}(1^\lambda, \alpha, k^*)$, *that is,*

$$\Pr[\mathcal{A}^{\mathsf{H}(\cdot)}(1^\lambda, \alpha, k^*) = \mathsf{Eval}(k, \alpha) \mid \neg\mathsf{bad}_\lambda] \leq \frac{1}{|\mathcal{Y}_\lambda| - Q(\lambda)},$$

where Q *is the number of oracle queries by* $\mathcal{A}$ *to* H *(and the randomness is taken over the choice of the random oracle* H *and the choice of* k^**).*

To give some intuition behind these definitions, suppose H is a bijection on $\{0,1\}^\lambda \to \{0,1\}^\lambda$, and consider a randomly chosen $x \xleftarrow{\$} \{0,1\}^\lambda$. Then x is strongly unpredictable given $z := \mathsf{H}(x)$, in the sense that unless an adversary is lucky enough to have x among its Q queries to H, then its ability to predict x is no better than uniform among the remaining $2^\lambda - Q$ possible values. More formally, the corresponding event bad_λ in the definition above is an adversary that on input z queries x to H. Since z does not reveal any information about x, the probability of this event can be upper bounded by

$$\Pr[\mathsf{bad}_\lambda] \leq \frac{Q(\lambda)}{2^\lambda},$$

where Q corresponds to the number of queries asked by $\mathcal{A}$. Further, the adversary cannot gain any more information about x than potentially ruling out the $Q(\lambda)$ previously asked values. It therefore holds

$$\Pr[\mathcal{A}^{\mathsf{H}(\cdot)}(z) = x \mid \neg\mathsf{bad}_\lambda] \leq \frac{1}{2^\lambda - Q(\lambda)},$$

as desired.

Note, however, that even strong unpredictability does not imply that x is indistinguishable from random from the view of the adversary. This is because given a candidate value $\tilde{x}$, the adversary can efficiently check whether $\mathsf{H}(\tilde{x}) = z$.

An important difference between *unpredictability* and *strong unpredictability* is that for an unpredictable value $x \in \{0,1\}^\lambda$ giving out $x \oplus r$ for $r \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random could potentially leak, e.g., half the bits of r (since one half of x might be predictable), therefore giving out $x \oplus r$ and $x' \oplus r$ for two unpredictable values x and x' could potentially leak r completely. If x is *strongly unpredictable*, on the other hand, giving out $x \oplus r$, the adversary obtains *almost no information* about r (except with negligible probability). Therefore, even given $x_i \oplus r$ for polynomially many *strongly unpredictable* values x_i, r remains unpredictable to an adversary. This will be a necessary condition to prove that a strong UPF suffices to instantiate Silent OT extension, in Section 6.4.

Strong unpredictable punctured function in the random oracle model. We present in Figure 15 a generic template for a a strong unpredictable punctured function, parameterized by two length-preserving functions $\mathsf{H}_0, \mathsf{H}_1$. For an appropriate choice of $\mathsf{H}_0, \mathsf{H}_1$, this construction is exactly the standard GGM construction of puncturable pseudorandom function; for example, instantiating H_0 and H_1 as two independent random oracles gives a PPRF in the ROM. In the following, we will put forth alternative choices for the functions $\mathsf{H}_0, \mathsf{H}_1$ such that:

- on an input x, $\mathsf{H}_0(x)$ and $\mathsf{H}_1(x)$ can be *simultaneously computed* using a *single* call to a length-preserving cryptographic primitive, and
- with this instantiation, the construction of Figure 15 is indeed a strong unpredictable punctured function.

- Setup on input 1^λ outputs a key $k \xleftarrow{\$} \{0,1\}^\lambda$.
- $\mathsf{Puncture}$ on input of a key k and a value $\alpha = (\alpha_1, \ldots, \alpha_m) \in \{0,1\}^m$ outputs the punctured key $k^* = (k_1^*, \ldots, k_m^*)$, where

$$k_i^* := \mathsf{H}_{\overline{\alpha}_i}(\mathsf{H}_{\alpha_{i-1}}(\mathsf{H}_{\alpha_{i-2}}(\ldots(\mathsf{H}_{\alpha_1}(k))\ldots))).$$

- Eval on input of a key k and a value $x \in \{0,1\}^m$ outputs

$$y := \mathsf{H}_{x_m}(\mathsf{H}_{x_{m-1}}(\ldots(\mathsf{H}_{x_1}(k))\ldots)) \in \{0,1\}^\lambda.$$

- $\mathsf{PuncEval}$ on input of a key $k^* = (k_1^*, \ldots, k_m^*) \in (\{0,1\}^\lambda)^m$, restriction α and value $x \neq \alpha$ sets $j \in \{1, \ldots, m\}$ as the minimal index such that $x_j \neq \alpha_j$ and outputs

$$y := \mathsf{H}_{x_m}(\mathsf{H}_{x_{m-1}}(\ldots(\mathsf{H}_{x_{j+1}}(k_j^*))\ldots)) \in \{0,1\}^\lambda.$$

Figure 15: General template for a strong unpredictable punctured function $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ over domain $[N]$ and range $\{0,1\}$, based on public functions $\mathsf{H}_0, \mathsf{H}_1$, where $[N]$ is interpreted as $\{0,1\}^m$ for $m := \log N$.

Our first theorem shows that instantiating $\mathsf{H}_0, \mathsf{H}_1$ as $\mathsf{H}_0(k) := \mathsf{H}(k)$ and $\mathsf{H}_1(k) := \mathsf{H}(k) \oplus k$, where H is modeled as a random oracle, indeed leads to a strong unpredictable punctured function (note that evaluating H_0 and H_1 on a common input k requires a single call to the random oracle).

Theorem 6.3 *Let N be a power of 2. For all $\lambda \in \mathbb{N}$, let $\mathsf{H}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ be modeled as a random oracle, and define $\mathsf{H}_0(k) := \mathsf{H}(k)$ and $\mathsf{H}_1(k) := \mathsf{H}(k) \oplus k$ for all $k \in \{0,1\}^\lambda$. Then, UPF as defined in Figure 15 with domain $[N]$ and range $\{0,1\}^\lambda$, instantiated with $\mathsf{H}_0, \mathsf{H}_1$ is a strong unpredictable punctured function, where for any PPT adversary $\mathcal{A}$ asking at most Q queries to H the probability of the bad event $\mathsf{bad} = \mathsf{bad}_\lambda$ occurring is bounded by*

$$\Pr[\mathsf{bad}_\lambda] \leq \frac{2 \cdot \log^2 N + \cdot \log N}{2^\lambda} + \frac{2 \cdot Q \cdot \log N}{2^\lambda}$$

for all sufficiently large $\lambda \in \mathbb{N}$.

Proof. Let $m := \log N$ and let $\alpha \xleftarrow{\$} [N]$ denote the value sampled by the experiment interpreted as bitstring $\alpha = \alpha_1 \| \ldots \| \alpha_m \in \{0,1\}^m$. By the *path of* α in the tree defined by k we denote the values $p^* := (p_1^*, \ldots, p_m^*)$, where

$$p_i^* := \mathsf{H}_{\alpha_i}(\mathsf{H}_{\alpha_{i-1}}(\mathsf{H}_{\alpha_{i-2}}(\ldots(\mathsf{H}_{\alpha_1}(k))\ldots)) \in \{0,1\}^\lambda$$

(recall that $m = \log N$) on the path of α. Note that $k^* = (k_1^*, \ldots, k_m^*)$ as defined in Figure 15 is equal to the *co-path of* α, that is all values in the tree defined by k *neighboring* the path of α.

By the event bad we denote the event $\mathsf{bad} := \mathsf{bad}_1 \vee \mathsf{bad}_2$, where

- bad_1: There is a collision on the values $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots k_m^*$.
- bad_2: The adversary $\mathcal{A}$ queries H on any of the values $k, p_1^*, \ldots, p_{m-1}^*$.

Claim 1: If bad does not occur, then $\mathsf{Eval}(k, x)$ is strongly unpredictable. That is,

$$\Pr[\mathcal{A}^{\mathsf{H}(\cdot)}(1^\lambda, \alpha, k^\star) = \mathsf{Eval}(k, \alpha) \mid \mathsf{bad}_\lambda] \leq \frac{1}{2^\lambda - Q(\lambda)},$$

where Q is the number of oracle queries by $\mathcal{A}$ to H (and the probability is taken over the random choice of H and k^*).

Proof. We change the setup procedure as follows. Instead of sampling $k \leftarrow \mathsf{Setup}(1^\lambda)$ and setting $k^* \leftarrow \mathsf{Puncture}(k, \alpha)$, the experiment now proceeds as follows:

- Sample $k_1^*, \ldots, k_m^* \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random.
- Sample $p_m^* \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random.
- For $j \in [m-1]$ set $p_j^* := p_{j+1}^* \oplus k_{j+1}^*$.
- Set $k := p_1^* \oplus k_1^*$.
- If there is any collision on the values $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots, k_m^*$ resample, else program the random oracle as:
  - $\mathsf{H}(k) := \begin{cases} p_1^* & \text{if } \alpha_1 = 0 \\ k_1^* & \text{else} \end{cases}$
  - For $j \in [m-1]$: $\mathsf{H}(p_j^*) := \begin{cases} p_{j+1}^* & \text{if } \alpha_{j+1} = 0 \\ k_{j+1}^* & \text{else} \end{cases}$

We will first show that the distribution of the experiment above is identical to the original experiment if bad_1 does not occur, i.e., if there is no collision on the values $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots, k_m^*$.

To this end, note that for all $j \in [m-1]$ we have

$$\begin{aligned} p_{j+1}^* \oplus k_{j+1}^* &= \mathsf{H}_{\alpha_j}(p_j^*) \oplus \mathsf{H}_{\overline{\alpha}_{j+1}}(p_j^*) \\ &= \mathsf{H}(p_j^*) \oplus \mathsf{H}(p_j^*) \oplus p_j^* \\ &= p_j^* \end{aligned}$$

and similarly

$$p_1^* \oplus k_1^* = k.$$

Further, since H is modeled as a random oracle and there are no collisions on $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots, k_m^*$, we have that $k, k_1^*, \ldots, k_m^*, p_1^*, \ldots, p_m^* \xleftarrow{\$} \{0,1\}^\lambda$ are sampled uniformly at random from $\{0,1\}^\lambda$ conditioned on

1. no collision occurring on $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots, k_m^*$
2. the above equations being satisfied.

This is equivalent to sampling $k_1^*, \ldots, k_m^* \xleftarrow{\$} \{0,1\}^\lambda$ and $p_m^* \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random and defining $k, p_1^*, \ldots, p_{m-1}^*$ as done in the alternative experiment above.

Now, if $\mathcal{A}$ does not query any of the values $k, p_1^*, \ldots, p_{m-1}^*$ to H, then the distribution of $\mathsf{Eval}(k, \alpha) = p_m^*$ is fully independent of $k_1^*, \ldots, k_m^*$, and therefore the adversary can at most rule out $Q(\lambda)$ of the possible 2^λ candidate values for p_m^*. This concludes the proof of Claim 1.

□

It is left to show that we can upper bound the probability of the event $\mathsf{bad} = \mathsf{bad}_1 \vee \mathsf{bad}_2$ occurring. We have $\Pr[\mathsf{bad}] = \Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2] \leq \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}_2 \mid \neg\mathsf{bad}_1]$. We can thus proceed the proof by considering the two cases separately.

Claim 2: There is no collision on the values $k, p_1^*, \ldots, p_{m-1}^*, k_1^*, \ldots, k_m^*$ except with negligible probability. More precisely, it holds

$$\Pr[\mathsf{bad}_1] \leq \frac{2\log^2 N + \log N}{2^\lambda}$$

(where the probability is taken over the random choice of α, k and the randomness of H).

Proof. For all $j \in [m]$ let $L_j := \{k, p_1^*, \ldots, p_j^*, k_1^*, \ldots, k_j^*\}$. Then, we define the event bad_j as follows:

- $\widetilde{\mathsf{bad}}_j$: The set L_j does *not* contain $2j+1$ distinct values.

For $\Pr[\widetilde{\mathsf{bad}_1}]$ it holds

$$\begin{aligned}\Pr[\widetilde{\mathsf{bad}_1}] &= \Pr[\mathsf{H}(k) \neq k \wedge \mathsf{H}(k) \oplus k \neq k \wedge \mathsf{H}(k) \neq \mathsf{H}(k) \oplus k] \\ &= \Pr[\mathsf{H}(k) \neq k \wedge \mathsf{H}(k) \neq 0 \wedge k \neq 0] \leq \frac{3}{2^\lambda}.\end{aligned}$$

For all $j \in [m]$ with $j \geq 2$ in the following we will bound $\Pr[\widetilde{\mathsf{bad}_j} \mid \neg\widetilde{\mathsf{bad}_{j-1}}]$. Since we condition on the event that L_{j-1} contains $2j+1$ distinct values, we have that $\mathsf{H}(p^*_{j-1})$ is distributed uniformly at random (independently from all values in L_{j-1}). It thus follows

$$\begin{aligned}&\Pr[\widetilde{\mathsf{bad}_j} \mid \neg\widetilde{\mathsf{bad}_{j-1}}] \\ &= \Pr\left[\mathsf{H}(p^*_{j-1}) \notin L_{j-1} \wedge \mathsf{H}(p^*_{j-1}) \oplus p^*_j \notin L_{j-1} \wedge \mathsf{H}(p^*_{j-1}) \neq \mathsf{H}(p^*_{j-1}) \oplus p^*_{j-1}\right] \\ &\leq \frac{2 \cdot (2(j-1)+1)+1}{2^\lambda} = \frac{4j-1}{2^\lambda}.\end{aligned}$$

With this we obtain

$$\begin{aligned}\Pr[\mathsf{bad}_1] = \Pr[\widetilde{\mathsf{bad}_{\log N}}] &\leq \Pr[\widetilde{\mathsf{bad}_1}] + \sum_{j=2}^{\log N} \Pr[\widetilde{\mathsf{bad}_j} \mid \neg\widetilde{\mathsf{bad}_{j-1}}] = \sum_{j=1}^{\log N} \frac{4j-1}{2^\lambda} \\ &= \frac{4 \cdot \log N \cdot (\log N + 1)}{2 \cdot 2^\lambda} - \frac{\log N}{2^\lambda} \leq \frac{2\log^2 n + \log N}{2^\lambda}\end{aligned}$$

as required. □

Claim 3: If there are no collisions on $k, p^*_1, \ldots, p^*_{m-1}, k^*_1, \ldots, k^*_m$, then the adversary does not query any of the values $k, p^*_1, \ldots, p^*_{m-1}$ to H except with negligible probability. More precisely, it holds

$$\Pr[\mathsf{bad}_2 \mid \neg\mathsf{bad}_1] \leq \frac{2 \cdot Q \cdot \log N}{2^\lambda}.$$

Proof. Recall that if there are no collisions on $k, p^*_1, \ldots, p^*_{m-1}, k^*_1, \ldots, k^*_m$ (i.e., bad_1 does not occur), we have that $k, k^*_1, \ldots, k^*_m, p^*_1, \ldots, p^*_m \xleftarrow{\$} \{0,1\}^\lambda$ are sampled uniformly at random from $\{0,1\}^\lambda$ conditioned on

1. no collision occurring on $k, p^*_1, \ldots, p^*_{m-1}, k^*_1, \ldots, k^*_m$, and
2. for all $j \in [m-1]$ it holds

$$\begin{aligned}p^*_{j+1} \oplus k^*_{j+1} &= \mathsf{H}_{\alpha_j}(p^*_j) \oplus \mathsf{H}_{\overline{\alpha}_{j+1}}(p^*_j) \\ &= \mathsf{H}(p^*_j) \oplus \mathsf{H}(p^*_j) \oplus p^*_j \\ &= p^*_j\end{aligned}$$

as well as

$$p^*_1 \oplus k^*_1 = k.$$

This implies that from the view of $\mathcal{A}$ (before asking any queries to H) each of the values p^*_i *individually* is distributed uniformly at random conditioned on yielding pairwise distinct values $k, p^*_1, \ldots, p^*_{m-1}, k^*_1, \ldots, k^*_m$.

We can loosely lower bound the set of possible values to be of size at least $\frac{3}{4} \cdot 2^\lambda$, since $m^2 = \log^2 N \ll \frac{1}{4} \cdot 2^\lambda$ for all sufficiently large $\lambda \in \mathbb{N}$. Note that the choice of $\frac{3}{4}$ here is somewhat arbitrary, but allows that even after Q attempts, the set of possible values is still of size at least $\frac{1}{2} \cdot 2^\lambda$, since $Q \ll \frac{1}{4} \cdot 2^\lambda$ for all sufficiently large $\lambda \in \mathbb{N}$.

Therefore, the adversary $\mathcal{A}$ that cannot do better than guessing from this restricted set of values. The probability that bad_2 occurs, i.e. $\mathcal{A}$ queries any of the $m = \log N$ values $k, p^*_1, \ldots, p^*_{m-1}$ to H, therefore happens at most with probability $Q \cdot \log N \cdot 2/2^\lambda$ by a union bound.

This yields

$$\Pr[\mathsf{bad}_2 \mid \neg\mathsf{bad}_1] \leq \frac{2 \cdot Q \cdot \log N}{2^\lambda}$$

as claimed.

□

This concludes the proof. □

Unpredictable punctured function in the random permutation model. When instantiating the construction of Figure 15 in the real world, it might be desirable to build upon an *invertible permutation* rather than a random oracle: this is because fixed-key AES provides such an invertible permutation, and the current Intel support for AES (the AES-NI instructions) make the use of this primitive significantly faster than most alternatives. 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. We believe, nonetheless, that our construction from Theorem 6.3 remains the most natural efficient construction. Its inefficiency compared to an AES-based construction is an unfortunate artifact of the fact that AES has hardware support while hash functions do not; but were hash functions to have hardware support in a not-so-distant future, they would likely be even faster than AES, since invertible permutations are a much more structured object.

Still, given the current state of hardware support, it is interesting to investigate whether one can construct an unpredictable punctured function with the same efficiency gains (a single call to the primitive for computing the two children of a node) in the *random invertible permutation* model; this question is also a natural question of theoretical interest. We bring forward a candidate construction which we believe to satisfy unpredictability. In the following we describe the candidate and give an explanation why we believe it secure. We leave a full analysis for future work.

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.

Note though that we cannot hope to show *strong* unpredictability for this candidate. To see this, suppose that $H(x) \oplus x$ equals 1 at the i-th position. By the property of the $\oplus$-operation, we know that in this case either $H(x)_i = 1$ and $x_i = 0$, or $H(x)_i = 0$ and $x_i = 1$ must hold. Therefore, whenever there is a length-ℓ sequence of 1's, we know that there has to be a length-$(\ell - 1)$ sequence of 1's in $H(x) + x \bmod 2^\lambda$, contradicting strong unpredictability of $H(x) + x \bmod 2^\lambda$. It is therefore not secure to use this candidate directly within the Silent OT extension, but by applying the transformation of Section 6.3 as done for the RO candidate (where we implement the hash functions to hash the left leaves of the last level with H_0) we conjecture that one can achieve a 25% improvement compared to the standard PPRF construction based on AES.

Conjecture 6.4 *Let N be a power of 2. For all $\lambda \in \mathbb{N}$, let $\mathsf{H}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ be modeled as a random invertible permutation oracle, and define $\mathsf{H}_0(k) := \mathsf{H}(k) \oplus k$ and $\mathsf{H}_1(k) := \mathsf{H}(k) + k \bmod 2^\lambda$ for all $k \in \{0,1\}^\lambda$ (where k is also viewed as an integer between 0 and $2^\lambda - 1$). Then, UPF as defined in Figure 15 with domain $[N]$ and range $\{0,1\}^\lambda$, instantiated with $\mathsf{H}_0, \mathsf{H}_1$ is an unpredictable punctured function.*

Underlying intuition. As before, let $m := \log N$, let $\alpha \in [N]$ denote the value sampled by the experiment interpreted as bitstring $\alpha = \alpha_1 \| \ldots \| \alpha_m \in \{0,1\}^m$, let $p^* = (p_1^* \cdots p_m^*)$ the path of α defined by the key k, and let $(k_1^* \cdots k_m^*)$ the co-path of α. A possible strategy to prove unpredictability is as follows:

1. Show that given the punctured key $(k_1^* \ldots k_m^*)$ except with negligible probability it is hard to predict the root key k with probability $\varepsilon_{\mathcal{A}}$ using significantly less than $2^{\lambda - \log \frac{1}{\varepsilon_{\mathcal{A}}}}$ queries to H.
2. Show that given the punctured key $(k_1^* \ldots k_m^*)$ if there was an adversary who could predict $\mathsf{Eval}(k, \alpha)$ with probability $\varepsilon_{\mathcal{A}}$ using at most Q hash queries, then there is an adversary $\mathcal{A}'$ which can predict k with probability at least $\varepsilon_{\mathcal{A}}$ using on average (over the randomness of k and H) $\ll 2^{\lambda - \log \frac{1}{\varepsilon_{\mathcal{A}}}}$ queries to H.

In the following we provide a proof of the latter claim and leave it for future work to provide a proof of the former. (Note that for a depth-1 tree the former is obviously true, since both $\mathsf{H}(k) \oplus k$ and $\mathsf{H}(k) + k \bmod 2^\lambda$ individually reveal no information about k.)

Claim: If an adversary $\mathcal{A}$ can predict p_m^* with probability $\epsilon_{\mathcal{A}}$ using at most Q hash queries, then there is an adversary $\mathcal{A}'$ which can predict k with probability $\epsilon_{\mathcal{A}}$ using on average (over the randomness of k and H) $Q + m \cdot 2^{c \cdot \lambda - \log \frac{1}{\varepsilon_{\mathcal{A}}}}$ hash queries, where $c \approx 0.58$ is a constant.

Proof. We show that given $\{p^*_m, k^*_m\} = \{x \oplus \mathsf{H}(x), x + \mathsf{H}(x) \bmod 2^\lambda\}$, it is possible to recover $x = p^*_{m-1}$ using on average $2^{\lambda/2}$ queries; the claim will follow directly by traversing the tree back to its root, spending $2^{\lambda/2}$ to recover the parent node from two children nodes.

Let $y = \mathsf{H}(x), z = x \oplus \mathsf{H}(x), a = x + \mathsf{H}(x) \bmod 2^\lambda$, and define $\mathsf{carry}_0 = 0$. Then for any $i \in \{1, \cdots, \lambda\}$, it holds that $a_i = x_i \oplus y_i \oplus \mathsf{carry}_{i-1}$, and $\mathsf{carry}_i = \mathsf{majority}(x_i, y_i, \mathsf{carry}_{i-1})$. Now, given (a, z), $\mathcal{A}'$ can reconstruct $\mathsf{carry}_{i-1} = a_i \oplus z_i$. Whenever $z_i = 0$, observe that $x_i \oplus y_i = 0$, hence either $x_i = y_i = 1$ or $x_i = y_i = 0$; in both cases, it holds that $\mathsf{carry}_i = \mathsf{majority}(x_i, y_i, \mathsf{carry}_{i-1}) = x_i$. Therefore, for all positions where $z_i = 0$, $\mathcal{A}'$ recovers $x_i = \mathsf{carry}_i$. To recover the missing bits, $\mathcal{A}'$ can query the $2^{\mathsf{HW}(z)}$ possible inputs x to H until it finds an input that matches z and a.

By linearity of expectation, when traversing the entire tree, $\mathcal{A}'$ can recover the root key k using on average $m \cdot \mathbb{E}[2^{\mathsf{HW}(z)}]$ queries. Given a uniformly random key k and a random permutation H, each k^*_i (resp. p^*_i) is individually distributed as a uniformly random bitstring – in particular, z is. However, some care must be taken with the expectation in this formula, since the random variable z is conditioned on $\mathcal{A}$ succeeding (which happens with probability $\varepsilon_{\mathcal{A}}$). We can bound $\mathbb{E}_{\mathcal{A}\text{ wins}}[2^{\mathsf{HW}(z)}]$ as follows:

$$\begin{aligned}
\mathbb{E}_{\mathcal{A}\text{ wins}}[2^{\mathsf{HW}(z)}] &= \sum_{i=0}^{\lambda} 2^i \cdot \Pr[\mathsf{HW}(z) = i \mid \mathcal{A}\text{ wins}] \\
&= \varepsilon_{\mathcal{A}} \cdot \sum_{i=0}^{\lambda} 2^i \cdot \Pr[\mathsf{HW}(z) = i \ \wedge\ \mathcal{A}\text{ wins}] \\
&\leq \varepsilon_{\mathcal{A}} \cdot \sum_{i=0}^{\lambda} 2^i \cdot \Pr[\mathsf{HW}(z) = i] \\
&= \varepsilon_{\mathcal{A}} \cdot \mathbb{E}[2^{\mathsf{HW}(z)}] = \varepsilon_{\mathcal{A}} \cdot \prod_{i=1}^{\lambda} \mathbb{E}[2^{z_i}] = \varepsilon_{\mathcal{A}} \cdot 1.5^\lambda = 2^{c\lambda + \log \varepsilon_{\mathcal{A}}},
\end{aligned}$$

where the third to last equality is because the bits z_i of z are independently random and equal to $1/2$ (when we do not condition anymore on $\mathcal{A}$ winning the game), and $c = \log(1.5)$ is a constant. This concludes the proof of the claim. □

6.3 From UPF to Puncturable Pseudorandom Functions

In the following we show that a UPF implies a PPRF in the random oracle model by hashing the outputs of the Eval algorithm, if the size of the input domain $\mathcal{X}$ is polynomially bounded (i.e., $|\mathcal{X}_\lambda| \leq \mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly} \in \mathbb{N}[X]$).[12] This general approach will render the optimizations of the previous section in general ineffective (since it requires $|\mathcal{X}_\lambda|$ extra RO calls for a full evaluation), but we will show that using a more careful tailored reduction we obtain PPRFs with a 25% and more improvement regarding the RO calls required for a full evaluation.

PPRF from Generic UPF

- F on input of a key k and $x \in X_\lambda$, computes $y \leftarrow \mathsf{Eval}(k, x)$ and outputs $z := \mathsf{H}(y)$.
- $F.\mathsf{KeyGen}$ on input 1^λ outputs $k \leftarrow \mathsf{Setup}(1^\lambda)$.
- $F.\mathsf{Puncture}$ on input of a key k and a singleton set $\{\alpha\}$, computes $k^* \leftarrow \mathsf{Puncture}(k, \alpha)$ and outputs (k^*, α).
- $F.\mathsf{Eval}$ on input of a punctured key (k, α) and input value x, computes $y \leftarrow \mathsf{PuncEval}(k^*, \alpha, x)$ and outputs $z := \mathsf{H}(y)$.

Figure 16: Puncturable PRF $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ from UPF $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ and hash function $\mathsf{H}\colon \mathcal{Y}_\lambda \to \mathcal{Y}_\lambda$ modeled as a random oracle.

[12] Note that while this is a limitation, it constitutes the only regime where computing a full evaluation is feasible.

Theorem 6.5 *Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles of sets, such that $|\mathcal{X}_\lambda| \leq \mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly} \in \mathbb{N}[X]$. Let $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ be a UPF over domain $\mathcal{X}$ and range $\mathcal{Y}$. Let $\mathsf{H}\colon \mathcal{Y}_\lambda \to \mathcal{Y}_\lambda$ be modeled as a random oracle. Then F together with the tuple of algorithms $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ as given in Figure 16 defines a 1-puncturable pseudorandom function.*

Proof. Note that since the input domain $\mathcal{X}$ is polynomially bounded, it suffices to show that $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ satisfies selective security. From this it automatically follows that F is a PRF, since one can transform an adversary on the PRF security of F into an adversary on the selective security of $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ by guessing the point x^* on which the adversary wants to be challenged ahead of time and answering all oracle queries of the adversary by the key punctured on x^*. This incurs a loss of $\frac{1}{|X_\lambda|} \leq \frac{1}{\mathsf{poly}(\lambda)}$ for some polynomial $\mathsf{poly} \in \mathbb{N}[X]$, yielding the required.

It is left to show that $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ indeed satisfies selective security. To that end, let $\mathcal{A}$ be a PPT adversary on the selective security of $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ with success probability $\varepsilon_\mathcal{A}$. Then, we construct an adversary $\mathcal{B}$ breaking the unpredictability of UPF as follows. The adversary $\mathcal{B}$ obtains α, k^* by its own security experiment and awaits input $\{\alpha'\}$ by $\mathcal{A}$. If $\alpha \neq \alpha'$ it aborts. Otherwise, it forwards k^* to $\mathcal{A}$ and additionally a value $y \xleftarrow{\$} \mathcal{Y}_\lambda$ sampled at random. Further, $\mathcal{B}$ records all queries to H by $\mathcal{A}$ and answers consistently by keeping a list of queries and sampling a fresh value from $\mathcal{Y}_\lambda$ uniformly at random for each fresh query. When $\mathcal{A}$ outputs its guess after, say, Q hash queries, $\mathcal{B}$ samples a random index $i \xleftarrow{\$} [Q]$ and outputs the i-th query of $\mathcal{B}$ to H as its guess on $\mathsf{Eval}(k, \alpha)$.

First, note that since α is sampled at random, the probability that $\mathcal{B}$ aborts can be upper bounded by $1 - 1/|\mathcal{X}_\lambda|$. Further, the probability that $\mathcal{B}$ aborts is independent of the behaviour of $\mathcal{A}$, we thus have

$$\Pr[\mathcal{A} \text{ breaks PPRF } \wedge \alpha = \alpha'] \geq \frac{\epsilon_\mathcal{A}(\lambda)}{|X_\lambda|}.$$

If $\mathcal{A}$ is successful in distinguishing the case $b = 0$ (i.e., obtaining $y = \mathsf{H}(\mathsf{Eval}(k, \alpha))$) and $b = 1$ (i.e., obtaining $y \xleftarrow{\$} \{0,1\}^\lambda$) with probability at least $\varepsilon_\mathcal{A}$, given k^* it must query H on $Y := \mathsf{Eval}(k, \alpha)$ with at least probability $\varepsilon_\mathcal{A}$ (since otherwise the two distributions are *identical* from the view of $\mathcal{A}$). Further note that up to the point where $\mathcal{A}$ potentially queries Y, the adversary $\mathcal{B}$ perfectly simulates the selective security game for PPRFs.

Overall, we thus obtain that $\mathcal{B}$ succeeds with probability at least

$$\Pr[\mathcal{B} \text{ breaks UPF}] \geq \frac{\epsilon_\mathcal{A}(\lambda)}{|X_\lambda| \cdot Q(\lambda)}$$

and thus

$$\epsilon_\mathcal{A}(\lambda) \leq |X_\lambda| \cdot Q(\lambda) \cdot \Pr[\mathcal{B} \text{ breaks UPF}].$$

Since $\Pr[\mathcal{B} \text{ breaks UPF}]$ is negligible by assumption, $|\mathcal{X}_\lambda| \leq \mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly} \in \mathbb{N}[X]$, and $Q \in \mathbb{N}[X]$, we obtain that also $\epsilon_\mathcal{A}$ has to be negligible, which concludes the proof. □

We next show that our specific UPF construction from the previous section (Theorem 6.3) can be converted into a PPRF by applying the random oracle to only *half* the output evaluations (instead of all), thus saving $0.5N$ calls from the generic transformation.

Theorem 6.6 *Let $N = N(\lambda)$ be a power of 2, such that $N(\lambda) \leq \mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly} \in \mathbb{N}[X]$. For all $\lambda \in \mathbb{N}$, let $\mathsf{H}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ be modeled as a random oracle. Let $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ be the UPF with over domain $[N]$ and range $\{0,1\}^\lambda$ from Theorem 6.3. Then, F together with the tuple of algorithms $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ as given in Figure 17 defines a 1-puncturable pseudorandom function.*

Proof. Again, we only have to show that $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ satisfies selective security. Let $\mathcal{A}$ be an adversary on the selective security of $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ that succeeds with probability $\varepsilon_\mathcal{A}$ and let $\{\alpha'\}$ denote the input by $\mathcal{A}$. Again, we construct an adversary $\mathcal{B}$ that receives α, k^* by its own security experiment. If $\alpha \neq \alpha'$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ proceeds as follows. If $\alpha_{\log N} = 1$, $\mathcal{B}$ proceeds as in the proof of Theorem 6.5. Else, $\mathcal{B}$ parses $k^* =: (k^*_1, \ldots, k^*_{\log N})$ and forwards

$$K^* := (k^*_1, \ldots, k^*_{\log N - 1}, \mathsf{H}(k^*_{\log N}))$$

together with a random value $y \xleftarrow{\$} \{0,1\}^\lambda$ to $\mathcal{A}$. Recall that $\mathcal{A}$ attempts to distinguish $p^*_{\log N} = \mathsf{H}(p^*_{\log N - 1})$ from random, where $p^*_{\log N - 1}$ is such that $\mathsf{H}(p^*_{\log N - 1}) \oplus p^*_{\log N - 1} = k^*_{\log N}$. Similar to the

PPRF from UPF Construction of Theorem 6.3

- F on input of a key k and $x \in X_\lambda$, computes $y \leftarrow \mathsf{Eval}(k, x)$ and outputs

$$z := \begin{cases} y & \text{if } x_{\log N} = 0 \\ \mathsf{H}(y) & \text{else} \end{cases}.$$

- $F.\mathsf{KeyGen}$ on input 1^λ outputs $k \leftarrow \mathsf{Setup}(1^\lambda)$.
- $F.\mathsf{Puncture}$ on input of a key k and a singleton set $\{\alpha\}$, computes $k^* = (k_1^*, \ldots, k_{\log N}^*) \leftarrow \mathsf{Puncture}(k, \alpha)$ and outputs (k^*, α), where

$$K^* := \begin{cases} k^* & \text{if } 1 - \alpha_{\log N} = 0 \\ (k_1^*, \ldots, k_{\log N - 1}^*, \mathsf{H}(k_{\log N}^*)) & \text{else} \end{cases}.$$

- $F.\mathsf{Eval}$ on input of a punctured key (k, α) and input value x, computes $y \leftarrow \mathsf{PuncEval}(k^*, \alpha, x)$ and outputs

$$z := \begin{cases} y & \text{if } x_{\log N} = 0 \vee x = \alpha \\ \mathsf{H}(y) & \text{else} \end{cases}.$$

Figure 17: Puncturable PRF ($F.\mathsf{KeyGen}$, $F.\mathsf{Puncture}$, $F.\mathsf{Eval}$) from UPF $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ over domain $[N]$ and range $\{0,1\}$ from Theorem 6.3.

proof of Theorem 6.5, we have that if $\mathcal{A}$ is successful in distinguishing the case $b = 0$ (i.e., obtaining $y = \mathsf{H}(p_{N-1}^*)$) and $b = 1$ (i.e., obtaining $y \xleftarrow{\$} \{0,1\}^\lambda$) with probability at least $\varepsilon_\mathcal{A}$, it must query H on p_{N-1}^* with at least probability $\varepsilon_\mathcal{A}$. Further, again we have that up to the point where $\mathcal{A}$ queries p_{N-1}^*, the adversary perfectly simulates the selective secuirty game for PPRFs. The adversary $\mathcal{B}$ thus proceeds by guessing a query q_i for $i \xleftarrow{\$} [Q(\lambda)]$ and for this query outputting $\mathsf{H}(q_i)$ to its own experiment. Note that $\mathcal{B}$ succeeds if indeed $q_i = p_{N-1}^*$. We thus obtain that $\mathcal{B}$ succeeds with probability at least

$$\Pr[\mathcal{B} \text{ breaks UPF}] \geq \frac{\epsilon_\mathcal{A}(\lambda)}{|X_\lambda| \cdot Q(\lambda)},$$

which concludes the proof. □

Corollary 6.7 *Let* $\mathsf{H}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ *be modeled as a random oracle. Then, there exists a puncturable pseudorandom function over domain* $[N]$ *and range* $\{0,1\}^\lambda$*, which requires* $1.5N$ *calls to the random oracle for a full evaluation.*

Instantiating the construction in Figure 17 with the candidate construction of Conjecture 6.4 we obtain the following.

Corollary 6.8 *Let* $\mathsf{H}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$ *be modeled as a random permutation (with publicly accessible inverse). If Conjecture 6.4 is true, then there exists a puncturable pseudorandom function over domain* $[N]$ *and range* $\{0,1\}^\lambda$*, which requires* $1.5N$ *calls to the random permutation for a full evaluation.*

6.4 From UPF to Silent OT Extension

In this section we show that for some applications a strong unpredictable pseudorandom functions can serve as a replacement for a puncturable pseudorandom functions, thereby reducing the costs required for a full evaluation from $2N$ to N (where N is the domain size).

More precisely, we show that the protocol for silent OT extension of [BCG+19b, BCG+19a], which allows to generate n instances of random OT with sublinear communication complexity, can directly be instantiated with a strong UPF instead of a PPRF.

Construction G_{OT}

PARAMETERS:

- Security parameter 1^λ, integers $N > n$, $q = p^r = \lambda^{\omega(1)}$, and noise weight t.
- A code generation algorithm $\mathbf{C}$ and $H_{n,N} \xleftarrow{\$} \mathbf{C}(n, N, \mathbb{F}_p)$ returning matrices in $\mathbb{F}_p^{n \times N}$.
- A UPF scheme $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ over domain $[N]$ and range $\mathbb{F}_q$.
- A hash function $\mathsf{H}\colon [n] \times \mathbb{F}_q \to \{0,1\}^\lambda$ modeled as a random oracle.

CORRELATION: Outputs $(R_0, R_1) = \big(\{(u_i, w_{i,u_i})\}_{i\in[n]}, \{w_{i,j}\}_{i\in[n],j\in[p]}\big)$, where $w_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$ and $u_i \xleftarrow{\$} \{1, \ldots, p\}$, for $i \in [n], j \in [p]$.

GEN:

- Pick a random size-t subset $S = \{s_1, \ldots, s_t\}$ of $[N]$, sorted in increasing order.
- Pick a random vector $\vec{y} = (y_1, \ldots, y_t) \in (\mathbb{F}_p^*)^t$ and $x \xleftarrow{\$} \mathbb{F}_q$.
- For $i \in [t]$ proceed as follows:
  - Draw $k_i \leftarrow \mathsf{Setup}(1^\lambda)$.
  - Compute $k_i^* \leftarrow \mathsf{Puncture}(k_i, s_i)$.
  - Compute $\Delta_i := \mathsf{Eval}(k, s_i) - y_i \cdot x$.
  - Set $K_0^i := (k_i^*, \Delta_i)$ and $K_1^i := k_i$.
- Let $\mathsf{k}_0 := (n, N, \{K_0^i\}_{i\in[t]}, S, \vec{y})$ and $\mathsf{k}_1 := (n, N, \{K_1^i\}_{i\in[t]}, x)$.
- Output $(\mathsf{k}_0, \mathsf{k}_1)$.

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

1. If $\sigma = 0$, parse k_0 as $(n, N, \{K_0^i\}_{i\in[t]}, S, \vec{y})$ and proceed as follows:
   - Define $\vec{\mu} \in \mathbb{F}_p^N$ to be the vector with
   $$\mu_j := \begin{cases} y_i & \text{if } j = s_i \text{ for some } i \in [t] \\ 0 & \text{else} \end{cases}.$$
   - For all $i \in [t]$, define $\vec{v}_0^i \in \mathbb{F}_p^N$ to be the vector with
   $$v_{0,j}^i := \begin{cases} \mathsf{PuncEval}(k_i^*, s_i, j) & \text{if } j \neq s_i \\ \Delta_i & \text{else} \end{cases}.$$
   - Set $\vec{u} := H_{n,N} \cdot \vec{\mu} \in \mathbb{F}_p^n$ and $\vec{v}_0 := H_{n,N} \cdot \left(\sum_{i=1}^t \vec{v}_0^i\right) \in \mathbb{F}_q^n$.
   - Compute $w_i' := \mathsf{H}(i, v_{0,i})$ for $i = 1, \ldots, n$ and output $\{(u_i, w_i')\}_{i,j\in[n]}$.
2. If $\sigma = 1$, parse k_1 as $(n, N, \{K_1^i\}_{i\in[t]}, x)$ and proceed as follows:
   - Define $\vec{v}_1^i \in \mathbb{F}_p^N$ to be the vector with $v_{1,j}^i := \mathsf{Eval}(k_i, j)$ for all $i \in [t]$.
   - Compute $\vec{v}_1 := H_{n,N} \cdot \left(\sum_{i=1}^t v_1^i\right) \in \mathbb{F}_q^n$.
   - Compute $w_{i,j} := \mathsf{H}(i, v_{1,i} - j \cdot x)\ \forall i \in [n], j \in \mathbb{F}_p$.
   - Output $\{w_{i,j}\}_{i\in[n],j\in[p]}$.

Figure 18: PCG for n sets of 1-out-of-p random OT

Theorem 6.9 *Suppose* $\mathsf{H}\colon [n] \times \mathbb{F}_q \to \{0,1\}^\lambda$ *is a hash function modeled as a random oracle, the* $(\mathsf{HW}_t^N, \mathbf{C}, \mathbb{F}_p)$-dual-$\mathsf{LPN}(n, N)$ *assumption holds, and* $\mathsf{UPF} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{PuncEval})$ *is a UPF over domain* $[N]$ *and range* $\mathbb{F}_q$ *that satisfies strong unpredictability relative to oracle* $\tilde{\mathsf{H}}$*. Then the silent OT construction* G_{OT} *(Fig. 18) is a secure PCG for the random* 1*-out-of-*p *OT correlation.*

Proof. *Correctness.* We first have to show that the correlation (R_0, R_1) obtained via the PCG is computationally indistinguishable from (R_0, R_1) sampled truly at random conditioned on satisfying the OT correlation.

First, note that $\vec{\mu}$ is sampled as a random vector over $\mathbb{F}_p^N$ conditioned on having exactly t non-zero entries, which corresponds exactly to the distributed HW_t. The choice vector $\vec{u} = H_{n,N} \cdot \vec{\mu} \in \mathbb{F}_p^n$ is thus computationally indistinguishable to a truly random vector by the $(\mathsf{HW}_t^N, \mathbf{C}, \mathbb{F}_p)$-dual-$\mathsf{LPN}(n, N)$ assumption.

It is left to show that the following holds:

- $\{w_{i,j}\}_{i\in[n],j\in[p]}$ are computationally indistinguishable from values drawn uniformly at random from $\{0,1\}^\lambda$, and
- for all $i \in [n]$: $w'_i = w_{i,u_i}$.

The former is a straightforward consequence from assuming H is modeled as a random oracle. If $x \neq 0$, then we have $v_i - j \cdot x \neq v_i - j' \cdot x$ for all $j \neq j' \in \mathbb{F}_p$. Since additionally i is an input to H, we have that $\{w_{i,j}\}_{i\in[n],j\in[p]}$ corresponds to the output on H on $n \cdot p$ pairwise distinct values (unless $x = 0$) and is therefore even perfectly indistinguishable from chosen uniformly at random. Since by assumption we have $q = \lambda^{\omega(1)}$, the case $x = 0$ only occurs with negligible probability, therefore yielding the required.

Further, by the correctness of UPF, for all $i \in \mathbb{F}_p$, $j \neq s_i$ we have that

$$v^i_{0,j} = \mathsf{PuncEval}(k^*_i, s_i, j) = \mathsf{Eval}(k_i, j) = v^i_{1,j}$$

as well as

$$v^i_{0,s_i} = \Delta_i = \mathsf{Eval}(k_i, s_i) - y_i \cdot x = v^i_{1,s_i} - \mu_{s_i} \cdot x.$$

This implies

$$\sum_{i=1}^{t} \vec{v}^i_0 = \sum_{i=1}^{t} \vec{v}^i_1 - \vec{\mu} \cdot x,$$

since $\mu_j = 0$ for all $j \in [N] \backslash S$. Altogether, we obtain

$$\vec{v}_0 = \vec{v}_1 - \vec{u} \cdot x,$$

which implies

$$w'_i = \mathsf{H}(i, v_{0,i}) = \mathsf{H}(i, v_{0,i} - u_i \cdot x) = w_{i,u_i}$$

as required.

Security. First, consider the case (k_0, R_1). Here, in the real distribution, the adversary is given a key $\mathsf{k}_0 = (n, N, \{K^i_0\}_{i\in[t]}, S, \vec{y})$ as well as the expanded output $R_1 = \{w_{i,j}\}_{i\in[n],j\in[p]}$, where $w_{i,j} = \mathsf{H}(i, v_{1,i} - j \cdot x)$. We need to show that this is indistinguishable from the ideal distribution, where $R_1 \xleftarrow{\$} \mathsf{RSample}(0, R_0)$.

Recall that $\mathsf{RSample}(0, R_0)$ proceeds by setting $w_{i,u_i} := w'_i$ (where w'_i is as specified in $\mathsf{Expand}(0, k_0)$) and sampling the remaining values $\{w_{i,j}\}_{i\in[n],j\in[p]\backslash\{u_i\}}$ uniformly at random. By correctness, we have $w_{i,u_i} := w'_i$, it is thus left to show that the remaining values are distributed uniformly at random from the view of the adversary, even given k_0.

We first assume $x \neq 0$, which happens except with probability $\frac{1}{q}$. Further, for simplicity we will assume the t UPF instances to be t independent instances punctured at a random position s_i. Note that in typical instances we have $t \ll \sqrt{N}$ and therefore expect no collision to occur for positions $s_1, \ldots, s_t$ chosen uniformly at random (instead of uniformly at random conditioned on being distinct). For the general case, note that we can consider the UPF punctured at a chosen instead of random point add the cost of an additional loss of factor N in the bad event of the strong unpredictability definition.

Let $\mathsf{bad} = \mathsf{bad}_\lambda$ be the bad event from the strong unpredictability definition relative to UPF. Then, by assumption there exists a negligible function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that

$$\Pr[\mathsf{bad}_\lambda] \leq \mathsf{negl}(\lambda)$$

for all $\lambda \in \mathbb{N}$. By a union bound, we thus obtain that except with probability at most $t \cdot \mathsf{negl}(\lambda)$, it holds that

$$\Pr\underbrace{\left[\bigvee_{i=1}^{t} \mathsf{bad}_\lambda^i\right]}_{=:\mathsf{bad}'_\lambda} \leq t \cdot \mathsf{negl}(\lambda),$$

where by bad_λ^i we denote the bad_λ event relative to the i-th instance of the UPF.

Thus, we have that except with negligible probability, $\mathsf{Eval}(k_i, s_i)$ is strongly unpredictable for all $i \in [t]$, i.e.,

$$\Pr[\mathcal{A}^{\tilde{\mathsf{H}}(\cdot)}(1^\lambda, k_i^*, s_i) = \mathsf{Eval}(k_i, s_i) \mid \neg\mathsf{bad}_\lambda^i] \leq \frac{1}{q - Q_{\tilde{\mathsf{H}}}(\lambda)}.$$

Recall that $\Delta_i = \mathsf{Eval}(k, s_i) - y_i \cdot x$. Since $x \xleftarrow{\$} \mathbb{F}_q$ is chosen uniformly at random, the above implies

$$\Pr[\mathcal{A}^{\tilde{\mathsf{H}}(\cdot)}(1^\lambda, k_i^*, s_i, \Delta_i, y_i) = x \mid \neg\mathsf{bad}_\lambda^i \wedge x \neq 0] \leq \frac{1}{q - Q_{\tilde{\mathsf{H}}}(\lambda)}.$$

Phrased differently, for each i the adversary can rule out at most $Q_{\tilde{\mathsf{H}}}(\lambda)$ out of q potential values for x. Altogether, the adversary can thus rule out at most $t \cdot Q_{\tilde{\mathsf{H}}(\lambda)}$ out of q potential values for x. This implies

$$\Pr[\underbrace{\mathcal{A}^{\tilde{\mathsf{H}}(\cdot)}(1^\lambda, \{k_i^*, s_i, \Delta_i, y_i\}_{i\in[t]}) = x}_{=:\widetilde{\mathsf{bad}}_\lambda} \mid \mathsf{bad}'_\lambda \wedge x \neq 0] \leq \frac{1}{q - t \cdot Q_{\tilde{\mathsf{H}}}(\lambda)}.$$

Since x is independent from all other values of the protocol, we obtain that the probability that $\mathcal{A}$ queries H on $(i, v_{1,i} - j \cdot x)$ for some $j \neq s_i$ if $\widetilde{\mathsf{bad}}_\lambda$ does not occur is bounded by

$$\Pr[\underbrace{\mathcal{A} \text{ queries } \mathsf{H} \text{ on } (i, v_{1,i} - j \cdot x) \text{ for } j \neq s_i}_{=:\mathsf{bad}^{i,j}} \mid \neg\mathsf{bad}'_\lambda \wedge \neg\widetilde{\mathsf{bad}}_\lambda \wedge x \neq 0] \leq \frac{Q_{\mathsf{H}}(\lambda)}{q},$$

where Q_{H} is the number of queries by $\mathcal{A}$ to H.

Altogether, we obtain that $\{w_{i,j}\}_{i\in[n],j\in[p]\setminus\{s_i\}}$ is distributed perfectly uniformly at random from the view of $\mathcal{A}^{\tilde{\mathsf{H}}(\cdot),\mathsf{H}(\cdot)}(1^\lambda, \mathsf{k}_0)$ except with probability at most

$$\begin{aligned}
&\Pr[x = 0] + \Pr[\mathsf{bad}'_\lambda] + \Pr[\widetilde{\mathsf{bad}}_\lambda \mid \mathsf{bad}'_\lambda \wedge x \neq 0] \\
&+ \Pr\left[\bigvee_{i\in[n],j\in[p]\setminus\{s_i\}} \mathsf{bad}^{i,j} \;\middle|\; \neg\mathsf{bad}'_\lambda \wedge \neg\widetilde{\mathsf{bad}}_\lambda \wedge x \neq 0\right] \\
&\leq \frac{1}{q} + t \cdot \mathsf{negl}(\lambda) + \frac{1}{q - t \cdot Q_{\tilde{\mathsf{H}}}(\lambda)} + \frac{n \cdot (p-1) \cdot Q_{\mathsf{H}}(\lambda)}{q},
\end{aligned}$$

which is negligible as required.

It is left to consider the case (k_1, R_0). Here, in the real distribution the adversary is given a key $\mathsf{k}_1 = (m, n, \{K_1^i\}_{i\in[t]}, x)$ as well as the expanded output $\{(u_i, w'_i)\}_{i,j\in[n]}$, where $\vec{u} = H_{n,N} \cdot \vec{\mu}$ and $w'_i = w_{i,u_i}$. Therefore, it suffices to show that the distribution of $\vec{u}$ is computationally indistinguishable from uniformly at random even given k_1. First note that k_1 does not leak any information about $S, \vec{y}$, since the distribution of $k_i \leftarrow \mathsf{Setup}(1^\lambda)$ is independent of s_i, y_i. It therefore follows that $\vec{u}$ is distributed computationally indistinguishable from uniformly at random by the $(\mathsf{HW}_t^N, \mathbf{C}, \mathbb{F}_p)$-dual-$\mathsf{LPN}(n, N)$ assumption, as in the proof of correctness.

□

Together with the two-round distributed setup for a PPRF scheme of [BCG+19a] which directly applies to our UPF constructions given in 6.2, we obtain the following corollary, reducing the random oracle calls to $\tilde{\mathsf{H}}$ from $2 \cdot t \cdot n$ to $t \cdot n$ per party.

Corollary 6.10 *Assuming the $(\mathsf{HW}_t^N, \mathbf{C}, \mathbb{F}_p)$-dual-$\mathsf{LPN}(n, N)$ assumption, there exists a semi-honest 2-round OT extension with silent preprocessing for generating n 1-out-of-p OTs, which makes $o(n)$ black-box uses of a 2-round semi-honest 1-out-of-2 OT, and $t \cdot n$ black-box calls to a random oracle $\tilde{\mathsf{H}} \cdot \{0,1\}^\lambda \to \{0,1\}^\lambda$ and $n \cdot (p-1)$ calls to a random oracle $\mathsf{H} \cdot [n] \times \mathbb{F}_q \to \{0,1\}^\lambda$ per party.*

7 Acknowledgements

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

[AHI+17] Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In *ITCS 2017*, January 2017.

[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.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT 2011*, May 2011.

[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.

[BMS08] Louay Bazzi, Mohammad Mahdian, and Daniel A Spielman. The minimum distance of turbo-like codes. *IEEE Transactions on Information Theory*, 55(1):6–15, 2008.

[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.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *EUROCRYPT 2020, Part I*, May 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.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *ITC 2021*, 2021.

[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.

[LP04] Carlos A León and François Perron. Optimal hoeffding bounds for discrete reversible markov chains. *The Annals of Applied Probability*, 14(2):958–970, 2004.

[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.

[MZRA21] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. Incremental offline/online PIR (extended version). *IACR Cryptol. ePrint Arch.*, page 1438, 2021.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO 2012*, August 2012.

[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.

[Shp09] Amir Shpilka. Constructions of low-degree and error-correcting ε-biased generators. *computational complexity*, 18(4):495, 2009.

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

[TZ06] Jean-Pierre Tillich and Gilles Zémor. On the minimum distance of structured ldpc codes with two variable nodes of degree 2 per parity-check equation. In *2006 IEEE International Symposium on Information Theory*, pages 1549–1553. IEEE, 2006.

[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.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *42nd IEEE Symposium on Security and Privacy, SP 2021*, pages 1074–1091, 2021.

[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.

[ZWR] Wangda Zhang, Yanbin Wang, and Kenneth A Ross. Parallel prefix sum with simd. *Algorithms*, 5:31.