BKW Meets Fourier
New Algorithms for LPN with Sparse Parities

Dana Dachman-Soled[1⋆], Huijing Gong[2⋆⋆], Hunter Kippen[1⋆⋆⋆], and Aria Shahverdi[1]

[1] University of Maryland, College Park, USA
{danadach, hkippen, ariash}@umd.edu
[2] Intel Labs
huijing.gong@intel.com

Abstract. We consider the Learning Parity with Noise (LPN) problem with sparse secret, where the secret vector $\mathbf{s}$ of dimension n has Hamming weight at most k. We are interested in algorithms with asymptotic improvement in the *exponent* beyond the state of the art. Prior work in this setting presented algorithms with runtime $n^{c \cdot k}$ for constant $c < 1$, obtaining a constant factor improvement over brute force search, which runs in time $\binom{n}{k}$. We obtain the following results:

– We first consider the *constant* error rate setting, and in this case present a new algorithm that leverages a subroutine from the acclaimed BKW algorithm [Blum, Kalai, Wasserman, J. ACM '03] as well as techniques from Fourier analysis for p-biased distributions. Our algorithm achieves asymptotic improvement in the exponent compared to prior work, when the sparsity $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, where $c \in o(\log\log(n))$ and $c \in \omega(1)$. The runtime and sample complexity of this algorithm are approximately the same.

– We next consider the *low noise* setting, where the error is subconstant. We present a new algorithm in this setting that requires only a *polynomial* number of samples and achieves asymptotic improvement in the exponent compared to prior work, when the sparsity $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$ and noise rate of $\eta \neq 1/2$ and $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$, for $f(n) \in \omega(1) \cap n^{o(1)}$. To obtain the improvement in sample complexity, we create subsets of samples using the *design* of Nisan and Wigderson [J. Comput. Syst. Sci. '94], so that any two subsets have a small intersection, while the number of subsets is large. Each of these subsets is used to generate a single p-biased sample for the Fourier analysis step. We then show that this allows us to bound the covariance of pairs of samples, which is sufficient for the Fourier analysis.

⋆ Supported in part by NSF grants #CNS-1933033, #CNS-1453045 (CAREER), and by financial assistance awards 70NANB15H328 and 70NANB19H126 from the U.S. Department of Commerce, National Institute of Standards and Technology.

⋆⋆ Most of the work was done while the author was a student at the University of Maryland, College Park.

⋆⋆⋆ Supported in part by the Clark Doctoral Fellowship from the Clark School of Engineering, University of Maryland, College Park.

– Finally, we show that our first algorithm extends to the setting where the noise rate is very high $1/2 - o(1)$, and in this case can be used as a subroutine to obtain new algorithms for learning DNFs and Juntas. Our algorithms achieve asymptotic improvement in the exponent for certain regimes. For DNFs of size s with approximation factor ϵ this regime is when $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log \log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, for $c \in n^{1-o(1)}$. For Juntas of k the regime is when $k \in \omega\left(\frac{c}{\log n \log \log c}\right)$, and $k \in n^{1-o(1)}$, for $c \in n^{1-o(1)}$.

1 Introduction

The *(search) Learning Parity with Noise (LPN)* problem with dimension n and noise rate η, asks to recover the secret parity $\mathbf{s}$, given samples $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e)$, where $\mathbf{x} \in \{0,1\}^n$ is chosen uniformly at random, $\mathbf{s} \in \{0,1\}^n$, error $e \in \{0,1\}$ is set to 1 with probability η and 0 with probability $1 - \eta$, and the dot product is taken modulo 2.

While solving a linear system of n equations over $\mathbb{F}_2$ to recover a secret of dimension n can be done in polynomial time via Gaussian elimination, even adding a small amount of noise e renders the above a seemingly hard learning problem, even given a large number of samples. Specifically, the search LPN problem, which typically assumes the noise rate is a small constant, is believed to be hard, with the asymptotically best algorithm (known as BKW) requiring runtime $2^{\Theta(n/\log(n))}$ and $2^{\Theta(n/\log(n))}$ number of samples to recover $\mathbf{s}$ of dimension n. Some evidence of its hardness comes from the fact that it provably cannot be learned efficiently in the so called *statistical query (SQ)* model under the uniform distribution [3,5].

Though originally arising in the fields of computational learning theory and coding theory, the LPN problem has found numerous applications in cryptography (see e.g. [4,17,18,13] for a partial list of applications) due to the fact that (1) there is a search-to-decision reduction, meaning that the decision version—which is more amenable to cryptographic applications and asks to distinguish $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e)$ from $(\mathbf{x}, b)$, where b is random—is as hard as the search version (which asks to recover $\mathbf{s}$) and (2) the LPN problem is believed to be *quantum-hard*, as opposed to other standard cryptographic assumptions such as discrete log and factoring which are known to have polynomial time quantum algorithms [26].

Variants of the LPN problem have also been considered in the literature: Sparse LPN [6], where the $\mathbf{x}$ vectors in the LPN problem statement are sparse, LPN with structured noise, where the noise across multiple samples is guaranteed to satisfy some constraint [2], and Ring LPN [16]. While typically the error rate is assumed to be constant, LPN with low noise rate has also been considered with applications to cryptography [8]. Indeed, LPN with noise rate even as low as $\Omega(\log^2(n)/n)$ is considered a hard problem [8]. We further note that WLOG can assume that the secret is drawn from the same distribution as the noise, as there is a reduction from LPN with secret $\mathbf{s}$ to LPN with secret $\mathbf{e}$, where $\mathbf{e}$ is the error vector obtained after n samples are drawn [1].

In this work we consider LPN with sparse parities (i.e. the "sparsity" or Hamming weight k of the secret vector is significantly less than $\eta \cdot n$, where η is the error rate). We consider both the constant noise and the low noise setting (where the error rate is subconstant). Motivations for considering this variant of LPN include the fact that sparse secrets may be used in practical cryptosystems for efficiency purposes (as is the case for some fully homomorphic encryption implementations [9]), or some bits of the secret may be leaked via a side-channel attack. More generally, analyzing the security of LPN with sparse parities tests the robustness of the standard LPN assumption, since a lack of polynomial-time algorithms in the sparse parities setting (when k is super-constant) would then raise our confidence in the security of the standard setting. We also consider applications of our results to other learning problems, such as learning DNFs and Juntas. Prior work on LPN with sparse parities, has mainly considered obtaining algorithms with runtime $n^{c \cdot k}$ for constant $c < 1$ [14,27]. This beats the trivial brute force search with runtime $\binom{n}{k}$ in the regime where $k \ll n$. In this work, our focus is to achieve an algorithm which, for certain regimes of k, beats the prior best algorithms asymptotically *in the exponent.* Since our goal is to achieve asymptotic improvement in the exponent, we will compare our algorithm's runtime against brute force search and not the prior work of [14,27], since the latter algorithms are equivalent to brute force search in terms of asymptotics in the exponent.

1.1 Our Results

We obtain new LPN algorithms for sparse parities that improve upon the state-of-the-art in certain regimes, which will be discussed below.

Our first result pertains to the constant noise setting, where the noise rate $\eta \in \Theta(1)$. In the theorem below, $p \in (0, 1)$ is a free parameter that we set later to optimize our runtime.

Theorem 1.1. *For $\delta \in [0, 1]$, $p \in (0, 1)$, LPN for parities of sparsity k out of n variables and constant noise rate can be learned with total number of samples and total computation time of*

$$\mathsf{poly}\left(\frac{1}{(1-2\eta)^{\sqrt{np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{n}{\delta}) \cdot \left(2^{\frac{np}{\log(np)}} \cdot \log(np)\right)\right),$$

and success probability of $1 - \delta - \left(\frac{16}{(1-2\eta)^{\sqrt{8np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{2n}{\delta}) \cdot \exp(\frac{-pn}{8})\right)$.

By setting the parameter p appropriately, we obtain the following:

Corollary 1.2. *For sparisty $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, where $c \in o(\log\log(n))$ and $c \in \omega(1)$, the runtime of our new learning algorithm is contained in both $\log(n)^{o(k)}$ and $2^{o(n/\log(n))}$, and it succeeds with constant probability. For this range of k, Brute Force search requires runtime $\log(n)^{\Omega(k)}$ and* BKW *requires runtime of $2^{\Omega(n/\log(n))}$.*

Our second result pertains to the low noise setting, where the noise rate $\eta \in o(1)$. Again, $p \in (0, 1)$ is a free parameter that we set later to optimize our runtime.

Theorem 1.3. *Assuming parameters are set such that*

$$\log\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \in o(1/\eta \cdot \log(np)),$$

and that $\delta \in [0, 1]$, $p \in (0, 1)$, LPN for parities of sparsity k out of n variables and noise rate $\eta \in o(1)$ can be learned using $(2np + 1)^2 \cdot \log(n)$ number of samples, total computation time of $N := \mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and achieves success probability of

$$1 - \delta - \left(N \cdot \left(2 \cdot \exp(-p \cdot n/8) + \exp(-n/48) + 1/2^{np/4}\right)\right)$$

By setting the parameter p appropriately, we obtain the following:

Corollary 1.4. *For sparsity $k(n)$ such that $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$, noise rate $\eta \neq 1/2$ such that $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$, for $f(n) \in \omega(1) \cap n^{o(1)}$, the Learning Algorithm of Figure 4 runs in time $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2k}} \cdot \log(n) \cdot (np)^3\right) \in \left(\frac{n}{k}\right)^{o(k)}$ with constant probability. In this setting, the running time Brute Force is $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ and the running time of Lucky Bruteforce is $e^{\eta n} \in \left(\frac{n}{k}\right)^{\omega(k)}$.*

Finally, applying known reductions to LPN [12] and solving LPN using our algorithm, we also obtain applications to learning other classes of functions such as DNF and juntas:

– Our algorithm can be applied to learn DNFs of size s and approximation factor ϵ, with asymptotic improvements over Verbeurgt's bound [28] of $O\left(n^{\log \frac{s}{\epsilon}}\right)$, and with negligible failure probability when $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.
– Our algorithm can be applied to learn Juntas of size k with a runtime of $n^{o(k)}$ and a negligible failure probability when $k \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $k \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.

1.2 Technical Overview

Fourier Analysis of Boolean Functions. Every Boolean function, $f : \{0,1\}^n \to \{0,1\}$—equivalently $f : \{-1,1\}^n \to \{-1,1\}$—can be represented as a linear combination $f(\mathbf{x}) = \sum_{S \subseteq [n]} \hat{f}(S) \cdot \chi_{S,p}(\mathbf{x})$, known as the Fourier representation of f. Typically, we consider the uniform distribution over examples $\mathbf{x}$, in which case $\chi_{S,p}(\mathbf{x})$ is defined as $\prod_{j \in S} \mathbf{x}[j]$ and $\hat{f}(S) = \mathbb{E}_{\mathbf{x} \sim \{-1,1\}^n}[f(\mathbf{x}) \cdot \chi_{S,p}(\mathbf{x})]$. However, for any product distribution $[p_1, \ldots, p_n]$, where $\mathbb{E}[\mathbf{x}[j]] = p_j$, we can also define

$\chi_{S,p}(\mathbf{x}) := \prod_{j \in S} \frac{\mathbf{x}[j]-p_j}{\sqrt{1-p_j^2}}$ and $\hat{f}(S) := \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_p}\left[f(\mathbf{x}) \cdot \chi_{S,p}(\mathbf{x})\right]$, where $\mathcal{D}_p$ is a product distribution defined over $\{-1,1\}^n$ and is parameterized by its mean vector $[p_1, \ldots, p_n]$. Fourier analysis is a strong tool in computational learning theory for learning under the uniform distribution (and can be extended to product distributions as well). Specifically, the Low Degree Algorithm of [20] guarantees that if most of the Fourier weight of a Boolean function is concentrated on low degree parities (i.e. $\chi_{S,p}$ with small $|S|$), then an approximate version of the function can be reconstructed, *even in the presence of noise.* However, for learning large parities under the uniform distribution Fourier analysis is not useful since for a parity corresponding to secret $\mathbf{s}$ of Hamming weight k, all of the Fourier weight is on a single Fourier coefficient of degree k and searching for this Fourier coefficient would require a brute force search that enumerates over all possible parities of size at most k. If the distribution is p-biased instead of uniform, however, then the above is no longer the case. Specifically, if we consider a product distributions where the example $\mathbf{x}$ is no longer uniformly random, but each coordinate of $\mathbf{x}$ is set to 0 with probability $1/2+p/2$ and 1 with probability $1/2-p/2$ (so the expectation $\mathbb{E}[x[j]] = p$ for each coordinate of $\mathbf{x}$), then the Fourier weight is now spread over all parities S such that $\forall j \in S, \mathbf{s}[j] = 1$. In particular, this means that by approximately computing the Fourier coefficient of all subsets consisting of a single element $S = \{\mathbf{s}[1]\}, \ldots, S = \{\mathbf{s}[n]\}$, we can distinguish the subsets of size 1 with non-zero versus zero Fourier weight and thus determine all i such that $\mathbf{s}[j] = 1$. We note that when the distribution is p-biased, the magnitude of the Fourier coefficients that we must approximate is of the order p^k, and we will therefore require $\mathsf{poly}((1/p)^k)$ samples to approximate the quantity (even without considering noise). We will see in the following that in order for our approach to improve upon known algorithms, we must consider *sparse parities* with $k \in o(n)$.

Attack Overview. Given the above discussion, the main idea of our attack is to convert samples drawn from the uniform distribution to samples drawn from a p-biased distribution and then use Fourier analysis techniques to learn the elements of the parity one by one.

In order for this approach to succeed, our algorithm first needs to generate a sufficient number of p-biased LPN samples, given uniformly random LPN samples. Specifically, the attacker has access to unbiased LPN oracle which outputs samples $\mathbf{x}_i$ and corresponding label b_i such that $b_i = \langle \mathbf{x}_i, \mathbf{s} \rangle + e_i$, noise e_i has rate η meaning that error e_i is 1 with probability η and 0 with probability $1-\eta$. The attacker will generate new samples $\mathbf{x}'_i$, which are p-biased, and a corresponding label b'_i, with a higher error rate η'. We then approximate the Fourier coefficient of coordinate j, constructed as above, by $\hat{b}_p(\{j\}) := \mathbb{E}_{\mathbf{x}' \sim \mathcal{D}_p}[b' \cdot \chi_{\{j\},p}(\mathbf{x}')]$. The main observation is that for the secret key coordinate j such that $\mathbf{s}[j] = 0$ we have $\hat{b}_p(\{j\}) = 0$ and for the coordinates j such that $\mathbf{s}[j] = 1$ we have $\hat{b}(\{j\}) = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$. The value of $\hat{b}_p(\{j\})$ is estimated by using a sample mean with a sufficient number of generated p-biased samples to approximate the expectation.

We present two algorithms for generating the p-biased samples, each algorithm is appropriate for a different scenario. Specifically, our first algorithm is appropriate for the standard case where the noise rate is constant, while our second algorithm is appropriate for the *low noise* case where the noise rate is sub-constant. After generating the p-biased samples, the Fourier estimation step is similar in both settings. We next elaborate on our algorithm for each of the two settings.

Constant Noise. In the case where the noise rate is constant, to generate the p-biased samples, we apply a variant of the BKW algorithm. The BKW algorithm gives an $2^{O(n/\log(n))}$-time algorithm for the LPN problem that also requires $2^{O(n/\log(n))}$ number of samples. An intermediate step of the BKW algorithm uses access to its LPN oracle to generates samples $(\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle \oplus e')$, where $\mathbf{x}$ is a vector that has all 0's except in a single position, and e' is an error term with higher noise rate than the original error. The key idea of our algorithm is that in order to create p-biased samples, we can choose a random set of coordinates, $R \subseteq [n]$, by including each $i \in [n]$ in the set R independently with probability p, and then run the subroutine of the BKW algorithm on the *smaller set* R, of expected size pn, in order to create a sample $\mathbf{x}$ that is set to 0 for all $i \in R$. Such a sample $\mathbf{x}$ is now distributed identically to a p-biased sample. The error rate increases, but since Fourier analysis is robust against noise, these p-biased samples can still be used to estimate the Fourier Coefficients corresponding to $S = \{\mathbf{s}[1]\}, \ldots, S = \{\mathbf{s}[n]\}$ to determine the secret $\mathbf{s}$. Crucially, our algorithm gains over simply running BKW on the entire instance because the set of coordinates we run BKW on is of size $O(pn)$ instead of size n. Thus, generating the biased samples runs in time $2^{O(pn/\log(pn))}$ instead of time $2^{O(n/\log(n))}$. When p is subconstant, we achieve an asymptotic gain in the exponent. In contrast, the Fourier estimation step runs in time $\mathsf{poly}((1/p)^k)$, so we must also set p large enough so that this step achieves asymptotic gain in the exponent beyond the brute force search time of $\binom{n}{k}$. We discuss at the end of the section the regime in which it is possible to set the parameter p so that our algorithm improves asymptotically in the exponent beyond the best known algorithms.

Low Noise. When the noise rate is sufficiently low, we can generate p-biased samples using a simpler approach. As before, we randomly select a set $R \subseteq [n]$, by including each $i \in [n]$ in the set R independently with probability p. Now, instead of running BKW on the coordinates in the set R, we simply choose $O(np)$ number samples (since R has expected size np) from the non-biased oracle and find a linear combination (guaranteed to exist) that sets all the coordinates in R to 0. Again, the noise increases in the generated sample. Nevertheless, we gain over the trivial approach (which instead of p-biasing the oracle simply creates linear combinations that have $\mathbf{x}$ set to all 0 except for in a single coordinate) because the linear combination we generate is over at most $O(np)$ versus $O(n)$

vectors, which in turn guarantees that the noise rate will be lower.[3] We gain from this technique by choosing p small enough to lower the noise rate but large enough to ensure that the $(1/p)^k$ necessary to estimate the Fourier coefficient still beats brute force search asymptotically in the exponent.

In the low noise case we further show that we can generate the large number of samples needed for the Fourier analysis using only a *polynomial size* set of examples from the original LPN oracle. In this case, the generated samples will not be i.i.d., but we will use a construction inspired by the *designs* of Nisan and Wigderson to generate an exponentially large set of samples, where each pair of samples from the generated set has low covariance.[4] See Section 4.1 for more details. This will be enough to then run the Fourier analysis, which requires that one can use random sampling to estimate the mean of a random variable. We can bound the deviation from the mean using Chebyshev's inequality since we guarantee that the covariance between any two distinct samples is small.

Parameters. We now discuss the regime of k and η in which we improve on prior algorithms, and how to set the parameter p to achieve the optimal run time. For the constant noise setting, with secret $\mathbf{s}$ with sparsity in the form $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$, where $c \in o(\log\log(n))$ and $c \in \omega(1)$, we set $p = 1/\log^{1/(c)}(n)$ to obtain an algorithm that improves upon both Bruteforce and BKW asymptotically in the exponent. Recall that prior work on LPN with parities of sparsity k reduced the constant in the exponent beyond brute force, but did not achieve asymptotic improvement in the exponent. In our work we care about asymptotic improvement in the exponent and therefore do not compare against those algorithms. For the low noise setting we show that for sparsity $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$ and the noise rate of $\eta \neq 1/2$ and $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$, for $f(n) \in \omega(1) \cap n^{o(1)}$, by setting $p = \frac{1}{f(n)}$ and $\frac{1}{p} \in \left(\frac{n}{k}\right)^{o(1)}$, our algorithm improves upon both Bruteforce and "lucky Bruteforce"–i.e. an algorithm which gathers m samples until it has n noiseless samples with high confidence (where m depends on the noiserate) and then attempts Gaussian elimination with every possible subset of size n, giving runtime $\mathsf{poly}(\binom{m}{n})$–asymptotically in the exponent. To our knowledge, these are the best algorithms when considering asymptotics in the exponent.

Application to DNF and Juntas. In addition to parities, the reductions by Feldman et al. [12] provide a way to translate improvements in solving LPN to learning Juntas and DNFs. As such, we present a formulation of our constant noise algorithm that is parameterized according to these reductions, and provide parameter settings such that our algorithm, when applied to learning DNFs or

[3] We note that the above description is a bit inaccurate, since we must include an additional step to ensure that the added noise is independent of the set of samples. See discussion in Section 4.1, Figure 3 and Lemma 4.1 for more details.

[4] It is also possible to use a random choice of subsets in place of this design. However, the deterministic procedure allows for bounding the covariance of the newly generated samples which is crucial in our analysis as seen later.

Juntas, yields asymptotic improvements in the exponent. For DNFs, we present an asymptotic result similar to that of [14] in that we improve on Verbeurgt's bound of $O(n^{\log \frac{s}{\epsilon}})$ for learning DNFs of size s with approximation factor ϵ for a different regime of $\frac{s}{\epsilon}$, where $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, for $c \in n^{1-o(1)}$. Note that for Juntas, we present an algorithm that learns Juntas of k variables in $n^{o(k)}$ time for $k \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $k \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.

1.3 Related Work

LPN. Blum, Kalai and Wasserman [5] presented the first algorithm that improved upon the trivial $2^{\Omega(n)}$ time algorithm for LPN. They showed that LPN with constant error rate can be learned in slightly subexponential time $2^{O(n/\log n)}$ with the same amount of samples. To date, their algorithm remains the state-of-the-art in terms of asymptotics in the exponent in the constant error rate regime.

Lyubashevsky [22] extended the previous algorithm by Blum et al. [5] and reduced the overall sample complexity. Lyubashevsky developed an algorithm for creating a super-polynomial number of psuedorandom samples from a polynomial number of original samples. Thus, Lyubashevsky traded sample complexity for time complexity. More specifically, the algorithm solved LPN with constant error rate and parities of size n in time $2^{O(n/\log\log n)}$ using only $n^{1+\epsilon}$ samples.

In later work Bogos et al. [7] presented a unified framework for various improvements and optimizations of BKW. Specifically, they focused on tightening the analysis of several previous works [19,15] to give more accurate bounds for the time and sample complexity needed to solve the LPN problem. They improved the bounds of the variant of the BKW algorithm proposed by Leviel and Fouque [19] which is based on Walsh-Hadamard transform. Moreover, they analyzed the algorithm by Guo et al. [15] which used a "covering codes" technique to reduce the dimension of the problem. We note that the many of the improvements listed are heuristic in nature, while others provably improve the runtime. We also note that our usage of BKW in our algorithms is compatible with only some of these improvements. We only use the so-called "reduction" phase of the algorithm to generate our p-biased samples. Thus, improvements to this phase, such as covering codes, are applicable whereas others, such as the Walsh-Hadamard transform, are not.

LPN with sparse parities. Grigorescu et al. [14] showed an improvement of learning sparse parities with noise over brute force search, which has run time $\binom{n}{k}$. The algorithm ran in time $\mathsf{poly}\left(\log(\frac{1}{\delta}), \frac{1}{1-2\eta}\right) \cdot n^{\left(1+(2\eta)^2+o(1)\right)k/2}$ and had sample complexity of $\frac{k\log(n/\delta)\omega(1)}{(1-2\eta)^2}$ in the random noise setting under the uniform distribution. , where η is the noise rate and δ is the confidence parameter.

Valiant [27] showed that the learning parity with noise problem can be solved in time $\approx n^{0.8k}\mathsf{poly}(\frac{1}{1-2\eta})$. He also showed that noisy k-juntas can be learned in time $n^{0.8k}\mathsf{poly}\left(\frac{1}{1-2\eta}\right)$ and r-term DNF can be (ε, δ)-PAC learned in time $\mathsf{poly}\left(\frac{1}{\delta}, \frac{r}{\varepsilon}\right) n^{0.8\log(\frac{r}{\varepsilon})}$, respectively. We note that the improvements of Grigorescu et al. [14] and Valiant [27] do not improve upon the runtime of brute force search of n^k in terms of asymptotics in the exponent.

Learning DNF and Juntas. Mossel et al. [24] showed the first learning algorithm which achieves a polynomial factor improvement over trivial brute force algorithm which runs time $O(n^k)$. It shows that k-juntas can be learned in absence of noise with confidence $1-\delta$ from uniform random examples with run time of $\left(n^k\right)^{\frac{\omega}{\omega+1}} \cdot \mathsf{poly}\left(2^k, n, \log(1/\delta)\right)$ where $\omega < 2.376$ is the matrix multiplication exponent.

Feldman et al. [11] presented a foundational work for learning both DNFs and Juntas. They developed an oracle transformation procedure that enabled reductions from learning DNFs and Juntas to that of LPN. In addition, Feldman et al. presented a learning algorithm for agnostically learning parities by showing a reduction from learning parities with adverserial noise to learning parities with random noise. With this reduction, they showed that the algorithm by Blum et al. [5] can learn parities with an adverserial noise rate of η in time $O(2^{\frac{n}{\log n}})$. In a follow up work [12], Feldman et al. refined their reductions and included the influence of sample complexity on the the runtime. These reductions have streamlined the process of improving algorithms for learning DNFs and Juntas, as improved algorithms for learning parities can be directly applied to both problems. Both the work of Grigorescu et al. [14], and Valiant [27] were examples of this.

One can also consider natural restrictions to the Junta problem. For monotone Juntas, Dachman-Soled et al. [10] found lower bounds for solving monotone Juntas in the statistical query model. Lipton et al. considered the problem of learning symmetric Juntas [21] and showed they can be learned in $n^{o(k)}$ time. Note here that the symmetry requirement is orthogonal to restrictions on the size of k.

2 Preliminaries

2.1 Notations

In this section we remind the reader some of the preliminary results used throughout the paper. We use $:=$ as deterministic assignment and $\leftarrow$ as uniformly randomized assignment. We also use bold lowercase, e.g. $\mathbf{x}$, to denote vectors and bold uppercase, e.g. $\mathbf{A}$, to denote matrix. The set $\{1, 2, \ldots, n\}$ is often denoted by $[n]$. The i-th coordinate of vector $\mathbf{x}$ is denoted by $\mathbf{x}[i]$. For the vector $\mathbf{x}$ of dimension n and a set R that is a subset of $[n]$, we denote $\mathbf{x}|_R$ to be the restriction of $\mathbf{x}$ to the coordinates in R, namely $\mathbf{x}|_R = \mathbf{x}[i_1] \| \mathbf{x}[i_2] \| \ldots \mathbf{x}[i_{|R|}], \forall i \in R\}$. The indices in $\mathbf{x}$ are from 1 to n. For simplicity, we reset the indices in $\mathbf{x}|_R$ and have the indices from 1 to $|R|$.

2.2 Fourier Analysis

The boolean Fourier transform is defined for boolean functions defined over the domain $\{-1, 1\}$. Throughout the rest of the paper, when we discuss boolean functions, we will use this representation. To map a boolean function from $\{0, 1\} \in \mathbb{F}_2$ to $\{-1, 1\}$, we set $-1 := 1_{\mathbb{F}_2}$ and $1 := 0_{\mathbb{F}_2}$. We now present some additional notation regarding the representation of the LPN problem in the $\{-1, 1\}$ domain.

Notation. Assuming the LPN secret $\mathbf{s}$ is represented in $\mathbb{F}_2^n$, the following represent the boolean inner product of input $\mathbf{x}$ with $\mathbf{s}$ in different notation.

$$f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle \in \mathbb{F}_2 \text{ for } \mathbf{x}, \mathbf{s} \in \mathbb{F}_2^n$$
$$f_{\mathbf{s}}(\mathbf{x}) = \prod_{i=1}^{n} \mathbf{x}[i]^{\mathbf{s}[i]} \in \{-1, 1\} \text{ for } \mathbf{x} \in \{-1, 1\}^n \text{ and } \mathbf{s} \in \mathbb{F}_2^n$$

hence to represent a sample $(\mathbf{x}, b)$ from LPN oracle $\mathcal{O}_{0,\eta}^{\mathsf{LPN}}(\mathbf{s})$ we have the following two notations

$$b = f_{\mathbf{s}}(\mathbf{x}) + e \text{ for } \mathbf{x}, \mathbf{s} \in \mathbb{F}_2^n \text{ and } e \in \mathbb{F}_2$$
$$b = f_{\mathbf{s}}(\mathbf{x}) \cdot e \text{ for } \mathbf{x} \in \{-1, 1\}^n,\ \mathbf{s} \in \mathbb{F}_2^n \text{ and } e \in \{-1, 1\}$$

Consider a vector $\mathbf{x} \in \{-1, 1\}^n$. We denote by $\mathcal{D}_p$ the product distribution over $\{-1, 1\}^n$, where each bit of the vector is independent and has mean p.

Definition 2.1 (Fourier Expansion)**.** *For a product distribution $\mathcal{D}_p$ as above, every function $f : \{-1, 1\}^n \to \mathbb{R}$ can be uniquely expressed as the multilinear polynomial*

$$f(\mathbf{x}) = \sum_{S} \hat{f}_p(S)\chi_{S,p}(\mathbf{x}), \text{ where } \chi_{S,p}(\mathbf{x}) = \prod_{i \in S} \frac{\mathbf{x}[i] - p}{\sqrt{1 - p^2}}.$$

This expression is called the Fourier expansion of f with respect to $\mathcal{D}_p$, and the real numbers $\hat{f}(S)$ are called the Fourier coefficients of f on S.

The Fourier transform defines an inner product between two boolean functions f and g: $\langle f, g \rangle_p = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_p}[f(\mathbf{x}) \cdot g(\mathbf{x})]$. The Fourier coefficient for any $S \subset N$ over product distribution $\mathcal{D}_p$ is defined as follows:

$$\hat{f}_p(S) = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_p}[f(\mathbf{x}) \cdot \chi_S(\mathbf{x})].$$

Claim 2.2. Let $\mathsf{s}^p = (\mathbf{x}, b)$ be a p-biased sample and let $b = f_{\mathbf{s}}(\mathbf{x}) \cdot e$, where $e \in \{-1, 1\}$ is independent of $\mathbf{x}$ and $\mathbb{E}[e] = 1 - 2\eta'$. Define $\hat{b}_p(\{j\}) := \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_p}[b \cdot \chi_{\{j\},p}(\mathbf{x}))]$. If $\mathsf{s}^p.\mathbf{s}[j] = 0$, then $\hat{b}_p(\{j\}) = 0$. Whereas if $\mathsf{s}^p.\mathbf{s}[j] = 1$, then $\hat{b}_p(\{j\}) = (1 - 2\eta') \cdot p^{k-1}\sqrt{1 - p^2}$.

Proof. For the proof of the claim, refer to the full version of our paper available on ePrint. □

3 Constant Noise Setting

In the constant noise setting, our algorithm consists of two steps. First, using a modification of the acclaimed BKW algorithm [5], we implement a p-biased LPN Oracle with noise rate η' and secret value $\mathbf{s}$ which is denoted by $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ and is defined in Section A.2. We present this modification, entitled $\mathsf{BKW_R}$ (BKW restricted to set R), in subsection 3.1. In subsection 3.2 we present the integration of our p-biased oracle into the learning algorithm based on Fourier analysis. Finally, in subsections 3.3 and 3.4, we combine our analysis to present the regime in which we can set the free parameter p in order to improve on both BKW and brute force search asymptotically in the *exponent*.

3.1 $\mathsf{BKW_R}$

As a first step, we present our $\mathsf{BKW_R}$ algorithm in Figure 1. The $\mathsf{BKW_R}$ algorithm is given access to an unbiased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ and its goal is to produce a sample that is p-biased. The presented algorithm works similarly to BKW by successively taking linear combinations of samples to produce a sample with all zero entries one 'block' at a time. The algorithm accomplishes this by maintaining successive tables such that samples in each table are combined to fill the next table. The number of tables is a parameter of the algorithm denoted $\mathfrak{a}$. The tables $T^{(1)}, \ldots, T^{(\mathfrak{a})}$ are each of size $2^{\mathfrak{b}}$, where $\mathfrak{b}$ is the size of each block, except the last table $T^{(\mathfrak{a})}$ which might have a smaller number of entries, specifically $2^{|R| \bmod \mathfrak{b}}$. Each table $T^{(j)}$ is indexed by the value of the coordinates in the j-th block of $\mathbf{x}|_R$, namely $\mathbf{x}|_R\left[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1\right]$. The element in row i of table j is denoted by $\left[T^{(j)}_i\right]$. Importantly, while the size of R may vary, $\mathfrak{a}$ remains constant each time the algorithm is called. This ensures that a constant number of samples are combined to produce the output. This decouples the noise present in the output from the size of R, ensuring that all generated samples are independent.

Construction of p-biased Oracle given $\mathsf{BKW_R}$ The construction of the p-biased Oracle is quite simple. We sample an index set R where each index is selected independently with probability p. R is then passed as input to $\mathsf{BKW_R}$. By bounding the size of the set R, we can ensure that with overwhelming probability $\mathsf{BKW_R}$ outputs a p-biased sample in $2^{O(np/\log(np))}$ time. If the size of the set R exceeds this bound (captured by the event $\mathsf{Event1}$ occurring), the runtime may be longer. Thus, when we invoke $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ multiple times to generate a large number of p-biased samples for the Fourier analysis, we need to ensure that w.h.p. $\mathsf{Event1}$ never occurs. We bound the probability of $\mathsf{Event1}$ in Theorem 3.2.

Lemma 3.1. *The samples $(\mathbf{x}', b')$ outputted by $\mathsf{BKW_R}$ Algorithm with access to $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ are independent and distributed identically to samples drawn from a p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$.*

Proof. The proof can be found in Section A.4. □

Algorithm 1: $\mathsf{BKW_R}$

```
Result: Sample $(\mathbf{x}', b')$ such that the coordinates of $\mathbf{x}'$, which are defined by
        set $R$ are set to 0.
if $|R| \geq 2np \vee |R| \leq pn/2$ then
    $\mathsf{Event1}$ occurs.
end
Set $\mathfrak{a} := \lceil \log(2np)/2 \rceil$ and $\mathfrak{b} := \lceil |R|/\mathfrak{a} \rceil$;
Set $T^{(1)}, \ldots, T^{(\mathfrak{a})}$ to empty tables;
while True do
    Query a new sample from unbiased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ ;
    $j := 1$;
    while $j \leq \mathfrak{a}$ do
        if $\left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right] = \emptyset$ then
            $\left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right] := (\mathbf{x}, b)$;
            break;
        end
        if $\mathbf{x}|_R\left[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1\right] \neq 0$ then
            $(\mathbf{x}', b') := \left[T^{(j)}_{\mathbf{x}|_R[(j-1)\cdot\mathfrak{b}, j\cdot\mathfrak{b}-1]}\right]$;
            $\mathbf{x}'' := \mathbf{x} + \mathbf{x}', \quad b'' := b + b'$;
            $(\mathbf{x}, b) := (\mathbf{x}'', b'')$;
        end
        $j := j + 1$;
    end
    if $j = \mathfrak{a} + 1$ then
        break;
    end
end
$(\mathbf{x}', b') := (\mathbf{x}, b)$;
return $(\mathbf{x}', b')$;
```

Fig. 1. $\mathsf{BKW_R}$ "Zeroing" Algorithm

Theorem 3.2. *Given access to LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ which gives samples $\mathsf{s} = (\mathbf{x}, b)$, the oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ constructed from $\mathsf{BKW_R}$ requires $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ samples, and $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ runtime with probability at least $1 - 2\exp(-p \cdot n/8)$.*

Proof. The proof can be found in Section A.5. □

3.2 Learning Secret Coordinates

In this subsection we first present the Learning Algorithm in Figure 2. The Algorithm starts by sampling num number of samples from a p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$. As the samples are non-uniform, we can apply Fourier analysis technique described in Section 2.2.

```
The Learning Algorithm
The learning algorithm gets access to $p$-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ which
returns sample $\mathsf{s}^p = (\mathbf{x}, b)$.

1. Initialize $\mathcal{S}, \mathcal{S}' := \emptyset$
2. For $i \in \mathsf{num}$:
   (a) Set $\mathsf{s}_i^p \leftarrow \mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ to be the output sample from $p$-biased LPN Oracle.
   (b) Add $\mathsf{s}_i^p$ to the set $\mathcal{S}$.
3. Use the set $\mathcal{S}$ of $\mathsf{num}$ number of samples to estimate the Fourier coefficient of
   each coordinate of secret.
   – For each $j \in [n]$, approximate $\hat{b}_p(\{j\}) := \frac{1}{\mathsf{num}} \sum_{i=1}^{\mathsf{num}} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, where
     each coordinate of $\mathbf{x}_i, b_i$ is switched to $\{-1, 1\}$ from $\mathbb{F}_2$.
   – If $\hat{b}_p(\{j\}) > (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$, add $j$ to $\mathcal{S}'$.
4. Output $\mathbf{s}'$ such that $\mathbf{s}'[j] = 1$ for $j \in [n]$ if $j \in \mathcal{S}'$.
```

Fig. 2. LPN Algorithm for Constant Noise

Lemma 3.3. *For $\delta \in [0,1]$, $p \in (0,1)$, the learning algorithm presented in Figure 2 uses samples from Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ to estimate the secret value $\mathbf{s}'$. The algorithm runs in time $\frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$, requires* $\mathsf{num} = \frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$ *number of samples and outputs the correct secret key, i.e. $\mathbf{s} = \mathbf{s}'$ with probability $1-\delta$.*

Proof. The proof can be found in Section A.6. $\square$

3.3 Combining the Results

Combining the results of Sections 3.1 and 3.2 we obtain the following theorem:

Theorem 3.4. *For $\delta \in [0,1]$, $p \in (0,1)$, the Learning Parity with Noise algorithm presented in Figure 2, learns parity with k out of n variables with the total number of samples and total computation time of*

$$\mathsf{poly}\left(\frac{1}{(1-2\eta)^{\sqrt{np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{n}{\delta}) \cdot 2^{\frac{np}{\log(np)}} \cdot \log(np)\right),$$

and achieves success probability of $1-\delta-\left(\frac{16}{(1-2\eta)^{\sqrt{8np}} \cdot p^{2(k-1)}(1-p)^2} \cdot \ln(\frac{2n}{\delta}) \cdot \exp(\frac{-pn}{8})\right)$.

Proof. Using Lemma 3.3, we have that the number of p-biased samples required is $\mathsf{num} = \frac{8}{(1-2\eta')^2 \cdot p^{2(k-1)} \cdot (1-p)^2} \cdot \ln(2n/\delta)$ and using Lemma 3.1 we have that $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$. From Theorem 3.2 we have that with probability $1 - 2\exp(-p \cdot n/8)$ each p-biased sample can be obtained by an invocation of the $\mathsf{BKW_R}$ algorithm, which requires $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ samples and $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ runtime with probability $1 - 2\exp(-p \cdot n/8)$. Combining and taking a union bound, we have that the algorithm in Figure 2 requires at most $\mathsf{num} \cdot$

$O\left(2^{\frac{4np}{\log(2np)}} \cdot \log(2np)\right)$ samples and run time and succeeds with probability $1 - \delta - (2 \cdot \mathsf{num} \cdot \exp(-p \cdot n/8))$.

□

3.4 Parameter Settings

We consider the parameter setting for which our algorithm asymptotically outperforms the previous algorithms *in the exponent*. We consider two cases.

– The algorithm has to run faster than a brute force algorithm which tries all the $\binom{n}{k}$ combination to find the sparse secret. Note that the best algorithms for k-sparse LPN achieve only a constant factor improvement *in the exponent* beyond brute force search. Since we are concerned with asymptotic improvement in the exponent, these algorithms are equivalent to brute force search.
– The algorithm should run faster than the BKW algorithm for the length-n LPN problem, as BKW is the asymptotically best algorithm for length-n LPN.

Corollary 3.5. *For the sparsity* $k = k(n) = \frac{n}{\log^{1+1/c}(n)}$*, where* $c \in o(\log\log(n))$ *and* $c \in \omega(1)$*, the runtime of our learning algorithm in Figure 2 is contained in both* $\log(n)^{o(k)}$ *and* $2^{o(n/\log(n))}$*, with constant failure probability. For this range of* k*, Brute Force search requires runtime* $\log(n)^{\Omega(k)}$ *and* BKW *requires runtime of* $2^{\Omega(n/\log(n))}$.

Proof. Setting $1/p = \log^{1/(c)}(n)$ and $k = \frac{n}{\log^{(c+1)/c}(n)}$ in Theorem 3.4, we find that our LPN Algorithm for constant noise rate presented in Figure 2 succeeds with constant probability and has runtime

$$\left(\frac{1}{p}\right)^{2k} \cdot 2^{\frac{4np}{\log(2np)}} = \log(n)^{(1/c)\cdot\frac{n}{\log^{(c+1)/c}(n)}} \cdot 2^{\frac{4n/\log^{1/(c)}(n)}{\log(2n/\log^{1/(c)}(n))}} \in \log(n)^{O((1/c)\cdot k)}.$$

Note that if $c \in \omega(1)$, then our runtime is in $\log(n)^{o(k)}$. On the other hand, if $c \in o(\log\log(n))$ then our runtime

$$\log(n)^{O((1/c)\cdot k)} = 2^{O((\log\log(n)/c)\cdot k)} \in 2^{o(k)} \in 2^{o(n/\log(n))}.$$

and so asymptotically beats the above two algorithms *in the exponent* for any $c = c(n)$ that satisfies $c \in \omega(1)$ and $c \in o(\log\log(n))$. Plugging the above parameter into Theorem 3.4 yields probability of success of $1 - \delta - \mathrm{negl}(n) = 1 - \delta$.

□

4 Low Noise Setting

In this section we present an improved learning algorithm for the low noise setting. The algorithm will draw only a *polynomial number of samples* from the given LPN oracle, use them to construct a much larger set of p-biased samples that are not independent, but have certain desirable properties, and then present a learning algorithm that succeeds w.r.t. a set of p-biased samples satisfying these properties.

4.1 Sample Partition

In this section we present the SamP algorithm which draws a polynomial-sized set of samples from the original LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{\mathit{0},\eta}(\mathbf{s})$, and uses them to construct a far larger set of p-biased samples that are "close" to being pairwise independent. To achieve this, SamP constructs a large number of subsets of size $2np+1$ from the polynomial-sized set of samples, such that each pair of distinct subsets has at most $t \ll 2np+1$ number of samples in common Then, from each subset of size $2np+1$, we construct a single p-biased sample $\mathsf{s}^p = (\mathbf{x}', b')$ as follows: First, a random subset $R \subseteq [n]$ of coordinates is chosen, by placing each index $i \in [n]$ in R with independent probability p. Note that with overwhelming probability, $|R| \leq 2np$. Thus, given our set of $2np+1 \geq |R|+1$ samples, we construct a matrix $\mathbf{M}$ that contains the samples as rows and we compute the left kernel of the matrix to find a vector $\mathbf{u}$ to zero out the coordinates of R – i.e. $(\mathbf{u}\cdot\mathbf{M})\,|_R = 0^{|R|}$ and the returned sample is $(\mathbf{x}', b') := \mathbf{u}\cdot\mathbf{M}$. This procedure is denoted by RLK (see Definition A.11 for more details). Note that the procedure always succeeds when the size of R is at most $2np+1$.[5] We show that the samples resulting from distinct subsets are "close" to independent, due to the small intersection of any pair of subsets. We next provide some additional details on the construction and guarantees on independence, before formally describing the algorithm and its properties.

Constructing the subsets with small pairwise intersection. Our algorithm given in Figure 3 constructs the subsets using the *designs* of Nisan and Wigderson [25]: It first draws $(2np+1)^2$ samples from the original LPN distribution and associates each sample with an ordered pair (x, y) for $x, y \in \mathbb{F}$, for the field $\mathbb{F}$ of size $2np+1$. There are $(2np+1)^t$ polynomials of degree $t-1$ in $\mathbb{F}$, and each subset is associated with a particular polynomial, i.e. the samples contained in a particular subset correspond to the $2np+1$ points that lie on the associated polynomial. Note that the maximum number of subsets that can be constructed is $(2np+1)^t$ and that, furthermore, since any pair of distinct polynomials of degree $t-1$ in $\mathbb{F}$ intersect in at most t points, any two subsets have at most t samples in common. Note that this construction allows at most $\mathsf{maxnum} := (2np+1)^t$ number of p-biased samples to be generated. Looking ahead, in Section 4.2 we will present a learning algorithm that requires $O(\log(n))$ such independent sets of samples, each of size at most maxnum to learn the parity function.

[5] If the size of R is larger than this, a bad event Event1 occurs, and we must draw new independent samples from the oracle. We will later show that Event1 occurs with negligible probability.

Generating the p-biased samples

Obtain $(2np+1)^2$ independent samples $\mathcal{S} = \{\mathsf{s}_1, \ldots, \mathsf{s}_{(2np+1)^2}\}$ from the unbiased LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$. Run the following setup phase to create sets $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{\mathsf{maxnum}}$ each of size $2np+1$ such that for distinct i, j, $|\mathcal{O}_i \bigcap \mathcal{O}_j| \le t$.

Setup Phase :

1. Consider a Finite Field $\mathbb{F}$ of size $2np+1$. Define a bijection π from $[(2np+1)^2]$ to pairs $(x, y) \in \mathbb{F} \times \mathbb{F}$.
2. Consider all polynomials of degree $t-1$ in the ring $\mathbb{F}[x]$. There are $\mathsf{maxnum} := (2np+1)^t$ such distinct polynomials $\mathsf{poly}_1, \ldots, \mathsf{poly}_{\mathsf{maxnum}}$.
3. For $j \in [\mathsf{maxnum}]$, $\mathcal{O}_j$ contains s_i if and only if $\pi(i) = (x, y)$ and $\mathsf{poly}_j(x) = y$.

Algorithm 2: SamP(j)

```
Result: $p$-biased sample $(\mathbf{x}', b')$.
To respond to the $j$-th query, if $j > \mathsf{maxnum}$ then
    return $\perp$ and terminate.
end
Otherwise, sample a set $R_j$ such that each $i \in [n]$ is selected
  independently into $R_j$ with probability $p$;
if $|R_j| \ge 2np \vee |R_j| \le pn/2$ then
    Event1 occurs.;
    Sample a fresh set of $|R_j|+1$ samples from the LPN oracle and
      arrange them in rows of matrix $\mathbf{A}$ of size $(|R_j|+1 \times n)$.;
    Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$;    ▷ RLK is
      defined in Section A.3
    Go To L1;
end
Select set $\mathcal{O}_j$ and arrange them in rows of matrix $\mathbf{A}$ of size $(2np+1 \times n)$;
Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$;  ▷ RLK is defined
  in Section A.3
if $\mathbf{x}'|_{R_i} = 0^{|R_i|}$ for some $i \in [j-1]$ then
    Event2 occurs;
    Sample a fresh set of $2np+1$ samples from the LPN oracle and
      arrange them in rows of matrix $\mathbf{A}$ of size $(2np+1 \times n)$;
    Compute $(\mathbf{x}', \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R_j)$ such that $\mathbf{x}'|_{R_j} = 0^{|R_j|}$;
end
L1 : k := 1;
$(\mathbf{x}', b') := \mathbf{u} \cdot \mathbf{A}$;
while $k < 2np + 1 - \mathsf{weight}(\mathbf{u})$ do
                                        ▷ weight is defined in Section A.3
    $b' := b' + \tilde{\mathcal{O}}_\eta$
end
return $(\mathbf{x}', b')$;
```

Fig. 3. SamP "Zeroing" Algorithm

Near pairwise independence. We note that by construction, the Sample Partition Algorithm SamP presented in Figure 3 constructs sets of size $(2np+1)$ such the intersection of any two sets is at most t for $t \le (np+1)$. This will allow us to bound the covariance of the errors e'_i and e'_j obtained by taking linear combinations of elements in the sets $\mathcal{O}_i, \mathcal{O}_j$. Overall, the set of samples generated by SamP algorithm have certain properties enumerated in the following Lemma.

Lemma 4.1. *Consider an experiment in which the setup phase is run and two samples $\mathsf{s}^p_i = (\mathbf{x}'_i, b'_i)$ and $\mathsf{s}^p_j = \left(\mathbf{x}'_j, b'_j\right)$ are generated by running $\mathsf{SamP}(i)$ and $\mathsf{SamP}(j)$ for distinct $i, j \le \mathsf{maxnum}$ then the following hold:*

1. *Each individual sample $(\mathbf{x}'_i, b'_i)$ (resp. $\left(\mathbf{x}'_j, b'_j\right)$) outputted is distributed identically to a sample drawn from a p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{2np+1}$.*
2. *$\mathbf{x}'_i$ and $\mathbf{x}'_j$ are pairwise independent*
3. *Recall that $b'_i = f_{\mathbf{s}}(\mathbf{x}'_i) + e'_i$ and $b'_j = f_{\mathbf{s}}(\mathbf{x}'_j) + e'_j$. Then*

$$\mathrm{Cov}\left[e'_i, e'_j\right] \le (1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}.$$

Proof. The proof can be found in Section A.7. □

Finally, we analyze the runtime and sample complexity for each invocation of SamP.

Theorem 4.2. *Given access to LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ which gives samples $\mathsf{s} = (\mathbf{x}, b)$, the SamP algorithm requires $O\left((np)^2\right)$ samples in total, and $\mathsf{poly}(np)$ runtime per invocation with probability at least $1 - 2\exp(-p \cdot n/8) - (np)^t \cdot \exp(-n/48) - (np)^t \cdot 1/2^{np/4}$.*

Proof. The proof can be found in Section A.8. □

4.2 Learning Secret Coordinates

In this subsection we present our Learning Algorithm in Figure 4. The input to the algorithm is $8\log(n)$ independently generated sets of p-biased samples with the properties given in Lemma 4.1. The algorithm uses the p-biased samples to estimate the values of the Fourier Coefficients of the target function.

Lemma 4.3. *For $\delta \in [0,1]$, $p \in (0,1)$, given as input $8\log(n)$ independent sets of samples $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{8\log(n)}$ each of size $\mathsf{num} := O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and each satisfying the properties given in Lemma 4.1 for some $t \in \Theta(1/\eta)$, the Learning Algorithm presented in Figure 4 runs in time $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and outputs the correct secret key, i.e. $\mathbf{s} = \mathbf{s}'$ with probability $1-\delta$.*

Proof. The proof can be found in Section A.9. □

```
The Learning Algorithm
The learning algorithm starts by having access to $8\log(n)$ sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{8\log(n)}$
of randomly generated samples. Each set of samples is independent and satisfies
the properties given in Lemma 4.1.

1. Initilizate set $\mathcal{S}' := \emptyset$.
2. For $j \in [n]$
    – $\mathsf{count} := 0$
    – $T := 8\log(n)$
    – For $i' \in T$:
        (a) Use the set $\mathcal{S}_{i'}$ of $\mathsf{num}$ number of samples to approximate $\hat{b}_p(\{j\}) := \frac{1}{\mathsf{num}}\sum_{i=1}^{\mathsf{num}} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, where each coordinate of $\mathbf{x}_i, b_i$ is switched to $\{-1,1\}$ from $\mathbb{F}_2$.
        (b) If $\hat{b}_p(\{j\}) > (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$, $\mathsf{count} := \mathsf{count} + 1$
    – if $\mathsf{count} \geq T/2$
        • add $j$ to $\mathcal{S}'$
3. Output $\mathbf{s}'$ such that $\mathbf{s}'[j] = 1$ for $j \in [n]$, if $j \in \mathcal{S}'$.
```

Fig. 4. LOW-NOISE LPN ALGORITHM

4.3 Combining the Results

Combining the results of Sections 4.1 and 4.2 we obtain the following theorem:

Theorem 4.4. *Assuming parameters are set such that*

$$\log\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \in o(1/\eta \cdot \log(np)), \tag{4.1}$$

and with $\delta \in [0,1]$, $p \in (0,1)$, the Learning Parity from Noise Algorithm presented in Figure 4, learns parity with k out of n variables and noise rate η using $(2np+1)^2 \cdot \log(n)$ number of samples, total computation time of $N := \mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ and achieves success probability of

$$1 - \delta - \left(N \cdot \left(2 \cdot \exp(-p \cdot n/8) + \exp(-n/48) + 1/2^{np/4}\right)\right)$$

Proof. Using Lemma 4.3, we have that, for some $t \in \Theta(1/\eta)$, the number of p-biased samples with the following properties needed to succeed with probability $1-\delta$ is $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$. From Theorem 4.2, we have that as long as $\mathsf{num} = \mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \leq \mathsf{maxnum} = (2np+1)^t$ we can generate the required samples using $(2np+1)^2$ samples from the unbiased LPN oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$, and with $\mathsf{poly}(np)$ runtime per sample, with probability at least $1 - 2(np)^t \cdot \exp(-p\cdot n/8) - (np)^t \cdot \exp(-n/48) - (np)^t \cdot 1/2^{np/4}$. The fact that num and maxnum satisfy the above constraint is guaranteed by the assumption in the theorem on the setting of parameters and the fact that $t \in \Theta(1/\eta)$. Combining and taking a union bound, we have that the algorithm in Figure 2 requires $(2np+1)^2 \cdot 8\log(n)$ samples, has run time $\mathsf{poly}\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)} \cdot \log(n)\right)$, and succeeds with probability $1-\delta-\left(N \cdot \left(2 \cdot \exp(-p \cdot n/8) + \exp(-n/48) + 1/2^{np/4}\right)\right)$. □

4.4 Parameter Settings

We consider the parameter setting for which our algorithm's runtime asymptotically outperforms the previous algorithms' runtime *in the exponent.* We consider two cases.

- The algorithm has to run faster than a brute force algorithm which tries all the $\binom{n}{k}$ combinations to find the sparse secret. Note that there are known algorithms that improve upon brute force search, but the improvement is a *constant factor in the exponent.* Since we are concerned with asymptotic improvement in the exponent, these algorithms are equivalent to brute force search.
- The algorithm should run faster than the algorithm which just gets lucky and gets n noiseless samples, we call this algorithm "Lucky Bruteforce". For this algorithm to succeed, it needs $\frac{n}{1-\eta}$ samples from LPN Oracle to ensures that there are approximately n noiseless samples. The next step is to just randomly select n out of these $\frac{n}{1-\eta}$ samples and try Gaussian elimination on them. The run time of such an algorithm for small η can be approximate by $e^{\eta n}$.

Corollary 4.5. *For sparsity $k(n)$ such that $k = \frac{1}{\eta} \cdot \frac{\log(n)}{\log(f(n))}$, noise rate $\eta \neq 1/2$ such that $\eta^2 = \left(\frac{\log(n)}{n} \cdot f(n)\right)$for $f(n) \in \omega(1) \cap n^{o(1)}$, the Learning Algorithm of Figure 4 runs in time $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2k}} \cdot \log(n) \cdot (np)^3\right) \in \left(\frac{n}{k}\right)^{o(k)}$ with constant probability. In this setting, the running time Brute Force is $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ and the running time of Lucky Bruteforce is $e^{\eta n} \in \left(\frac{n}{k}\right)^{\omega(k)}$.*

Proof. For k, η defined as above, we choose the biased $p = \frac{1}{f(n)}$ and $\frac{1}{p} \in \left(\frac{n}{k}\right)^{o(1)}$, we have constraint (4.1) from Theorem 4.4 satisfied as follows:

$$\log\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right) \approx 4np\eta + 2k\log(\frac{1}{p}) \\ \in o(1/\eta \cdot \log(n)) \in o(1/\eta \cdot \log(np)),$$

the runtime of the Learning Algorithm of Figure 4 is bounded by

$$\frac{1}{(1-2\eta)^{2np}p^{2k}} \cdot \log(n) \cdot O\left((np)^3\right) \approx e^{4np\eta} \cdot \left(\frac{1}{p}\right)^{2k} \cdot \log(n) \cdot O\left((np)^3\right) \\ \in e^{o(k)\cdot\log(n/k)} \cdot \left(\frac{n}{k}\right)^{o(k)} \cdot \log(n) \cdot o(n^3) \\ \in \left(\frac{n}{k}\right)^{o(k)},$$

which outperforms Brute Force and Lucky Bruteforce under the same parameter settings. Plugging the above parameters into Theorem 4.4 yields probability of success of $1-\delta-\mathrm{negl}(n)$. □

5 Learning Other Classes of Functions

In the following we apply our LPN algorithms from Section 3 to learn other classes of functions. First, let us look at the reduction from learning DNFs to learning noisy parities.

Theorem 5.1 (Theorem 2 in [12]). *Let $\mathcal{A}$ be an algorithm that learns noisy parities of k variables on $\{0,1\}^n$ for every noise rate $\eta < 1/2$ in time $T(n,k,\frac{1}{1-2\eta})$ and using at most $S(n,k,\frac{1}{1-2\eta})$. Then there exists an algorithm that learns DNF expressions of size s in time $\tilde{O}\left(\frac{s^4}{\epsilon^2}\cdot T(n,\log B,B)\cdot S(n,\log B,B)^2\right)$, where $B=\tilde{O}(s/\epsilon)$.*

We are interested in determining the parameter range for which our algorithm yields an asymptotic improvement over the state of the art in the *exponent*. The work of Grigorescu [14] is the current state-of-the-art. They present an improvement of the bound from [28] of $2^{O(\log(n)\log\frac{s}{\epsilon})}$ for $\frac{s}{\epsilon}\in o\left(\frac{\log^{1/3} n}{\log\log n}\right)$. As we are similarly applying the reductions from Feldman, our algorithm yields a similar improvement on the bounds in [28] for a different range of $\frac{s}{\epsilon}$.

Note the reduction in Feldman [12] relates the ratio of the size of the DNF and its approximation factor to both the noise rate and sparsity of the parity function. Thus, the parameter range for which our algorithm is optimal will be expressed in terms of this ratio.

We begin by extending the runtime analysis of our algorithm from Section 3, which dealt with the constant noise setting, to the arbitrary noise $\eta < 1/2$.

Theorem 5.2. *The learning algorithm described in Figure 2 has a runtime of*

$$T\left(n,k,\frac{1}{1-2\eta}\right)=\left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}}\frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2}O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

and requires

$$S\left(n,k,\frac{1}{1-2\eta}\right)=\left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}}\frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2}O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

LPN samples in the high noise setting, and achieves a success probability of

$$1-\delta-\left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}}\frac{16\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2}e^{\frac{-np}{8}}$$

where $\mathfrak{a}\mathfrak{b}=np$.

Proof. The proof follows directly from Theorem 3.4. Instead of fixing a value for $\mathfrak{a}$ and $\mathfrak{b}$, we let them remain free parameters. As well, we no longer make assumptions on the noise rate η. Thus, we start with the runtime in terms of η'.

$$T(n,k,\eta') = \frac{8\ln(2n/\delta)}{(1-2\eta')^2 p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

$$T(n,k,\eta) = \frac{8\ln(2n/\delta)}{(1-2\eta)^{2^{\mathfrak{a}+1}} p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

$$T\left(n,k,\frac{1}{1-2\eta}\right) = \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2} O\left(\mathfrak{a}2^{\mathfrak{b}}\right)$$

The sample complexity of the algorithm is equal to its runtime complexity, and thus we need to just need to consider the success probability. In the high noise setting, the p-biased LPN oracle is called $\mathsf{num} = \left(\frac{1}{1-2\eta}\right)^{2^{\mathfrak{a}+1}} \frac{8\ln(2n/\delta)}{p^{2(k-1)}(1-p)^2}$ times, and the success probability calculation follows the same formula from Theorem 3.4. □

As we are concerned with asymptotic improvement in the *exponent* of the runtime we will take the logarithm of the runtime and compare it to the state of the art for learning DNFs and Juntas.

Corollary 5.3. *The learning algorithm described in Figure 2 learns DNFs of size s and approximation factor ϵ, with asymptotic improvements over Verbeurgt's bound [28] of $O\left(n^{\log \frac{s}{\epsilon}}\right)$, and with negligible failure probability when $\log \frac{s}{\epsilon} \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $\log \frac{s}{\epsilon} \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.*

Note here that the parameter regime in 5.3 requires setting the free parameters of the learning algorithm differently than in the constant noise setting. In order to minimize the runtime of the $\mathsf{BKW_R}$ step of the algorithm in the high noise setting, the value for $\mathfrak{a}$ must be changed from the description in Section 3. Thus we set $\mathfrak{a} = (1/2)\log\log(np)$. This change necessitates considerations for δ, the Fourier analysis confidence. This ensures that the failure probability of the full algorithm remains small, even after increasing the number of samples required. We set $\delta = 2^{-n}$. The free parameter p is set to $n^{-o(1)}$ to satisfy asymptotic requirements. These parameters are set similarly for Corollary 5.5.

Aside from DNFs we can also use our LPN algorithm to learn Juntas. By applying Feldman's reduction we are able to yield an algorithm that, for certain ranges for k, is able to improve on the $O(n^{0.7k})$ runtime cited in [27] asymptotically, not just by reducing the constant factor in the exponent.

Theorem 5.4 (Theorem 3 in [12]). *Let $\mathcal{A}$ be an algorithm that learns parities of k variables on $\{0,1\}^n$ for every noise rate $\eta < 1/2$ in time $T(n,k,\frac{1}{1-2\eta})$. Then there exists an algorithm that learns k-juntas in time $O\left(2^{2k}k \cdot T(n,k,2^{k-1})\right)$.*

Corollary 5.5. *The learning algorithm described in Figure 2 learns Juntas of size k with a runtime of $n^{o(k)}$ and a negligible failure probability when $k \in \omega\left(\frac{c}{\log n \log\log c}\right)$, and $k \in n^{1-o(1)}$, where $c \in n^{1-o(1)}$.*

6 Acknowledgments

The authors would like to thank the anonymous reviewers of TCC 2021 for their insightful comments.

References

1. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009). https://doi.org/10.1007/978-3-642-03356-8_35
2. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I. Lecture Notes in Computer Science, vol. 6755, pp. 403–415. Springer, Heidelberg, Germany, Zurich, Switzerland (Jul 4–8, 2011). https://doi.org/10.1007/978-3-642-22006-7_34
3. Blum, A., Furst, M.L., Jackson, J.C., Kearns, M.J., Mansour, Y., Rudich, S.: Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: 26th Annual ACM Symposium on Theory of Computing. pp. 253–262. ACM Press, Montréal, Québec, Canada (May 23–25, 1994). https://doi.org/10.1145/195058.195147
4. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 278–291. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994). https://doi.org/10.1007/3-540-48329-2_24
5. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**(4), 506–519 (2003). https://doi.org/10.1145/792538.792543, `https://doi.org/10.1145/792538.792543`
6. Bogdanov, A., Sabin, M., Vasudevan, P.N.: XOR codes and sparse learning parity with noise. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019. pp. 986–1004 (2019). https://doi.org/10.1137/1.9781611975482.61, `https://doi.org/10.1137/1.9781611975482.61`
7. Bogos, S., Tramer, F., Vaudenay, S.: On solving lpn using bkw and variants. Cryptography and Communications **8**(3), 331–369 (2016)
8. Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous IBE, leakage resilience and circular security from new assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 535–564. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78381-9_20
9. Cheon, J.H., Son, Y., Yhee, D.: Practical FHE parameters against lattice attacks. IACR Cryptol. ePrint Arch. **2021**, 39 (2021), `https://eprint.iacr.org/2021/039`
10. Dachman-Soled, D., Feldman, V., Tan, L.Y., Wan, A., Wimmer, K.: Approximate resilience, monotonicity, and the complexity of agnostic learning. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 498–511. SIAM (2014)

11. Feldman, V., Gopalan, P., Khot, S., Ponnuswami, A.K.: New results for learning noisy parities and halfspaces. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). pp. 563–574 (2006). https://doi.org/10.1109/FOCS.2006.51
12. Feldman, V., Gopalan, P., Khot, S., Ponnuswami, A.K.: On agnostic learning of parities, monomials, and halfspaces. SIAM Journal on Computing **39**(2), 606–645 (2009). https://doi.org/10.1137/070684914, https://doi.org/10.1137/070684914
13. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: How to encrypt with the LPN problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II. Lecture Notes in Computer Science, vol. 5126, pp. 679–690. Springer, Heidelberg, Germany, Reykjavik, Iceland (Jul 7–11, 2008). https://doi.org/10.1007/978-3-540-70583-3_55
14. Grigorescu, E., Reyzin, L., Vempala, S.: On noise-tolerant learning of sparse parities and related problems. In: International Conference on Algorithmic Learning Theory. pp. 413–424. Springer (2011)
15. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 1–20. Springer (2014)
16. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: An efficient authentication protocol based on ring-lpn. In: Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. pp. 346–365 (2012). https://doi.org/10.1007/978-3-642-34047-5_20, https://doi.org/10.1007/978-3-642-34047-5_20
17. Hopper, N.J., Blum, M.: Secure human identification protocols. In: Boyd, C. (ed.) Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science, vol. 2248, pp. 52–66. Springer, Heidelberg, Germany, Gold Coast, Australia (Dec 9–13, 2001). https://doi.org/10.1007/3-540-45682-1_4
18. Juels, A., Weis, S.A.: Authenticating pervasive devices with human protocols. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 293–308. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005). https://doi.org/10.1007/11535218_18
19. Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: International conference on security and cryptography for networks. pp. 348–359. Springer (2006)
20. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform, and learnability. J. ACM **40**(3), 607–620 (1993). https://doi.org/10.1145/174130.174138, https://doi.org/10.1145/174130.174138
21. Lipton, R.J., Markakis, E., Mehta, A., Vishnoi, N.K.: On the fourier spectrum of symmetric boolean functions with applications to learning symmetric juntas. In: 20th Annual IEEE Conference on Computational Complexity (CCC'05). pp. 112–119. IEEE (2005)
22. Lyubashevsky, V.: 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, pp. 378–389. Springer (2005)
23. Mitzenmacher, M., Upfal, E.: Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press (2017)

24. Mossel, E., O'Donnell, R., Servedio, R.P.: Learning juntas. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. pp. 206–212 (2003)
25. Nisan, N., Wigderson, A.: Hardness vs Randomness. Journal of computer and System Sciences **49**(2), 149–167 (1994)
26. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press, Santa Fe, NM, USA (Nov 20–22, 1994). https://doi.org/10.1109/SFCS.1994.365700
27. Valiant, G.: Finding correlations in subquadratic time, with applications to learning parities and juntas. In: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science. pp. 11–20. IEEE (2012)
28. Verbeurgt, K.: Learning DNF under the Uniform Distribution in Quasi-Polynomial Time. In: Proceedings of the Third Annual Workshop on Computational Learning Theory. p. 314–326. COLT '90, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)

A Appendix

A.1 Probability Bounds

The following inequality is used to bound the magnitude of an observed random variable with respect to the true expected value of that random variable. The Chernoff-Hoeffding bound extends the Chernoff bound to random variables with a bounded range. Another important fact is that Chernoff-Hoeffding bound assumes the random variables are independent whereas Chebyshev's bound applies to arbitrary random variables. The reader in encouraged to refer to [23] for more in depth reading.

Theorem A.1 (Multiplicative Chernoff Bounds). *Let $X_1, X_2, \dots, X_n$ be n mutually independent random variables. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$,*

$$\Pr[X \le (1-\beta)\mu] \le \exp\left(\frac{-\beta^2\mu}{2}\right) \text{ for all } 0 < \beta \le 1$$

$$\Pr[X \ge (1+\beta)\mu] \le \exp\left(\frac{-\beta^2\mu}{3}\right) \text{ for all } 0 < \beta \le 1$$

Theorem A.2 (Chernoff-Hoeffding). *Consider a set of n independent random variables $X_1, X_2, \dots, X_n$. If we know $a_i \le X_i \le b_i$, then let $\Delta_i = b_i - a_i$. Let $X = \sum_{i=1}^n X_i$. Then for any $\alpha \in (0, 1/2)$*

$$\Pr\left(\left|X - \mathbb{E}[X]\right| > \alpha\right) \le 2\exp\left(\frac{-2\alpha^2}{\sum_{i=1}^n \Delta_i^2}\right).$$

Theorem A.3 (Chebyshev's). *Consider a set of n arbitrary random variable $X_1, X_2, \dots, X_n$. Let $X = \sum_{i=1}^n X_i$. Then for any $\alpha > 0$,*

$$\Pr\left(\left|X - \mathbb{E}[X]\right| \ge \alpha\right) \le \frac{\mathrm{Var}\,[X]}{\alpha^2}.$$

The following lemma is being used to further simplify the Var[X] in Theorem A.3.

Lemma A.4. *Let $X_1, X_2, \ldots, X_n$ be n arbitrary random variables. Then*

$$\mathrm{Var}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{Var}\left[X_i\right] + 2\sum_{i=1}^{n}\sum_{j>i} \mathrm{Cov}\left[X_i, X_j\right].$$

A.2 Learning Parities

In this subsection, we define three Oracles . The first is the *standard* LPN Oracle, that samples $\mathbf{x}$ uniformly. The second is the noise Oracle, which sets $\mathbf{x}$ to the zero vector. The purpose of this Oracle is to return additional noise sampled identically to the noise found in a normal LPN sample. The third Oracle is the p-biased LPN Oracle, which samples $\mathbf{x}$ according to a p-biased Bernoulli distribution.

Definition A.5 (Bernoulli Distribution). *Let $p \in [0,1]$. The discrete probability distribution of a random variable which takes the value 1 with probability η and the value 0 with probability $1-\eta$ is called Bernoulli Distribution and it is denoted by Ber_η.*

Definition A.6 (LPN Oracle). *Let secret value $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ and let $\eta < 1/2$ be a constant noise parameter. Let Ber_η be a Bernoulli distribution with parameter η. Define the following distribution $\mathcal{L}^{(1)}_{\mathbf{s},\eta}$ as follows*

$$\left\{ (\mathbf{x}, b) \mid \mathbf{x} \leftarrow \mathbb{Z}_2^n,\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done module 2. Upon calling the LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$, a new sample $\mathsf{s} = (\mathbf{x}, b)$ from the distribution $\mathcal{L}^{(1)}_{\mathbf{s},\eta}$ is returned.

Definition A.7 (Noise Oracle). *Let secret value $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ and let $\eta < 1/2$ be a constant noise parameter. Let Ber_η be a Bernoulli distribution with parameter η. Define the following distribution $\mathcal{L}^{(2)}_{\mathbf{s},\eta}$ as follows*

$$\left\{ (\mathbf{x}, b) \mid \mathbf{x} := 0^n,\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done module 2. Upon calling the Noise Oracle $\tilde{\mathcal{O}}_\eta$ a new sample $\mathsf{s} = (\mathbf{x}, b)$ from the distribution $\mathcal{L}^{(2)}_{\mathbf{s},\eta}$ is returned.

Definition A.8 (p-biased LPN Oracle). *Let secret value $\mathbf{s} \leftarrow \mathbb{Z}_2^n$ and let $\eta < 1/2$ be a constant noise parameter. Let Ber_η be a Bernoulli distribution with parameter η and $\mathsf{Ber}^n_{(1-p)/2}$ be Bernoulli distribution with parameter $(1-p)/2$ over n coordinates. Define the following distribution $\mathcal{L}^{(3)}_{\mathbf{s},\eta,p}$ as follows*

$$\left\{ (\mathbf{x}, b) \mid \mathbf{x} \leftarrow \mathsf{Ber}^n_{(1-p)/2},\ f_{\mathbf{s}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{s} \rangle,\ b = f_{\mathbf{s}}(\mathbf{x}) + e,\ e \leftarrow \mathsf{Ber}_\eta \right\} \in \mathbb{Z}_2^{n+1}$$

with the additions being done modulo 2. Upon calling the p-biased LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{p,\eta}(\mathbf{s})$ a new sample $\mathsf{s}^p = (\mathbf{x}, b)$ from the distribution $\mathcal{L}^{(3)}_{\mathbf{s},\eta,p}$ is returned.

As our algorithms require linear combinations of LPN samples, we present the following lemma that describes the noise growth associated with the linear combination.

Lemma A.9 (New Sample Error [5]). *Given a set of ℓ samples $(\mathbf{x}_1, b_1), \ldots, (\mathbf{x}_\ell, b_\ell)$ from an LPN Oracle $\mathcal{O}^{\mathsf{LPN}}_{0,\eta}(\mathbf{s})$ with secret $\mathbf{s}$, where the choice of samples may depend on the values of $\mathbf{x}_i$ but not on the values of b_i, then the new sample $\mathsf{s}_{\ell+1}$ can be formed as follows $\mathsf{s}_{\ell+1} = \sum_{i=1}^{\ell} \mathsf{s}_i$ which has the property that $b_{\ell+1}$ is independent of $\mathbf{x}_{\ell+1}$ and the probability that the label of the constructed sample is correct is as follows: $\eta' = \Pr[b' = \langle \mathbf{x}_{\ell+1}, s\rangle] = \frac{1}{2} - \frac{1}{2}(1-2\eta)^\ell$.*

For reference we additionally provide the runtime of the original BKW algorithm:

Theorem A.10 (BKW [5]). *The length-n parity problem, for noise rate η for any constant less than $1/2$, can be solved with number of samples and total computation time of $2^{O(n/\log n)}$.*

For sample i, the j-th coordinate of $\mathbf{x}$ is denoted by $\mathsf{s}_i.\mathbf{x}[j]$ and the j-th coordinate of $\mathbf{s}$ is denoted by $\mathsf{s}_i.\mathbf{s}[j]$. For simplicity, given two sample pairs $\mathsf{s}_1 = (\mathbf{x}_1, b_1)$ and $\mathsf{s}_2 = (\mathbf{x}_2, b_2)$ a new sample $\mathsf{s}_3 = \mathsf{s}_1 + \mathsf{s}_2$ can be formed by $\mathsf{s}_3 = (\mathbf{x}_1 + \mathbf{x}_2, b_1 + b_2)$ with the additions being done mod 2.

A.3 Miscellaneous

Definition A.11 (Restricted Left Kernel). *Given a matrix $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ for $m \le n$ and set $R \subset [n]$ such that $|R| < m$, RLK first finds a vector $\mathbf{u} \in \mathbb{Z}_2^m$ such that $\mathbf{v} = \mathbf{u}\cdot\mathbf{A}$ and $\mathbf{v}|_R = 0^{|R|}$. The RLK algorithm returns $(\mathbf{v}, \mathbf{u}) := \mathsf{RLK}(\mathbf{A}, R)$.*

Note that the RLK algorithm mentioned above can be implemented by simply modifying matrix $\mathbf{A}$ and only takes the columns pointed by set R, i.e. restriction of $\mathbf{A}$ to only columns pointed by R. Let's denote the new matrix by $\mathbf{A}'$, find a vector in left kernel of $\mathbf{A}'$ and call it $\mathbf{u}$. Then $\mathbf{v}$ can simply be computed as $\mathbf{v} = \mathbf{u}\cdot\mathbf{A}$.

Definition A.12 (Hamming Weight). *Given a vector $\mathbf{u} \in \mathbb{Z}_2^m$, $\mathsf{weight}(\mathbf{u})$ returns the number of 1's in vector $\mathbf{u}$, i.e. the Hamming weight of $\mathbf{u}$.*

A.4 Proof of Lemma 3.1

We first show that each coordinate of $\mathbf{x}'$ is set to 0 with independent probability $(1+p)/2$. The probability that a coordinate j of $\mathbf{x}'$ in sample s^p is set to 0 after running $\mathsf{BKW_R}$ can be computed as follows:

$$\begin{aligned}\Pr[\mathbf{x}'[j] = 0] &= \Pr[\mathbf{x}'[j] = 0 \mid j \in R]\cdot\Pr[j \in R] + \Pr[\mathbf{x}'[j] = 0 \mid j \notin R]\cdot\Pr[j \notin R] \\ &= 1\cdot p + 1/2\cdot(1-p) = (1+p)/2\end{aligned}$$

To show that the label b' is correct with probability η' and that the correctness of the label is independent of the instance $\mathbf{x}', \mathbf{s}$, note that $\mathbf{x}'$ is always constructed by XOR'ing a set of exactly $2^{\mathfrak{a}}$ number of samples and that the choice of the set of XOR'ed samples depends only on the random coins of the algorithm and on the $\mathbf{x}$ values, which are independent of the e value. Therefore, we can apply Lemma A.9 to conclude that the noise is independent and that b' is correct with probability $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{\sqrt{2np}}$.

A.5 Proof of Theorem 3.2

From the description of $\mathsf{BKW_R}$, it is clear to see that it takes $O(\mathfrak{a}2^{\mathfrak{b}})$ LPN samples and running time to generate a p-biased sample, where $\mathfrak{a} = \log(2np)/2, \mathfrak{b} = \lceil |R|/\mathfrak{a} \rceil$. Remember that the $\mathsf{BKW_R}$ algorithm will abort if $|R| \geq 2pn$ or $|R| \leq pn/2$, i.e. $\mathsf{Event1}$ occurs. By showing that $\mathsf{Event1}$ occurs with probability at most $2\exp(-p \cdot n/8)$, we obtain that $\mathsf{BKW_R}$ runs in time $O(2^{\frac{4np}{\log(2np)}} \cdot \log(2np))$ with probability at least $1 - 2\exp(-p \cdot n/8)$.

To bound the probability of $\mathsf{Event1}$ occurring, we notice that by multiplicative Chernoff bounds in Theorem A.1, we can bound the size of set R as follows:

$$\begin{aligned}
\Pr\left[|R| \geq 2pn\right] &\leq \exp(-p \cdot n/3) \\
\Pr\left[|R| \leq pn/2\right] &\leq \exp(-p \cdot n/8) \\
\Pr\left[|R| \geq 2pn \vee |R| \leq pn/2\right] &\leq \exp(-p \cdot n/3) + \exp(-p \cdot n/8) \leq 2\exp(-p \cdot n/8) \\
\Pr\left[pn/2 < |R| < 2pn\right] &> 1 - 2\exp(-p \cdot n/8)
\end{aligned}$$

A.6 Proof of Lemma 3.3

Before proving Lemma 3.3, we present the following simple claims about the number of samples needed to estimate the Fourier Coefficient of a single index. Based on Claim 2.2, the magnitude of Fourier coefficient of the indexes with secret value of 0 is equal to 0, while for the secret coordinates 1 that is equal to $\varepsilon = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$. In the Following Claim we compute how many samples are needed to estimate the magnitude of Fourier coefficient within distance of $\varepsilon/2$ of correct value. We will bound the failure probability with δ/n.

Claim A.13. For every $j \in [n]$, $\hat{b}_p(\{j\}) = \mathbb{E}[b \cdot \chi_{\{j\},p}(\mathbf{x}))]$, where $(\mathbf{x}, b) \sim \mathcal{O}^{\mathsf{LPN}}_{p,\eta'}(\mathbf{s})$, can be estimated within additive accuracy $\frac{\varepsilon}{2}$ and confidence $1 - \frac{\delta}{n}$ using $\frac{8}{\varepsilon^2} \cdot \frac{1+p}{1-p} \cdot \ln(2n/\delta)$ number of samples.

Proof. The estimate of $\hat{b}_p(\{j\})$ based on the m samples $\mathsf{s}_i^p = (\mathbf{x}_i, b_i)$ is.

$$\hat{b}_{\text{estimate}}(\{j\}) = \frac{1}{m}\sum_{i=1}^{m} b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$$

and notice that $\mathbb{E}\left[\hat{b}_{\text{estimate}}(\{j\})\right] = \hat{b}_p(\{j\})$. Lets denote $X_i = \frac{1}{m} \cdot b_i \cdot \chi_{\{j\},p}(\mathbf{x}_i)$, then note that $|X_i| \leq (1/m)\sqrt{\frac{1+p}{1-p}}$. Finally by Chernoff-Hoeffding bound of Theorem A.2 we have the following.

$$\Pr\left[\left|\hat{b}_{\text{estimate}}(\{j\}) - \hat{b}_p\left(\{j\}\right)\right| \geq \varepsilon/2\right] \leq 2\ \exp\left(\frac{-m\varepsilon^2}{8} \cdot \frac{1-p}{1+p}\right)$$

Bounding the right hand side by δ/n and solving for m gives the desired value for number of samples.

□

Proof of Lemma 3.3. Invoking Claim 2.2, we have that for j such that $\mathbf{s}[j] = 1$ $\hat{b}_p(\{j\}) = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$ while for j such that $\mathbf{s}[j] = 0$, $\hat{b}_p(\{j\}) = 0$. It is clear by inspection that Algorithm 2 succeeds when it correctly estimates the values of $\hat{b}_p(\{j\})$ to within additive $\varepsilon/2 := (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}/2$ for all $j \in [n]$. By Claim A.13, $\frac{8}{\varepsilon^2} \cdot \frac{1+p}{1-p} \cdot \ln(2n/\delta)$ number of samples are sufficient to estimate a single coordinate within additive $\varepsilon/2$ of its correct value with confidence $1 - \frac{\delta}{n}$. By a union bound, the success probability of estimating all coordinates to within additive $\varepsilon/2$ is $1-\delta$.

□

A.7 Proof of Lemma 4.1

The proof is similar to the proof of Lemma 3.1 and noticing that the SamP algorithm uses $2np+1$ samples to generate a single p-biased sample. Two p-biased samples $\mathbf{x}'_i, \mathbf{x}'_j$, $j > i$ are pairwise independent, unless the same linear combination of samples in $\mathcal{S}$ was used to generate both of them. But in that case, during execution, the condition $\mathbf{x}'_j|_{R_i} = 0^{|R_i|}$ would evaluate to true, which means that Event2 occurred and so fresh samples (not from $\mathcal{S}$) would be used to generate $\mathbf{x}'_j$.

In the rest of the proof we switch to the ± 1 representation instead of the Boolean representation. The sample $\mathsf{s}_i^p = (\mathbf{x}'_i, b'_i)$ is obtained from the samples in set $\mathcal{O}_i$ alongside some extra error samples from Noise Oracle $\tilde{\mathcal{O}}_\eta$. In the following proof these are denoted by $e_1, e_2, \ldots, e_{2np+1}$. Moreover, notice that the sample $\mathsf{s}_j^p = \left(\mathbf{x}'_j, b'_j\right)$, obtained from set $\mathcal{O}_j$, has at most t elements in common with the sample obtained from the set $\mathcal{O}_i$. Hence we can represent the error in sample $\mathsf{s}_j^p = \left(\mathbf{x}'_j, b'_j\right)$ as $e_1, e_2, \ldots, e_t, e''_{t+1} \ldots e''_{2np+1}$. For the ease of notation we assumed that the t samples which are in common are at index 1 to t.

$$\begin{aligned}\mathrm{Cov}[e'_i, e'_j] &= \mathrm{Cov}[e_1 \cdot e_2 \dots e_t \cdot e_{t+1} \dots e_{2np+1}\ ,\ e_1 \cdot e_2 \dots e_t \cdot e''_{t+1} \dots e''_{2np+1}] \\ &= \mathbb{E}[e_1^2 \cdot e_2^2 \dots e_t^2 \cdot e_{t+1} \dots e_{2np+1} \cdot e''_{t+1} \dots e''_{2np+1}] \\ &\quad - \mathbb{E}[e_1 \cdot e_2 \dots e_{2np+1}]\ \mathbb{E}[e_1 \cdot e_2 \dots e_t \dots e''_{t+1} \dots e''_{2np+1}] \\ &= (1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}\end{aligned}$$

Where the last line follows from the independence of errors, $\mathbb{E}[e_i] = 1 - 2\eta$ and $\mathbb{E}[e_i^2] = 1$.

A.8 Proof of Theorem 4.2

Assuming Event1 and Event2 do not occur, the sample complexity and runtime can be verified by inspection and assuming RLK takes $\mathsf{poly}(np)$ time.

It remains to bound the probability of Event1 and Event2. We can upper bound the probability of Event1 by $2\exp(-p \cdot n/8)$, as in the proof of Theorem 3.2.

To upperbound the probability of Event2, we note that assuming Event1 does not occur, Event2 occurs only if one of the following two events occur:

- Event′1: For some distinct $i, j \in \mathsf{maxnum}$, $|R_i \cap R_j| \geq np/4$.
- Event′2: For some distinct $i, j \in \mathsf{maxnum}$, $|R_i \setminus R_j| \geq np/4$ and $\mathbf{x}'_j|_{R_i \setminus R_j} = 0^{|R_i \setminus R_j|}$.

Since for distinct i, j, each coordinate $\ell \in [n]$ is placed in *both* R_i and R_j with probability p^2, by a union bound over all pairs i, j and a standard Chernoff bound, Event′1 can be upperbounded by:

$$\mathsf{maxnum}^2 \cdot \exp(-n/48) = (np)^t \cdot \exp(-n/48).$$

Since for any $\mathbf{x}'_j$, the coordinates outside of R_j are uniformly random, Event′2 can be upperbounded by:

$$\mathsf{maxnum}^2 \cdot 1/2^{np/4} = (np)^t \cdot 1/2^{np/4}.$$

A.9 Proof of Lemma 4.3

Similar to subsection 3.2, before proving Lemma 4.3, we first present the following claim about the number of samples needed to estimate the Fourier Coefficient of a single index. The algorithm gets access to $8\log(n)$ sets of p-biased samples. In the following claim we first prove how many samples are needed to be able to approximate the Fourier coefficient within additive distance of $\epsilon/2$ and later discuss how by repeating the approximation step, i.e. step 2b in Figure 4, will reduce the error in approximation even further.

Claim A.14. For $\delta \in [0,1]$, $p \in (0,1)$, given $8\log(n)$ independent sets of samples $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{8\log(n)}$ that each of size $\mathsf{num} := O\left(\frac{1}{(1-2\eta)^{4np+2}p^{2(k-1)}(1-p^2)}\right)$ and each satisfying the properties given in Lemma 4.1 for some $t \in \Theta(1/\eta)$, then for every $j \in [n]$, $\hat{b}_p(\{j\}) = \mathbb{E}[b \cdot \chi_{\{j\},p}(\mathbf{x})]$ can be estimated within additive accuracy $\frac{\epsilon}{2} = (1-2\eta')p^{k-1}\sqrt{1-p^2}/2$ for $\eta' = \frac{1}{2} - \frac{1}{2}(1-2\eta)^{2np+1}$ with confidence $1 - \frac{\delta}{n}$.

Proof. Let $X = \frac{1}{m}\sum_{i=1}^{m} b_i \cdot \chi_{S,p}(\mathbf{x}_i)$. Let f be a parity function. Assuming $S = \{k\}$, let $X_i = \frac{1}{m} \cdot b_i \cdot \chi_{\{k\},p}(\mathbf{x}_i)$. First we compute $\mathrm{Cov}[X_i, X_j]$ for k such that $\mathbf{s}[k] = 1$

$$\mathrm{Cov}[X_i, X_j] = \mathrm{Cov}\left[\frac{1}{m} \cdot b'_i \cdot \chi_{\{k\},p}(\mathbf{x}'_i)\ ,\ \frac{1}{m} \cdot b'_j \cdot \chi_{\{k\},p}(\mathbf{x}'_j)\right]$$

$$\begin{aligned}
\mathrm{Cov}[X_i, X_j] &= \frac{1}{m^2} \cdot \mathrm{Cov}\left[b'_i \cdot \chi_{\{k\},p}(\mathbf{x}'_i)\ ,\ b'_j \cdot \chi_{\{k\},p}(\mathbf{x}'_j)\right] \\
&= \frac{1}{m^2} \cdot \mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u]\right) \cdot e'_i \cdot \frac{\mathbf{x}'_i[k]-p}{\sqrt{1-p^2}}\ ,\ \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v]\right) \cdot e'_j \cdot \frac{\mathbf{x}'_j[k]-p}{\sqrt{1-p^2}}\right] \qquad \text{(A.1)} \\
&= \frac{1}{m^2} \cdot \frac{1}{1-p^2}\Bigg(\mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}'_i[u]\right) \cdot e'_i\ ,\ \left(\prod_{v:\mathbf{s}[v]=1 \wedge v \neq k} \mathbf{x}'_j[v]\right) \cdot e'_j\right] - \\
&\quad \mathrm{Cov}\left[\left(\prod_{u:\mathbf{s}[u]=1 \wedge u \neq k} \mathbf{x}'_i[u]\right) \cdot e'_i\ ,\ p \cdot \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v]\right) \cdot e'_j\right] - \\
&\quad \mathrm{Cov}\left[p \cdot \left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u]\right) \cdot e'_i\ ,\ \left(\prod_{v:\mathbf{s}[v]=1 \wedge v \neq k} \mathbf{x}'_j[v]\right) \cdot e'_j\right] + \\
&\quad \mathrm{Cov}\left[p \cdot \left(\prod_{u:\mathbf{s}[u]=1} \mathbf{x}'_i[u]\right) \cdot e'_i\ ,\ p \cdot \left(\prod_{v:\mathbf{s}[v]=1} \mathbf{x}'_j[v]\right) \cdot e'_j\right]\Bigg) \qquad \text{(A.2)} \\
&= \frac{1}{m^2} \cdot \frac{1}{(1-p^2)}\left(p^{2(k-1)}\mathrm{Cov}\left[e'_i, e'_j\right] - 2p^{2k}\mathrm{Cov}\left[e'_i, e'_j\right] + p^{2(k+1)}\mathrm{Cov}\left[e'_i, e'_j\right]\right) \qquad \text{(A.3)} \\
&= m^{-2}p^{2(k-1)}(1-p^2)\mathrm{Cov}\left[e'_i, e'_j\right] \\
&= m^{-2}p^{2(k-1)}(1-p^2)\left[(1-2\eta)^{2(2np-t)+2} - (1-2\eta)^{4np+2}\right] \qquad \text{(A.4)}
\end{aligned}$$

Where equation (A.1) follows from definition of Fourier Coefficients and noting that b'_i is multiplications of $\mathbf{x}_i$s and error term e_i, equation (A.2) follows from properties of Covariance, equation (A.3) follows from independence of $\mathbf{x}'_i$s and equation (A.4) follows from Lemma 4.1. We can also bound $\mathrm{Var}[X_i]$ as follows

$$\begin{aligned}
\mathrm{Var}[X_i] &= \mathrm{Var}\left[\frac{1}{m}\cdot b_i'\cdot \chi_{\{k\},p}(\mathbf{x}_i')\right]\\
&= \frac{1}{m^2}\cdot \mathrm{Var}\left[\left(\prod_{u:\mathbf{s}[u]=1}\mathbf{x}_i'[u]\right)\cdot e_i'\cdot \frac{\mathbf{x}_i'[k]-p}{\sqrt{1-p^2}}\right]\\
&= \frac{1}{m^2}\cdot\frac{1}{1-p^2}\left(\mathrm{Var}\left[\left(\prod_{u:\mathbf{s}[u]=1\wedge u\neq k}\mathbf{x}_i'[u]\right)\cdot e_i'\right] - p^2\cdot \mathrm{Var}\left[\left(\prod_{v:\mathbf{s}[v]=1}\mathbf{x}_i'[u]\right)\cdot e_i'\right]\right)\\
&= \frac{1}{m^2}\cdot\frac{1}{1-p^2}\left(\mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1\wedge u\neq k}\mathbf{x}_i'^2[u]\right)\cdot e_i'^2\right] - \mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1\wedge u\neq k}\mathbf{x}_i'[u]\right)\cdot e_i'\right]^2 -\right.\\
&\qquad \left. p^2\cdot\mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1}\mathbf{x}_i'^2[u]\right)\cdot e_i'^2\right] + p^2\cdot \mathbb{E}\left[\left(\prod_{u:\mathbf{s}[u]=1}\mathbf{x}_i'[u]\right)\cdot e_i'\right]^2\right) \qquad \text{(A.5)}\\
&= \frac{1}{m^2}\cdot\frac{1}{1-p^2}\left(1-p^{2(k-1)}(1-2\eta)^{2np}-p^2+p^{2(k+1)}(1-2\eta)^{2np}\right) \qquad \text{(A.6)}\\
&= m^{-2}\left(1-p^{2(k-1)}(1+p^2)(1-2\eta)^{2np}\right)\le m^{-2}
\end{aligned}$$

Where equation (A.5) follows from properties of variance and equation (A.6) follows from independence of $\mathbf{x}_i'$s. Then we have the following bound from Chebyshev's bound of Theorem A.3

$$\begin{aligned}
\Pr\left[|X-\mathbb{E}[X]|\ge \varepsilon/2\right] &\le \frac{\sum_{i=1}^m \mathrm{Var}\left[X_i\right] + 2\sum_{i=1}^m\sum_{j>i}\mathrm{Cov}\left[X_i,X_j\right]}{\varepsilon^2/4}\\
&\le 4\cdot\frac{m^{-1}+p^{2(k-1)}(1-p^2)\left[(1-2\eta)^{2(2np-t)+2}-(1-2\eta)^{4np+2}\right]}{\varepsilon^2}
\end{aligned}$$

By substituting $\varepsilon = (1-2\eta')\cdot p^{k-1}\sqrt{1-p^2}$ for $\eta' = \frac{1}{2}-\frac{1}{2}(1-2\eta)^{2np+1}$, we can bound the right hand side by a constant less than $1/2$ by setting $t < -\frac{\ln(9/8-1/c)}{2\ln(1-2\eta)}$ and setting $m = c\cdot\frac{1}{(1-2\eta)^{4np+2}p^{2(k-1)}(1-p^2)}$, where $c>8$. We use random variable $Y_{i'}$ to represents whether the value of count in step i' is increased or not, specifically $Y_{i'}=1$ represents the event that count is increased in step i'. Assume we repeat the protocol for T rounds in total. Let $Y=(1/T)\cdot\sum_{i'=1}^T Y_{i'}$. First, take the case that j such that $\mathbf{s}[j]=0$, we know that in each step of loop over i', $\Pr[Y_{i'}=1]=1/2-\epsilon$. Note that the algorithm is run T times using independent sets $\mathcal{S}_{i'}$ each time and index j is only added if in the majority of the runs its estimated Fourier coefficient is more than $\varepsilon/2$. Using Chernoff bound, we can bound $\Pr[Y \geq T/2] \leq 1/n$.

$$\begin{aligned}\Pr[\text{index } j \text{ is added to set } \mathcal{S}'] &= \Pr[\mathsf{count} \geq T/2] \\ &= \Pr[\frac{\sum_{i'=1}^{T} Y_{i'}}{T} \geq \frac{1}{2}] \\ &\leq \Pr\left[|Y - E[Y]| > \varepsilon\right] \leq 2\exp(-2T\varepsilon^2)\end{aligned}$$

We can bound the right hand side by $\frac{\delta}{n}$ for constant δ by setting $T = 8\log(n)$ and $\varepsilon = 1/4$. Similar argument applies to the case for j such that $\mathbf{s}[j] = 1$.

□

Proof of Lemma 4.3. Invoking Claim 2.2, we have that for j such that $\mathbf{s}[j] = 1$ $\hat{b}_p(\{j\}) = (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}$ while for j such that $\mathbf{s}[j] = 0$, $\hat{b}_p(\{j\}) = 0$. It is clear by inspection that Algorithm in Figure 4 succeeds when it correctly estimates the values of $\hat{b}_p(\{j\})$ to within additive $\varepsilon/2 := (1-2\eta') \cdot p^{k-1}\sqrt{1-p^2}/2$ for all $j \in [n]$. By Claim A.14, we need $8\log(n)$ sets such that each set has $O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)}\right)$ number of p-biased samples. So in total $\mathsf{num} \cdot 8\log(n) = O\left(\frac{1}{(1-2\eta)^{2np+2}p^{2(k-1)}(1-p^2)} \cdot \log(n)\right)$ number of p-biased samples are sufficient to estimate a single coordinate within additive $\varepsilon/2$ of its correct value with confidence $1 - \frac{\delta}{n}$. By a union bound, the success probability of estimating all coordinates to within additive $\varepsilon/2$ is $1-\delta$.

□