Electronic Colloquium on Computational Complexity, Report No. 60 (2023)

The Planted k-SUM Problem: Algorithms, Lower Bounds, Hardness Amplification, and Cryptography

Sagnik Saha Nikolaj I. Schwartzbach* Prashant Nalini Vasudevan†

April 17, 2023

Abstract

In the average-case k-SUM problem, given r integers chosen uniformly at random from $\{0, \ldots, M-1\}$, the objective is to find a set of k numbers that sum to 0 modulo M (this set is called a "solution"). In the related k-XOR problem, given k uniformly random Boolean vectors of length $\log M$, the objective is to find a set of k of them whose bitwise-XOR is the all-zero vector. Both of these problems have widespread applications in the study of fine-grained complexity and cryptanalysis.

The feasibility and complexity of these problems depends on the relative values of k, r, and M. The *dense* regime of $M \leq r^k$, where solutions exist with high probability, is quite well-understood and we have several non-trivial algorithms and hardness conjectures here. Much less is known about the *sparse* regime of $M \gg r^k$, where solutions are unlikely to exist. The best answers we have for many fundamental questions here are limited to whatever carries over from the dense or worst-case settings.

We study the *planted* k-SUM and k-XOR problems in the sparse regime. In these problems, a random solution is planted in a randomly generated instance and has to be recovered. As M increases past r^k, these planted solutions tend to be the only solutions with increasing probability, potentially becoming easier to find. We show several results about the complexity and applications of these problems.

Conditional Lower Bounds: Assuming established conjectures about the hardness of average-case (non-planted) k-SUM when $M = r^k$, we show non-trivial lower bounds on the running time of algorithms for planted k-SUM when $r^k \leq M \leq r^{2k}$. We show the same for k-XOR as well.

Search-to-Decision Reduction: For any $M > r^k$, suppose there is an algorithm running in time T that can distinguish between a random k-SUM instance and a random instance with a planted solution, with success probability $(1-o(1))$. Then, for the same M, there is an algorithm running in time $\widetilde{O}(T)$ that solves planted k-SUM with constant probability. The same holds for k-XOR as well.

Hardness Amplification: For any $M \geq r^k$, if an algorithm running in time T solves planted k-XOR with success probability $\Omega(1/\mathrm{polylog}(r))$, then there is an algorithm running in time $\widetilde{O}(T)$ that solves it with probability $(1-o(1))$. We show this by constructing a rapidly mixing random walk over k-XOR instances that preserves the planted solution.

Cryptography: For some $M \leq 2^{\mathrm{polylog}(r)}$, the hardness of the k-XOR problem can be used to construct Public-Key Encryption (PKE) assuming that the Learning Parity with Noise (LPN) problem with constant noise rate is hard for $2^{n^{0.01}}$-time algorithms. Previous constructions of PKE from LPN needed either a noise rate of $O(1/\sqrt{n})$, or hardness for $2^{n^{0.5}}$-time algorithms.

Algorithms: For any $M \geq 2^{r^2}$, there is a constant c (independent of k) and an algorithm running in time r^c that, for any k, solves planted k-SUM with success probability $\Omega(1/8^k)$. We get this by showing an average-case reduction from planted k-SUM to the Subset Sum problem. For $r^k \leq M \ll 2^{r^2}$, the best known algorithms are still the worst-case k-SUM algorithms running in time $r^{\lceil k/2 \rceil - o(1)}$.

* Department of Computer Science, Aarhus University. Email: nikolaj@ignatieff.io. This project is funded by VILLUM FONDEN under the Villum Kann Rasmussen Annual Award in Science and Technology under grant agreement no 17911.

† Department of Computer Science, National University of Singapore. Email: prashant@comp.nus.edu.sg. Supported by the National Research Foundation, Singapore, under its NRF Fellowship programme, award no. NRF-NRFF14-2022-0010. Part of the work on this project was done when Sagnik and Nikolaj were visiting the National University of Singapore, and were also supported by this award.

ISSN 1433-8092

Contents

1 Introduction

In the k-SUM problem, given a set of r numbers from $[M]$, the task is to find a set of k of them that sum to 0 (modulo M), if such a set exists.[1] This problem has been central to studying the complexity of important problems in a variety of domains such as computational geometry, data structures, graph theory, etc. [AW14, Pat10, GO95, BHP01, SEO03, KPP16]. It also has several applications in cryptanalysis [Wag02, BCJ11].

The naïve algorithm of iterating through all k-sets takes time $O(r^k)$. The "meet-in-the-middle" algorithm that computes the sums of all $\lceil k/2 \rceil$-sets and looks for collisions runs in time $\widetilde{O}(r^{\lceil k/2 \rceil})$ [HS74]. Better algorithms are known that are faster than $r^{\lceil k/2 \rceil}$ by a few polylog factors [BDP08, GP18, Cha20]. There are also FFT-based algorithms that run in time $\widetilde{O}(M + r)$ [Bri17, JW19], which is faster if $M \ll r^{\lceil k/2 \rceil}$.

The 3-SUM hypothesis of Gajentaan and Overmars [GO95] states that it is not possible to do much better than the above – that any algorithm for the 3-SUM problem in general takes time at least $r^{2-o(1)}$. This hypothesis has been instrumental in establishing conditional lower bounds on the complexities of a variety of interesting problems. More generally, it is conjectured that any algorithm for k-SUM takes time at least $r^{\lceil k/2 \rceil - o(1)}$ [AL13].

Average-Case k-SUM. In the *average-case* k-SUM problem, the r numbers in the input are each chosen uniformly at random from $[M]$. The characteristics of the problem now change depending on the relative sizes of M, r, and k. In analogy to Subset Sum [LO85], we define the *density* of average-case k-SUM as the following ratio:

$$\Delta = \frac{\log \binom{r}{k}}{\log M}.$$

When this density Δ is 1, the expected number of k-SUM solutions in an instance is also 1. In general, the expected number of solutions is approximately $r^{k(1-1/\Delta)}$. For values of Δ larger than 1 (the *dense* regime), this number is polynomial in r, and for smaller values of Δ (the *sparse* regime), it is vanishing with r.

In the dense regime, several non-trivial algorithms are known for average-case k-SUM that are more efficient than the worst-case algorithms. For instance, the "birthday" algorithm that computes the sum of random sets of $k/2$ numbers and looks for collisions runs in expected time $O(\sqrt{M}) = O(r^{k/2\Delta})$. For densities larger than $\approx k/(1 + \log_2 k)$, Wagner's k-tree algorithm [Wag02] solves the problem in time $\widetilde{O}(r)$.

However, at density $\Delta = 1$, when there is only a single solution in expectation, no algorithms are known that outperform the best worst-case algorithms. The average-case k-SUM problem at this density is believed to be as hard as the worst-case variant, although no worst-case to average-case reduction is currently known [Pet15, LLW19, DKK21].

Conjecture 1.1 (Average-Case k-SUM Conjecture)**.** *Any algorithm for average-case k-SUM at density 1 with constant probability of success has running time at least $\Omega\left(r^{\lceil k/2 \rceil - o(1)}\right)$.*

Building on this conjecture, Dinur et al. [DKK21] showed lower bounds on the complexity of k-SUM at densities in the range $(1, 2)$. In particular, their results implied that Wagner's algorithm is optimal for $k = 3, 4$, and 5 for certain densities in this range. LaVigne et al. [LLW19] used a decision version of this conjecture to construct fine-grained One-Way Functions.

The Sparse Regime. In this work, we investigate the average-case complexity of k-SUM at densities smaller than 1, where random instances are unlikely to have k-SUM solutions. While there has been a lot of recent work on average-case k-SUM with densities 1 and higher [LLW19, BSV21, DKK21], very little is known about its complexity in the sparse regime. The natural version of the problem in this regime is *planted* k-SUM, where a randomly chosen set of k numbers that sum to 0 is planted at random locations in a random k-SUM instance. There are two problems to be considered here:

[1] The k-SUM problem is usually defined with the sum being over the integers. These variants are equivalent in complexity in the worst-case, and also in the average-case in certain regimes of parameters (see [BSV21, DKK21]).

- The *planted search* k-SUM problem is to recover a k-SUM solution given such an instance (at low densities, with high probability, the planted solution is the only one)
- The *planted decision* k-SUM problem is to distinguish a random instance with a planted solution from a random instance without a planted solution

We study these and the analogous versions of the k-XOR problem, where the elements in the input are Boolean vectors and the addition operation is replaced with bitwise-XOR.

Apart from being a natural and interesting problem on its own, the planted k-SUM problem in the sparse regime is interesting also for its potential applications to cryptography (some of which we realize in this work). Structured problems where a hidden solution can be planted are often useful in cryptographic constructions [ABW10, LLW19]. An immediately relevant illustration of this may be seen in the case of the Subset Sum problem, which is the unparametrized version of k-SUM where the size of the solution is not restricted. The average-case hardness of the planted Subset Sum problem at very low densities can be used to construct Public Key Encryption [LPS10], a primitive that only a handful of different hardness assumptions are known to imply.

1.1 Our Results

We study the planted k-SUM and k-XOR problems at densities 1 and lower. In either case, the density of an instance consisting of r elements is defined as the ratio $(k \log r / \log M)$, where M is the size of the domain from which the elements in the instance are drawn.[2] We show conditional lower bounds, search-to-decision reductions, hardness amplification, algorithms, and applications of the hardness of these problems to cryptography. Our results are summarized in Fig. 2. In this section, we briefly describe these results and present informal versions of our theorems; see the respective sections for the precise statements.

Conditional Lower Bounds. We start by looking at the search problem, showing an equivalence between planted search k-SUM and regular (non-planted) search k-SUM at any non-sparse density.

Theorem 1.2 (Section 4)**.** *At density $\Delta \geq 1$, any algorithm that solves planted search k-SUM with success probability ϵ also solves non-planted search k-SUM with success probability $\Omega(\epsilon^{3/2})$*

Following the above theorem, the average-case k-SUM conjecture (which is stated for density 1) also implies similar hardness for planted search k-SUM at density 1. This equivalence also implies that, assuming the conjecture, the algorithm that plants random k-SUM solutions into random instances is a fine-grained One-Way Function. We then show reductions for planted search k-SUM from density 1 to lower densities, implying conditional lower bounds on the complexity of the problem at these densities.

Theorem 1.3 (Section 5)**.** *Assuming the average-case k-SUM conjecture, any algorithm that solves planted search k-SUM at density $\Delta \in \left(\frac{1}{2}, 1\right)$ with constant success probability has to take time $\Omega\left(r^{k\left(1-\frac{1}{2\Delta}\right)-o(1)}\right)$*

Our reduction works by lowering the density of an instance by sampling a random subset of the instance, and then trying to use an algorithm for lower density k-SUM on the subset. Another way to lower density is to reduce the value of k in the k-SUM problem being solved, by combining elements of the input, in a similar way to Wagner's k-tree algorithm. This approach gives us similar lower bounds as sampling a subset, though only for some specific densities in the range $\left(\frac{1}{2}, 1\right)$. But interestingly, it gives lower bounds for k-SUM assuming the hardness of k'-SUM for a different k'. The following is an example of such a bound.

Theorem 1.4 (Section 5.2)**.** *Assuming the average-case 5-SUM conjecture, any algorithm that solves planted search 4-SUM at density 4/5 with constant success probability has to take time $\Omega\left(r^{2-o(1)}\right)$*

[2]This ratio is a good approximation to the more meaningful expression for density introduced earlier in this section when k is small, and we use this instead in the rest of the paper for simplicity.

Search to Decision Reduction. Having shown the above results for the search k-SUM problem, we show a search-to-decision reduction for planted k-SUM that, in particular, carries over the conditional lower bounds to the decision k-SUM problem. Note that a reduction in the other direction – from decision to search – is straightforward and conserves the running time and success probability.

Theorem 1.5 (Section 6)**.** *At any density $\Delta < 1$, suppose there is an algorithm that runs in time T and solves planted decision k-SUM with success probability $1 - o(1)$. Then, for every constant $\epsilon < 1$, there is an algorithm that runs in time $\widetilde{O}\left(T + \frac{r}{\Delta}\right)$ and solves planted search k-SUM at the same density with success probability at least ϵ.*

While a known search-to-decision reduction for Subset Sum [IN89] can be extended to show a search-to-decision reduction for k-SUM [ABRV23], our reduction uses a different approach that relies less on the group structure in the problem. We present it here in anticipation of its applicability to other problems.

Following the above lower bounds, our current understanding of the complexity of average-case k-SUM at various densities is depicted in Fig. 1. The results described so far hold not just for k-SUM over integers (modulo some M or otherwise), but also generalize to k-SUM over any suitable family of Abelian groups (see Section 3 for what "suitable" means here). Our remaining results are specific to either k-SUM over integers or the k-XOR problem and make use of the additional structure present in the respective groups.

Hardness Amplification. For the k-XOR problem, we show that the success probability of an algorithm for the planted search k-XOR can be amplified. We do this using a random walk over k-XOR instances that preserves the planted solution (with non-trivial probability) and is also rapidly mixing. Similar techniques have been used to prove direct product theorems in the past [IJKW10]. Hirahara and Shimizu [HS23] use a similar approach to show hardness self-amplification for a number of important problems (see Section 1.3 for a comparison of our techniques).

Theorem 1.6 (Section 7)**.** *At any density $\Omega\left(\frac{1}{\mathrm{polylog}(r)}\right) \leq \Delta \leq 1$, for $k \geq 3$, suppose there is an algorithm that runs in time T and solves planted search k-XOR with success probability $\Omega(1/\mathrm{polylog}(r))$. Then, there is an algorithm that runs in time $\widetilde{O}(T)$ and solves planted search k-XOR at the same density with success probability $1 - o(1)$.*

Public Key Encryption. Next, we show that the hardness of decision k-XOR at sufficiently low densities can be used to construct fine-grained Public-Key Encryption (PKE) assuming the hardness of the Learning Parity with Noise (LPN) problem with constant noise rate. It is known how to construct PKE assuming either that LPN with m-bit secrets at noise rate $O(1/\sqrt{m})$ is hard for $\mathrm{poly}(m)$-time algorithms [Ale03], or that LPN with constant noise rate is hard for $2^{m^{0.5}}$-time algorithms [YZ16]. We show that the hardness of planted k-XOR can be used to weaken the assumption on LPN needed for PKE.

Theorem 1.7 (Section 8)**.** *Under any of the following sets of assumptions about constant-noise LPN on m-bit secrets and planted decision k-XOR on instances of size r, there is a PKE scheme in which encryption and decryption can be done in time $r^{1+o(1)}$, which is secure against adversaries running in time $r^{\lceil k/2 \rceil - \Omega(1)}$:*

- *LPN is 2^{m^c}-hard for some constant c, and k-XOR is $r^{\lceil k/2 \rceil - o(1)}$-hard for all densities $\Delta = 1/\mathrm{polylog}(r)$*
- *LPN is m^c-hard for some $c = \omega(1)$, and k-XOR is $r^{\lceil k/2 \rceil - o(1)}$-hard for all densities $\Delta = 1/r^{o(1)}$*

At even lower densities of $\widetilde{O}(1/r)$, fine-grained PKE can be constructed assuming the hardness of k-SUM alone [ABRV23], by extending PKE constructions based on Subset Sum [LPS10]. Theorem 1.7 also works for super-constant k – assuming of the hardness of k-XOR for such k (even against $r^{o(k)}$-time algorithms) would give a PKE scheme secure against all polynomial-time adversaries. Note that these assumptions about k-XOR are quite strong as not much research has gone into algorithms for this problem at these low densities. Our theorem is to be seen, at this point, primarily as motivation for further research on this subject.

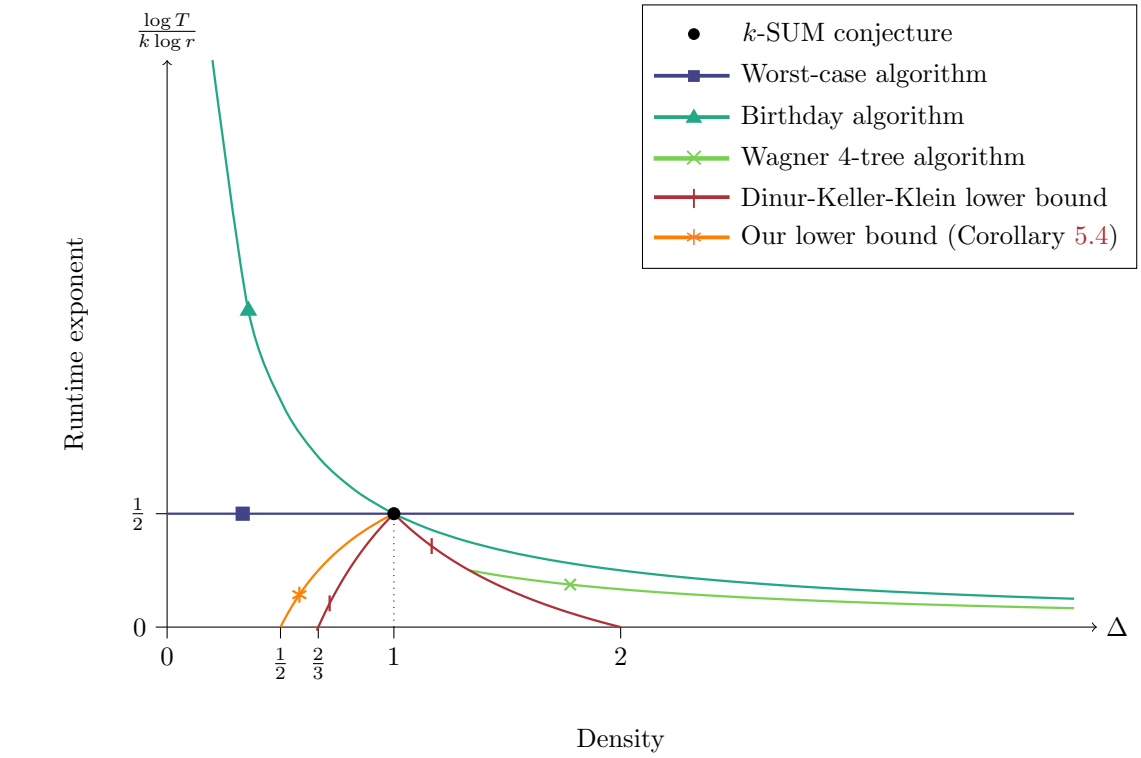

Figure 1: Landscape of the known bounds on the complexity of average-case (planted) k-SUM problems as $k \to \infty$ (except for the Wagner algorithm, which is depicted for $k = 4$). The x-axis represents the density $\Delta = \frac{k \log r}{m}$ of the instances, and the y-axis represents the runtime T, with $y = \frac{\log T}{k \log r}$. More specifically, the y-axis is the exponent of the runtime, such that if the runtime of an algorithm at density Δ is $r^{k\alpha}$, we plot the point (Δ, α). We have omitted the algorithm that works at density $O(\log(r)/r^2)$ presented in Section 9 since its runtime is independent of k.

Algorithms for Very Sparse k-SUM. Proceeding in this direction, we show an average-case reduction from very sparse planted k-SUM over integers to the average-case Subset Sum problem at low densities. Known algorithms for average-case Subset Sum [LO85, Ben22] then give us an algorithm in this regime that outperforms the worst-case k-SUM algorithms. Note that below, c is independent of k.

Theorem 1.8 (Section 9)**.** *For any density $\Delta \leq (k \log(r)/r^2)$, there is a constant $c \in \mathbb{N}$ such that for any $k \in \mathbb{N}$, there is an algorithm that runs in time r^c and solves planted search k-SUM (over integers or with prime moduli) at density Δ with success probability $\Omega(1/8^k)$.*

1.2 Technical Overview

In this section, we give a high-level overview of the techniques we use to show our results.

Equivalence of Planted and Non-Planted k-SUM. We show that any algorithm for planted search k-SUM at density 1 also works for non-planted search k-SUM with a small loss in success probability. This follows from showing that the planted and non-planted distributions at densities 1 and higher are statistically

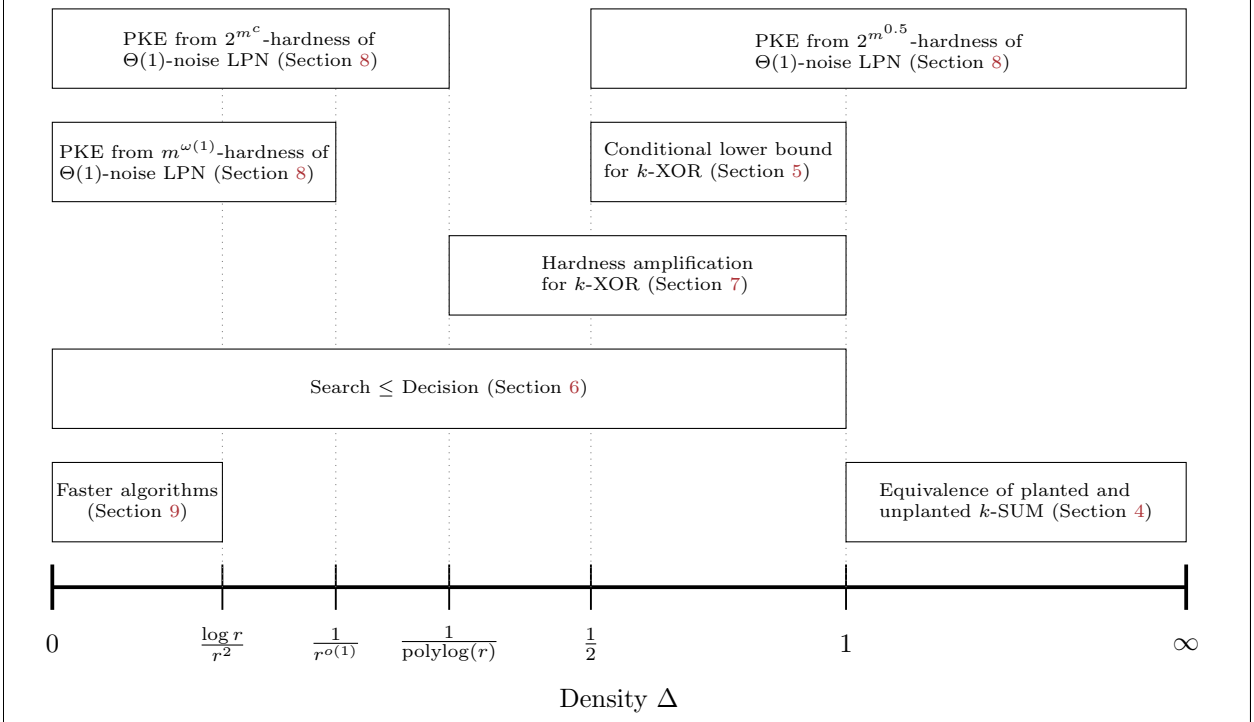

Figure 2: Overview of our results in terms of the density of the generated instances (excluding $\Delta \in \{0, \infty\}$), where r is size (number of elements) of the instance. The x-axis is not to scale. The PKE schemes for $\Delta \leq 1$ can be constructed from the specified hardness of LPN under the *assumption* that k-XOR is hard at the densities depicted on the plot. By contrast, the PKE scheme for $\Delta > 1$ works under no assumptions on the hardness of k-XOR, and for $\Delta \in (\frac{1}{2}, 1]$ under the standard average-case k-SUM conjecture.

close. Let D_0 denote the distribution of uniformly random k-SUM instances, and D_1 the distribution of such instances with random planted k-SUM solutions. We construct a class of hybrid distributions $D^{(\ell)}$ that "interpolates" between these. For $\ell \in [3, r^k]$, the distribution $D^{(\ell)}$ has the following properties:

- $D^{(\ell)}$ has total variation distance at most $1/(\ell-2)^2$ from D_1
- $D^{(\ell)}$ has Rényi divergence (of order ∞) at most $\ell+1$ relative to D_0

So if an algorithm succeeds with probability ϵ on D_1, then it succeeds with probability at least $\epsilon' = (\epsilon - 1/(\ell-2)^2)$ on $D^{(\ell)}$, and thus with probability at least $\epsilon'/(\ell+1)$ on D_0. Picking an appropriate $\ell = \Theta(1/\sqrt{\epsilon})$ then gives us what we want.

The distribution $D^{(\ell)}$ is defined as follows: it samples an instance from X from D_1, checks if the instance at most ℓ solutions: if so, it outputs X, and otherwise it outputs a fresh sample from D_0. The above bounds on the distances are then shown by expressing the probability mass functions of $D^{(\ell)}$ and D_1 in terms of that of D_0, and using bounds on the probability of an instance from D_0 having more than ℓ solutions.

Conditional Lower Bounds. Following the above equivalence, the average-case k-SUM conjecture implies that planted search k-SUM at density 1 needs at least $r^{k/2-o(1)}$ time. Assuming this, we show lower bounds for lower densities. Suppose there is an algorithm $\mathcal{A}$ that runs in time $T(r)$ and solves planted search k-SUM at some density $\Delta < 1$ with constant probability. The idea is, given an instance X at density 1, to convert X

into an instance of density Δ that still contains the planted solution, and then use $\mathcal{A}$ to recover the solution. We implement this in two different ways.

In the first approach, given an instance X consisting of r elements sampled at density 1, we choose a random subset X' of X of size r^Δ (thus reducing the density to Δ) and run $\mathcal{A}$ on X'. If we condition on all elements of the solution planted in X being copied to X' in this process, then X' is distributed identically to a planted k-SUM instance of size r^Δ sampled at density Δ. In this case, $\mathcal{A}$ will find this solution with constant probability (over X). The event we conditioned on happens with probability at least $\Omega(1/r^{k(1-\Delta)})$, and so if we repeat this process $O(r^{k(1-\Delta)})$ times, it happens at least once with constant probability and we can find a solution in X. The k-SUM conjecture now implies that $r^{k(1-\Delta)} \cdot T(r^\Delta) \geq r^{k/2}$. This, in turn, implies that $T(r) \geq r^{k(1-\frac{1}{2\Delta})}$, which is the bound we show.

The second approach is to reduce the density by combining elements in the input and reducing the k in the k-SUM problem being considered, not unlike the Wagner k-tree algorithm. For example, given instance X for planted k-SUM at density 1, randomly choose $r/4$ disjoint pairs of elements to remove from the instance, compute their sum and put the result back in to get instance X'. If it happened that, out of the k elements in the solution in X, two were picked as one of these pairs to be combined and the remaining were left untouched, then this leads to a set of $(k-1)$ elements in X' that sum to 0. Seen as an instance of $(k-1)$-SUM, X' has density $\approx (1-1/k)$, and an algorithm solving it can be used to solve X. Computing the probability of this happening then leads to a similar lower bound of $r^{k(1-\frac{1}{2\Delta})}$ for density Δ, with two important differences. First, it translates between different values of k, inferring lower bounds on k'-SUM from the hardness of k-SUM for $k' \neq k$. Second, it only works for a discrete set of densities for a given value of k, whereas the first lower bound works "continuously" as depicted in Fig. 1.

Search-to-Decision Reduction. We show that an algorithm for decision k-SUM at density less than 1 with large enough success probability can be used to solve planted search k-SUM. Given a planted instance X, we use the decision algorithm to obtain information on where the planted solution is using a non-recursive binary search. Consider replacing half of the elements in X with random elements, and running the decision algorithm on the resulting instance X'. If none of the k elements of the planted solution were replaced in this process, then X' is identical to a random planted k-SUM instance. If any of the elements of the planted solution were replaced, then X' is identical to a random non-planted k-SUM instance. We use the ability of the decision algorithm to distinguish between these distributions to work out where the k elements of the solution are.

We perform the above replacement $\text{polylog}(r)$ times, starting from the original input each time. We maintain a counter C_i for each element of the input that counts the number of iterations in which this element was not replaced and the decision oracle reported there was a solution. We show that the expected value of this counter is noticeably larger for elements that are part of the planted solution. A concentration bound for the counters then implies that, with high probability at the end of the iterations, the k largest counters correspond to the elements in the planted solution.

Hardness Amplification. We show that an algorithm that solves planted k-XOR recovery at density in the range $(1/\text{polylog}(r), 1]$ with probability $\Omega(1/\text{polylog}(r))$ in time T implies an algorithm that solves it at the same density with probability $(1-o(1))$ in time $\widetilde{O}(T)$. We briefly describe our approach here. For simplicity, we will start with the stronger assumption that there is an algorithm $\mathcal{A}$ that solves planted search k-XOR with probability $\Omega(1)$ and, further, is deterministic.

Let $T_\mathcal{A} \subseteq (\mathbb{F}_2^m)^r$ be the set of k-XOR instances for which $\mathcal{A}$ correctly finds a solution; note that $T_\mathcal{A}$ consists of an $\Omega(1)$ fraction of *planted* instances (if the density is sufficiently less than 1). Our approach, given an instance $X \in (\mathbb{F}_2^m)^r$, is to find an $X' \in T_\mathcal{A}$ such that a solution for X can be recovered from $\mathcal{A}(X')$. If there is an efficient procedure that finds such an X' given X and fails for at most a $o(1)$-fraction of X's, then this would prove the required amplification. We do this using the following process.

Walk(X, t):

1. Set $X^0 \leftarrow X$

2. For i from 1 to t:

   2.1. Sample $j \leftarrow [r]$

   2.2. Replace the j^{th} element $X^{i-1}[j]$ with a random element $x \leftarrow \mathbb{F}_2^m$ such that $x \neq X^{i-1}[j]$

   2.3. Set X^i to be the resulting instance

3. Output X^t

Consider a graph where each vertex corresponds to an instance in $(\mathbb{F}_2^m)^r$, with an edge between two vertices iff they differ in exactly one column. This is a well-studied graph known as the Hamming graph, here defined over length-r strings and alphabet of size 2^m. The above process is a t-step random walk on this graph starting from the vertex corresponding to X. The expansion properties of the Hamming graph imply that random walks of length $\omega(r)$ mix quite well. In other words, with $t = \omega(r)$, for any sets S and T that each contain an $\Omega(1)$ fraction of instances, at least an $\Omega(1)$ fraction of t-step random walks that start from S end in T.

With $T = T_{\mathcal{A}}$, this is reminiscent of what we want – by the above property, if $T_{\mathcal{A}}$ contains an $\Omega(1)$-fraction of instances, then the set of instances X from which a constant fraction of walks *do not* lead to $T_{\mathcal{A}}$ has to be of relative size $o(1)$. There are some issue here, though – first, this random walk does not preserve solutions so it is not clear how to use it to solve X; and second, this graph mostly consists of *non-planted* instances, and $T_{\mathcal{A}}$ does not actually contain a constant fraction of these. We deal with both of these by considering a conditioning of this random walk.

For simplicity, we restrict our attention to planted instances X that have a unique solution. Denote by $\text{span}(X)$ the set of all instances X' such that the solution in X' appears at the same locations and consists of exactly the same elements as that in X. Now, conditioning on all the X^i's being contained in $\text{span}(X)$, the process $\text{Walk}(X, t)$ is again a t-step random walk over a Hamming graph, this time defined over $\text{span}(X)$. This conditioned random walk does preserve the solution of X, as X^t is now in $\text{span}(X)$. Further, for densities less than 1, with high probability no additional solutions are introduced during this walk.

Suppose the fraction of instances in $\text{span}(X)$ that are contained in $T_{\mathcal{A}}$ is at least $\Omega(1)$. Then the set of $X' \in \text{span}(X) = \text{span}(X')$ for which an $\Omega(1)$-fraction of conditioned t-step random walks starting from X' *do* end in $T_{\mathcal{A}}$ is of relative size at least $(1 - o(1))$. For each such X', the event we conditioned on happens with probability at least $(1 - k/r)^t$. So for all but a $o(1)$ fraction of $X' \in \text{span}(X)$, the unconditioned t-step random walk starting from X' ends in $T_{\mathcal{A}}$ with probability at least $\Omega((1 - k/r)^t)$. We can set $t = r \cdot \log\log r$ so that it is large enough for the walk to mix, and also $(1 - k/r)^t = 1/\text{polylog}(r)$ is not too small so that success can then be amplified by repetition.

It remains to show that the fraction of instances in $\text{span}(X)$ contained in $T_{\mathcal{A}}$ is at least $\Omega(1)$. We show that this property can be achieved for all but a $o(1)$-fraction of instances X by obfuscating the solution. Our obfuscation works by multiplying the elements of X by a random full-rank matrix and permuting the elements of the resulting instance before running $\mathcal{A}$ on it. This hides most properties of the solution and ensures that for most instances X, the fraction of $\text{span}(X)$ that is solved correctly by the algorithm is the same, and hence is at least $\Omega(1)$. This argument works at every density $\leq 1 - \frac{\log\log r}{\log r}$, and we extend the result to density 1 using a couple of other reductions.

Public-Key Encryption. We present a Public-Key Encryption (PKE) scheme whose security is based on the hardness of the planted k-XOR problem at low densities and on the hardness of Learning Parity with Noise (LPN) with constant noise rate. The scheme is similar to the dual Regev encryption [Reg09, GPV08], which uses the hardness of the Learning With Errors problem.

Key Generation: To generate a public key, Alice samples a planted k-XOR instance $X \in \mathbb{F}_2^{m \times r}$ at density Δ (to be determined later), with $m = k \log(r)/\Delta$. The secret key is the location $K \subseteq [r]$ of the planted solution. We denote also by $K \in \mathbb{F}_2^r$ the characteristic vector corresponding to the set K.

Encryption: To encrypt a bit $b = 0$, Bob simply samples a random vector $c \leftarrow \mathbb{F}_2^r$ and sends c to Alice. To encrypt $b = 1$, he computes a random linear combination of the rows of X, adds noise to the entries and

sends the result to Alice. Specifically, he samples a random vector $s \in \mathbb{F}_2^m$ as well as a Bernoulli-noise vector Ber_η^r (for some constant $\eta = \Theta(1)$) and computes the ciphertext $c \leftarrow s^\top X + e$.

Decryption: The idea is that Alice, knowing the location $K \in \mathbb{F}_2^r$ of the planted solution, may compute the inner product $c \cdot K$: if Bob encrypted a 0, the result is a random $\mathrm{Ber}_{\frac{1}{2}}$; if he encrypted a 1, unless the noise destroyed the solution, the result will be zero. So Alice guesses that Bob encrypted a one, if $c^\top K = 0$ and guesses zero otherwise. This process can be amplified to achieve vanishing decryption error by repeating it $\ell = \exp(k) \cdot \omega(\log r)$ times.

By the hardness of k-XOR, the public key X generated by Alice is indistinguishable from a random matrix $Y \in \mathbb{F}_2^{m \times r}$ *without* a planted k-XOR solution. If such a Y was the public key, the hardness of LPN then implies that encryptions of 0 and 1 are indistinguishable. Security for the ℓ repetitions then follows from a hybrid argument.

Suppose that LPN with constant noise rate, say $\eta = 1/3$, is hard for $T(m)$-time algorithms. We then consider different ways of instantiating k and m to strike a trade-off between the hardness required for LPN and the density at which k-XOR is assumed to be hard, given the level of security we want. For instance, let us assume that LPN with constant noise is hard for $2^{m^{0.01}}$-time algorithms and determine at what density we need to assume that k-XOR is hard. Suppose that k-XOR takes time $r^{k/2}$ time at some density Δ. We want to have $T(m) \geq r^{k/2}$ for the LPN hardness to match the security afforded by k-XOR. This implies that we must have $m^{0.01} \geq k \log(r)/2$, which when combined with the identity $m = k \log(r)/\Delta$ implies that $\Delta \leq \frac{2^{100}}{(k \log(r))^{99}} = \frac{1}{\mathrm{polylog}(r)}$. So assuming $2^{m^{0.01}}$-hardness for LPN and hardness of k-XOR at some $1/\mathrm{polylog}(r)$ density gives us PKE secure against $r^{k/2}$-time adversaries. The key generation, encryption, and decryption all run in time $r^{1+o(1)}$. So if we make these assumptions for super-constant k, this gives us PKE that is secure against all polynomial-time adversaries.

More generally, we get PKE from 2^{m^c}-hardness of constant-noise LPN (for any constant $c > 0$), assuming k-XOR is hard at every density $1/\mathrm{polylog}(r)$. A similar argument allows us to construct PKE from m^c-hardness of constant-noise LPN for any $c = \omega(1)$, assuming k-XOR is hard at every density $1/r^{o(1)}$.

Algorithms at Very Low Densities. We show a general reduction from the average-case planted k-SUM problem modulo p (where p is prime) at density $\frac{k\Delta \log r}{r}$ to the average-case Subset Sum problem modulo p at density Δ whenever $\Delta < 1/2$. Polynomial-time algorithms are known for the average-case Subset Sum problem at densities lower than $1/r$ [LO85, Ben22]. This immediately implies an algorithm for k-SUM modulo p at density at or below $\frac{k \log r}{r^2}$ whose runtime complexity is a fixed polynomial in r independent of k. This is asymptotically much faster than existing worst-case k-SUM algorithms. Following known reductions from k-SUM over integers to modular k-SUM [DKK21], this also implies an algorithm for average-case k-SUM over integers at the same density with the same runtime.

In the Subset Sum problem modulo p, given r numbers from $[p]$ and a target t, the task is to find some subset of them (of unrestricted size) that sums to t modulo p. In average-case Subset Sum, the r numbers are chosen uniformly at random, and t is set to be the sum of a random subset of these. A naïve attempt at a reduction simply passes the k-SUM instance directly to the Subset Sum algorithm, with target sum 0. At low enough densities, with high probability the only subset that sums to 0 would indeed be the k-SUM solution. The problem with this reduction is that the resulting Subset Sum instance doesn't look like a randomly sampled instance at all. We modify this reduction in a couple of different ways to rectify that.

First, we add to each element in the k-SUM instance (call it A) a randomly chosen element α in $\mathbb{Z}_p$. This shifts the target sum by $k\alpha$ since we know that the subset we are looking for has size k. Note that A was already chosen uniformly at random before replacing exactly one element by the negated sum of $k-1$ other elements (corresponding to planting a k-SUM solution). As a result of adding α to everything, this becomes equivalent to replacing one element by $k\alpha -$ (the sum of $k-1$ other elements). Since α is random and p is prime, this is equivalent to randomly sampling a new element. Thus, we have ensured that the array of numbers passed to the Subset Sum oracle has a uniform distribution.

We still have the issue that the solution we are looking for (call it K) has length exactly k, and our Subset Sum algorithm could be biased against such solutions. To fix this, we choose a completely random

subset $S \subseteq A$, and add $\sum_{e \in S} e$ to the target. If S and K are disjoint (which happens with probability 2^{-k}), we end up with an instance of Subset Sum that is closer to being from the right distribution. The solution we are looking for is now $S \bigcup K$. While this conditional distribution does not actually have the distribution of a random Subset Sum instance, we can show that it is statistically close.

These two modifications now guarantee that a random instance of k-SUM gets mapped to an almost random instance of Subset Sum, which the provided Subset Sum algorithm must still be able to solve with constant success probability. This completes the reduction.

1.3 Related Work

The worst-case complexity of the k-SUM problem has been studied extensively in the field of fine-grained complexity due to its reductions to a large number of other interesting problems [GO95, BHP01, SEO03, BDP08, Pat10, AW14, KPP16, DSW18, Cha20, ...]. We refer the reader to the survey by Williams [Wil18] for details. The complexity of the k-SUM problem in other computational models has also been studied, and it is known to have non-trivial decision trees [GS17, GP18], non-deterministic algorithms [CGI+16], and lower bounds in some of these models [Eri95, AC05]. Questions regarding data structures for it have also been studied [KP19, GGH+20, CL23].

Some conditional bounds for worst-case k-SUM are known in certain settings. For super-constant k, an algorithm that runs in time $r^{o(k)}$ would contradict the Exponential Time Hypothesis (ETH) [PW10]. And an algorithm for k-SUM with numbers in the range $[M]$ that runs in time $M^{1-\Omega(1)}$ would contradict the Strong Exponential Time Hypothesis (SETH) [ABHS19].

Average-Case k-SUM. Average-case k-SUM and k-XOR in the dense regime have several applications in cryptanalysis and has been the subject of substantial work in the area, most involving better algorithms and applications [Wag02, MS12, NS15, Nan15, Din19, LS19, BDJ21].

More recently, different conditional lower bounds have been shown in this regime. Brakerski et al. [BSV21] show that Wagner's algorithm is near-optimal for k-SUM at large densities as k tends to infinity, using reductions from worst-case lattice problems. Dinur et al. [DKK21], as discussed above, show lower bounds at densities in $(1, 2)$ assuming the k-SUM conjecture at density 1. Dalirrooyfard et al. [DLW20] show the average-case hardness of counting solutions in a "factored" version of k-SUM assuming SETH. They also show search-to-decision reductions for the average-case Zero-k-Clique problem.

The study of average-case fine-grained complexity in general has proliferated in the past few years [BRSV17, DLW20]. Of particular relevance here is line of work on worst-case to average-case reductions for counting k-cliques, which focuses on reducing from and to the same problem [GR18, BBB19]. The general paradigm of looking for small hidden solutions in random instances is common in problems studied in statistical inference, such as planted clique, Sparse PCA, etc. [Jer92, BR13b, BR13a, GZ19]. Worst-case versions of these problems have also been subjects of interest in fine-grained complexity [Wil18, GV21].

Hardness Amplification. Approaches similar to ours for hardness amplification have been used to prove direct product theorems in the past [IJKW10], but its use in amplifying the hardness of a fixed natural problem is new. In concurrent independent work, Hirahara and Shimizu [HS23] use a similar framework to show hardness amplification for the planted clique problem, triangle counting, matrix multiplication, and online matrix-vector multiplication. We briefly describe below the high-level similarities and differences in our approaches.

Our approach to amplifying the hardness of planted search k-XOR is as follows. Given an instance, we perform a random walk over instances of the same size where each step consists of adding some noise to the instance and then randomizing it in a way that preserves solutions. We show that the graph defined over the instances by these steps has sufficient expansion properties for the random walk to mix well before the noise added destroys the initial solution. Then, for most instances as starting point, with a large enough probability, the random walk leads to an instance that still has the original solution and at which the weak average-case algorithm is correct.

Hirahara and Shimizu's approach, roughly, is to embed the given instance in a randomized instance of larger size – note that this never destroys the original solution. They then show, in each of their reductions, that the bipartite graph that captures this random embedding has sufficient expansion properties that again, with most instances as starting point, with a large enough probability, taking a random edge on the bipartite graph leads to a larger instance at which the weak average-case algorithm is correct. This approach is closer to that of Impagliazzo et al. [IJKW10], who also relied similarly on bipartite graphs with expansion properties.

The approach in [HS23] is likely to be more generally applicable because it never destroys solutions. In problems where solutions are larger (rather than just a constant k out of r elements), it is harder to repeatedly randomize instances the way we do without destroying solutions. One benefit of our approach, at least in the case of k-SUM, is that the graph that results from our steps is much simpler and easier to analyze than the graph that would have resulted from their approach.

Fine-Grained Cryptography. The question of constructing cryptographic primitives with fine-grained security guarantees assuming fine-grained hardness conjectures has been studied alongside average-case fine-grained complexity [BRSV18, LLW19, BC22]. LaVigne et al. [LLW19] use an assumption about the hardness of decision k-SUM to construct a fine-grained One-Way Function. They also construct fine-grained Public-Key Encryption (with quadratic security) assuming the average-case hardness of the Zero-k-Clique problem. Juels and Peinado [JP00] similarly constructed One-Way Functions from the conjectured hardness of planted clique for certain parameters.

1.4 Open Problems

There are a number of questions raised by our work that remain to be answered. Answering any of these is likely to involve interesting new ideas and lead to better understanding of the average-case complexity of fundamental problems.

1. Are there algorithms for planted k-SUM at densities in $\left(k \log r/r^2, 1\right)$ that are better than the worst-case algorithms?
2. Can our conditional lower bounds be improved? In particular, could similar bounds be shown for densities smaller than $1/2$?
3. Similarly, can conditional lower bounds for search k-SUM be shown for densities larger than 2?
4. Is there a fine-grained reduction from worst-case k-SUM to average-case k-SUM at any density?
5. Can our approach to hardness amplification (or that of [HS23]) be applied to other problems in fine-grained complexity?

Paper Organization

The paper is organized as follows. In Section 3, we formally define the planted search and decision k-SUM problems. In Section 4, we establish an equivalence between planted k-SUM and non-planted k-SUM at density 1. In Section 5, we establish our conditional lower bounds for sparse k-SUM. In Section 6, we show a search-to-decision reduction for k-SUM. In Section 7, we present a success amplification theorem for search k-XOR. In Section 8, we propose a public-key encryption scheme based on the hardness of sparse planted k-XOR and LPN. Finally, in Section 9, we describe our algorithm for very sparse k-SUM by reducing the problem to Subset Sum.

2 Preliminaries

We denote by $\log x$ the base-2 logarithm of x. We denote by $[n] = \{1, 2, \ldots, n\}$ the set containing the first n positive integers. We use the notation $X \xleftarrow{\$} G$ to denote that X is sampled uniformly from G when G is

finite. We let $\mathbb{1}[\cdot]$ be the indicator variable for the validity of the statement in the brackets, with 1 denoting true and 0 denoting false. If A, B are two sets, we denote by $A \Delta B = (A \cup B) \setminus (A \cap B)$ the symmetric difference between A, B.

We say a function $f(\cdot)$ is *negligible* if it grows slower than the inverse of any polynomial, i.e. if for any constant c, it holds that $f(x) = o(x^c)$. We denote by negl a generic negligible function.

Probability theory. If D is a probability distribution on a countable set Ω, and $X \in \Omega$, we denote by $D(X)$ the probability mass of D on X. We use the notation $X \sim D$ to denote that X is sampled according to D. If D, D' are two probability distributions, we denote by $SD(D, D')$ the total variation distance, defined as,

$$SD(D, D') = \frac{1}{2} \sum_{X \in \Omega} |D(X) - D(X')|.$$

It is well-known that the total variation distance gives an upper bound on the advantage of distinguishing between two probability distributions.

Lemma 2.1 (Rényi divergence, [BLRL+18])**.** *Let P, Q be two probability distributions, with $\mathrm{supp}(P) \subseteq \mathrm{supp}(Q)$, and let $E \subseteq \mathrm{supp}(Q)$ be an event. Then,*

$$Q(E) \geq P(E)/R(P\|Q),$$

where $R(P\|Q)$ is the Rényi divergence (of order ∞), defined as,

$$R(P\|Q) = \max_{x \in \mathrm{supp}(P)} \frac{P(x)}{Q(x)}.$$

The Rényi divergence between two distributions can be used to obtain multiplicative bounds on the success probabilities of average-case algorithms whose inputs are sampled from those distributions.

We denote by Ber_η the Bernoulli distribution on support $\mathbb{F}_2$ with parameter η. In a similar vein, we let Ber_η^r be the distribution of r i.i.d. Bernoulli distributions with support $\mathbb{F}_2^r$ where $X \sim \mathrm{Ber}_\eta^r$ means that $X_i \sim \mathrm{Ber}_\eta$ and that X_i and X_j are independent for $i \neq j$, and likewise for $\mathrm{Ber}_\eta^{m \times r}$ with support $\mathbb{F}_2^{m \times r}$.

Concentration bounds. We will make use of a variety of concentration bounds that we include here for the purpose of self-containment. Markov's inequality gives concentration of a non-negative random variable in terms of its first moment.

Lemma 2.2 (Markov's inequality, [SS05])**.** *Let X be a non-negative random variable. Then for every $t > 0$,*

$$\Pr[X > t\,\mathbb{E}[X]] < \frac{1}{t}.$$

Chebyshev's inequality bounds it in terms of its second moment.

Lemma 2.3 (Chebyshev's inequality, [Tch67])**.** *Let X be a random variable with finite variance. Then for every $t > 0$,*

$$\Pr[|X - \mathbb{E}[X]| > t\,\mathrm{Std}[X]] < \frac{1}{t^2}.$$

where $\mathrm{Std}[X] = \sqrt{\mathrm{Var}[X]}$ is the standard deviation of X.

The Paley-Zygmund inequality gives an anticoncentration bound it in terms of its first two moments.

Lemma 2.4 (Paley-Zygmund inequality, [PZ32])**.** *Let X be a non-zero random variable with finite variance. Then for every $c \in [0, 1]$,*

$$\Pr[X > c\,\mathbb{E}[X]] \geq (1 - c)^2 \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

A slightly stronger (and rewritten) version of the inequality is as follows.

$$\Pr[X > c\,\mathbb{E}[X]] \geq \frac{(1-c)^2\,\mathbb{E}[X]^2}{\mathrm{Var}[X] + (1-c)^2\,\mathbb{E}[X]^2}$$

Finally, the Chernoff bounds gives strong concentration for the mean of n i.i.d. 0-1 random variables.

Lemma 2.5 (Chernoff bound, [Che52])**.** *Let $X_1, X_2, \ldots, X_n$ be i.i.d random variables on $\{0,1\}$, and let $X = \sum_{i=1}^{n} X_i$. Then for every $\varepsilon > 0$,*

$$\Pr[X > (1+\varepsilon)\,\mathbb{E}[X]] < e^{-\frac{\varepsilon^2\,\mathbb{E}[X]}{2}},$$
$$\text{and,} \quad \Pr[X < (1-\varepsilon)\,\mathbb{E}[X]] < e^{-\frac{\varepsilon^2\,\mathbb{E}[X]}{2}}.$$

Similarly, the following also holds,

$$\Pr\left[\frac{1}{n}X > \frac{1}{n}\mathbb{E}[X] + \varepsilon\right] < e^{-2\varepsilon^2 n},$$
$$\text{and,} \quad \Pr\left[\frac{1}{n}X < \frac{1}{n}\mathbb{E}[X] - \varepsilon\right] < e^{-2\varepsilon^2 n}.$$

Spectral graph theory We will analyze one of our constructions by representing it as a graph and obtain bounds on its edge expansion to argue correctness. Formally, an undirected graph $G = (V, E)$ consists of a set of n vertices V, with $|V| = n$, and a set of m edges $E \subseteq V \times V$, such that $(u, v) \in E$ iff $(v, u) \in E$. Let n denote the number of nodes, and m the number of edges. If $A, B \subseteq V$, we denote by $E(A, B)$ the set of edges connecting A and B, i.e. $(u, v) \in E(A, B)$ iff $u \in A, v \in B$ and $(u, v) \in E$. The degree of a node is the number of edges that includes it. A graph is said to be d-regular if all nodes have degree d. The graph may also be represented using its adjacency matrix $A \in \mathbb{F}_2^{n \times n}$. Fix any ordering of the vertices and let (i, j) denote the edge between the i^{th} and the j^{th} node. With slight overload of notation, we let G refer also to the $n \times n$ matrix defined as $G_{ij} = \mathbb{1}[(i, j) \in E]$. A *multigraph* is a graph that is allowed to have multiple edges between the same nodes. We may represent such graphs using matrices of the form $A \in \mathbb{N}^{n \times n}$, where the value A_{ij} represents the number of edges from i to j. A graph remains a special case of a multigraph where $A_{ij} \in \{0, 1\}$ for every $i, j \in [n]$.

Let G be a multigraph with adjacency matrix A. We associate to G the eigenvalues of A. Now, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the eigenvalues of G. We then define the *algebraic expansion* as $\lambda(G) = \max_{i=2\ldots n} |\lambda_i| = \max(|\lambda_2|, |\lambda_n|)$. The algebraic expansion measures the extent to which nodes are connected, with smaller values of $\lambda(G)$ meaning a graph that is more connected. In particular, the following lemma allows us to lower bound the number of edges between any two sets of vertices in terms of $\lambda(G)$.

Lemma 2.6 (Expander Mixing Lemma, [AC88])**.** *Let $G = (V, E)$ be a d-regular graph, and let $S, T \subseteq V$. Then,*

$$\left| E(S,T) - \frac{d \cdot |S| \cdot |T|}{|V|} \right| \leq \lambda(G)\sqrt{|S| \cdot |T|}$$

where $\lambda(G)$ is the algebraic expansion of the graph.

Hamming Graphs Fix a set Q with $|Q| = q$. The *Hamming graph* $H(d, q)$ is defined as the graph (V, E) whose vertex set $V = Q^d = Q \times Q \times \cdots \times Q$ is the Cartesian product of Q with itself d times, where $(u, v) \in E$ if u and v differ in precisely one coordinate, i.e. if there is an index $j \in [d]$ such that $u_i = v_i$ if and only if $i = j$. The graph $H(d, q)$ is a regular graph of diameter d, whose eigenvalues can be characterized as follows.

Lemma 2.7 (Hamming Graph Eigenvalues, [DKT16])**.** *The i^{th} eigenvalue of the adjacency matrix of $H(d, q)$ satisfies,*

$$\lambda_i(H(d,q)) = (q(d-i) - d)^{\binom{d}{i}(q-1)^i}.$$

3 The Planted k-SUM Problem

We now formally define the average-case k-SUM problem over general Abelian groups, and present existing hardness conjectures for certain interesting groups. We start with a framework for discussing the general groups in this setting, and some descriptive quantities we will use for them.

Group Ensembles. We fix some underlying countably infinite sequence of finite Abelian groups $\mathsf{G} = \{G^{(r)}\}_{r\in\mathbb{N}^+}$ that we refer to as a *group ensemble*. Informally, an instance of "size" r of the k-SUM problem over G will consist of r elements chosen uniformly at random from the group $G^{(r)}$. With slight abuse of notation, we denote the group operation in all of these groups by $+$ (thus removing its dependency on r), and trust that it is clear from the context to which group it belongs.

Definition 3.1. *For any $k \in \mathbb{N}$ and group ensemble $\mathsf{G} = \{G^{(r)}\}$, we define the* k-SUM density *of the r^{th} group as,*

$$\Delta_k^{(r)}(\mathsf{G}) = \frac{\log \binom{r}{k}}{\log |G^{(r)}|}. \tag{1}$$

We then define the k-SUM density *of the ensemble G as the limit of $\Delta^{(r)}$ as r tends to infinity, i.e.,*

$$\Delta_k(\mathsf{G}) = \lim_{r\to\infty} \Delta_k^{(r)}(\mathsf{G}). \tag{2}$$

When k is clear from the context, we will simply refer to the above quantity as the density of G*, and denote it by $\Delta(\mathsf{G})$ or even Δ.*

We only work with group ensembles that have well-defined density, though many of our techniques can be applied to specific groups (rather than all groups in an ensemble) if needed. We will also need the groups in the ensembles to be efficiently sampleable and have group operations that can be efficiently performed. This is both so that hard problems defined over them can be used, and because our reductions sometimes need to sample random group elements.[3]

Definition 3.2 (Admissibility)**.** *For $k \in \mathbb{N}$, a group ensemble $\mathsf{G} = \left\{G^{(r)}\right\}$ is* admissible for k-SUM *if it satisfies the following properties:*

- Efficient sampling: *There exists an algorithm that, on input $r \in \mathbb{N}$, samples a uniformly random group element from $G^{(r)}$ and runs in time $O\left(\text{polylog}\left|G^{(r)}\right|\right)$*
- Efficient operations: *There exists an algorithm that, on input $r \in \mathbb{N}$ and group elements $g, h \in G^{(r)}$, outputs the result of the corresponding group operation on g and h, and runs in time $\widetilde{O}\left(\log\left|G^{(r)}\right|\right)$*
- Convergent density: *$\Delta_k(\mathsf{G})$ exists and is finite.*

All our statements are to be taken to be made only for ensembles that are admissible for k-SUM for k that will be clear from the context, and we leave out this specification in the rest of the paper.

Special Ensembles. We now define two classes of group ensembles that will be of particular interest to us. Each class is parameterized by the density Δ of the ensemble. The first is the class of modular k-SUM ensembles, i.e., ensembles associated with the k-SUM problem modulo some integer. The ensemble corresponding to density $\Delta > 0$ is defined as follows.

$$\mathsf{G}_{k\text{-SUM}}^{(\Delta)} = \left\{\mathbb{Z}_{2^{m(r)}}\right\}_{r\in\mathbb{N}}, \quad \text{where } m(r) = \left\lceil \frac{k \log r}{\Delta} \right\rceil. \tag{3}$$

[3]Note that $\log\left|G^{(r)}\right|$ is the number of bits required to represent elements of $G^{(r)}$, and we ask that random elements be sampleable in time quasi-linear in this. This asks for a *uniform* algorithm that samples elements for any $G^{(r)}$. All the theorems stated in the paper are for uniform algorithms. All of our reductions are uniform except where they use this group sampler and compute group operations. So if the group ensembles in consideration only have non-uniform samplers and non-uniform algorithms for group operations, the non-uniform versions of our theorems are still true for them.

Another class that we will pay special attention to is the class of k-XOR group ensembles, i.e. k-SUM defined over GF_{2^m} for an approriately chosen m. We define it as follows.

$$\mathsf{G}^{(\Delta)}_{k\text{-XOR}} = \{GF_{2^{m(r)}}\}_{r\in\mathbb{N}}, \quad \text{where } m(r) = \left\lceil \frac{k \log r}{\Delta} \right\rceil. \tag{4}$$

The Non-Planted k-SUM Problem. Fix some $k \in \mathbb{N}$, a group ensemble $\mathsf{G} = \{G^{(r)}\}_{r\in\mathbb{N}}$, and define the related ensemble of "null distributions" as follows.

Distribution $D_0^{(r)}$

1. Sample r group elements $X_1, X_2, \ldots, X_r$ i.i.d. uniformly at random from $G^{(r)}$
2. Return $X = (X_1, \ldots, X_r)$

In the (non-planted) search k-SUM problem, given such an X, the task is to find a set of k elements in X that sum to zero (the identity element of the group).

Definition 3.3 (Non-Planted Search k-SUM)**.** *For $k \in \mathbb{N}$ and an ensemble G, an algorithm $\mathcal{A}$ is said to solve the (non-planted)* search k-SUM problem over G *with success probability ϵ if, on input an instance X of size r it outputs an $S = \mathcal{A}(X) \subseteq [r]$ with $|S| = k$ such that:*

$$\Pr_{\substack{X\sim D_0^{(r)} \\ S\leftarrow\mathcal{A}(X)}}\left[\sum_{i\in S} X_i = 0\right] \geq \epsilon$$

If $\epsilon = \Omega(1)$, we simply say that $\mathcal{A}$ solves the search k-SUM problem over G.

We will refer to any set S of size k that satisfies $\sum_{i\in S} X_i = 0$ as a *k-SUM solution* for X. Not all instances X necessarily have a k-SUM solution. However, if G has density 1, there is (asymptotically) at least a constant probability that X drawn from D_0 has at least one k-SUM solution. Such a solution, can be found in time $\widetilde{O}(r^{\lceil k/2\rceil})$ using a simple meet-in-the-middle algorithm. So for any ensemble G of density 1, there is an algorithm that runs in time $\widetilde{O}(r^{\lceil k/2\rceil})$ and solves search k-SUM over G with success probability $\Omega(1)$.

For certain ensembles of density 1, it is conjectured that it is not possible to do much better than this. That is, that there is no algorithm that is significantly faster that still solves the search k-SUM problem with constant success probability. The following conjectures were formalized by Dinur et al. [DKK21], where they are stated to be folklore.[4] Weaker versions appear in [LLW19] and [Pet15].

Conjecture 3.4 (Average-Case k-SUM Conjecture)**.** *For any $k \in \mathbb{N}$, any algorithm that solves (non-planted) search k-SUM over $\mathsf{G}^{(1)}_{k\text{-}SUM}$ with constant success probability has expected running time at least $\Omega(r^{\lceil k/2\rceil - o(1)})$.*

Conjecture 3.5 (Average-Case k-XOR Conjecture)**.** *For any $k \in \mathbb{N}$, any algorithm that solves (non-planted) search k-SUM over $\mathsf{G}^{(1)}_{k\text{-}XOR}$ with constant success probability has expected running time at least $\Omega(r^{\lceil k/2\rceil - o(1)})$.*

The Planted k-SUM Problem. We now define a different distribution — the planted distribution — where, again we sample a random instance, but now we additionally plant a solution at random before outputting it. We may define this process formally as follows.

[4] To be accurate, Dinur et al. state their conjecture for the k-SUM problem where the sum is performed over integers (rather than modulo some number as in $\mathsf{G}_{k\text{-SUM}}$). They show, however, that k-SUM over integers is equivalent to k-SUM over $\mathsf{G}_{k\text{-SUM}}$ at approximately the same density, roughly implying the conjecture above.

Distribution $D_1^{(r)}$

1. Sample r group elements $X_1, X_2, \ldots, X_r$ i.i.d. uniformly at random from $G^{(r)}$
2. Choose a random set $S \subseteq [r]$ with $|S| = k$
3. Let $i \in S$ be the smallest index and let $X_i \leftarrow -\sum_{\substack{j \in S \\ j \neq i}} X_j$
4. Return X

Once again, we may define a (planted) search k-SUM problem for the planted distribution in the same way as we did in Definition 3.3.

Definition 3.6 (Planted Search k-SUM)**.** *For $k \in \mathbb{N}$ and an ensemble G, an algorithm $\mathcal{A}$ is said to solve the* planted search k-SUM problem *over G with success probability ϵ if, on input an instance X of size r it outputs an $S = \mathcal{A}(X) \subseteq [r]$ with $|S| = k$ such that:*

$$\Pr_{\substack{X \sim D_1^{(r)} \\ S \leftarrow \mathcal{A}(X)}} \left[\sum_{i \in S} X_i = 0 \right] \geq \epsilon$$

If $\epsilon = \Omega(1)$, we simply say that $\mathcal{A}$ solves the planted search k-SUM problem over G.

Note that the $\widetilde{O}(r^{\lceil k/2 \rceil})$-time algorithm mentioned above can solve the planted search k-SUM problem with probability $(1 - o(1))$. In addition, we also define a decision version of the k-SUM problem, which is to distinguish between these above two distributions. Now the algorithm is given a sample from either $D_0^{(r)}$ or $D_1^{(r)}$, and has to guess from which distribution its input was sampled.

Definition 3.7 (Decision k-SUM)**.** *For $k \in \mathbb{N}$ and an ensemble G, an algorithm $\mathcal{A}$ is said to solve the* decision k-SUM problem *over G with success probability ϵ if, for $b \in \{0, 1\}$:*

$$\Pr_{X \sim D_b^{(r)}} [\mathcal{A}(X) = b] \geq \epsilon$$

If $\epsilon = 1 - o(1)$, we simply say that $\mathcal{A}$ solves the decision k-SUM problem over G.

An algorithm that randomly guesses can solve the decision k-SUM problem with success probability $1/2$, so the interesting task is doing better than this. It follows from our proofs in Section 4 that at density 1, the best success probability any algorithm can have is some constant $\epsilon < 1$. At lower densities, it is possible to achieve success probability that is $(1 - o(1))$ by checking whether any solution exists.

Notation. We will use the following notation for ease of discussion of the number of solutions in k-SUM instances.

Definition 3.8. *For $k \in \mathbb{N}$, an ensemble G, and an instance $X = (X_1, \ldots, X_r)$ where each $X_i \in G^{(r)}$, we denote by $c^{(k,\mathsf{G})}(X)$ the number of sets $S \subseteq [r]$ with $|S| = k$ such that $\sum_{i \in S} X_i = 0$. When k and G are clear from context, we simply denote this by $c(X)$.*

In later sections, we will occasionally be 'sloppy' with our use of these formal definitions. It will often be the case that the choice of r is fixed and unambiguous, and hence we will sometimes refer to the group ensemble simply as G, thus removing the dependence on the superscript. Similarly, we may denote the null distribution as simply D_0, or the planted distribution as D_1. Similarly, the underlying group ensemble may be implicitly given in terms of the density; when we talk about 'sampling at density Δ_0', we refer to a group ensemble $G^{(\Delta_0)}$ that satisfies $\Delta(\mathsf{G}) = \Delta_0$. These group ensembles will often be $G_{k\text{-SUM}}^{(\Delta_0)}$ and $G_{k\text{-XOR}}^{(\Delta_0)}$, though we may omit formally specifying this and trust it is clear from the context what we mean.

4 Equivalence of Planted and Non-Planted k-SUM

In this section, we prove an equivalence between planted and non-planted k-SUM at densities ≥ 1 for any finite Abelian group. We first show that at $\Delta = 1$, any algorithm that solves the planted problem can be used to solve the non-planted problem. The precise theorem we show is the following.

Theorem 4.1. *For any $k \in \mathbb{N}$ and ensemble G of density 1, suppose there exists an algorithm that runs in time $T(r)$ and solves* planted *search k-SUM over G with success probability at least $\epsilon(r)$. Then, the same algorithm also solves* non-planted *search k-SUM over G with success probability at least $\epsilon(r)^{3/2}/9$.*

This theorem implies that planting is a one-way function. Let $f_r : \binom{r}{k} \to \mathbb{F}_2^{\lceil k \log r \rceil \times r}$ be the 'planting function' that samples a random matrix at density 1 — such that $m = k \log r$ — and plants a solution in the location specified by the first index, where these locations are ordered lexicographically among all subsets of $[r]$ of size k. For instance, the output $f_r(1)$ is a random matrix that has a solution in the set $[k]$. Then it follows immediately from Theorem 4.1 that f_r is a fine-grained one-way function. This function was also considered by [LLW19] who show that it constitutes a fine-grained one-way function based on a decision version of the k-SUM conjecture. In the dense regime, the two distributions are equivalent in a stronger sense.

Theorem 4.2. *At any density $\Delta > 1$, if $k \leq \alpha r$ for some constant $\alpha < 1$, then it holds that,*

$$SD\left(D_0^{(r)}, D_1^{(r)}\right) = O\left(1/r^{k\left[1-\frac{1}{\Delta}\right]}\right).$$

This means that if we modify f_r to have $k = \omega(1)$ and $\Delta > 1$ be some constant, then planting is an 'actual' one-way function against any polynomial-time algorithm assuming the k-XOR conjecture is true. This is similar to [JP00] who show that planting a clique of a certain size in an Erdős–Rényi graph also constitutes a one-way function, assuming it is hard to find planted cliques of size $(1+\epsilon)\log n$ for some constant $\epsilon > 0$ in an Erdős–Rényi graph of size n.

Proof Strategy. At a high-level, we wish to show that any algorithm $\mathcal{A}$ that solves the planted k-SUM recovery problem with some constant probability $\epsilon > 0$ also solves the non-planted k-SUM recovery problem with constant probability $\epsilon' > 0$ for a possibly different constant. To do so, we proceed using a hybrid argument where we define a intermediate distribution, parameterized by some integer $\ell > 0$, whose distance to both D_0 and D_1 can be bounded. By transitivity, this shows that D_0 and D_1 are also close and allows us to bound the error probability. In the former case, we are able to bound the Rényi divergence, and in the latter the statistical distance. This allows us to express ϵ' as an affine function of ϵ, i.e. $\epsilon' = \alpha\epsilon - \beta$ where $\alpha = \alpha(\ell)$ and $\beta = \beta(\ell)$ are functions of ℓ. We will show that, for each ϵ, there is a choice of ℓ such that $\epsilon' > 0$ for sufficiently large r, which would conclude the proof. Specifically, we define the following family of probability distributions,

Distribution D^ℓ

1. Sample $X \xleftarrow{\$} D_1$.
2. Let $c(X)$ be the number of solutions.
3. If $c(X) > \ell$, let $X \xleftarrow{\$} D_0$.
4. Output X.

Note that this distribution 'interpolates' between D_0 and D_1 - in particular, we have $D^0 = D_0$ and $D^{\binom{r}{k}} = D_1$.

Lemma 4.3. *For any X, if $c(X)$ is the number of solutions in X, we have,*

1. $D_1(X) = c(X)\, D_0(X)$.

2. $D^\ell(X) = \left(\mathbb{1}[c(X) \le \ell]\, c(X) + \Pr_{X \sim D_1}[c(X) > \ell] \right) D_0(X)$.

Proof. To prove the first statement, we break down the expression for $D_1(X)$ using the definition of the planted distribution as follows.

$$D_1(X) = \Pr_{\substack{Y \sim D_0 \\ S \subset [r] \\ |S|=k \\ i \leftarrow \min(S)}} \left[X_j = Y_j \forall j \neq i \bigwedge X_i = -\sum_{\substack{j \in S \\ j \neq i}} Y_j \right]$$

$$= \mathop{\mathbb{E}}_{Y \sim D_0} \left[\Pr_{\substack{S \subset [r] \\ |S|=k \\ i \leftarrow \min(S)}} \left[X_j = Y_j \forall j \neq i \bigwedge X_i = -\sum_{\substack{j \in S \\ j \neq i}} Y_j \right] \right].$$

Observe that the planting process ensures there is at least one solution in the resulting vector. Hence, $c(X) = 0 \Rightarrow D_1(X) = 0 = c(X) \cdot D_0(X)$. Now let us assume that X has $c(X) \ge 1$ distinct solutions, and it was obtained by choosing $Y \sim D_0$, $S \subset [r]$ and $i \in S$ in the planting process. Clearly, S can be any of the $c(X)$ solutions of k-SUM in X, and i is the minimum index in S. Since Y_i is completely replaced whereas the other elements in Y remain unchanged, Y can be any vector that agrees with X in all indices other than i; there are 2^m such vectors corresponding to each possible group element as Y_i. Starting from X, we can therefore make $2^m\, c(X)$ choices for the pair (Y, S). Since all the choices made in the planting process are uniformly random, the probability of any particular pair is $\frac{1}{2^{mr}} \frac{1}{\binom{r}{k}}$. Multiplying the two expressions, we get

$$D_1(X) = \frac{2^m}{\binom{r}{k}} \cdot c(X) \cdot \frac{1}{2^{mr}} = c(X) \cdot \frac{1}{2^{mr}} = c(X) D_0(X)$$

To prove the second statement, observe that $D^\ell(X)$ is the sum of the probability of choosing X in step 1 and that of choosing X in step 3 of the sampling procedure. The first term is clearly 0 if $c(X) > \ell$ (since step 3 would override it in that case) and $D_1(X)$ otherwise. The second term is the product of the probability of re-sampling in step 3 (which is exactly $\Pr_{X \sim D_1}[c(X) > \ell]$) and the probability of getting X from re-sampling (which is just $D_0(X)$). The statement now follows from adding the two terms and expanding $D_1(X) = c(X) D_0(X)$. □

Next, we will bound the Rényi divergence of D^ℓ and D_0 using Lemma 2.1 which gives a multiplicative bound on the error. In fact, applying Lemma 4.3, it is straight-forward to bound the Rényi divergence for our use-case.

Corollary 4.4. $R(D^\ell \| D_0) = \ell + \Pr_{X \sim D_1}[c(X) > \ell]$.

This establishes that D^ℓ is not 'too far' from D_0 and establishes a multiplicative bound on the error probabilities for an algorithm that solves the hybrid distribution, and the non-planted distribution. Next, we will bound the statistical distance between D^ℓ and D_1 to get an additive bound.

Lemma 4.5. $SD(D^\ell, D_1) < \frac{1}{(\ell-2)^2}$.

Proof. At a high level, our proof strategy is to bound the statistical distance in terms of the probability that a planted instance has at least a certain number of solutions that we can then bound using Chebyshev's

inequality by bounding its first two moments.

$$\begin{aligned}
&SD(D^\ell, D_1) \\
&\quad = \frac{1}{2} \sum_{X \in G^r} \left| D^\ell(X) - D_1(X) \right| \\
&\quad = \frac{1}{2} \left(\sum_{\substack{X \in G^r \\ c(X) \leq \ell}} \Pr_{X \sim D_1}[c(X) > \ell]\, D_0(X) + \sum_{\substack{X \in G^r \\ c(X) > \ell}} \left[c(X) - \Pr_{X \sim D_1}[c(X) > \ell] \right] D_0(X) \right)
\end{aligned}$$

Now identify those instances X for which $D^\ell(X) \geq D_1(X)$. This is exactly the probability that $c(X) \leq \ell$ which means the statistical distance is just the difference in probability between $D^\ell(X)$ and $D_1(X)$ which we may also write as follows.

$$\begin{aligned}
SD(D^\ell, D_1) &= \sum_{\substack{X \in G^r \\ c(X) \leq \ell}} \Pr_{X \sim D_1}[c(X) > \ell]\, D_0(X) \\
&= \Pr_{X \sim D_1}[c(X) > \ell] \frac{|\{X \in G^r \mid c(X) \leq \ell\}|}{2^{mr}} \\
&= \Pr_{X \sim D_1}[c(X) > \ell] \cdot \Pr_{X \sim D_0}[c(X) \leq \ell] \\
&\leq \Pr_{X \sim D_1}[c(X) > \ell]
\end{aligned}$$

Now, suppose we sample X from D_1 and consider the number of solutions in X. Clearly, the number of solutions is invariant to where exactly the solution is planted, so we may condition on the event that the planted solution is $[k]$.

Let $X \sim D_1$, let $S \subseteq [r]$ be a k-set and let $I_S = \mathbb{1}[\sum_{i \in S} X_i = 0]$ be the random variable, indicating whether S is a solution for the instance X. Clearly, as X is sampled randomly, we have $I_S \sim \mathrm{Ber}_2^{-m}$; hence $\mathbb{E}[I_S] = 2^{-m}$, and $\mathrm{Var}(I_S) = 2^{-m}(1-2^{-m})$. Now, let $I = \sum_{\substack{S \subseteq [r] \\ |S|=k}} I_S = c(X)$ be the total number of solutions. First, note that by linearity of expectation,

$$\mathbb{E}[I] = \sum_{\substack{S \subseteq [r] \\ |S|=k}} \mathbb{E}[I_S] = 1 + \sum_{\substack{S \subseteq [r] \\ |S|=k,\, S \neq [k]}} 2^{-m} = 1 + \left(\binom{r}{k} - 1 \right) 2^{-m} = 2 - 2^{-m}$$

Next, we may decompose the variance of I as follows.

$$\mathrm{Var}(I) = \sum_{\substack{S \subseteq [r] \\ |S|=k}} \mathrm{Var}(I_S) + \sum_{\substack{S \subseteq [r] \\ |S|=k}} \sum_{\substack{T \subseteq [r] \\ |T|=k \\ T \neq S}} \mathrm{Cov}(I_S, I_T)$$

We shall make use of the fact that for the vast majority of tuples, the covariance is zero. In fact, we note the following identity.

$$\mathrm{Cov}(I_S, I_T) = \begin{cases} 0 & \text{if } S \Delta T \neq [k] \\ 2^{-m}\left(1 - 2^{-m}\right) & \text{if } S \Delta T = [k] \end{cases}$$

To see why this is true, note that when $S \Delta T = [k]$, since we know $[k]$ is a solution, we have that $I_S = I_T$ and hence $\mathrm{Cov}(I_S, I_T) = \mathrm{Var}(I_S) = 2^{-m}\left(1 - 2^{-m}\right)$. On the other hand, if this is not the case, then S must contain at least one group element contained in neither T nor $[k]$, and hence S is a solution for exactly one

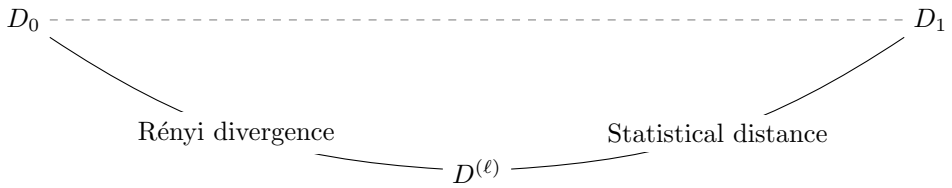

Figure 3: Depiction of the argument for statistical closeness of D_0 and D_1. We define a class of hybrid distributions $D^{(\ell)}$ for which we can bound the distance to D_0 and D_1, specifically we bound the Rényi divergence between D_0 and $D^{(\ell)}$ and the statistical distance between $D^{(\ell)}$ and D_1. We show that for large enough values of r, there is an ℓ such that the nonplanted error probability on of any algorithm that solves the planted problem is some constant > 0.

choice of this element, meaning I_S is independent of I_T. This allows us to write,

$$\begin{aligned}
\mathrm{Var}(I) &= 1 - 2^{-m} + \binom{k}{k/2}\binom{r-k}{k/2} 2^{-m}\left(1 - 2^{-m}\right) \\
&\leq 1 - 2^{-m} + c_k \binom{r}{k-1} 2^{-m}\left(1 - 2^{-m}\right) \\
&\leq 1 - 2^{-m} + c_k \frac{1}{r}\binom{r}{k} 2^{-m}\left(1 - 2^{-m}\right) \\
&= 1 - 2^{-m} + \frac{c_k(1 - 2^{-m})}{r}
\end{aligned}$$

Here c_k is a constant that depends only on k. This means that $\mathrm{Std}(I) = 1 + o(1)$ and so for sufficiently large r, we have that $\mathrm{Std}(I) \leq \sqrt{2}$. We now put things together.

$$\begin{aligned}
SD(D^\ell, D_1) &\leq \Pr_{X \sim D_1}[c(X) > \ell] \\
&\leq \Pr_{X \sim D_1}[|c(X) - 2| > \ell - 2] \\
&\leq \Pr_{X \sim D_1}[|I - \mathbb{E}[I]| > \frac{(\ell - 2)}{\sqrt{2}} \mathrm{Std}(I)] \\
&\leq \frac{2}{(\ell - 2)^2}
\end{aligned}$$

Where the latter inequality follows from application of Chebyshev's inequality (Lemma 2.3). □

This establishes that D^ℓ is not 'too far' from D_1, and implies a bound on the additive error between the success of an algorithm for the planted distribution and its success on D^ℓ. This also immediately establishes the following bound.

Lemma 4.6. $\Pr_{X \sim D_1}[c(X) > \ell] \leq \frac{2}{(\ell-2)^2}$.

We are now ready to show the main result of this section.

Proof of Theorem 4.1. We now prove the main result of this section. Let $\mathcal{A}$ be an algorithm that solves planted k-SUM, and let $\epsilon > 0$ be a constant that lower bounds its error probability. We shall prove that $\mathcal{A}$ solves non-planted k-SUM with probability $\geq \epsilon'$ for some other constant $\epsilon' > 0$. By Lemmas 4.5 and 4.6 and Corollary 4.4, we get that,

$$\epsilon' \geq \frac{\epsilon - \Pr_{X \sim D_1}[c(X) > \ell]}{\ell + \Pr_{X \sim D_1}[c(X) > \ell]} \geq \frac{\epsilon - \frac{2}{(\ell-2)^2}}{\ell + \frac{2}{(\ell-2)^2}}$$

To ensure that $\epsilon' > 0$, solving for ℓ, we get that $\ell > 2+\sqrt{\frac{2}{\epsilon}}$. Note that for any ϵ and k, there is a viable such ℓ for sufficiently large r (note that ℓ is confined to the interval $[0, \binom{r}{k}]$, which concludes the proof. Specifically, we let $\ell = 2 + \frac{2}{\sqrt{\epsilon}}$ which gives a bound of,

$$\epsilon' \geq \frac{\epsilon^{3/2}}{\epsilon^{3/2} + 4\sqrt{\epsilon} + 4} > \frac{\epsilon^{3/2}}{9} = \Omega\left(\epsilon^{3/2}\right) \qquad \square$$

Stronger Equivalence in the Dense Regime. We now show the second theorem of this section, namely that the two distributions are close in a stronger sense in the dense regime

Proof of Theorem 4.2. Let M be the set of instances without a solution. By Lemma 4.3, we know that D_1 has the property that $D_1(X) = 0$ for every $X \in M$ and that $D_1(X) \geq D_0(X)$ for every $X \in \text{supp}(D_1)$. Hence, we get that,

$$\Delta\left(D_0, D_1\right) = \sum_{X \in M} D_0(X) = \frac{|M_r|}{2^{mr}} = \Pr_{X \sim D_0}[c(X) = 0] = 1 - \Pr_{X \sim D_0}[c(X) \geq 1]$$

Our goal is thus to lower bound the probability that X has at least one solution. We note that by linearity of expectation the expected number of solutions satisfies,

$$\mathop{\mathbb{E}}_{X \sim D_0}[c(X)] = \binom{r}{k} 2^{-m} = \binom{r}{k}^{1-\frac{1}{\Delta}}.$$

Similarly, since the indicator variables for two separate solutions are pairwise independent, we get that,

$$\mathop{\text{Var}}_{X \sim D_0}[c(X)] = \binom{r}{k}^{1-\frac{1}{\Delta}} (1 - 2^{-m}).$$

And thus we get that by application of Paley-Zygmund (Lemma 2.4) that,

$$\begin{aligned}
\Pr_{X \sim D_0}[c(X) > 0] &\geq \frac{\mathbb{E}[c(X)]^2}{\binom{r}{k}^{1-\frac{1}{\Delta}} + \mathbb{E}[c(X)]^2} \\
&= 1 - \frac{\binom{r}{k}^{1-\frac{1}{\Delta}}}{1 + \binom{r}{k}^{1-\frac{1}{\Delta}} \left(\binom{r}{k}^{1-\frac{1}{\Delta}} - 1\right)} \\
&\geq 1 - \frac{2}{\binom{r}{k}^{1-\frac{1}{\Delta}}}.
\end{aligned}$$

This means we can upper bound the statistical distance as follows.

$$SD\left(D_0, D_1\right) \leq \frac{2}{\binom{r}{k}^{1-\frac{1}{\Delta}}} \leq \frac{2}{\left(\frac{r}{k}\right)^{k[1-\frac{1}{\Delta}]}} = O\left(r^{k\left[\frac{1}{\Delta}-1\right]}\right),$$

where the last bound follows since $k \leq \alpha r$ for some constant $\alpha < 1$. $\square$

5 Conditional Lower Bounds for Sparse k-SUM

In this section, we establish two different conditional lower bounds for planted k-SUM in the sparse regime. In Section 5.1, we describe a sparsification procedure on k-SUM that reduces the size of the input array to decrease the density of an instance. The resulting reduction establishes a conditional lower bound for recovery and detection that is non-trivial at any density $\Delta \in \left[\frac{1}{2}, 1\right)$. Next in Section 5.2, we use a different

method to lower density by changing the value of k; this gives us non-trivial bounds at some particular densities in $\left[\frac{2}{3}, 1\right)$.

Before going into further details, let us describe the conditional lower bound by Dinur, Keller and Klein [DKK21]. They establish a conditional lower bound for the dense regime $\Delta \in (1, 2]$. We describe their reduction at a high level for the case of $G = GF_{2^m}$ with k even for simplicity of exposition.[5] Here, we may interpret the input as a matrix $X \in \mathbb{F}_2^{m \times r}$, with the goal being to find k columns that XOR to the all-zero vector. Their lower bound is established by giving a reduction from an instance of density 1 to a dense instance by removing rows from the instance and giving this instance to a dense oracle.

Now suppose we wish to convert a density 1 instance to having density $\Delta \in (1, 2]$. In order to do this, we need to remove $t = k \log r - m$ rows from the instance. This process introduces $2^m / r^k = r^{\left[k\left(\frac{1}{\Delta} - 1\right)\right]}$ new solutions in expectation. Hence, ignoring constant factors, assuming that the oracle returns a random solution, we need to invoke the oracle $r^{k\left[\frac{1}{\Delta} - 1\right]}$ many times to obtain constant success probability. Now suppose the dense oracle takes time T, then we can solve a density 1 instance in time $r^{k\left[\frac{1}{\Delta} - 1\right]} T$ which by Conjecture 3.4[6] must satisfy $r^{k\left[\frac{1}{\Delta} - 1\right]} T \geq r^{k/2 - o(1)}$, and hence we must have that $T \geq r^{k\left[\frac{1}{2} - \frac{1}{\Delta}\right] - o(1)}$. This establishes a lower bound for the dense case, and assuming the oracle returns a random solution. However, this is not the case of a malicious oracle as the inputs as described are highly correlated. Thus, the main technical contribution of [DKK21] is an obfuscation procedure that ensures the oracle gives (mostly) random responses, whose correctness is analyzed using discrete Fourier analysis. The lower bound they obtain is known to be optimal for $k = 3, 4, 5$.

Theorem 5.1 (Dinur, Keller, Klein [DKK21])**.** *Suppose Conjecture 3.4 (resp. Conjecture 3.5) is true. Then, for $k \in \mathbb{N}$ and $\Delta \in (1, 2]$, any algorithm that solves search k-SUM in $\mathsf{G}^{(\Delta)}_{k\text{-}SUM}$ (resp. $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$) with constant success probability has to take expected time $\Omega\left(r^{\left[k\left(\frac{1}{\Delta} - \frac{1}{2}\right)\right] - o(1)}\right)$.*

As a first observation, note that this lower bound is easily adaptable to the sparse setting (at least in the case of k-XOR). Here, instead of removing rows to increase the density, we will add random rows to lower the density and give the resulting instance to the sparse oracle. Here, we do not need to worry about correlations between instances, as we are not introducing new solutions. In fact, the oracle cannot be malicious as it has to be correct over the randomness of the instance which is distributed exactly according to what it expects. Note that by adding t rows, the original solution is preserved with probability 2^{-t} and hence we will have to invoke to oracle $\Omega(2^t)$ times to recover the solution with constant probability. Now suppose we start with a density 1 instance: in order to convert this to a density Δ instance, we need to add $t = k \log r \left(\frac{1}{\Delta} - 1\right)$ such rows. Assuming it takes time T to solve the instance at density Δ, by Conjecture 3.4 we get a bound of $2^t T \geq r^{k/2}$, i.e. $r^{k\left[\frac{1}{\Delta} - 1\right]} T \geq r^{k/2}$, and thus $T \geq r^{k\left[\frac{3}{2} - \frac{1}{\Delta}\right]}$ which is non-trivial for $\Delta \geq \frac{2}{3}$. This reduction establishes the following lower bound.

Theorem 5.2 (Follows from techniques in [DKK21])**.** *Suppose Conjecture 3.5 is true. Then, for $k \in \mathbb{N}$ and $\Delta \in (\frac{2}{3}, 1]$, any algorithm that solves search k-SUM in $G^{(\Delta)}_{k\text{-}XOR}$ with constant success probability has to take expected time $\Omega\left(r^{\left[k\left(\frac{3}{2} - \frac{1}{\Delta}\right)\right] - o(1)}\right)$.*

5.1 Lower Bound for Densities in $\left(\frac{1}{2}, 1\right)$

In this section, we show how to generalize Theorem 5.2 to k-SUM in arbitrary groups. Specifically, we will prove the following theorem.

Theorem 5.3. *Consider some $k, r \in \mathbb{N}$, $\Delta \in \left(\frac{1}{2}, 1\right)$, $\epsilon \in (0, 1]$, and Abelian group G. Suppose there is an algorithm that runs in time T and, given an instance of r^{Δ} uniformly random group elements from G with*

[5] See Footnote 4 in Section 3 discussing the slightly different definition of the k-SUM problem as considered by [DKK21].

[6] Throughout this section, we use a weaker version of Conjectures 3.4 and 3.5 that state a lower-bound of $r^{k/2 - o(1)}$ rather than $r^{\lceil k/2 \rceil - o(1)}$. This is done for simplicity in our expressions. Note that this relaxation only weakens our lower bounds, which are hence actually stronger than stated for certain values of k and Δ.

a planted k-SUM solution, outputs a k-SUM solution for it with probability ϵ. Then, for some constants $c \in \mathbb{N}, \epsilon' \in (0,1]$, there is an algorithm that runs in time $(c \cdot T \cdot r^{k(1-\Delta)} \cdot \log(|G|))$ that, given an instance of r uniformly random group elements from G with a planted k-SUM solution, outputs a k-SUM solution for it with probability ϵ'.

Proof. Let $\mathcal{A}^{(\Delta)}$ be the algorithm that given r^Δ random group elements from the group G with a planted solution, outputs a k-SUM solution with probability at least ϵ. We then construct the following algorithm $\mathcal{A}^{(1)}$ for recovering k-SUM solutions given r random elements from G with a planted solution.

Algorithm $\mathcal{A}^{(1)}(X)$

1. Repeat $2r^{k(1-\Delta)}$ times:
   1.1. Initialize X' to be an empty array.
   1.2. Randomly choose r^Δ elements from X and copy them to X'.
   1.3. Define P to be the indexing function such that $X'[i] = X[P(i)]$.
   1.4. Let $K \leftarrow \mathcal{A}^{(\Delta)}(X')$
   1.5. If K is a solution, return $P(K)$.

By definition of planted k-SUM, we know that X has at least one solution K. On any given iteration, the probability of all k of those elements being copied to X' is $\frac{r^\Delta}{r} \cdot \frac{r^\Delta - 1}{r} \cdots \frac{r^\Delta - k + 1}{r} > \frac{r^{k(\Delta-1)}}{2}$. Therefore, the probability that we call A on an array containing all the elements of κ at least once is at least,

$$1 - \left(1 - \frac{r^{k(\Delta-1)}}{2}\right)^{2r^{k(1-\Delta)}} \geq 1 - \frac{1}{e} = \Omega(1).$$

We claim that the probability distribution induced on X' conditioned on the original solution being preserved is just the planted distribution on density Δ. Observe that the elements of X outside K are uniformly i.i.d from G, and K independently contains a uniformly random k-tuple from G that sums to 0. Therefore, the elements of X' outside K are also uniformly i.i.d from G, and K still contains a uniformly random k-tuple from G that sums to 0. We can conclude that if $\mathcal{A}^{(\Delta)}$ gets called on an array where the solution is preserved, its input will look like an average-case instance sampled from the planted distribution, and $\mathcal{A}^{(\Delta)}$ will succeed with probability ϵ. The overall success probability of $\mathcal{A}^{(1)}$ is therefore at least $\epsilon\left(1 - \frac{1}{e}\right) = \Omega(1)$. The runtime of $\mathcal{A}^{(1)}$ is $O(r^{k(1-\Delta)}\, T)$, as desired. □

This reduction immediately gives a lower bound on k-SUM in terms of the density.

Corollary 5.4. *Suppose Conjecture 3.4 (resp. Conjecture 3.5) is true. Then, for $k \in \mathbb{N}$ and $\Delta \in \left[\frac{1}{2}, 1\right)$, any algorithm that solves planted search k-SUM in $\mathsf{G}^{(\Delta)}_{k\text{-}SUM}$ (resp. $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$) with constant success probability has to take expected time $\Omega\left(r^{\left[k\left(1-\frac{1}{2\Delta}\right)\right]-o(1)}\right)$.*

5.2 Reducing Between k-SUM for Different k's

In this section, we will present a different sparse conditional lower bound. This bound also applies to any group. Recall that we previously decreased the density by reducing the number of elements in the instance. Instead, now we will reduce the density by compressing elements of the inputs and hope that the resulting instance has a 'nice' structure. An interesting feature of this lower bound is that it relates the hardness of k-SUM to the hardness of k'-SUM at a different density (where $k \neq k'$).

Theorem 5.5. *Consider $k_1, k_2 \in \mathbb{N}$ such that $k_1 \geq 3$ and $k_2 \in [k_1 + 1, 2k_1 - 1]$, and an ensemble G of density $\Delta_1 \leq \frac{k_1}{k_2}$. Suppose there exists an algorithm that runs in time $T(r)$ and solves planted search k_1-SUM on G with constant success probability. Then, there is an algorithm B that runs in time $O(r^{k_2 - k_1} T(r))$ and solves planted search k_2-SUM on G with constant success probability.*

Proof. We start by describing the new algorithm.

Algorithm $B(X)$

1. Repeat $3^{2k_2} r^{k_2-k_1}$ times:
   1.1. Initialize X' to be an empty array.
   1.2. Randomly choose $\frac{r}{2}$ elements from X and copy them to X'.
   1.3. Randomly split the remaining elements of X into $\frac{r}{4}$ disjoint pairs.
   1.4. Insert the sums of each of the above $\frac{r}{4}$ pairs into X'.
   1.5. Add $\frac{r}{4}$ random elements of G to X'.
   1.6. Apply a random permutation to X'.
   1.7. Let $S \leftarrow A(X')$.
   1.8. If S is a solution and it depends on exactly k_2 elements of X, return those k_2 elements.

By definition of planted k-SUM, we know that X has at least one solution S. Recall that $|S| = k_2$. We are interested in the event where $2k_1 - k_2$ of the elements in S were copied directly to X' and the remaining $2k_2 - 2k_1$ elements of S were paired with each other such that their sums got copied to X'. Clearly, this would give rise to a $(2k_1 - k_2) + \frac{2k_2-2k_1}{2} = k_1$-SUM solution in X'. We call solutions of this type *valid*. The probability of exactly $2k_1 - k_2$ elements of S being copied directly to X' in step 1.2. is at least $\left(\frac{r/2-(2k_1-k_2)}{r}\right)^{2k_2-2k_1} \geq \frac{1}{3^{2k_2}}$. The probability that the $2k_2 - 2k_1$ remaining elements get paired amongst themselves in step 1.3. is at least $\left(\frac{1}{r}\right)^{k_2-k_1}$. Therefore, the probability that we call A on an array containing k_1-SUM solution at least once is at least $1 - (1 - \frac{1}{3^{2k_2} r^{k_2-k_1}})^{3^{2k_2} r^{k_2-k_1}} \geq 1 - \frac{1}{e} = \Omega(1)$.

We claim that the probability distribution induced on X' conditioned on it having a *valid* k_1-SUM solution is just the planted distribution on density Δ_1. Observe that the elements of X outside S are uniformly i.i.d from G. Therefore, the $\frac{r}{2}$ elements added in step 1.2. and the $\frac{r}{4}$ elements added in step 1.5. are uniformly i.i.d. from G. Since G is a group, the sum of two random elements is also random; this implies that the other $\frac{r}{4}$ elements of X' are uniformly i.i.d. too (excluding the solution). The density of X' is clearly $\frac{k_1 \log(r)}{m} = \frac{k_1}{k_2} \cdot \Delta_2 = \Delta_1$.

We can conclude that if A gets called on an array where a valid solution exists, its input will look like an average-case instance sampled from the planted distribution, and it will succeed with constant probability. The overall success probability of B is therefore also a constant. The runtime of B is clearly $O(r^{k_2-k_1} T(r))$, as required. □

Corollary 5.6. *Suppose Conjecture 3.4 (resp. Conjecture 3.5) is true. Then, for $k \in \mathbb{N}$ and $\Delta \in \left(\frac{1}{2}, 1\right)$ such that $\Delta = \frac{k}{k'}$ for some $k' \in [k+1, 2k-1]$, any algorithm that solves planted search k-SUM in $\mathsf{G}^{(\Delta)}_{k\text{-}SUM}$ (resp. $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$) with constant success probability has to take expected time $\Omega\left(r^{\left\lceil k(1-\frac{1}{2\Delta})\right\rceil - o(1)}\right)$.*

Proof. This follows directly from the contrapositive of the previous theorem and the k-SUM conjecture. If there is an algorithm at density $\Delta = \frac{k}{k'}$ for k-SUM with runtime T, Theorem 5.5 implies that we can solve k'-SUM at density 1 in time $O(r^{k'-k} T)$. We assume that is at least $\Omega(r^{\left\lceil \frac{k'}{2} \right\rceil - o(1)})$. This, along with the definition of Δ, implies $T = \Omega\left(r^{\left\lceil k(1-\frac{1}{2\Delta})\right\rceil - o(1)}\right)$. □

6 Search to Decision Reduction

We will prove that decision is at least as hard as search by reducing search to decision using an algorithm reminiscent of binary search. The precise statement we will show is the following.

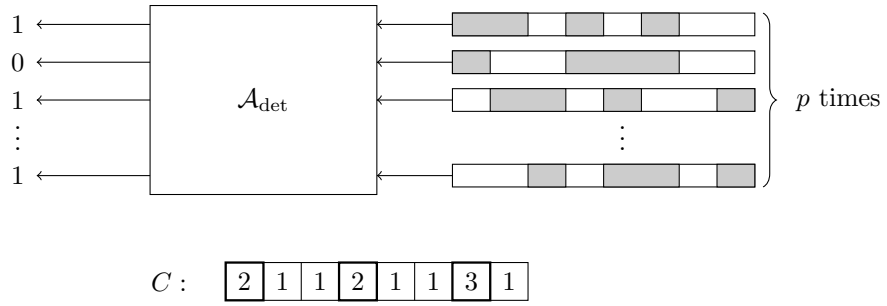

Figure 4: Illustration of the structure of the algorithm $\mathcal{A}_{\text{rec}}$ shown for an instance X of size $r = 8$ and $k = 3$. It chooses a random subset S of the elements of size $r/2 = 4$ and replaces all elements in X from S with fresh samples from the group and gives the resulting instance Y to the decision oracle $\mathcal{A}_{\text{det}}$. If $\mathcal{A}_{\text{det}}$ reports there is a solution, we increment a counter for each of the elements we did not replace. We then repeat this process $p = \text{polylog}(r)$ times and output the k elements whose counters are the highest. In the illustration, we have $p = 4$ and have marked each set S chosen. At the end, the three elements with the highest counters are X_1, X_4, X_7, so the search algorithm outputs $\{X_1, X_4, X_7\}$ as the solution.

Theorem 6.1. *For $k \in \mathbb{N}$, and ensemble G of density $\Delta < 1$, suppose there is an algorithm that runs in time $T(r) = \Omega(r/\Delta)$, and solves the* decision *k-SUM problem over G with success probability $(1 - o(1))$. Then, for any constant $\gamma < 1$, there is an algorithm that runs in time $\widetilde{O}(T(r))$, and solves the* planted search *k-SUM problem in G with success probability at least γ.*

We first describe the reduction at a high level and then prove its correctness. Our algorithm is vaguely related to binary search. We will repeatedly guess a random half of the inputs to replace with fresh random elements, invoke the decision oracle on the resulting instance, and record whether or not the oracle reported there was a solution. Specifically, we will maintain a counter for every element in the original input that we increment whenever an element was found to belong to an unreplaced half of the inputs for which the decision algorithm reported there was a solution. Finally, we output the indices corresponding to the k largest counters. Our hope is that this process is biased in favor of the indices in the solution, and that we do not introduce too many new solutions in the process.

Algorithm $R(X)$

1. Sample $r/2$ elements from $[r]$ at random (with replacement), and let $S \subseteq [r]$ be the resulting set.
2. Let $X'_i \leftarrow X_i$ for every $i \notin S$, and let $X'_i \xleftarrow{\$} G$ otherwise.
3. Return X'

We now describe our reduction formally. Let $X \sim D_1$ be some planted instance and let $\mathcal{A}_{\text{det}}$ be an algorithm that solves the decision problem with probability $1 - o(1)$, and let T be its runtime.

Algorithm $\mathcal{A}_{\text{rec}}(X)$

1. Let $C \leftarrow 0^r$ be a list of counters.
2. Repeat p times:
   2.1. Let $Y \leftarrow R(X)$, and let S be the subset chosen in this execution of R.
   2.2. If $\mathcal{A}_{\text{det}}(Y) = 1$, increment C_i for every $i \notin S$.
3. Output $K \subseteq [r]$ with $|K| = k$ that maximizes $\sum_{i \in K} C_i$.

Sampling an element from G takes time $O(k\log(r)/\Delta) = \widetilde{O}(1/\Delta)$, and so $R(X)$ can be computed in time $\widetilde{O}(r/\Delta)$. Also note that the last step can be done in time $O(r)$ (or faster if using a secondary data structure such as a heap), and hence the total time complexity of the algorithm is $O((T + r/\Delta)\, p)$. This means we are done if we can show that $\Pr[\sum_{i\in K} x_i = 0] \geq \gamma$ for some $p = \mathrm{polylog}(r)$.

We now explain our proof strategy at a high level. Intuitively, the above procedure will assign higher counts for the indices belonging to a solution. Indeed, we will give a concentration bound on the value of each counter, conditioned on it being a solution or not. We will then bound the probability of these values belonging to two disjoint intervals, such that the desired solution is output. Finally, we will union bound over all values to achieve the desired result. We may assume w.log. that $\mathcal{A}_{\mathrm{det}}$ is deterministic.

Lemma 6.2. *Let $K \subseteq [r]$ denote the planted set. Then for sufficiently large r it holds that in any given iteration of $\mathcal{A}_{\mathrm{rec}}$, the probability that there is a set $S \neq K$ such that $\sum_{i\in S} Y_i = 0$ is at most $4r^{k\,[1-\frac{1}{\Delta}]}$.*

Proof. Consider the $R(X)$ used in a given iteration and denote by $\widetilde{c}(R(X))$ the number of solutions in $R(X)$ different from the planted set K. Let $A \subseteq [r]$ be a set with $|A| = k$ and let I_A be the indicator for whether or not A is a solution in $R(X)$. Let K denote the planted set and note that,

$$\widetilde{c}(R(X)) = \sum_{\substack{A\subseteq r\\ |A|=k\\ A\neq K}} I_A.$$

Each $I_A \sim \mathrm{Ber}\,(2^{-m})$, and hence by linearity of expectation,

$$\mathbb{E}[\widetilde{c}(R(X))] = \left[\binom{r}{k} - 1\right] 2^{-m} < r^{k\,[1-\frac{1}{\Delta}]}.$$

Here, the inequality holds since $\binom{r}{k} \leq \left(\frac{er}{k}\right)^k < r^k$ for $k \geq 3 > e$. To bound the variance, we will use the fact that most I_A, I_B for $A \neq B$ are independent. In fact, unless $A\Delta B = K$, the variables I_A and I_B will each contain one random element that is not determined by the planted solution. This means that $\Pr[I_A = 1 \mid I_B] = 2^{-m} = \Pr[I_A = 1]$, and hence I_A, I_B are independent. If indeed, $A\Delta B = K$, then A is a solution if and only if B is a solution, in which case $\mathrm{Cov}[I_A, I_B] = \mathrm{Var}[I_A] = 2^{-m}(1 - 2^{-m})$. We proceed to bound the variance using the bounds discussed above.

$$\begin{aligned}
\mathrm{Var}[\widetilde{c}(R(X))] &= \sum_{\substack{A\subseteq[r]\\ |A|=k\\ A\neq K}} \mathrm{Var}[I_A] + \sum_{\substack{A,B\subseteq[r]\\ |A|=|B|=k\\ A\neq B}} \mathrm{Cov}[I_A, I_B] \\
&< \binom{r}{k} 2^{-m}\,(1 - 2^{-m}) + \sum_{\substack{A,B\subseteq[r]\\ |A|=|B|=k\\ A\Delta B=K}} 2^{-m}(1 - 2^{-m})
\end{aligned}$$

To count the number of such sets, we may arbitrarily choose $k/2$ elements from the planted set and choose $k/2$ elements from the remaining $r - k$ elements. This uniquely determines the say A, and by consequence also the set B.

$$\begin{aligned}
&= \binom{r}{k} 2^{-m}\,(1 - 2^{-m}) + \binom{k}{k/2}\binom{r-k}{k/2} 2^{-m}\,(1 - 2^{-m}) \\
&\leq \left[\binom{r}{k} + \binom{r}{k/2}\right] 2^{-m}\,(1 - 2^{-m}) \\
&\leq 2\binom{r}{k} 2^{-m}\,(1 - 2^{-m}) \\
&< 2\,r^{k\,[1-\frac{1}{\Delta}]}
\end{aligned}$$

Note that this gives a bound of $\mathrm{Std}[\widetilde{c}(R(X))] \leq \sqrt{2}\, r^{k\,[1-\frac{1}{\Delta}]/2}$ which allows us to bound the probability that $\widetilde{c}(R(X)) > 1$.

$$\begin{aligned}\Pr[\widetilde{c}(R(X)) > 1] &\leq \Pr\left[|\widetilde{c}(R(X)) - \mathbb{E}[\widetilde{c}(R(X))]| \geq 1 - r^{k\,[1-\frac{1}{\Delta}]}\right] \\ &\leq \Pr\left[|\widetilde{c}(R(X)) - \mathbb{E}[\widetilde{c}(R(X))]| \geq \left(\frac{1 - r^{k\,[1-\frac{1}{\Delta}]}}{\sqrt{2}\, r^{k\,[1-\frac{1}{\Delta}]/2}}\right) \mathrm{Std}[\widetilde{c}(R(X))]\right]\end{aligned}$$

Now suppose that $r > (2+\sqrt{2})^{\frac{\Delta}{k(\Delta-1)}}$ is sufficiently large. Then we get the upper bound,

$$\begin{aligned}&\leq \Pr\left[|\widetilde{c}(R(X)) - \mathbb{E}[\widetilde{c}(R(X))]| \geq \frac{1}{2r^{k\,[1-\frac{1}{\Delta}]/2}} \mathrm{Std}[\widetilde{c}(R(X))]\right] \\ &\leq 4r^{k\,[1-\frac{1}{\Delta}]},\end{aligned}$$

where the last inequality follows from Chebyshev's inequality (Lemma 2.3). □

In the following, we will bound the values of the counters. Denote by $\mathrm{Det}(X)$ the correct answer for the instance X, i.e.,

$$\mathrm{Det}(X) = \begin{cases} 1 & \text{if } X \text{ has a solution,} \\ 0 & \text{if } X \text{ does not have a solution.}\end{cases}$$

Lemma 6.3. *At any constant density $\Delta < 1$, if $X \sim D_b$ then $\mathrm{Det}(X) = b$, except with probability $o(1)$.*

Proof. We need to show that with probability $1 - o(1)$, an instance $X \sim D_b$ has a solution iff $b = 1$. The case of $b = 1$ is true by definition, while for $b = 0$ we need to upper bound the probability that an instance $X \sim D_0$ has a solution. Let $S \subseteq [r]$ be a set with $|S| = k$, and let I_S be the indicator variable for S being a solution. Let $c(X)$ be the number of solutions in X and note that $c(X) = \sum_{\substack{S\subseteq[r]\\|S|=k}} I_S$. By linearity of expectation, we have $\mathbb{E}[c(X)] \leq r^{k\,[1-\frac{1}{\Delta}]}$ which is subconstant for $\Delta < 1$. Hence,

$$\Pr_{X\sim D_0}[\mathrm{Det}(X) \neq 0] = \Pr_{X\sim D_0}[c(X) \geq 1] = \Pr_{X\sim D_0}\left[c(X) \geq \frac{1}{r^{k\,[1-\frac{1}{\Delta}]}}\mathbb{E}[c(X)]\right] \leq r^{k\,[1-\frac{1}{\Delta}]},$$

by Markov's inequality (Lemma 2.2) which as remarked is subconstant. □

Say an instance X is *bad* if $\Pr[\mathcal{A}_{\mathrm{det}}(R(X)) \neq \mathrm{Det}(R(X))] > \frac{1}{2^{2k}}$, with randomness taken over R and $\mathcal{A}_{\mathrm{det}}$.

Lemma 6.4. *If $\mathcal{A}_{\mathrm{det}}$ solves the decision problem with success probability $1 - o(1)$, then for sufficiently large r, an instance $X \sim D_1$ is bad with probability at most $\frac{1}{2^{8k}}$.*

Proof. At a high level, the result essentially follows using a Markov bound at a sufficiently high value of r. Let $X \sim D_1$ be an instance from the planted distribution. First note that the distribution of $R(X)$ is a convex combination of D_0 and D_1, determined by whether or not the set chosen by $R(\cdot)$ intersects with the solution. Hence by correctness $\mathcal{A}_{\mathrm{det}}$ has to mostly agree with Det. Let S be the subset chosen by R. Clearly, if S is disjoint from the solution, the resulting instance is distributed as D_1. Otherwise, we do not preserve the solution and the instance is distributed as D_0. Note that as $\Delta < 1$ is constant, it follows from Lemma 6.3 that except with probability $o(1)$, $X \sim D_{\mathrm{Det}(X)}$. By convexity, in the former case, $\mathcal{A}$ has to output 1 except with probability $o(1)$, while in the latter case, it has to output 0 except with probability $o(1)$. Let $Z(X) = \Pr_{Y\leftarrow R(X)}[\mathcal{A}(Y) \neq \mathrm{Det}(Y)]$, and note that an instance X is bad if $Z(X) \geq \frac{1}{2^{2k}}$. For large enough r, we thus get a bound of $\mathbb{E}[Z(X)] \leq \frac{1}{2^{10k}}$ with the expectation taken over X and $R(\cdot)$. We can now bound the probability that X is bad.

$$\Pr_{X\sim D_1}[X \text{ is bad}] = \Pr_{X\sim D_1}\left[Z(X) \geq \frac{1}{2^{2k}}\right] \leq \Pr_{X\sim D_1}\left[Z(X) \geq 2^{8k}\,\mathbb{E}[Z(X)]\right] \leq \frac{1}{2^{8k}},$$

where the latter follows from Markov's inequality (Lemma 2.2). □

Proof of Theorem 6.1. At a high level, we will give a concentration bound on each counter using a Chernoff bound, and conclude that, with high probability, the range of the counters for the indices in the solution is disjoint from the range of counters outside the solution by employing a union bound on all the counters.

Formally, suppose $\mathcal{A}_{\text{det}}$ is a deterministic algorithm, and let us fix X. Now, in the reduction, for each choice of $R(X)$, the counter will be incremented by some specific vector which is either zero if the solution was destroyed, or a balanced vector if the solution is preserved. Let E_{ij} be the event that the i^{th} counter was incremented in the j^{th} iteration. Note that $C_i = \sum_{j=1}^{p} E_{ij}$. Now consider an index i belonging to the planted solution, and suppose that $\mathcal{A}_{\text{det}}$ has no errors and that $R(\cdot)$ did not introduce any new solutions. Then C_i is incremented if all of the indices belonging to the solution were not replaced, and thus $\mathbb{E}[E_{ij} \mid i \text{ solution}] = \frac{1}{2^k}$. Analogously, for an index not belonging to a solution, it will be incremented if the solution were preserved and also this index was preserved, and so the error-free expectation will be $\mathbb{E}[E_{ij} \mid i \text{ not solution}] = \frac{1}{2^{k+1}}$. By Lemma 6.4, even if $\mathcal{A}_{\text{det}}$ has a $o(1)$ probability of error, we know that for each counter the error is $\epsilon < \frac{1}{2^{2k}} < \frac{1}{2^{k+3}}$ (since $k \geq 3$) with probability at least $1 - \frac{1}{2^{8k}}$, where the randomness is taken over the instance. In addition, even if we destroyed the solution, $R(\cdot)$ might inadvertently create a new solution which happens with probability $< 4r^{k(1-\frac{1}{\Delta})}$. To account for the errors, we assume, as a worst-case precaution using a union bound, that the expectations change by at most ϵ, such that by linearity of expectation,

$$\begin{aligned}
\mathbb{E}[C_i] &\geq p\left(\frac{1}{2^k} - \epsilon\right) && \text{when } i \text{ is solution} \\
\mathbb{E}[C_i] &\leq p\left(\frac{1}{2^{k+1}} + \epsilon + 4r^{k(1-\frac{1}{\Delta})}\right) && \text{when } i \text{ is not solution}
\end{aligned}$$

We now wish to say that the range of values of indices belonging to the solution is disjoint from the range of those not belonging to the solution. We say a counter is *bad* if it deviates from its expectation by more than $\frac{p}{2^{k+4}}$. This ensures that when no counters are bad, for sufficiently large r, the range of counts for the indices belonging to a solution is disjoint from those not belonging to a solution. To see this, we compute the distance Δ to the midpoint of the expectations, i.e.,

$$\begin{aligned}
\Delta = \frac{p\left(\frac{1}{2^k} - \epsilon\right) - p\left(\frac{1}{2^{k+1}} + \epsilon + 4r^{k\left(1-\frac{1}{\Delta}\right)}\right)}{2} &> \frac{p\left(\frac{1}{2^k} - \frac{1}{2^{k+3}} - \frac{1}{2^{k+1}} - \frac{1}{2^{k+3}} - 4r^{k\left(1-\frac{1}{\Delta}\right)}\right)}{2} \\
&= p\left(\frac{1}{2^{k+3}} - 2r^{k\left(1-\frac{1}{\Delta}\right)}\right) \\
&> \frac{p}{2^{k+4}},
\end{aligned}$$

where the first inequality follows as $\epsilon < \frac{1}{2^{2k}} < \frac{1}{2^{k+3}}$ is true for any $k \geq 3$, and second inequality follows since $\Delta < 1$ is constant and $2r^{k\left(1-\frac{1}{\Delta}\right)} < \frac{1}{2^{k+4}}$ for sufficiently large r. Note that when X is fixed, each E_{ij} and E_{ik} are independent for $j \neq k$ and are supported on $\{0, 1\}$. We may thus we may bound the probability of a bad counter as function of p using a Chernoff bound (Lemma 2.5). Suppose that i is an index belonging to the solution, then we get the following bound, (Lemma 2.5).

$$\Pr\left[i^{\text{th}} \text{ counter is bad} \mid i \text{ solution}\right] = \Pr\left[C_i < \mathbb{E}[C_i] - \frac{p}{2^{k+4}}\right] = \Pr\left[\frac{1}{p} C_i < \frac{1}{p}\mathbb{E}[C_i] - \frac{1}{2^{k+4}}\right] < e^{-\frac{p}{2^{2k+7}}}.$$

Now let $0 < \gamma \leq 1$ be any constant and let $p = 2^{2k+7} \ln \frac{r}{\gamma}$. Then we get an upper bound of $\frac{\gamma}{r}$ for a solution counter going bad. We get the same bound for the indices not belonging to a solution. By a union bound on all the counters, we bound the total error rate by γ. □

7 Hardness Amplification for k-XOR

In this section, we propose a success amplification procedure for k-XOR recovery outside the dense regime. The procedure amplifies the success of any algorithm that solves recovery with success probability $\Omega(1/\text{polylog}(r))$

in GF_{2^m} to a success probability of $1 - o(1)$, at the cost of increasing its runtime by a polylogarithmic factor. Since we already proved that the k-SUM detection problem is at least as hard as k-SUM recovery in the previous section, this also establishes that detection and recovery are equivalent for planted k-SUM in $G = GF_{2^m}$.

Note that any element of GF_{2^m} can be represented by a bit vector of size m, hence we will represent the input as an $m \times r$ Boolean matrix whose columns represent the element. Note that addition in this group is equivalent to bitwise XOR. At density Δ, we have the relation $m\Delta = k \log r$. As long as $1 \geq \Delta = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$, this implies that $\log r \leq m = \Theta(\text{polylog}(r))$. To sample from D_0, we simply choose each bit in the matrix independently and uniformly at random. To sample from D_1, we do the same thing and then replace a random column with the sum of $k-1$ randomly chosen other columns. Given an arbitrary matrix M sampled from D_1, the recovery problem asks for a k-tuple of indices S such that,

$$\bigoplus_{j \in S} M_{ij} = 0 \quad \forall i \in [m] \tag{5}$$

Theorem 7.1. *For $k \geq 3$ and density $\Omega\left(\frac{1}{\text{polylog}(r)}\right) \leq \Delta \leq 1$, suppose there exists an algorithm $\mathcal{A}_{\text{rec}}$ that runs in time $T(r) = \Omega(r)$, and solves the planted search k-SUM problem over $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$ with success probability $\gamma = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$. Then, there is an algorithm that runs in time $\widetilde{O}(T(r))$, and solves the planted search k-SUM problem over $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$ with success probability $(1 - o(1))$.*

At a high level, we will simply invoke the recovery oracle multiple times on inputs related to the original input. To deal with the oracle potentially being malicious, we employ an obfuscation procedure to hide the original solution. We will raise the success probability in several steps. For the sake of simplicity, we will assume that $\gamma = \Theta\left(\frac{1}{\text{polylog}(r)}\right)$ in the rest of the proof. Of course, if our starting algorithm $\mathcal{A}_{\text{rec}}$ has a higher success probability, we can always make it less reliable by randomly failing on an appropriate fraction of inputs.

Let $M \in \mathbb{F}_2^{m \times r}$ be an input. For most of the construction, we will impose the following restrictions on the matrix M:

1. **The r columns are all unique.** Note that the probability of any 2 particular columns being equal is exactly 2^{-m}. This holds even when one (or both) of the 2 columns are part of the planted solution. Since there are $\binom{r}{2} < r^2$ pairs of columns, we can conclude via union bound that the probability of having any duplicate columns in the matrix is at most $\frac{r^2}{2^m} = r^{2-k/\Delta} \leq r^{2-k} \leq \frac{1}{r} = o(\gamma)$. Therefore, the r columns are all distinct with probability at least $1 - o(\gamma)$.

2. **There is exactly one solution.** Since M is sampled from D_1, there is always the solution we explicitly planted. The probability of another k-XOR solution among the $r-1$ original columns is at most $\frac{\binom{r-1}{k}}{2^m} < r^{k(1-1/\Delta)}$. The new column creates an extraneous solution exactly when there are some other $k-1$ columns which have the same sum as the unchanged columns in S; this happens with probability at most $\frac{\binom{r-1}{k-1}}{2^m} < r^{k(1-1/\Delta)-1} < \frac{1}{r} = o(\gamma)$. Therefore, we conclude that this condition is satisfied except with probability $O(r^{k(1-1/\Delta)})$. For the first several steps in the construction, we will assume that $\Delta \leq 1 - \frac{\log\log\log r}{\log r}$. The following calculation shows that the probability of multiple solutions becomes $o(\gamma)$ in that case.

$$r^{k(1-\frac{1}{\Delta})} \leq r^{k\left(1-\frac{1}{1-\log\log\log r/\log r}\right)} = 2^{k\log r\left(\frac{-\log\log\log r}{\log r - \log\log\log r}\right)} < 2^{k\log r\left(\frac{-\log\log\log r}{\log r}\right)} = (\log\log r)^{-k}$$
$$= o\left(\frac{1}{\text{polylog}(r)}\right) = o(\gamma).$$

3. **The k columns comprising the solution have rank $k-1$.** The column we replace is obviously linearly dependent on the other $k-1$ columns of S. This condition is therefore fulfilled if and only if

those other $k-1$ columns are linearly independent. Note that these columns were all independently chosen uniformly at random from GF_{2^m}. The probability of an arbitrary element being outside the subspace spanned by t elements is $1-2^{t-m}$. Therefore, the probability of this condition holding is at least

$$(1-2^{k-1-m})^{k-1} \geq 1 - \frac{(k-1)2^{k-1}}{2^m} \geq 1 - \frac{1}{r} \geq 1 - \frac{1}{\text{polylog}(r)} = 1 - o(\gamma),$$

since k is some constant and $m \geq \log r$.

4. **The matrix M has rank m.** This condition holds as long as all the m rows are linearly independent. For sake of simplicity, let us ignore the planted column; the other values are all uniformly randomly chosen. The probability of an arbitrary row being outside the subspace spanned by t rows is $1-2^{t-r+1}$. Therefore, the probability of this condition holding is at least

$$(1-2^{m-r})^m \geq 1 - m2^{m-r} = 1 - \frac{m2^m}{2^r} \geq 1 - \frac{2^{\text{polylog}(r)}}{2^r} = 1 - o(\gamma)$$

since $m = O(\text{polylog}(r))$.

Definition 7.2 (Permissible Matrix)**.** *We call a matrix that satisfy all 4 of the above conditions* permissible*. Define D_{perm} to be the distribution D_1 conditioned on the sampled matrix being permissible.*

Remark 7.3. *Since it is nontrivial to check if a given matrix is permissible, there is no obvious efficient sampling process for D_{perm}. We only use this distribution to make theoretical arguments.*

Lemma 7.4. *The following is true.*

1. *If $\Delta \leq 1 - \dfrac{\log\log\log r}{\log r}$, then $\Pr_{M\sim D_1}[M \text{ is permissible}] = 1 - o(\gamma)$*
2. *D_{perm} is uniform on its support.*

Proof. The first statement follows from applying the union bound over the probabilities of each of the 4 conditions being violated. The second statement follows from Lemma 4.3 and the fact that each permissible matrix has exactly one k-XOR solution. □

Our first step is creating an intermediate algorithm $\mathcal{A}^{\text{obf}}$ that runs in $\widetilde{O}(T)$ time and solves the same problem somewhat more reliably. Let $\mathcal{A}_{\text{rec}}$ be the recovery algorithm with success probability γ promised in the statement of Theorem 7.1; it takes as input a Boolean matrix sampled from D_1 and returns a Boolean vector of size r which only contains 1 at the k indices corresponding to a solution. Observe that $\mathcal{A}_{\text{rec}}$ can be adversarially biased in at least two ways. It can only look for solutions in some particular positions (e.g. it only looks at the first $r/2$ vectors and always fails in the $1-1/2^k$ fraction of the inputs where the solution involves the other vectors). To deal with this issue, we *anonymize* the solution by applying an arbitrary permutation to the columns of M. The original oracle can also only look for solutions where the columns have some particular property (e.g. it can only find solutions where the total Hamming weight of the k vectors in S is less than $km/2$ and always fails for the other half of the input space). To fix this issue, we *randomize* the solution by multiplying M with a random full-rank matrix on the left. This will transform the solution to an arbitrary solution which also has rank $k-1$, since full-rank linear transformations keep rank intact.

Algorithm $\mathcal{A}^{\text{obf}}(M)$

1. Choose a random full-rank $m \times m$ matrix C.
2. Choose a random $r \times r$ permutation matrix Π.
3. Let $K \leftarrow \mathcal{A}_{\text{rec}}(CM\Pi)$.

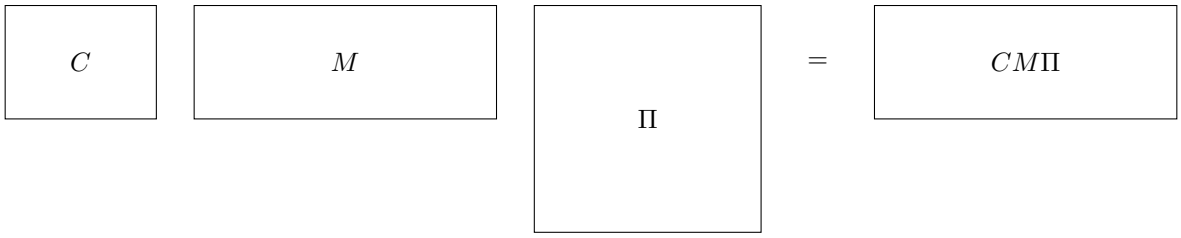

Figure 5: Visualization of the obfuscation procedure. The input $M \in \mathbb{F}_2^{m \times r}$ is *randomized* by multiplying with a random full-rank matrix $C \in \mathbb{F}_2^{m \times m}$ (which preserves the kernel and, in particular, the location of the k-SUM solution) and then *anonymized* by permuting the columns of the result with a permutation matrix $\Pi \in \mathbb{F}_2^{r \times r}$. The resulting matrix $CM\Pi$ is then given to the oracle $\mathcal{A}_{\text{rec}}$ for k-XOR recovery.

4. If K is a solution, return $K\Pi^{-1}$.

For efficiency reasons, we never explicitly construct Π, which would take $O(r^2)$ time. Instead, we use sparse representations of Π and Π^{-1}, which we can respectively sample and construct in $O(r)$ time. Note that the algorithm never returns a wrong solution since the two full-rank matrix multiplications cannot create a new solution. It is also clear that the runtime of this is $T + rm^2 = T + r \cdot \text{polylog}(r) = \widetilde{O}(T)$.

Before stating our main lemma regarding $\mathcal{A}^{\text{obf}}$, we introduce some new notation. We will denote the entire j^{th} column of a matrix M by $M[j]$. For an array A of size k containing unique numbers between 1 and r, we define $M[A]$ to be the matrix obtained by concatenating the k columns of M corresponding to the indices in A. Formally, $M[A][j] := M[A[j]]$. Adding bit vectors is defined as taking bitwise XOR for each position.

Lemma 7.5. *Let S_1 and S_2 be any two arrays of indices in $[r]$ of length k. Let V_1 and V_2 be any two $m \times k$ Boolean matrices of rank $k-1$ such that $\sum_{j=1}^{k} V_1[j] = \sum_{j=1}^{k} V_2[j] = \vec{0}$. Recall that γ is the success probability of $\mathcal{A}_{\text{rec}}$ Then, for large enough r,*

$$\Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_1] = V_1\right] = \Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_2] = V_2\right] \tag{6}$$

$$\Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds}\right] = \Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_1] = V_1\right] \tag{7}$$

$$\Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds}\right] \geq \frac{\gamma}{2} \tag{8}$$

Proof. Since the columns of V_1 sum to $\vec{0}$, its last column is a linear combination of its first $k-1$ columns. Therefore $V_1[[k-1]]$ has the same rank as V_1. Since V_1 has rank $k-1$ and $V_1[[k-1]]$ only has $k-1$ columns, all those columns must be linearly independent. Similarly, the first $k-1$ columns of V_2 must also be linearly independent. Since $k-1 < \log r \leq m$, we can find a full-rank matrix C_0 such that $C_0 V_1[[k-1]] = V_2[[k-1]]$. Since linear transformations preserve linear relations, we also have $C_0 V_1 = V_2$. Similarly, we can find a permutation Π_0 on $[r]$ that takes S_1 to S_2. Since both C_0 and Π_0 are invertible, $M[S_1] = V_1$ if and only if $C_0 M \Pi_0[S_2] = V_2$. Also, note that a matrix X is permissible if and only if $C_0 X \Pi_0$ is permissible, since permissibility is closed under full-ranked linear transformations of rows as well as permutations of columns. We can now prove the Eq. (6) as follows. Note that M is permissible if and only if $C_0 M \Pi_0$ is. Since D_{perm} is uniform on its support by Lemma 7.4, sampling M is the same as sampling $C_0 M \Pi_0$, we may write,

$$\begin{aligned}
&\Pr_{M \sim D_{perm}}[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_1] = V_1] \\
&\quad = \Pr_{C_0 M \Pi_0 \sim D_{perm}}[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_1] = V_1]
\end{aligned}$$

Since $C_0 V_1 = V_2$, and Π_0 takes S_1 to S_2,

$$= \Pr_{C_0 M \Pi_0 \sim D_{perm}} [\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid C_0 M \Pi_0[S_2] = V_2]$$
$$= \Pr_{C_0 M \Pi_0 \sim D_{perm}} [\mathcal{A}^{\text{obf}}(C_0^{-1}(C_0 M \Pi_0)\Pi_0^{-1}) \text{ succeeds} \mid C_0 M \Pi_0[S_2] = V_2]$$

Denoting the matrix $C_0 M \Pi_0$ by M', we get,

$$= \Pr_{M' \sim D_{perm}} [\mathcal{A}^{\text{obf}}(C_0^{-1} M' \Pi_0^{-1}) \text{ succeeds} \mid M'[S_2] = V_2]$$

Renaming M' to M.

$$= \Pr_{M \sim D_{perm}} [\mathcal{A}^{\text{obf}}(C_0^{-1} M \Pi_0^{-1}) \text{ succeeds} \mid M[S_2] = V_2]$$

Expanding the definition of $\mathcal{A}^{\text{obf}}$.

$$= \Pr_{\substack{M \sim D_{perm} \\ C \sim GL(m, \mathbb{F}_2) \\ \Pi \sim S_r}} [\mathcal{A}_{\text{rec}}(C C_0^{-1} M \Pi_0^{-1} \Pi) \text{ succeeds} \mid M[S_2] = V_2]$$

C is a full-rank $m \times m$ Boolean matrix if and only if CC_0^{-1} is. Similarly, $\Pi \in S_r$ if and only if $\Pi_0^{-1}\Pi \in S_r$. We can therefore rewrite as follows.

$$= \Pr_{\substack{M \sim D_{perm} \\ CC_0^{-1} \sim GL(m, \mathbb{F}_2) \\ \Pi_0^{-1}\Pi \sim S_r}} [\mathcal{A}_{\text{rec}}(C C_0^{-1} M \Pi_0^{-1} \Pi) \text{ succeeds} \mid M[S_2] = V_2]$$

Denoting CC_0^{-1} by C' and $\Pi_0^{-1}\Pi$ by Π'.

$$= \Pr_{\substack{M \sim D_{perm} \\ C' \sim GL(m, \mathbb{F}_2) \\ \Pi' \sim S_r}} [\mathcal{A}_{\text{rec}}(C' M \Pi') \text{ succeeds} \mid M[S_2] = V_2]$$

Using the definition of $\mathcal{A}^{\text{obf}}$.

$$= \Pr_{M \sim D_{perm}} [\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S_2] = V_2]$$

To prove Eq. (7), observe that every permissible matrix M has one and only one pair (S, V) such that V is a $m \times k$ Boolean matrix of rank $k-1$ with $\sum_{j=1}^{k} V[j] = \vec{0}$ and $M[S] = V$. Since the success probability of $\mathcal{A}^{\text{obf}}$ is the same regardless of S and V, the overall success probability must be the same as the success probability conditioned on any particular values of S and V.
We are now ready to prove Eq. (8).

$$\Pr_{M \sim D_{perm}} [\mathcal{A}^{\text{obf}}(M) \text{ succeeds}]$$
$$= \Pr_{\substack{M \sim D_{perm} \\ C \sim GL(m, \mathbb{F}_2) \\ \Pi \sim S_r}} [\mathcal{A}_{\text{rec}}(C M \Pi) \text{ succeeds}]$$

M is permissible if and only if $CM\Pi$ is. Since D_{perm} is uniform on its support by Lemma 7.4, sampling M is the same as sampling $CM\Pi$.

$$= \Pr_{\substack{CM\Pi \sim D_{perm} \\ C \sim GL(m,\mathbb{F}_2) \\ \Pi \sim S_r}} [\mathcal{A}_{\text{rec}}(CM\Pi) \text{ succeeds}]$$

Rewriting $CM\Pi$ as M'.

$$= \Pr_{M' \sim D_{perm}} [\mathcal{A}_{\text{rec}}(M') \text{ succeeds}]$$

From the definition of D_{perm},

$$\geq \Pr_{M \sim D_1} [\mathcal{A}_{\text{rec}}(M) \text{ succeeds}] - \Pr_{M \sim D_1} [M \text{ is not permissible}]$$

Using Lemma 7.4, we get,

$$= \gamma - o(\gamma) > \frac{\gamma}{2},$$

where the last inequality holds for sufficiently large r. □

7.1 Hardness Amplification at Density $\leq 1 - \frac{\log\log\log r}{\log r}$

Our next step is building the algorithm $\mathcal{A}^{\text{amp}}$ that will achieve the required success probability for densities below $1 - \frac{\log\log\log r}{\log r}$. The algorithm on input M is formally described below.

Algorithm $\mathcal{A}^{\text{amp}}(M)$

1. Repeat $\frac{64 \log r}{\gamma^{2k+2}}$ times:
   - 1.1. Make a new copy of M and call it M'
   - 1.2. Repeat $r \log \frac{1}{\gamma}$ times:
     - 1.2.1. Pick i uniformly at random from $[r]$
     - 1.2.2. Replace $M'[i]$ with a random element from GF_{2^m} different from $M'[i]$
   - 1.3. Call $\mathcal{A}^{\text{obf}}(M')$
   - 1.4. If the previous call succeeds and returns a k-tuple that was unchanged from M to M', return that.

It is easy to see that the algorithm never returns a wrong answer. Note that the inner loop takes $O(m)$ time. Therefore, the outer loop takes $\widetilde{O}(T) + O(mr \log \frac{1}{\gamma} + mr)$ time. Since $m = O(\text{polylog}(r))$, $\frac{1}{\gamma} = O(\text{polylog}(r))$ and $T = \Omega(r)$, the expression simplifies to $\widetilde{O}(T)$. The runtime of the whole algorithm is therefore $O(\frac{\log r}{\gamma^{2k+2}}) \cdot \widetilde{O}(T) = \widetilde{O}(T)$, as desired. To analyze the success probability of $\mathcal{A}^{\text{amp}}$, we start with a couple of definitions.

Definition 7.6 (Correspondence Graph)**.** *For a permissible matrix M with the k-XOR solution $S \subset [r]$, we define its* correspondence graph G_M *as follows: create a vertex for every Boolean matrix $M' \in \mathbb{F}_2^{m \times r}$ such that $M[S] = M'[S]$. Two vertices are connected by an edge if and only if the corresponding matrices only differ in one column.*

Lemma 7.7. *G_M is isomorphic to $H(r-k, 2^m)$.*

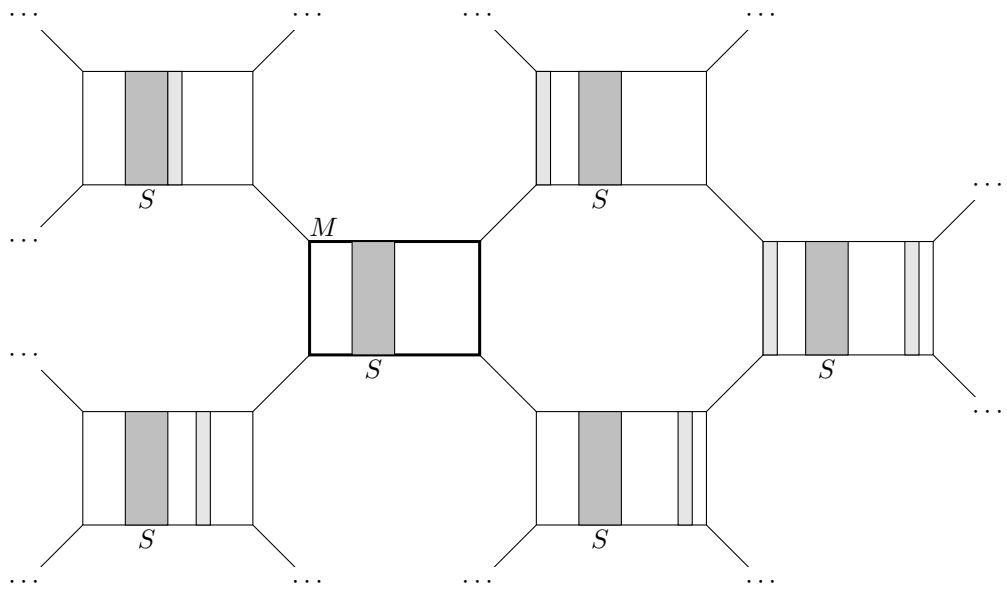

Figure 6: The Hamming graph representing the inner loop of $\mathcal{A}^{\text{amp}}(M)$. Each iteration resamples a random column of M and corresponds to taking a random step in the graph, assuming this does not destroy the solution S. Intuitively, the oracle cannot consistently fail on all the vertices we visit, and thus the process must eventually return the set S. The proof of this statement amounts to bounding the edge expansion of this graph.

Proof. We are allowed to change $r-k$ columns of M, and there are 2^m possibilities for each of those columns. The isomorphism follows from definition. □

Lemma 7.8. *The fraction of vertices corresponding to matrices which are not permissible is $o(\gamma)$ in any correspondence graph G_M as long as the number of columns of the matrix, r, is large enough.*

Proof. We will show that for any permissible M, if we choose the $r-k$ columns outside its k-XOR solution uniformly at random from GF_{2^m}, we end up with a permissible matrix with probability $1-o(\gamma)$. This is equivalent to the lemma statement.

The proof of Lemma 7.4 establishes this result almost directly.

1. All the columns are unique with probability at least $1-\frac{1}{r}$, as before.
2. With very high probability, there will be no new solutions other than the k fixed columns. We can bound the probability of a new solution by

$$\frac{1}{2^m}\left(\binom{r-k}{k}\binom{k}{0}+\binom{r-k}{k-1}\binom{k}{1}+\cdots\binom{r-k}{1}\binom{k}{k-1}\right)$$

   Since k is a constant, this sum is $O(r^{k(1-1/\Delta)})=o(\gamma)$
3. Since we start with a permissible matrix and do not disrupt the k columns containing its solution, the solution will always have rank $k-1$.
4. The matrix has rank m even ignoring the k fixed columns with probability at least

$$(1-2^{m+k-r-1})^m \geq 1-m2^{m+k-r} = 1-\frac{m2^{m+k}}{2^r} \geq 1-\frac{2^{\text{polylog}(r)}}{2^r} = 1-o(\gamma)$$

   since $m=O(\text{polylog}(r))$ and $k<m$ for large enough r.

As before, the result follows from an application of the union bound. □

For a pictorial representation of G_M, see Fig. 6. Note that if we are lucky enough to not destroy the solution while running the inner loop, we end up taking $r \log \frac{1}{\gamma}$ steps in this graph.

Definition 7.9. *For a permissible matrix M, its* correspondence power graph G_M^* *is a multigraph which has the same vertices as G_M, and has t edges between M_1 and M_2 where t is the number of paths of length $r \log \frac{1}{\gamma}$ from M_1 to M_2 in G_M. In other words, we can obtain the adjacency matrix of G_M^* by raising the adjacency matrix of G_M to the power $r \log \frac{1}{\gamma}$.*

Lemma 7.10. *The algebraic expansion of G_M^* is $((r-k)(2^m-1) - 2^m)^{r \log \frac{1}{\gamma}}$.*

Proof. By Lemmas 2.7 and 7.7, we know that for G_M, the highest eigenvalue is $(r-k)(2^m-1)$, the second highest eigenvalue is $(r-k)(2^m-1) - 2^m$, and the lowest eigenvalue is $k-r$. Note that raising a matrix to a certain power also raises its eigenvalues to the same power, and the algebraic expansion of a graph with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ is defined as $\max(|\lambda_2|, |\lambda_n|)$. Since $(r-k)(2^m-1) - 2^m > r-k$ for any $r > k$, the lemma follows. □

Lemma 7.11. *Each vertex of G_M^* has degree $((r-k)(2^m-1))^{r \log \frac{1}{\gamma}}$*

Proof. Lemma 7.7 implies that G_M is a regular graph of degree $d = (r-k)(2^m-1)$. Therefore, starting from any vertex v, there are exactly d^l paths of length l. By definition, this implies G_M^* is $((r-k)(2^m-1))^{r \log \frac{1}{\gamma}}$-regular. □

Note that taking $r \log \frac{1}{\gamma}$ steps in G_M is equivalent to taking 1 step in G_M^*.

Definition 7.12 (Good Vertices)**.** *We call a matrix X* good *if it is permissible and $\mathcal{A}^{\text{obf}}$ has at least a $\frac{\gamma}{4}$ success probability on it.*

$$\Pr[\mathcal{A}^{\text{obf}}(X) \text{ succeeds}] \geq \frac{\gamma}{4}$$

We call a vertex good *if it corresponds to a good matrix.*

Lemma 7.13. *Let M be any permissible matrix whose number of columns r is large enough. At least a $\frac{\gamma}{8}$ fraction of the vertices of G_M are good.*

Proof. Note that only a $o(\gamma)$ fraction of the vertices in G_M are not permissible. We choose r large enough so that this fraction doesn't exceed $\frac{\gamma}{8}$. The rest of the vertices exactly correspond to the support of D_{perm} which has the same k-XOR solution in the same position as M. Since D_{perm} is uniform on its support (Lemma 7.4), conditioning on a random vertex of G_M being permissible is the same as sampling from D_{perm} and conditioning on it having the same k-XOR solution in the same place as M. Lemma 7.5 now yields

$$\mathop{\mathbb{E}}_{M' \sim G_M} [\Pr[\mathcal{A}^{\text{obf}}(M') \text{ succeeds}] \mid M' \text{ is permissible}] \geq \frac{\gamma}{2} \tag{9}$$

We proceed via a proof by contradiction. Let us assume that the lemma is false. This implies that at least a $1 - \frac{\gamma}{8}$ fraction of the vertices in G are either not permissible or has a success probability less than $\frac{\gamma}{4}$. Since the fraction of vertices not permissible is at most $\frac{\gamma}{8}$, at least a $1 - \frac{\gamma}{4}$ fraction of the vertices are permissible but not good. Then, we have:

$$\mathop{\mathbb{E}}_{M' \sim G_M} [\Pr[\mathcal{A}^{\text{obf}}(M') \text{ succeeds}] \mid M' \text{ is permissible}]$$

This expected value is maximized when all the good vertices have success probability 1 and all the other permissible matrices in G_M have success probability $\frac{\gamma}{4}$.

$$\leq \frac{\frac{\gamma}{8}}{1 - \frac{\gamma}{8}} + \frac{1 - \frac{\gamma}{4}}{1 - \frac{\gamma}{8}} \cdot \frac{\gamma}{4}$$

This simplifies as follows.

$$= \frac{\gamma}{8-\gamma} + \frac{(4-\gamma)\gamma}{2(8-\gamma)} = \frac{6\gamma - \gamma^2}{2(8-\gamma)} < \frac{8\gamma - \gamma^2}{2(8-\gamma)} = \frac{\gamma}{2}$$

This inequality directly contradicts Eq. (9), and hence we are done. □

Definition 7.14 (Bad Vertices)**.** *Let M be a permissible matrix. We call a vertex X of G^*_M* bad *if it is permissible and less than $\frac{\gamma}{16}$ of its outgoing edges connect to a good vertex.*

Lemma 7.15. *Let M be any permissible matrix whose number of columns r is large enough. The fraction of bad vertices in G^*_M is $o(1)$.*

Proof. Let S be the set of bad vertices and T be the set of good vertices. We denote the set of all vertices of G^*_M by V. We know from Lemma 7.11 that G^*_M is a regular graph; we shall call its degree d. We will also denote the algebraic expansion of G^*_M by λ. The expander mixing lemma (Lemma 2.6) now implies

$$\left| E(S,T) - \frac{d \cdot |S| \cdot |T|}{|V|} \right| \leq \lambda \sqrt{|S| \cdot |T|} \quad \implies \quad E(S,T) \geq \frac{d \cdot |S| \cdot |T|}{|V|} - \lambda \sqrt{|S| \cdot |T|}.$$

The number of edges between S and T is at most $\frac{d\gamma|S|}{16}$ since each bad vertex, by Definition 7.14, has at most $\frac{d\gamma}{16}$ edges connecting to a good vertex. Therefore,

$$\begin{aligned}
\frac{d\gamma|S|}{16} \geq \frac{d \cdot |S| \cdot |T|}{|V|} - \lambda \sqrt{|S| \cdot |T|} \quad &\implies \quad \frac{d\gamma}{16} \cdot \frac{|S|}{|V|} \geq d \cdot \frac{|S|}{|V|} \cdot \frac{|T|}{|V|} - \lambda \sqrt{\frac{|S|}{|V|} \cdot \frac{|T|}{|V|}} \\
&\implies \quad \frac{|S|}{|V|} \left(\frac{d|T|}{|V|} - \frac{d\gamma}{16} \right) \leq \lambda \sqrt{\frac{|S|}{|V|} \cdot \frac{|T|}{|V|}} \\
&\implies \quad \sqrt{\frac{|S|}{|V|}} \leq \frac{\lambda \sqrt{\frac{|T|}{|V|}}}{\frac{d|T|}{|V|} - \frac{d\gamma}{16}} \\
&\implies \quad \frac{|S|}{|V|} \leq \left(\frac{\lambda}{d} \right)^2 \cdot \frac{\frac{|T|}{|V|}}{\left(\frac{|T|}{|V|} - \frac{\gamma}{16} \right)^2}.
\end{aligned}$$

Since T is the set of good vertices, Lemma 7.13 implies $\frac{|T|}{|V|} \geq \frac{\gamma}{8}$. Therefore,

$$\frac{|S|}{|V|} \leq \left(\frac{\lambda}{d} \right)^2 \cdot \frac{\frac{\gamma}{8}}{\left(\frac{\gamma}{8} - \frac{\gamma}{16} \right)^2} = \left(\frac{\lambda}{d} \right)^2 \cdot \frac{32}{\gamma}$$

We now plug in the values of λ and d from Lemmas 7.10 and 7.11 respectively to get

$$\frac{|S|}{|V|} \leq \left(1 - \frac{2^m}{(r-k)(2^m-1)} \right)^{2r \log \frac{1}{\gamma}} \cdot \frac{32}{\gamma} < \left(1 - \frac{1}{r} \right)^{2r \log \frac{1}{\gamma}} \cdot \frac{32}{\gamma}$$

We can further bound the above expression by using the inequality $\left(1 - \frac{1}{r}\right)^r \leq \frac{1}{e}$

$$\frac{|S|}{|V|} < \frac{32 e^{-2 \log \frac{1}{\gamma}}}{\gamma} = 32\gamma = \Theta\left(\frac{1}{\text{polylog}(r)} \right) = o(1)$$

We have thus shown that the fraction of bad vertices in the correspondence power graph of any permissible matrix is $o(1)$. □

We are now ready to conclude our second step.

Lemma 7.16. *The average-case success probability of $\mathcal{A}^{\text{amp}}$ is $1 - o(1)$. This algorithm works at any $\Omega\left(\frac{1}{\text{polylog}(r)}\right)$ density $\Delta \leq 1 - \frac{\log\log\log r}{\log r}$ for large enough r.*

Proof. With probability $1 - o(1)$, the input matrix M is permissible. In that case, M has a unique solution. Lemma 7.15 tells us that among all matrices that contain this exact solution at this exact position, only a $o(1)$ fraction can be *bad*. So we can assume with probability $1 - o(1)$ that M has at least a $\frac{\gamma}{16}$ fraction of its edges leading to good vertices.

Each iteration of the outer loop of $\mathcal{A}^{\text{amp}}$ makes $r \log \frac{1}{\gamma}$ column replacements before calling $\mathcal{A}^{\text{obf}}$. The original solution is preserved if we choose one of the $r - k$ columns not in the solution at every step. This happens with probability $(1 - \frac{k}{r})^{r \log \frac{1}{\gamma}} > e^{-2k \log \frac{1}{\gamma}} = \gamma^{2k}$, since $(1 - \frac{1}{t})^t > \frac{1}{e^2}$ for $t > 2$. Note that when we preserve the solution, the inner loop takes one random step in G^*_M. With probability at least $\frac{\gamma}{16}$, this lands us into a good vertex where $\mathcal{A}^{\text{obf}}$ succeeds with probability at least $\frac{\gamma}{4}$. Thus, each iteration of the outer loop has a probability at least $\frac{\gamma^{2k+2}}{64}$ of recovering the original solution for any input matrix which is permissible and not a bad vertex. Observe that different iterations of the outer loop are independent once we condition on the input matrix. Since the outer loop runs $\frac{64 \log r}{\gamma^{2k+2}}$ times, the probability of recovering the solution in at least one of the iterations is

$$1 - \left(1 - \frac{\gamma^{2k+2}}{64}\right)^{\frac{64 \log r}{\gamma^{2k+2}}} > 1 - \frac{1}{e^{\log(r)}} = 1 - \frac{1}{r} = 1 - o(1) \qquad \square$$

7.2 Hardness Amplification at Density ≤ 1

So far, we have only proved Theorem 7.1 for all densities $\leq 1 - \frac{\log\log\log r}{\log(r)}$. To complete the proof, we will now dispense with this assumption and only assume $\Delta \leq 1$. Note that this implies that permissible matrices may no longer take up a $1 - o(1)$ fraction of the input space, and we can no longer pretend the input is permissible without a significant loss in success probability anymore.

Proof of Theorem 7.1. Let us define

$$\Delta_0 := 1 - \frac{\log\log\log r}{\log r}$$

To complete the proof, we will assume that $\mathcal{A}_{\text{rec}}$ is an average-case algorithm for k-XOR at some fixed density $\Delta \in (\Delta_0, 1]$ with success probability $\gamma = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$ and time complexity $T(r) = \Omega(r)$. We will construct a new algorithm $\mathcal{B}$ to solve the same problem at the same density Δ with success probability $1 - o(1)$.

First, we use $\mathcal{A}_{\text{rec}}$ to construct an algorithm $\mathcal{A}_0$ for solving the same problem at density Δ_0. On input M_0 (which is sampled from $D_1^{\Delta_0}$), this algorithm $\mathcal{A}_0$ simply throws away a randomly chosen $1 - \frac{\Delta_0}{\Delta}$ fraction of the rows to get a density Δ instance M. It then checks if $\mathcal{A}_{\text{rec}}(M)$ returns a k-tuple which is also a solution for the original input M_0, and if so, returns it. Observe that since the rows of M_0 are independently chosen, M has the same distribution as D_1^{Δ}. Therefore, $\mathcal{A}_{\text{rec}}$ returns a solution with probability γ. Any k-tuple that sum to 0 in M_0 obviously also sum to 0 in M. If M has c solutions of k-XOR, we can argue by symmetry that they are all equally likely to be a solution for M_0. Now, note that since M has density ≤ 1, the expected value of c is less than 3. If the planted solution is S,

$$\mathbb{E}[c] = \sum_{\substack{|\kappa|=k \\ \kappa \subset [r]}} \Pr[\kappa \text{ is a solution}] = 1 + \sum_{\substack{|\kappa|=k \\ \kappa \subset [r] \\ \kappa \neq S}} \Pr[\kappa \text{ is a solution}] = 1 + \frac{\binom{r}{k} - 1}{2^m} < 3.$$

We can therefore apply Markov's inequality (Lemma 2.2) to conclude that with probability at least $1/2$, there are less than 6 k-XOR solutions for M. So with probability at least $\gamma \cdot \frac{1}{2} \cdot \frac{1}{5}$, we will call $\mathcal{A}_{\text{rec}}$ on a matrix with at most 5 solutions, $\mathcal{A}_{\text{rec}}$ will succeed, and the solution it returns will also be a solution for our

original input M_0. The success probability of $\mathcal{A}_0$ is therefore at least $\frac{\gamma}{10} = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$. The runtime of $\mathcal{A}_0$ is clearly $O(mr) + T = \widetilde{O}(T)$.

We now use our success amplification procedure described in Section 7.1 to obtain an average-case recovery algorithm $\mathcal{A}_0^{\text{amp}}$ with runtime $\widetilde{O}(T)$ that solves density Δ_0 instances with probability $1 - o(1)$.

Now we are ready to construct the final algorithm $\mathcal{B}$. Given a density Δ instance M with dimensions $m \times r$, our algorithm does the following:

Algorithm $\mathcal{B}$

1. Define $t := (\log\log r)^{k+2}$
2. Repeat t times:
   - 2.1. Add $k\log(r)\left(\frac{1}{\Delta_0} - \frac{1}{\Delta}\right)$ rows to M. Each new bit added is chosen i.i.d. from $\text{Ber}_{\frac{1}{2}}$.
   - 2.2. Call $\mathcal{A}_0^{\text{amp}}$ on the new matrix M_0.
   - 2.3. If a solution was returned in the last step, check if that is also a solution for M. If so, return it.

Observe that M is guaranteed to have at least one solution S since it is sampled from D_1^{Δ}. The probability of S being a k-XOR solution for M_0 is exactly $2^{k\log(r)\left(\frac{1}{\Delta} - \frac{1}{\Delta_0}\right)}$ since each additional row halves the chance of the original solution being preserved. We now bound the probability p that at least one iteration of the inner loop ran $\mathcal{A}_0^{\text{amp}}$ on a matrix where S was a solution.

$$p \geq 1 - \left(1 - 2^{k\log(r)\left(\frac{1}{\Delta} - \frac{1}{\Delta_0}\right)}\right)^t$$

Now, since $\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{e}$ and $\Delta \leq 1$, it follows that,

$$\geq 1 - \exp\left(-t \cdot r^{k\left(1 - \frac{1}{\Delta_0}\right)}\right)$$

We now substitute in $\Delta_0 = 1 - \frac{\log\log\log r}{\log r}$ to get,

$$= 1 - \exp\left(-t \cdot r^{\frac{k\log\log\log r}{\log\log\log r - \log r}}\right)$$

We now let $t = (\log\log r)^{k+2}$ and obtain,

$$= 1 - \exp\left(-(\log\log r)^{k+2} \cdot (\log\log r)^{\frac{k\log(r)}{\log\log\log r - \log r}}\right)$$
$$= 1 - \exp\left(-\log\log r \cdot (\log\log r)^{\frac{\log(r) - (k+1)\log\log\log r}{\log(r) - \log\log\log r}}\right)$$

We now choose a large enough r such that $\log(r) > (k+1)\log\log\log r$.

$$\geq 1 - \exp\left(-\log(r)\right) = 1 - o(1)$$

When M_0 does have a solution, it is easy to see that it has the same distribution as $D_1^{(\Delta_0)}$ and hence, $\mathcal{A}_0^{\text{amp}}$ returns a solution with probability $1 - o(1)$. Taking a union bound, we conclude that with probability $1 - o(1)$, we call $\mathcal{A}_0^{\text{amp}}$ on a matrix with a k-XOR solution at least once *and* it returns that solution. Any solution to M_0 is always a solution to M, and our algorithm therefore returns it. We thus have the required success probability.

The runtime of $\mathcal{B}$ is clearly t times the runtime of $\mathcal{A}_0^{\text{amp}}$. Since $\mathcal{A}_0^{\text{amp}}$ runs in $\widetilde{O}(T)$ and $t = O(\log r)$, our algorithm also runs in $\widetilde{O}(T)$ time. □

8 Public-Key Encryption from k-XOR and Learning Parity with Noise

In this section, we propose a class of public-key bit encryption schemes based on the planted k-XOR problem and the learning parity with noise (LPN) problem. By instantiating the class appropriately, we strike various trade-offs between the hardness required for LPN and the densities at which we assume k-XOR is hard. Specifically, the cryptosystem can be instantiated assuming either 1) $2^{m^{0.5}}$-hardness of LPN for secrets of size m; or, 2) 2^{m^c}-hardness of LPN (for any constant $c > 0$) and hardness of k-XOR at density $1/\text{polylog}(r)$; or, 3) $2^{\omega(1)}$-hardness of LPN and hardness of k-XOR at density $1/r^{o(1)}$.

At a high level, we simply generate an instance of planted k-XOR and use the result as the public key, with the location of the planted solution being used as the secret key. The security parameter is the number r of vectors to generate. Recall that such an instance can be interpreted as a matrix $X \in \mathbb{F}_2^{m \times r}$ where $m = k \lg r/\Delta$. We then encrypt a bit as follows. To encrypt zero, we sample a uniform random vector of length r. To encrypt one, we take a random linear combination of the rows of the public key, i.e. we sample a random vector $s \leftarrow \mathbb{F}_2^m$ and output the ciphertext $s^\top X$. Our hope is that only a recipient who knows the location of planted vector can distinguish $s^\top X$ from a random vector. Unfortunately, this transformation preserves the kernel of X which makes distinguishing between an encryption of zero and one easy. To circumvent this issue, we add i.i.d. noise to each entry of the ciphertext, i.e. we sample $e \leftarrow \text{Ber}_\eta^r$ where $\eta \in (0, 1)$ is some *noise parameter*. Distinguishing such a noisy linear combination from a random vector is known as the learning parity with noise (LPN) problem which is widely considered to be hard even for lower noise rates $\eta = \Omega\left(1/\sqrt{m}\right)$ [Ale03, Pie12].

Technically speaking, we will obtain a *weak* public-key encryption scheme in the sense that the advantage of the adversary is not negligible in the security parameter but only vanishing. This may be amplified using error-correction with an appropriate hardcore lemma (such as [HR05, Hol05] for full PKE). Formally speaking, our definition of (weak) PKE is as follows.

Definition 8.1 (Weak Public-Key Encryption)**.** *Let $PKE = \langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ be a triple of functions. We say PKE is a* (weak) public key bit encryption scheme *if it satisfies the following two properties.*

- (Correctness). *It holds that,*

$$\Pr\left[\mathsf{Dec}(sk, \mathsf{Enc}(pk, b)) = b\right] = 1 - o(1),$$

  whenever $\langle pk, sk \rangle \leftarrow \mathsf{KeyGen}(1^r)$, where the randomness is taken over uniform choice of b and the random coins used by PKE.

- (Indistinguishability). *For any PPT adversary $\mathcal{A}$, it holds that,*

$$\Pr\left[\mathcal{A}(pk, \mathsf{Enc}(pk, b)) = b\right] \leq \frac{1}{2} + o(1),$$

  whenever $\langle pk, sk \rangle \leftarrow \mathsf{KeyGen}(1^r)$, where the randomness is taken over uniform choice of b, the random coins used by PKE, as well as the random coins used by $\mathcal{A}$.

If instead, indistinguishability holds only against any adversary $\mathcal{A}$ that runs in fixed polynomial time $O(r^c)$ for some constant $c > 0$ then we say PKE is a (weak) r^c-fine-grained public-key bit encryption scheme*.*

8.1 The Cryptosystem

In this section, we propose a class of cryptosystems $PKE_{\eta,k,\ell,m}$ that is parameterized by four functions $\eta : \mathbb{N} \to (0, 1)$ and $k, \ell, m : \mathbb{N} \to \mathbb{N}$, where for security parameter r, $\eta(r)$ is the noise-rate, $k(r)$ is the size of the solution, $m(r)$ is the dimension of the vectors, and $\ell(r)$ is the number of repetitions of the cryptosystem we described (this is necessary to satisfy correctness). We will show that $PKE_{\eta,k,\ell,m}$ can be instantiated in various ways to obtain public-key encryption, by striking a trade-off between the assumed hardness of LPN and the densities at which k-XOR is assumed hard.

Key Generation $\mathsf{KeyGen}(1^r)$**.**

1. Let $pk \xleftarrow{\$} \mathbb{F}_2^{m(r)\times r}$.
2. Choose a random set $sk \subseteq [r]$ with $|sk| = k(r)$.
3. Let $i \in sk$ be the smallest index and let $pk_i \leftarrow -\sum_{\substack{j\in sk\\ j\neq i}} pk_j$.
4. Return $\langle pk, sk\rangle$.

Here, we will interpret each $sk \in \mathbb{F}_2^r$ as the characteristic vector for the set $sk \subseteq [r]$.

Encryption $\mathsf{Enc}(pk, b)$**.**

1. If $b = 0$.
   1.1. Return $C \xleftarrow{\$} \mathbb{F}_2^{\ell(r)\times r}$.
2. If $b = 1$.
   2.1. Let $S \xleftarrow{\$} \mathbb{F}_2^{\ell(r)\times m(r)}$.
   2.2. Let $E \leftarrow \mathrm{Ber}_{\eta(r)/2}^{\ell(r)\times r}$.
   2.3. Return $C \leftarrow S\,pk + E$.

Decryption $\mathsf{Dec}(sk, C)$**.**

1. Return 0 if $\|C\,sk\|_0 > \ell(r)\left(\frac{1}{2} - \frac{(1-\eta(r))^{k(r)}}{4}\right)$; else return 1.

Moving forward, we will implicitly fix the value of r and thus treat η, k, ℓ, m as constants such that we may drop their dependence on r.

Lemma 8.2. *The cryptosystem is ε-correct when $\ell \geq 32\,(1-\eta)^{-2k}\ln(1/\varepsilon)$.*

Proof. It will be convenient for our proof to think of the error of $\mathrm{Ber}_{\eta/2}$ added to each bit as $\mathrm{Ber}_\eta \cdot \mathrm{Ber}_{1/2}$ – for each location, with probability η, XOR it with a uniformly random bit.

Suppose for $\ell = 1$ that we encrypt $b = 0$ such that c is a uniformly random vector from $\{0,1\}^r$. It is clear in this case, as $sk \neq 0^r$, that $sk^\top c$ a uniformly random bit, i.e. if $\langle pk, sk\rangle \leftarrow \mathsf{KeyGen}(r)$, then we get that, $\Pr[sk^\top c = 1] = \frac{1}{2}$. If instead we encrypt $b = 1$ and there are no errors in the k positions corresponding to the planted set, then $sk^\top c = 0$. This happens with probability $(1-\eta)^k$. If instead, some error occurred in the planted set, $sk^\top c$ will be uniformly distributed. As such, if c is obtained by encrypting 1,

$$\Pr[sk^\top c = 0] = (1-\eta)^k + \frac{1-(1-\eta)^k}{2} = \frac{1}{2} + \frac{(1-\eta)^k}{2}.$$

This does not provide useful correctness by itself so we amplify the difference by repeating the process ℓ times and take the majority vote. Now suppose we have ℓ iterations and consider the case of $b = 0$. Let $E_i = c_i^\top sk$, for $i = 1 \ldots \ell$. Then all E_i are i.id. and satisfy $\mathbb{E}[E_i] = \frac{1}{2}$. It follows by a Chernoff bound (Lemma 2.5) that,

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(pk, 0), sk) = 1] = \Pr\left[\sum_{i=1}^{\ell} E_i \leq \ell\left(\frac{1}{2} - \frac{(1-\eta)^k}{4}\right)\right] \leq e^{-\left(\frac{1}{4}(1-\eta)^k\right)^2 \ell/2}.$$

This gives a decryption error of at most ε whenever $\ell \geq 32\,(1-\eta)^{-2k}\ln(1/\varepsilon)$. The case for $b = 1$ is identical. □

We aim to show that no adversary can efficiently distinguish between an encryption of zero and an encryption of one. Formally, we will define the search LPN problem as follows.

Definition 8.3 (LPN Search Problem)**.** *An algorithm $\mathcal{A}$ is said to solve the learning parity with noise (LPN) search problem with noise rate η with probability ϵ if, given $\langle X, Xs + e\rangle$ where $X \leftarrow \mathbb{F}_2^{r\times m}$, $s \leftarrow \mathbb{F}_2^m$, and $e \leftarrow \mathrm{Ber}_\eta^r$, it outputs s with probability at least ϵ, where the randomness is taken over the instance and the random coins used by the algorithm.*

The best known algorithm for search LPN when $\eta = \Theta(1)$ is by Blum, Kalai and Wasserman [BKW03] that for an m-dimensional secret runs in subexponential time $2^{O(m/\log(m))}$. There is another algorithm by Esser, Kübler and May [EKM17] that runs in time $2^{O(\eta m)}$ and thus outperforms the BKW algorithm when the noise-rate is small. We also consider the decision LPN problem that we formally define as follows.

Definition 8.4 (LPN Decision Problem)**.** *The (decision) Learning Parity with Noise (LPN) problem over $\mathbb{F}_2$ with noise rate $\eta \in (0, 1)$ is to distinguish between the following distributions:*

1. *$\langle X, Xs + e\rangle$, where $X \leftarrow \mathbb{F}_2^{r\times m}$, $s \leftarrow \mathbb{F}_2^m$, and $e \leftarrow \mathrm{Ber}_\eta^r$.*
2. *$\langle X, r\rangle$, where $X \leftarrow \mathbb{F}_2^{r\times m}$, and $r \leftarrow \mathbb{F}_2^r$.*

We say that an algorithm $\mathcal{A}$ solves the decision LPN problem with advantage ϵ if,

$$\Big|\Pr[\mathcal{A}(X, Xs + e) = 1] - \Pr[\mathcal{A}(X, r) = 1]\Big| \geq \epsilon,$$

where the randomness is taken over the instance and the random coins used by $\mathcal{A}$.

The search and decision LPN problems are known to be polynomially equivalent, as showed in [KSS10]. We restate their result as follows.

Lemma 8.5 (Katz, Shin, Smith, [KSS10])**.** *If there is an algorithm that runs in time T and solves the decision LPN problem with advantage ϵ, then there is an algorithm that runs in time $O\left(\frac{T\, m \log m}{\epsilon^2}\right)$ and solves the search LPN problem with probability $\epsilon/4$.*

Definition 8.6 (Hardness of LPN)**.** *We say that η-*noise LPN is hard for time $T(m)$ *if any algorithm $\mathcal{A}$ that runs in time $O(T(m))$ has an advantage $o(1)$ in solving the search LPN problem for secrets of size m with noise rate $\Omega(\eta)$.*

We can leverage the equivalence of Lemma 8.5 to to translate hardness of LPN into a bound on the advantage of an algorithm that solves decision LPN.

Corollary 8.7. *If η-noise LPN is hard for time $T(m)$ then there is no algorithm that runs in time $O(T(m))$ and solves the decision LPN problem with advantage $\omega(1/T(m))$.*

Definition 8.8 (Hardness of k-XOR)**.** *We say that k-*XOR is hard at density Δ *if any algorithm $\mathcal{A}$ that runs in time $O(r^{k/2\,(1-\Omega(1))})$ has a probability $o(1)$ of solving the average-case k-XOR search problem in $\mathsf{G}_{k\text{-}XOR}^{(\Delta)}$.*

Lemma 8.9 (Indistinguishability of Cryptosystem)**.** *Suppose the following conditions are satisfied.*

(1) η-noise LPN is $T(m)$-hard for secrets of size m.

(2) k-XOR is hard at density $\frac{2\lg T(m)}{m}$.

(3) $\ell = o(T(m))$.

Then any adversary that runs in time $O(r^{k/2\,(1-\Omega(1))})$ has an advantage $o(1)$ in distinguishing an encryption of zero from an encryption of one in $PKE_{\eta,k,\ell,m}$.

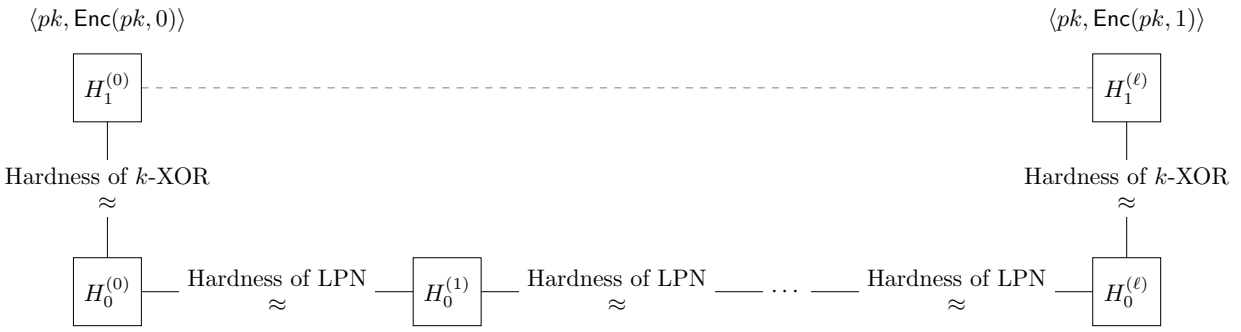

Figure 7: Structure of the hybrid argument. The distribution $H_b^{(i)}$ generates a public key pk using the distribution D_b, generates i vectors of the form $s^\top pk + e^\top$ and $\ell - i$ random vectors from $\mathbb{F}_2^r$. The distribution $H_1^{(0)}$ thus corresponds to an encryption of 0, while $H_1^{(\ell)}$ is an encryption of 1.

Proof. We proceed using a hybrid argument (see Fig. 7) and define a class of distributions $H_b^{(i)}$, each of which generates a public key pk using the distribution D_b, generates i vectors of the form $s^\top pk + e^\top$ and $\ell - i$ random vectors from $\mathbb{F}_2^r$. Note that $H_1^{(0)}$ is an encryption of 0 and $H_1^{(\ell)}$ is an encryption of 1. By Definition 8.8, any adversary that runs in time $O(r^{k/2\,(1-\Omega(1))})$ has advantage $o(1)$ in distinguishing $H_1^{(0)}$ from $H_0^{(0)}$ (respectively, $H_1^{(\ell)}$ from $H_0^{(\ell)}$). Now, by condition (3) and Corollary 8.7 we have that $H_0^{(i)}$ is $O(1/T(m))$-indistinguishable from $H_0^{(i+1)}$ for each $0 \le i < \ell$ which combined with condition (1) implies that $H_0^{(0)}$ is $o(1)$-indistinguishable from $H_0^{(\ell)}$. We then conclude that $H_1^{(0)}$ and $H_1^{(\ell)}$ are $o(1)$-indistinguishable to any adversary running in time $\min\left(T(m), r^{k/2\,(1-\Omega(1))}\right)$. We thus need that $T(m) \ge r^{k/2} = 2^{m\Delta/2}$ which is equivalent to condition (2). □

Using Lemma 8.9, we can identify various tradeoffs between the hardness assumed for LPN and the densities at which k-XOR is assumed to be hard.

Theorem 8.10 (PKE from k-XOR and LPN)**.** *There are values of $\eta = \Theta(1)$, $k = O(\log r)$, $m = r^{o(1)}$ and $\ell = \mathrm{poly}(r)$ such that the cryptosystem $PKE_{\eta,k,\ell,m}$ is a (weak) public-key bit encryption scheme, with all operations running in time $r^{1+o(1)}$, if either of the following conditions are satisfied.*

(1) Constant-noise LPN is $2^{m^{0.5}}$-hard; or,

(2) Constant-noise LPN is 2^{m^c}-hard (for any constant $c > 0$) and k-XOR is hard at every density $\frac{1}{\mathrm{polylog}(r)}$;

(3) Constant-noise LPN is $m^{\omega(1)}$-hard and k-XOR is hard at every density $\frac{1}{r^{o(1)}}$.

Proof. Suppose we write the parameters to the cryptosystem as follows (where $\alpha, \beta, \epsilon > 0$ are parameters we will define later).

$$\eta = \frac{1}{3} \qquad k = \epsilon\,(\log r)^\alpha \qquad m = (\log r)^\beta \qquad \ell = 32 \cdot 1.5^{2k}\,(\log r)^2$$

Here, we need $\alpha > 0$ for k to be superconstant. The constraint $\ell = \mathsf{exp}(k)\,\omega(\log r)$ is needed by Lemma 8.2 to ensure a vanishing decryption error which, moving forward is guaranteed by our choice of parameters. Note that in order for the ciphertext to be superconstant and of polynomial size, we need to have $1/\log\log r < \alpha \le 1$. Also note that the density in our case is $\epsilon \frac{k \log r}{m} = \epsilon\,(\log r)^{\alpha+1-\beta}$.

Now suppose LPN is 2^{m^c}-hard for some constant $c > 0$. Suppose we wish to instantiate the cryptosystem at constant density > 1. In this case, since $k = \omega(1)$, it follows by Theorem 4.2 that $H_1^{(0)}$ (respectively, $H_1^{(\ell)}$) is, in fact, statistically indistinguishable from $H_0^{(0)}$ (respectively, $H_0^{(\ell)}$). Note that we have $T(m) = 2^{m^c} = 2^{(\log r)^{\beta c}}$ which must satisfy $\beta c > 1$ for $T(m)$ to be superpolynomial. Note that in order to have density ϵ, we

need that $\beta = \alpha + 1$ and thus the cryptosystem is only secure if $c \geq \frac{1}{2}$ (in which case we may let $\alpha = 1 - \delta$ and $\beta = 2 - \delta$ for a small constant $\delta > 0$). This shows condition (1) of the theorem.

Suppose instead we let the density be subconstant such that we may circumvent this lower bound. By Lemma 8.9, the cryptosystem satisfies indistinguishability if we assume k-XOR is hard at density $\Delta = 2\frac{\log T(m)}{m} = 2m^{c-1}$. Using the fact that $m = k\log(r)/\Delta$, this implies indistinguishability at density,

$$\Delta = \left(\frac{2}{(k\log(r))^{1-c}}\right)^{\frac{1}{c}} = \frac{1}{\text{polylog}(r)}.$$

We can now let $m = \text{polylog}(r)$ such that Δ matches the upper bound. By assumption, k-SUM is hard at this density and we know LPN takes time at least $r^{k/2}$ at this density. This shows condition (2).

Finally, suppose instead that $T(m) = m^c$ for some $c = \omega(1)$. In this case, we need $m^c \geq r^{k/2}$, and hence using $m = k\log(r)/\Delta$, we need $\Delta \leq \frac{k\lg r}{r^{\frac{k}{2c}}}$. Now let $k = \sqrt{c}$, then we can choose $m = r^{o(1)}$ such that $\Delta = \frac{k\lg r}{r^{o(1)}}$, for which again, we know k-SUM is hard by assumption.

For each construction, note that the runtime is limited by $O(\ell m r) = O(\exp(k)\text{polylog}(r)\, r^{o(1)}\, r) = r^{1+o(1)}$ by letting $\alpha < 1$. □

The first of these results was previously shown by Yu and Zhang [YZ16] who use a different construction to build public-key encryption from $2^{m^{0.5}}$-hardness of constant-noise LPN.

9 Algorithms for k-SUM at Very Low Densities

In this section, we reduce the planted k-SUM problem on integers to the subset sum problem. We will show both a worst-case as well as an average-case reduction. We then use existing algorithms for low-density subset sum to get non-trivial algorithms for planted k-SUM at low densities. Surprisingly, the two constructions are quite different and can not be combined.

Definition 9.1 (Worst Case Algorithm for Subset Sum)**.** *A subset sum problem instance comprises a vector* $\mathbf{A}$ *containing* r *integers, and a target value* t*. An worst-case algorithm* Alg *solves the problem in time* $T(r)$ *if and only if it can find a Boolean vector* $\mathbf{x}$ *of length* r *such that* $\mathbf{A}\cdot\mathbf{x} = t$ *whenever such a vector* $\mathbf{x}$ *exists.*

Note that the above problem is known to be NP-complete.

Definition 9.2 (Average-Case Algorithm for Subset Sum)**.** *The average case problem is parametrized by two integers* r *and* N*, where* N *must be a prime power. To sample an instance, we choose a vector* $\mathbf{A}$ *from* $\mathbb{Z}_N^r$ *and a vector* $\mathbf{x}$ *from* $\{0,1\}^r$ *uniformly at random. An average case algorithm returns a vector* $\mathbf{x}'$ *given* $\mathbf{A}$ *and* $\mathbf{A}\cdot\mathbf{x} \mod N$ *such that* $\mathbf{A}\cdot\mathbf{x} = \mathbf{A}\cdot\mathbf{x}' \mod N$ *with constant probability. Note that this probability is taken over the randomness of the input as well as the randomness used inside the algorithm.*

Remark 9.3. *The quantity* $\Delta = \frac{r}{\log N}$ *is called the density of the subset sum instance. The runtime of an average case algorithm is expressed as a function of* r *for some fixed* Δ*. It is known that the problem is hardest when* $\Delta = 1$*, and there exist polynomial time algorithms when* $\Delta \leq \frac{1}{r}$ *[LO85, Ben22].*

Remark 9.4. *In fact, the specific algorithms described in [LO85, Ben22] can be applied directly to solve k-SUM at the densities in Corollaries 9.7 and 9.8 and would give better success probability than that stated there. Nevertheless, we present our results as corollaries of our reduction to Subset Sum, as this reduction works for a wider range of densities, and would transfer improvements in algorithms for Subset Sum immediately to k-SUM.*

It is possible to define the average-case version of the problem without using modular arithmetic such that the two versions correspond to each other more obviously. The modular version of the problem can be solved by using an oracle for non-modular Subset Sum by calling it r times with multiples of N added

to the target. On the other hand, the non-modular version can be solved by simply calling the oracle for modular Subset Sum once; there exists an unique solution with high probability in low densities. We chose this definition because it is cleaner and still equivalent to the more intuitive translation of the subset sum problem to the average-case setting.

9.1 Worst-Case Reduction to Subset Sum

In this section, we show the following worst-case reduction from the k-SUM search problem to Subset Sum.

Theorem 9.5. *Suppose there exists a worst-case algorithm A with time complexity $T(r) = \Omega(r)$ that solves subset sum. Then, there is an algorithm B with the same time complexity $T(r)$ that solves the worst-case search k-SUM problem.*

Proof. We start by describing the new algorithm:

Algorithm $B(X)$

1. Construct a new array Y of r integers.
2. Let $M := \max(|X_i|) + 1$.
3. Set $Y[i] := (k+1)M + X[i]$ for all $1 \leq i \leq r$.
4. Run A on the set Y with target $t = k(k+1)M$.
5. Return the set of indices returned by the previous call.

The above algorithm clearly has the same runtime complexity as A does. To show correctness, recall that the input array X must have a k-SUM solution. So there is a set κ of size k such that $\sum_{i\in\kappa} X_i = 0$. By our construction, $\sum_{i\in\kappa} Y_i = k(k+1)M + \sum_{i\in\kappa} X_i = t$. Therefore, we ensure that there is at least one solution to the subset sum problem instance we create. Hence, A must return a valid subset sum solution.

Suppose that A returns a set S. By construction,

$$|X_i| < M \Rightarrow M + X_i > 0 \Rightarrow Y_i = (k+1)M + X_i = kM + (M + X_i) > kM$$

Therefore, if $|S| \geq k+1$, the value of $\sum_{i\in S} Y_i > (k+1) \cdot kM = t$. This is a contradiction since the aforementioned sum is supposed to equal t, and we can thus conclude that $|S| \leq k$. Similarly, observe that

$$|X_i| < M \Rightarrow X_i < M \Rightarrow Y_i = (k+1)M + X_i < (k+2)M$$

Therefore, if $|S| \leq k-1$, the value of $\sum_{i\in S} Y_i < (k-1) \cdot (k+2)M < k(k+1)M = t$. This is also a contradiction since the aforementioned sum is supposed to equal t, and we can thus conclude that $|S| \geq k$. Together, these two constraints imply that $|S| = k$. Now we can easily calculate

$$\sum_{i\in S} X_i = \sum_{i\in S} (Y_i - (k+1)M) = \sum_{i\in S} Y_i - k(k+1)M = 0$$

This concludes our proof that B returns a set of exactly k indices which correspond to a k-SUM solution.

Note that the above reduction works unchanged in the case where A is a randomized algorithm with some constant success probability; B will also then have the same success probability in that case. □

9.2 Average-Case Reduction to Subset Sum

The above reduction unfortunately does not generalize to the average case setting. An average-case oracle could potentially be biased against any Y of the above form (which can be constructed from an array X with a k-SUM solution). To get around this, we reduce from the modular k-SUM problem with prime moduli. In the average case, this is known to be equivalent to the integer k-SUM problem.

Theorem 9.6. *Suppose there exists an algorithm A of time complexity $T(r) = \Omega(r)$ that solves the average-case subset sum problem with constant success probability at some density $\Delta < \frac{1}{2}$. Then there is an algorithm B with the same time complexity that can solve the planted search k-SUM problem on groups of the form $\mathbb{Z}_p$ (where p is a prime larger than k) at density $\frac{k\Delta \log r}{r}$ with constant success probability.*

Proof. We start by describing the new algorithm:

Algorithm $B(X)$

1. Construct a new array Y of r integers.
2. Choose α uniformly at random from $\mathbb{Z}_p$.
3. Set $Y[i] := \alpha + X[i] \pmod{p}$ for all $1 \leq i \leq r$.
4. Choose a random $S \subseteq [r]$.
5. Run A on the set Y with target $t = k\alpha + \sum_{i \in S} Y_i \pmod{p}$.
6. If A succeeds and returns S' such that $S \subset S'$ and $|S' \setminus S| = k$, return $S' - S$.

Since both the problems here involve members of $\mathbb{Z}_p$, we implicitly treat all numbers in the following analysis modulo p.

We will first prove that Y *looks random.* Specifically, we'll demonstrate that

$$\Pr\left[Y[i] = x \;\middle|\; \bigcup_{j \neq i} Y_j\right] = \frac{1}{p} \qquad \forall i \in [r] \forall x \in \mathbb{Z}_p$$

Note that X comes from a planted k-SUM instance, and therefore, was sampled from the planted distribution. Let us consider the effect of combining that sampling process with the first 3 steps of our algorithm B, which is how we obtain the array Y.

Sampling Algorithm for Y

1. Sample r elements $Y_1, Y_2, \ldots, Y_r$ i.i.d. uniformly at random from $\mathbb{Z}_p$.
2. Choose a random set $T \subseteq [r]$ with $|T| = k$
3. Let $t \in T$ be the smallest index and let $Y_t \leftarrow -\sum_{\substack{j \in S \\ j \neq t}} Y_j$
4. Choose α uniformly at random from $\mathbb{Z}_p$.
5. Increment Y_i by α for all $1 \leq i \leq r$.

Steps 2 and 4 are just independent uniformly random choices; we can obviously move both steps to the very beginning. Observe that we can move step 3 to the end if we simply modify the assignment to $Y_t \leftarrow k\alpha - \sum_{\substack{j \in S \\ j \neq t}} Y_j$. This is because step 3 does not affect the other $r - 1$ indices of Y and the modified assignment represents the net effect of steps 3 and 5 on Y_i. Therefore the sampling algorithm is equivalent to the following:

Equivalent Sampling Algorithm for Y

1. Sample r elements $Y_1, Y_2, \ldots, Y_r$ i.i.d. uniformly at random from $\mathbb{Z}_p$.
2. Choose a random set $T \subseteq [r]$ with $|T| = k$. Let $t \in T$ be its smallest index.
3. Choose α uniformly at random from $\mathbb{Z}_p$.
4. Increment Y_i by α for all $1 \leq i \leq r$.
5. Let $Y_t \leftarrow k\alpha - \sum_{\substack{j \in S \\ j \neq t}} Y_j$

In the new algorithm, it is easy to see that Y is uniformly distributed in $\mathbb{Z}_p^r$ at the end of step 3. That remains true after step 4, since $\mathbb{Z}_p$ is a cyclic group and adding α is merely applying an invertible translation. Finally, note that $k\alpha - \sum_{\substack{j \in S \\ j \neq t}} Y_j$ is a uniformly random member of $\mathbb{Z}_p$ since so is α. Thus, the last step simply replaces Y_t with a freshly chosen uniformly random element of $\mathbb{Z}_p$. Therefore, the result of this entire sampling procedure Y is distributed uniformly in $\mathbb{Z}_p^r$.

Recall that in the planted k-SUM problem, the existence of a k-tuple $T \subset [r]$ such that $\sum_{i \in T} X_i = 0$ is guaranteed. Therefore, $\sum_{i \in T} Y_i = k\alpha$. With probability 2^{-k}, the set S is disjoint from T. In that event, $S' = S \bigcup T$ is a valid solution to the subset sum instance in line 5 (of algorithm B), and it also satisfies the conditions in line 6. Since Y is uniformly distributed in $\mathbb{Z}_p^r$ and our subset sum instance has density $\Delta < \frac{1}{2}$, the probability that there exists a second subset whose sum also equals t is negligible. So if A succeeds on the problem instance constructed in line 5 with probability p, we can conclude that B successfully solves the k-SUM instance with probability at least $\frac{p}{2^k}$.

A subset sum problem instance is characterized by a vector and a subset. We have already concluded that the vector input in our constructed problem instance is distributed uniformly in $\mathbb{Z}_p^r$. Since the density of the provided k-SUM is $\frac{k\Delta \log r}{r}$, we have $\frac{k \log r}{\log p} = \frac{k\Delta \log r}{r} \Rightarrow \frac{r}{\log p} = \Delta$. Therefore the density of our constructed subset sum problem is exactly Δ, and so A would succeed with constant probability if the subset was chosen uniformly. Note that the subset (S') is the union of S and T where S is chosen uniformly at random in line 4, T is a random subset of size k, and we only condition on $S \bigcap T = \emptyset$. We can therefore use Lemma 9.9 to conclude that A solves the subset sum problem we construct with constant probability. Since k is a constant, this coupled with the observation from the previous paragraph implies that B has constant success probability. □

Corollary 9.7. *There exists an average-case algorithm for k-SUM over $\mathbb{Z}_p$ groups at density at most $\frac{k \log r}{r^2}$ whose runtime complexity is polynomial in r and does not depend on k.*

Proof. The subset sum problem can be solved in the average-case in polynomial time for density at most $(\frac{1}{r})$ with constant success probability (see Remark 9.3). The statement then follows directly from Theorem 9.6. □

Corollary 9.8. *There exists an average-case algorithm for k-SUM over integers (where the integers are chosen at random from $[-N, N]$ such that $\Delta = \frac{k \log r}{\log N}$) at density at most $(\frac{k \log r}{r^2})$ whose runtime complexity is polynomial in r and doesn't depend on k.*

Proof. Let p be a prime between N and $2N$. Corollary 9.7 gives us a polynomial algorithm A that solves k-SUM in $\mathbb{Z}_p$ for density approximately Δ. We can now use Theorem 4.5 in [DKK21] to obtain the desired algorithm. □

Lemma 9.9. *Assume that k is some fixed constant. Let D_0 be the uniform distribution on subsets of $[r]$. Let D_1 be the distribution induced by the following sampling procedure:*

1. *Choose k distinct integers uniformly at random from $[r]$; call this set T.*
2. *Choose a random subset S of $[r]$*

3. If $S \bigcap T = \emptyset$, return $S \bigcup T$. Otherwise, repeat the process.

If an average-case algorithm A has success probabilities $p_0 = \Omega_k(1)$ on inputs from D_0 and p_1 on inputs from D_1, then $p_1 \geq 2^{-(2k+1)} p_0$.

Proof. Let D_2 be the uniform distribution on subsets of $[r]$ with size at least $\frac{r}{4}$, and let p_2 be the success probability of A on inputs from D_2. Our proof will bound the total variation distance between D_0 and D_2, and the Rényi divergence of order ∞ between D_1 and D_2. We will thus derive bounds on $|p_0 - p_2|$ as well as $\frac{p_1}{p_2}$, which will together imply the given statement.

Note that $\mathrm{supp}(D_2) \subset \mathrm{supp}(D_0)$. Since both of these distributions are uniform, we have,

$$\begin{aligned}
&\Delta(D_0, D_2) \\
&\quad = \sum_{S \in \mathrm{supp}(D_2)} (D_2(S) - D_0(S)) \\
&\quad = \sum_{S \in \mathrm{supp}(D_0) - \mathrm{supp}(D_2)} D_0(S) \\
&\quad = \sum_{S \subset [r], |S| < \frac{r}{4}} D_0(S) \\
&\quad = \Pr_{S \sim D_0} \left[|S| < \frac{r}{4} \right] \\
&\quad \leq \exp\left(-2r \left(\frac{1}{2} - \frac{1}{4} \right)^2 \right) && \text{by Hoeffding's inequality} \\
&\quad = \exp\left(-\frac{r}{8} \right) \\
&\quad = o(1).
\end{aligned}$$

This implies that $p_2 \geq p_0 - o(1) > \frac{p_0}{2}$. Let us now calculate an explicit expression for $D_1(Q)$ for a subset Q of size at least k. To get an output of Q, we need two events to happen simultaneously. T must be a subset of Q, which happens with probability $\binom{|Q|}{k} / \binom{r}{k}$. Given any such choice of T, S must be exactly $Q - T$. There are 2^{r-k} choices of S which do not intersect with T, so the probability of this particular set being chosen is 2^{k-r}. Since the above two events are independent, we have

$$D_1(Q) = \frac{\binom{|Q|}{k}}{\binom{r}{k}} \cdot \frac{2^k}{2^r}$$

Note that all sets of size at least k are in the support of D_1. Therefore, as long as $r > 4k$, we have

$\mathrm{supp}(D_2) \subseteq \mathrm{supp}(D_1)$. We can now calculate the following Rényi divergence using Lemma 2.1 as follows.

$$
\begin{aligned}
&R(D_2 \| D_1) \\
&= \max_{Q \in \mathrm{supp}(D_2)} \frac{D_2(Q)}{D_1(Q)} \\
&= \max_{Q \subseteq [r], |Q| \geq \frac{r}{4}} \left(D_2(Q) \cdot \frac{\binom{r}{k}}{\binom{|Q|}{k}} \cdot 2^{r-k} \right) \\
&= \max_{Q \subseteq [r], |Q| \geq \frac{r}{4}} \left(\frac{1}{\texttt{\# subsets of } [r] \texttt{ with size at least } r/4} \cdot \frac{\binom{r}{k}}{\binom{|Q|}{k}} \cdot 2^{r-k} \right) \\
&\leq \left(\frac{1}{2^{r-1}} \cdot \frac{\binom{r}{k}}{\binom{r/4}{k}} \cdot 2^{r-k} \right) \\
&= \frac{1}{2^{k-1}} \cdot \frac{r(r-1)\cdots(r-k+1)}{(r/4)(r/4-1)\cdots(r/4-k+1)} \\
&< \frac{2}{2^k} \cdot \left(\frac{r-k+1}{r/4-k+1} \right)^k \\
&< \frac{1}{2^k} \cdot \left(\frac{2r-8k+8}{r/4-k+1} \right)^k && \text{for large enough } r \\
&= 4^k
\end{aligned}
$$

Applying the Rényi divergence lemma on the event that algorithm A succeeds, we get that,

$$p_1 \geq p_2 / R(D_2 \| D_1) > p_2 / 4^k.$$

Combining the two above relations, we get $p_1 \geq 2^{-(2k+1)} p_0$. □

Acknowledgments

Nikolaj thanks his advisor Ivan Damgård for discussions related to the public-key encryption scheme. We also thank Rachel Lin for encouraging us to think about the case of k-SUM with super-constant k, and Eldon Chung for helpful discussions.

References

[ABHS19] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 41–57. SIAM, 2019.

[ABRV23] Gal Arnon, Marshall Ball, Alon Rosen, and Prashant Nalini Vasudevan. Unpublished Manuscript. 2023.

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 171–180. ACM, 2010.

[AC88] N. Alon and F.R.K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72(1):15–19, 1988.

[AC05] Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

[AL13] Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307. IEEE Computer Society, 2003.

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443, 2014.

[BBB19] Enric Boix-Adserà, Matthew S. Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1256–1280. IEEE Computer Society, 2019.

[BC22] Chris Brzuska and Geoffroy Couteau. On building fine-grained one-way functions from strong average-case hardness. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 584–613. Springer, 2022.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.

[BDJ21] Charles Bouillaguet, Claire Delaplace, and Antoine Joux. Algorithms for Sparse Random 3XOR: The Low-Density Case. working paper or preprint, October 2021.

[BDP08] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, Apr 2008.

[Ben22] Huck Bennett. Solving Random Low-Density Subset Sum Using Babai's Algorithm. (https://web.engr.oregonstate.edu/ bennethu/low-density-subset-sum-via-babai.pdf), 2022.

[BHP01] Gill Barequet and Sariel Har-Peled. Polygon containment and translational min-hausdorff-distance between segment sets are 3sum-hard. *Int. J. Comput. Geometry Appl.*, 11:465–474, 08 2001.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, jul 2003.

[BLRL+18] Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. *Journal of Cryptology*, 31(2):610–640, Apr 2018.

[BR13a] Quentin Berthet and Philippe Rigollet. Complexity theoretic lower bounds for sparse principal component detection. In *Annual Conference Computational Learning Theory*, 2013.

[BR13b] Quentin Berthet and Philippe Rigollet. Optimal detection of sparse principal components in high dimension. *The Annals of Statistics*, 41(4):1780 – 1815, 2013.

[Bri17] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1073–1084. SIAM, 2017.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 483–496, New York, NY, USA, 2017. Association for Computing Machinery.

[BRSV18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 789–819. Springer, 2018.

[BSV21] Zvika Brakerski, Noah Stephens-Davidowitz, and Vinod Vaikuntanathan. On the hardness of average-case k-sum. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 29:1–29:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[CGI+16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, page 261–270, New York, NY, USA, 2016. Association for Computing Machinery.

[Cha20] Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020.

[Che52] Herman Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *The Annals of Mathematical Statistics*, 23(4):493 – 507, 1952.

[CL23] Eldon Chung and Kasper Green Larsen. Stronger 3sum-indexing lower bounds. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 444–455. SIAM, 2023.

[Din19] Itai Dinur. An algorithmic framework for the generalized birthday problem. *Des. Codes Cryptogr.*, 87(8):1897–1926, 2019.

[DKK21] Itai Dinur, Nathan Keller, and Ohad Klein. Fine-grained cryptanalysis: Tight conditional bounds for dense k-sum and k-xor. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 80–91. IEEE, 2021.

[DKT16] Edwin R. Van Dam, Jack H. Koolen, and Hajime Tanaka. Distance-regular graphs. *The Electronic Journal of Combinatorics*, 1000, apr 2016.

[DLW20] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 774–785. IEEE, 2020.

[DSW18] Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3xor. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 59:1–59:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. Lpn decoded. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 486–514, Cham, 2017. Springer International Publishing.

[Eri95] Jeff Erickson. Lower bounds for linear satisfiability problems. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 388–395. ACM/SIAM, 1995.

[GGH+20] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. Data structures meet cryptography: 3sum with preprocessing. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 294–307. ACM, 2020.

[GO95] Anka Gajentaan and Mark H Overmars. On a class of o(n2) problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

[GP18] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4), apr 2018.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 197–206, New York, NY, USA, 2008. Association for Computing Machinery.

[GR18] Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 77–88. IEEE Computer Society, 2018.

[GS17] Omer Gold and Micha Sharir. Improved Bounds for 3SUM, k-SUM, and Linear Degeneracy. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[GV21] Aparna Gupte and Vinod Vaikuntanathan. The fine-grained hardness of sparse linear regression. *CoRR*, abs/2106.03131, 2021.

[GZ19] David Gamarnik and Ilias Zadik. The landscape of the planted clique problem: Dense subgraphs and the overlap gap property. *CoRR*, abs/1904.07174, 2019.

[Hol05] Thomas Holenstein. Key agreement from weak bit agreement. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 664–673, New York, NY, USA, 2005. Association for Computing Machinery.

[HR05] Thomas Holenstein and Renato Renner. One-way secret-key agreement and applications to circuit polarization and immunization of public-key encryption. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 478–493, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.

[HS23] Shuichi Hirahara and Nobutaka Shimizu. Hardness self-amplification: Simplified, optimized, and unified. *Electron. Colloquium Comput. Complex.*, TR23-026, 2023.

[IJKW10] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM J. Comput.*, 39(4):1637–1665, 2010.

[IN89] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 236–241. IEEE Computer Society, 1989.

[Jer92] Mark Jerrum. Large cliques elude the metropolis process. *Random Struct. Algorithms*, 3(4):347–360, 1992.

[JP00] Ari Juels and Marcus Peinado. Hiding cliques for cryptographic security. *Designs, Codes and Cryptography*, 20(3):269–280, 2000.

[JW19] Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 17:1–17:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[KP19] Tsvi Kopelowitz and Ely Porat. The strong 3sum-indexing conjecture is false. *CoRR*, abs/1907.11206, 2019.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 1272–1287, USA, 2016. Society for Industrial and Applied Mathematics.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and concurrent security of the hb and hb+ protocols. *Journal of Cryptology*, 23(3):402–421, 2010.

[LLW19] Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-key cryptography in the fine-grained setting. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 605–635. Springer, 2019.

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 382–400. Springer, 2010.

[LS19] Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-mansour using the 3-xor problem. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 210–235. Springer, 2019.

[MS12] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. *Journal of Cryptology*, 25(2):349–382, Apr 2012.

[Nan15] Mridul Nandi. Revisiting security claims of XLS and COPA. *IACR Cryptol. ePrint Arch.*, page 444, 2015.

[NS15] Ivica Nikolic and Yu Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 683–703. Springer, 2015.

[Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 603–610, New York, NY, USA, 2010. Association for Computing Machinery.

[Pet15] Seth Pettie. Higher Lower Bounds from the 3SUM Conjecture, talk at the Computational Complexity of Low-Polynomial Time Problems workshop at the Simons Institute. (https://simons.berkeley.edu/talks/higher-lower-bounds-3sum-conjecture), 2015.

[Pie12] Krzysztof Pietrzak. Cryptography from learning parity with noise. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012.

[PW10] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010.

[PZ32] R. E. A. C. Paley and A. Zygmund. On some series of functions, (3). *Mathematical Proceedings of the Cambridge Philosophical Society*, 28(2):190–205, 1932.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[SEO03] Michael Soss, Jeff Erickson, and Mark Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry*, 26(3):235–246, 2003.

[SS05] Elias M Stein and Rami Shakarchi. *Real analysis: measure theory, integration, and Hilbert spaces*. Princeton lectures in analysis. Princeton Univ. Press, Princeton, NJ, 2005.

[Tch67] P. Tchébychef. Des valeurs moyennes (traduction du russe, n. de khanikof. *Journal de Mathématiques Pures et Appliquées*, pages 177–184, 1867.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 288–304, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Wil18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

[YZ16] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise lpn. In *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9814*, page 214–243, Berlin, Heidelberg, 2016. Springer-Verlag.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il