Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 60 (2023)

k-SUM in the Sparse Regime

Shweta Agrawal* Sagnik Saha† Nikolaj I. Schwartzbach‡ Akhil Vanukuri*
Prashant Nalini Vasudevan§

November 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 "solution" set of k numbers that sum to 0 modulo M. In the *dense* regime of $M \leq r^k$, where solutions exist with high probability, the complexity of these problems is well understood. Much less is known in the *sparse* regime of $M \gg r^k$, where solutions are unlikely to exist.

In this work, we initiate the study of the *sparse* regime for k-SUM and its variant k-XOR, especially their *planted* versions, where a random solution is planted in a randomly generated instance and has to be recovered. We provide evidence for the hardness of these problems and suggest new applications to cryptography. Our contributions are summarized below.

Complexity. First we study the complexity of these problems in the sparse regime and show:

- *Conditional Lower Bounds.* Assuming established conjectures about the hardness of average-case (non-planted) k-SUM/k-XOR when $M = r^k$, we provide non-trivial lower bounds on the running time of algorithms for planted k-SUM when $r^k \leq M \leq r^{2k}$.
- *Hardness Amplification.* We show that for any $M \geq r^k$, if an algorithm running in time T solves planted k-SUM/k-XOR with success probability $\Omega(1/\text{polylog}(r))$, then there is an algorithm running in time $\widetilde{\mathcal{O}}(T)$ that solves it with probability $(1-o(1))$. This in particular implies hardness amplification for 3-SUM over the integers, which was not previously known.

  Technically, our approach departs significantly from existing approaches to hardness amplification, and relies on the locality of the solution together with the group structure inherent in the problem.
- *New Reductions and Algorithms.* We provide reductions for k-SUM/k-XOR from search to decision, as well as worst-case and average-case reductions to the Subset Sum problem from k-SUM. Additionally, we present a new algorithm for average-case k-XOR that is faster than known worst-case algorithms at low densities.

Cryptography. We show that by additionally assuming mild hardness of k-XOR, we can construct Public Key Encryption (PKE) from a weaker variant of the Learning Parity with Noise (LPN) problem than was known before. In particular, such LPN hardness does not appear to imply PKE on its own – this suggests that k-XOR/k-SUM can be used to bridge "minicrypt" and "cryptomania" in some cases, and may be applicable in other settings in cryptography.

*IIT Madras. This work was supported in part by the DST "Swarnajayanti" fellowship, Cybersecurity Center of Excellence, IIT Madras, National Blockchain Project and the Algorand Centres of Excellence programme managed by Algorand Foundation.

†Computer Science Department, Carnegie Mellon University.

‡CIFRA Institute, Bocconi University. Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019547). Part of this work was done while Nikolaj was a Ph.D. student at Aarhus University and was 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. 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 $\{0, \ldots, M-1\}$, 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 and graph theory [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 $\mathcal{O}(r^k)$. The "meet-in-the-middle" algorithm that computes the sums of all $\lceil k/2 \rceil$-sets and looks for collisions runs in time $\widetilde{\mathcal{O}}(r^{\lceil k/2 \rceil})$ [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 $\widetilde{\mathcal{O}}(M+r)$ [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 $\{0, \ldots, M-1\}$. 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 Δ smaller than 1 (the *sparse* regime), the number of solutions is vanishing with r. When k is small, the density can be approximated by the ratio $(k \log r / \log M)$ and we will use this simplification in the remainder of this paper[2].

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 $\mathcal{O}(\sqrt{M}) = \mathcal{O}(r^{k/(2\Delta)})$ which is $r^{O(1)}$ for densities larger than k. For densities larger than $\approx k/(1+\log_2 k)$, Wagner's k-tree algorithm [Wag02] solves the problem in time $\widetilde{\mathcal{O}}(r)$.

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, Keller and Klein [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, Lincoln and Williams [LLW19] used a decision version of a weaker form of this conjecture to construct fine-grained One-Way Functions.

[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]).

[2] Please see Remark 3.2 for a discussion of the accuracy of this approximation.

The Sparse Regime. In this work, we investigate the average-case complexity of k-SUM at densities at most 1, where random instances are unlikely to have solutions. Unlike the dense regime, i.e. with densities 1 and higher [Wag02, LLW19, BSV21, DKK21], very little is known about the complexity of average-case k-SUM in the sparse regime. Addressing this basic gap in our understanding of k-SUM is the goal of the present work.

In the sparse regime where solutions are unlikely, a meaningful variant of the k-SUM problem which we introduce is the *planted* k-SUM problem – here 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 that arise naturally in this setting:

- 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.

Aside from being interesting problems that warrant study in their own right, we are also interested in these problems from the standpoint of applications to cryptography.

1.1 Our Results

In this work, we initiate the study of k-SUM and its variants, namely the k-XOR and vector k-SUM problems, in the sparse regime. In vector k-SUM, the elements in the input are vectors from $\mathbb{Z}_q^m$ for some m, and addition is done over this vector space; k-XOR is the special case of $q = 2$. Our results are described below. Please also see Fig. 2 for a summary.

Complexity. To begin with, we provide conditional lower bounds on the complexity of the planted k-SUM problem in the sparse regime via the following theorem, assuming the hardness of the regular (non-planted) k-SUM problem at density 1.

Theorem 1.2 (Corollary 4.9, Section 4.2)**.** *Assuming the average-case k-SUM conjecture (Conjecture 1.1), 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 $\Omega\left(r^{k\left(1-\frac{1}{2\Delta}\right)-o(1)}\right)$. This generalizes to the k-SUM problem defined over any abelian group.*

To establish the above, we first show a reduction from non-planted search k-SUM to planted search k-SUM at density 1. Then, for any $\Delta \in (1/2, 1)$, we reduce planted k-SUM at density 1 to planted k-SUM at density Δ. We demonstrate two such reductions, one of which additionally lets us show lower bounds for k-SUM assuming the hardness of k'-SUM for a different k'. Please see Sections 4.1 and 4.2 for details. Following these lower bounds, our current understanding of the complexity of average-case k-SUM at various densities is depicted in Fig. 1.

We then connect the complexity of the search k-SUM problem to related problems such as the subset sum problem and the decision variant of the k-SUM problem. We show an average-case reduction from very sparse planted k-SUM over integers to the average-case subset sum problem at low densities, as well as a search-to-decision reduction for planted k-SUM that, in particular, carries over the above conditional lower bounds to the decision k-SUM problem. Please see Sections 4.3 and 4.4 for details.

Finally, we show an algorithm for the k-XOR problem that, at densities less than $O(1/\sqrt{r})$, is faster than the best known worst-case algorithms. More precisely, we show that the planted search k-XOR problem can be solved in time $\widetilde{\mathcal{O}}\left(r^k \left(\frac{1}{\Delta}\right)^{3-k}\right)$ for any $\Delta \geq 1/r$. Please see Section 4.5 for details.

Hardness Amplification. For the k-SUM and vector k-SUM problems (including k-XOR) at density 1 or less, we show that the success probability of an algorithm for the planted search problem can be amplified. We do this using a random walk over instances that preserves the planted solution (with non-trivial probability)

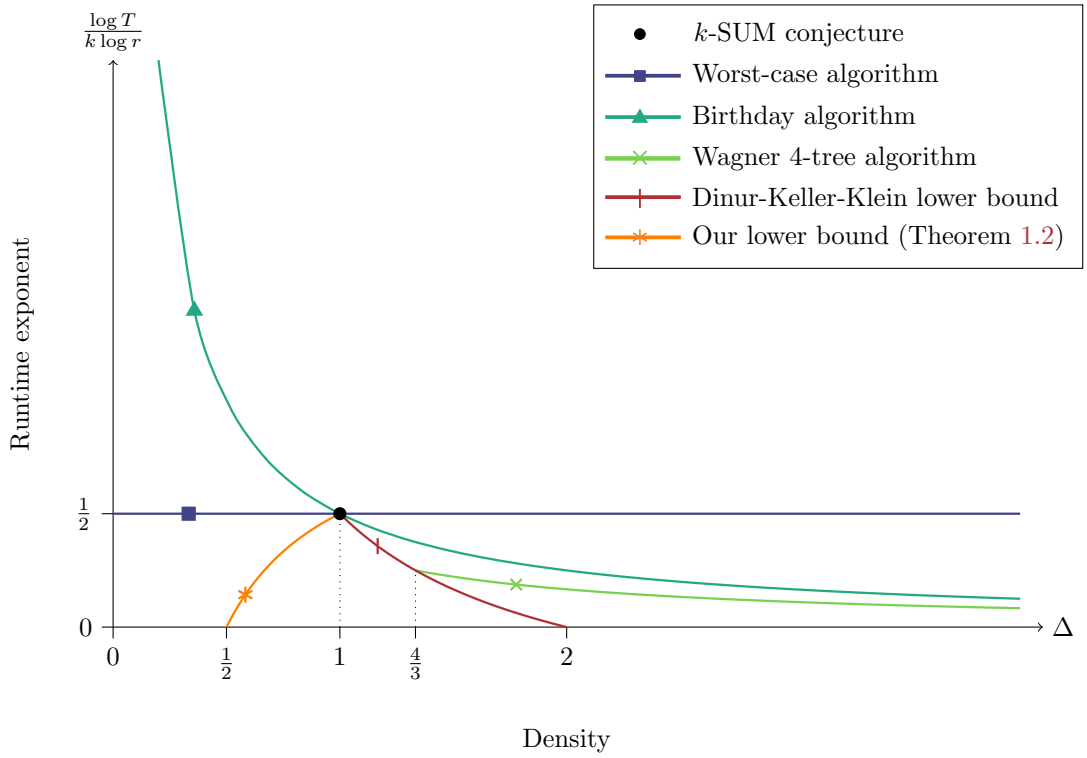

Figure 1: Landscape of the known bounds on the complexity of average-case (planted) k-SUM problems as $k \to \infty$ (except for the Wagner algorithm, which is depicted for $k = 4$). The x-axis represents the density $\Delta = \frac{k \log r}{m}$ of the instances, and the y-axis represents the runtime 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 $\mathcal{O}(\log(r)/r^2)$ presented in Section 4.4 since its runtime is independent of k. Similarly, we have omitted the algorithm for k-XOR that works at density $\mathcal{O}(1/r^{0.5+\epsilon})$ presented in Section 4.5.

and is also rapidly mixing. This amplification also extends to k-SUM over general groups, albeit for density slightly smaller than 1.

Theorem 1.3 (Theorem 5.20, Corollary 5.18)**.** *At any density* $\Omega\left(\frac{1}{\text{polylog}(r)}\right) \le \Delta \le 1$*, for any constant* $k \ge 3$*, suppose there is an algorithm that runs in time* T *and solves planted search* k*-SUM (resp. vector* k*-SUM) with success probability* $\Omega(1/\text{polylog}(r))$*. Then, there is an algorithm that runs in time* $\widetilde{\mathcal{O}}(T)$ *and solves planted search* k*-SUM (resp. vector* k*-SUM) at the same density with success probability* $\left(1 - o\left(\frac{1}{\log r}\right)\right)$*.*

The above also extends to super-constant values of k, with some additional loss in the running time. This hardness amplification, together with a search-to-decision reduction, enables us to use relatively mild hardness of k-SUM (or its variants) in applications – for instance in the public key encryption scheme we construct, it is sufficient for us to assume hardness of solving search k-XOR with success probability $(1 - o(1/\log r))$. Without the hardness amplification, we would have had to assume the hardness of solving it with some $\Omega(1)$ success probability.

Cryptography. Next, we show that somewhat mild hardness of planted search k-XOR at sufficiently low densities can be used to construct Public-Key Encryption (PKE) assuming weaker hardness of the Learning Parity with Noise (LPN) problem than was known before. Previously, it has been shown how to construct PKE assuming either that LPN with m-bit secrets at noise rate $\mathcal{O}(1/\sqrt{m})$ is hard for poly(m)-time algorithms [Ale03], or that LPN with constant noise rate is hard for $2^{m^{0.5}}$-time algorithms [YZ16]. In contrast, adding k-XOR enables us to use just 2^{m^c} hardness (for any constant $c > 0$) of LPN with constant noise rate.

Intriguingly, the level of hardness needed from LPN in our construction does not appear to imply public-key encryption by itself. This suggests the possibility of the k-SUM family of problems serving as a bridge for problems from the world of "Minicrypt" (where one way functions exist) to the world of "Cryptomania" (where public-key encryption exists) — see also [Imp95]. Qualitatively, our technique allows to interpret the k-SUM family of problems as a computational variant of the famous Leftover Hash Lemma [HILL99], which provides statistical guarantees and is used ubiquitously in cryptography [BDK+11][3]. Looking ahead, this can help to not only weaken the required hardness from the "core" problem being used in the cryptographic construction, but may also improve overall efficiency of the construction. We demonstrate this phenomenon in two PKE schemes, one based on LPN and another (with lesser improvement) based on its large-field analog Learning With Errors (LWE) [Reg09]. Please see Section 6 and Appendix A for details. We are optimistic that this technique will find other applications in cryptography.

Theorem 1.4 (Theorem 6.12, Section 6)**.** *Suppose that constant-noise LPN with an m-bit secret is 2^{m^c}-hard for some constant c, and that any algorithm for planted-search k-XOR at densities $\Delta = 1/\text{polylog}(r)$ with success probability $(1 - o(1/\log r))$ has running time at least $r^{\lceil k/2 \rceil - o(1)}$. Then, there is a PKE scheme which is secure against adversaries running in time $r^{\lceil k/2 \rceil - \Omega(1)}$.*

1.2 Technical Overview

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

Relating 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 close. At densities somewhat larger than 1, it is not hard to show that these are, in fact, very close in total variation distance. At density 1, however, their total variation distance is some constant. Nevertheless, we still show that any algorithm that has any constant success probability over planted instances also has some constant success probability over non-planted instances. We sketch our approach below.

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 $\ell \in [4, r^k]$, the distribution $D^{(\ell)}$ has the following properties:

- $D^{(\ell)}$ has total variation distance at most $2/(\ell - 3)^2$ from D_1
- $D^{(\ell)}$ has Rényi divergence[4] (of order ∞) at most $c_k\,(\ell + 1)$ relative to D_0, where c_k is a constant that depends only on k.

So if an algorithm succeeds with probability ϵ on D_1, then it succeeds with probability at least $\epsilon' = (\epsilon - 2/(\ell-3)^2)$ on $D^{(\ell)}$, and thus with probability at least $\frac{\epsilon'}{c_k(\ell+1)}$ on D_0. Picking an appropriate $\ell = \Theta(1/\sqrt{\epsilon})$ then gives us what we want.

[3]Technically, we are using k-XOR as a substitute for a specific strong extractor – the family of all linear functions $\mathbf{Ax} + \mathbf{b}$. Indeed, LHL is more general – it says any pairwise independent hash family is a strong extractor, but we only replace this specific family with k-SUM. However, this family suffices for most applications in cryptography.

[4]To be precise, this bound is on the maximum ratio of probability values between the distributions. Technically, the Rényi divergence of order ∞ is actually the log of this quantity, but for simplicity, throughout the paper we use the term to refer to this maximum ratio itself instead.

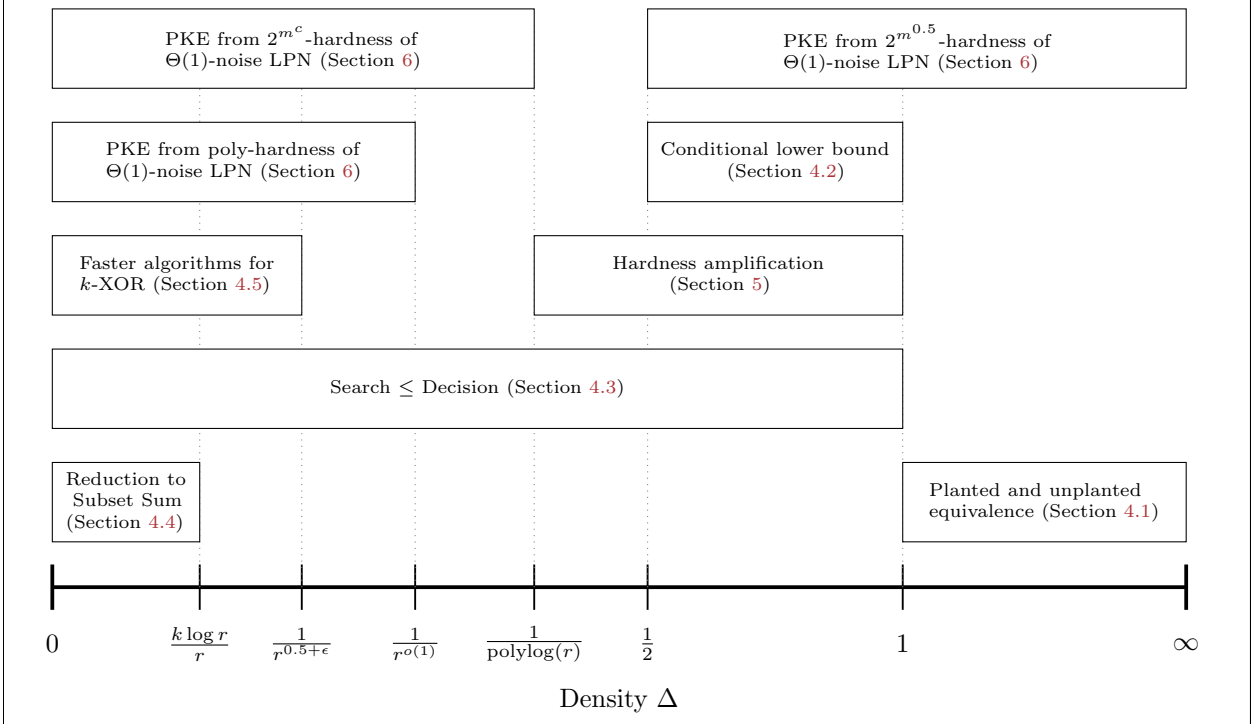

Figure 2: Overview of our results in terms of the density of the generated instances (excluding $\Delta \in \{0, \infty\}$), where r is size (number of elements) of the instance. The x-axis is not to scale. The PKE schemes for $\Delta \leq 1$ can be constructed from the specified hardness of LPN under 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.

It remains to construct the distribution $D^{(\ell)}$, which 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.

We briefly mention the relevance of this reduction to cryptography. Previously, it was shown by LaVigne, Lincoln and Williams [LLW19] that the hardness of (a variant of) the planted k-SUM problem yields a fine-grained one way function. Our aforementioned reduction (Theorem 4.1) shows that this can rely on the hardness of the standard *non-planted* k-SUM problem instead. For more details, please refer to Section 4.1.

Conditional Lower Bounds. Following the above reduction, the average-case k-SUM conjecture implies that planted search k-SUM at density 1 needs at least $r^{\lceil k/2 \rceil - o(1)}$ time. Assuming this, we show lower bounds for lower densities. In more detail, 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 $\mathcal{O}(r^{k(1-\Delta)})$ 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 $T(r) \geq r^{k\left(1-\frac{1}{2\Delta}\right)}$, 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 $r^{k\left(1-\frac{1}{2\Delta}\right)}$ 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 $k' \neq k$. This allows us to e.g. establish that solving 4-SUM at density $\Delta = \frac{4}{5}$ requires $r^{2-o(1)}$ time, assuming that 5-SUM is hard to solve at density $\Delta = 1$ (see Corollary 4.11). Second, the lower bound 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. Please see Section 4.2 for details.

Hardness Amplification. We show that an algorithm that solves planted search k-SUM (resp. k-XOR) 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 $\widetilde{\mathcal{O}}(T)$. Our procedure also works for k-SUM over general abelian groups, though in this case it only works for densities slightly less than 1. We briefly describe our approach here, using the specific case of k-XOR for illustration; all the steps described below except the final reduction from density 1 can be applied with any abelian group.

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. Our approach, given an instance $X \in (\mathbb{F}_2^m)^r$ with a planted solution, 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 $\Omega\left((1 - k/r)^t\right)$. 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 planted instances X by obfuscating the planted solution. Our obfuscation works by sampling a random set of k vectors E that sum to 0, and then adding a random element from this set to each column of the given instance X. With probability at least $1/k^k$, a distinct element from E is added to each element of the solution planted in X, and thus the existence and location of the solution are preserved, while the set of vectors that form the solution is fully randomized. The columns of the modified X are then randomly permuted. This ensures that the location of the solution is also randomized.

By repeating the above obfuscation process (and the entire reduction) $\mathcal{O}(k^k)$ times, we can ensure that a solution in X is preserved in at least one of the iterations with high probability. This process 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 entire argument works at every density $\leq 1 - \frac{\log\log r}{\log r}$, and in fact works for k-SUM over any abelian group. For the cases of k-SUM over integers and vector k-SUM, we can further extend the result to density 1 using a couple of other reductions. Please see Section 5 for details.

Public Key Encryption. Finally, we demonstrate an application of the planted search k-XOR problem to cryptography. We construct a Public-Key Encryption (PKE) scheme whose security is based on the hardness of the planted search k-XOR problem at low densities together with the hardness of the Learning Parity with Noise (LPN) problem. The hardness required from LPN here is weaker than what was previously known to imply PKE. At a high level, this is possible because the hardness of (decision) k-XOR serves as a computational analogue of the leftover hash lemma – this allows us to set the LPN parameters to result in public keys that are only computationally close to random, rather than statistically close to random, allowing us to weaken the hardness needed from LPN.

In our construction, 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. Note that the secret key can be interpreted as a k-sparse vector. The security parameter is the number r of vectors which must be generated. 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 $e \leftarrow \mathrm{Ber}_\eta^r$ where $\eta \in (0,1)$ is some noise parameter. Distinguishing such a noisy linear combination from a random vector is now hard by LPN, implying indistinguishability of ciphertexts. Decryption follows by using the location of the planted solution to annihilate the large term $s^\top X$ in the encryption of one. Sparsity of the secret key vector ensures that the added noise does not blow up too much.

The reason the hardness of k-XOR helps weaken the assumption on LPN is as follows. Suppose we wish to work with LPN with some constant noise rate η. In order to be able to decrypt correctly in the above construction, we would need to plant a set of fewer than $k = (1/\eta)$ vectors in the public-key matrix that sum to 0. Doing so might alter the distribution of the public matrix, whereas the hardness of LPN is only with respect to a public matrix that is uniformly random. If we want the distribution of the planted matrix to be close to uniform, then it needs to at least have enough rows so that sets of k vectors that sum to 0 occur naturally in the uniform distribution. This ends up requiring around $2^{m^{0.5}}$ rows, and so LPN had to be hard for algorithms running in this time. If decision k-XOR was hard, we would not need to rely on this statistical closeness to uniform, and the number of rows in the public matrix can be much smaller while keeping it computationally indistinguishable from uniform. This lets us weaken the hardness required from constant-noise LPN. Please see Section 6 for details.

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]. Additionally, an algorithm for k-SUM with numbers in the range $\{0, \ldots, M-1\}$ 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, Stephens-Davidowitz and Vaikuntanathan [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, Keller and Klein [DKK21], as discussed above, show lower bounds at densities in $(1,2)$ assuming the k-SUM conjecture at density 1. Dalirrooyfard, Lincoln and Williams [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-SUM/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, Jaiswal, Kabanets and Wigderson [IJKW10], who also relied similarly on bipartite graphs with expansion properties.

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, Lincoln and Williams [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.

Structured problems where a hidden solution can be planted have also been used to construct cryptography in [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 has been used to construct Public Key Encryption by Lyubashevsky, Palacio and Segev [LPS10]. We stress that our PKE construction based on LPN and k-XOR is not a simple modification of this construction. In fact, the appropriate adaptation of their construction to k-XOR would be insecure[5].

1.4 Open Problems

Our work raises multiple interesting questions, some of which we state below.

1. Are there algorithms for planted k-XOR at densities in $\left(1/r^{0.5}, 1\right)$ 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 be applied to other problems in fine-grained complexity?
6. Can the hardness of k-SUM or k-XOR help to weaken assumptions made for other cryptographic constructions?

[5]Briefly, the construction by [LPS10] relies on the hardness of subset sum (or possibly k-SUM) at a density where the number of bits in each element is roughly equal to the number of elements in an instance. At this very low density, k-XOR (unlike subset sum or k-SUM) can be easily solved using Gaussian elimination (see Section 4.5).

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 use standard notation for asymptotics, $\mathcal{O}(\cdot), o(\cdot), \Omega(\cdot), \omega(\cdot)$, and use a subscript $\mathcal{O}_k(\cdot)$ to hide factors that only depend on k. Similarly, we use the tilde $\tilde{\mathcal{O}}(\cdot)$ to hide polylogaritmic factors in the main parameter (usually r). 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,

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

The total variation distance gives an upper bound on the advantage of any algorithm in distinguishing between the two probability distributions D, D'.

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 $\varepsilon > 0$,*

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

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 $\varepsilon > 0$,*

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

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

The Paley-Zygmund inequality gives an anti-concentration 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 $\varepsilon \in [0,1]$,*

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

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

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

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 $\varepsilon > 0$,*

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

Similarly, the following also holds,

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

Finally, the Hoeffding also bounds the probability with which a sum exceeds a certain threshold.

Lemma 2.6 (Hoeffding's Inequality, [Hoe63])**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables on $\{0,1\}$, and let $X = \sum_{i=1}^{n} X$. Then for every $\varepsilon > 0$,*

$$\Pr\left[X > \mathbb{E}[X] + \varepsilon\right] < \exp\left(-\frac{2\varepsilon^2}{n}\right).$$

Spectral Graph Theory. We will analyze our construction for the hardness amplification by representing it as a graph and obtain bounds on its edge expansion to argue correctness (see Section 5. 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 $S, T \subseteq V$, we denote by $E(S, T)$ the set of edges connecting S and T, i.e. $(u, v) \in E(S, T)$ iff $u \in S, v \in T$ 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 $i, j \in [n]$ [Bon82].

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 2.7 (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.

Definition 2.8 (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 2.9 (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 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. For brevity, we may simply refer to the ensemble as $\left\{G^{(r)}\right\}$ (omitting the subscript).

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

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

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.*

Remark 3.2. *A more natural definition for density, as described in Section 1, is $\Delta(G) = \log \binom{r}{k} / \log |G|$, which corresponds more closely to the expected number of solutions. The above definition, however, is much more convenient to use in analysis, and is still a good approximation to this quantity – the difference between them is roughly $\mathcal{O}\left(\frac{\Delta \log k}{\log r}\right)$. So we use this instead, as Dinur, Keller and Klein [DKK21] also implicitly do.*

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.[6]

Definition 3.3 (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 $\mathcal{O}\left(\log \left|G^{(r)}\right| \operatorname{polylog} \log \left|G^{(r)}\right|\right)$.*
- 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 $\mathcal{O}\left(\operatorname{polylog} \left|G^{(r)}\right|\right)$.*
- Convergent density: $\Delta_k(\mathsf{G})$ *exists and is finite. Further, we have:*

$$\frac{\left|\Delta_k^{(r)}(\mathsf{G}) - \Delta_k(\mathsf{G})\right|}{\Delta_k(\mathsf{G})} \leq \frac{1}{\log \left|G^{(r)}\right|}. \tag{3}$$

If the density $\Delta_k^{(r)}(\mathsf{G})$ were to be equal to the limit $\Delta_k(\mathsf{G}) = \Delta$, then the size of the group $G^{(r)}$ would have to be equal to $r^{k/\Delta}$. As these quantities are discrete, this exact equality cannot be achieved for arbitrary values of Δ. Instead, we have the above convergence condition, which ensures that the size of $G^{(r)}$ is always within a factor of 2 of its ideal value $r^{k/\Delta}$.

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. For most of the paper, we will also ignore the convergence error, and assume that $\Delta_k^{(r)}(\mathsf{G}) = \Delta_k(\mathsf{G})$ in our analysis. This error is only of size $(\Delta_k(\mathsf{G})/\log \left|G^{(r)}\right|)$. As we almost always work with small values of density, this will not affect our results substantially.

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.

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

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 appropriately chosen m. We define it as follows.

$$\mathsf{G}_{k\text{-XOR}}^{(\Delta)} = \left\{GF_{2^{m(r)}}\right\}, \quad \text{where } m(r) = \left\lceil \frac{k \log r}{\Delta} \right\rceil. \tag{5}$$

We will refer to the k-SUM problem over $\mathsf{G}_{k\text{-XOR}}^{(\Delta)}$ simply as *the k-XOR problem*. We introduce a natural generalisation of k-XOR that we call *vector k-SUM* which is defined as follows.

$$\mathsf{G}_{\text{VECTOR-}(q,\,k)\text{-SUM}}^{(\Delta)} = \left\{(\mathbb{F}_q)^{m(r)}\right\}, \quad \text{where } m(r) = \left\lceil \frac{k \log r}{\Delta \lg q} \right\rceil. \tag{6}$$

The k-XOR problem remains a special case with $q = 2$. If q is given by the context, we may refer to this problem as simply *vector k-SUM*. It may be verified that all three of these ensembles are admissible.

[6]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 quasilinear 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.

3.1 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). If r is clear from the context, we may refer to the distribution simply as D_0 (omitting the superscript r).

Definition 3.4 (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,*

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

Where the randomness is taken over the instance and the random coins used by $\mathcal{A}$. 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 (or a *k-XOR solution* for the k-XOR problem). 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 $\widetilde{\mathcal{O}}(r^{\lceil k/2\rceil})$ using a simple meet-in-the-middle algorithm. So for any ensemble G of density 1, there is an algorithm that runs in time $\widetilde{\mathcal{O}}(r^{\lceil k/2\rceil})$ 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, Keller and Klein [DKK21], where they are stated to be folklore.[7] Weaker versions appear in [LLW19] and [Pet15].

Conjecture 3.5 (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 3.6 (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 $\Omega(r^{\lceil k/2\rceil - o(1)})$.*

3.2 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.

[7] To be accurate, Dinur, Keller and Klein 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

If r is clear from the context, we may refer to the distribution simply as D_1 (omitting the superscript r).

Remark 3.7. *Another natural distribution to study in the low-density regime is the uniform distribution conditioned on there being at least one solution. However, there is no simple way to sample from this distribution. It is worth noting that, at density $(1-\epsilon)$ for any constant $\epsilon > 0$, this distribution is statistically close to, but not the same as, the planted distribution.*

Once again, we may define a (planted) search k-SUM problem for the planted distribution in the same way as we did in Definition 3.4.

Definition 3.8 (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,*

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

Where the randomness is taken over the distribution $D_1^{(r)}$ and the random coins used by $\mathcal{A}$. If $\epsilon = \Omega(1)$, we simply say that $\mathcal{A}$ solves the planted search k-SUM problem over G.

Note that the $\widetilde{\mathcal{O}}\left(r^{\lceil k/2 \rceil}\right)$-time algorithm mentioned above can solve the planted search k-SUM problem with probability $(1 - o(1))$. For certain group ensembles with additional structure, the k-SUM problem becomes easy to solve at very low densities. For instance, in the k-XOR problem, each element in the instance is a vector. If the length of these vectors is larger than r, then with high probability the planted solution will be the only linear dependence among these vectors, and can be found by Gaussian elimination (see Section 4.5 for details).

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 3.9 (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 both $b \in \{0, 1\}$,*

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

Where the randomness is taken over the random coins chosen by $\mathcal{A}$. 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 4.1 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.

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 $G^{(\Delta_0)}_{k\text{-SUM}}$ and $G^{(\Delta_0)}_{k\text{-XOR}}$, though we may omit formally specifying this and trust it is clear from the context what we mean.

3.3 Statistics on the Number of Solutions

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

Definition 3.10 (Number of Solutions)**.** *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 $c(X)$.*

We will now prove certain properties about $c(X)$ for the uniform as well as the planted distribution that will be useful for proving several different results about the k-SUM problem.

Lemma 3.11. *For a vector X sampled from the uniform distribution D_0,*

$$\mathbb{E}\left[c(X)\right] = \binom{r}{k}/|G|. \tag{7}$$

$$\mathrm{Var}(c(X)) = \frac{\binom{r}{k}}{|G|}\left(1 - \frac{1}{|G|}\right). \tag{8}$$

If X is instead sampled from the planted distribution D_1,

$$\mathbb{E}\left[c(X)\right] = 1 + \frac{\binom{r}{k} - 1}{|G|}. \tag{9}$$

$$\mathrm{Var}(c(X)) < \frac{\binom{r}{k}}{|G|}\left(1 + \frac{2^k k^2}{r}\right). \tag{10}$$

Proof. For each $S \subset [r]$ with $|S| = k$, let I_S be the indicator random variable for whether S represents a k-SUM solution. Formally,

$$I_S = \mathbb{1}\left[\sum_{i \in S} X_i = 0\right].$$

We will first consider the case where X is sampled uniformly, i.e. $X \sim D_0$. Since G is a finite group, the sum of a set of elements is uniformly random as long as at least one of those elements is chosen randomly. Therefore, each I_S is a Bernoulli random variable with success probability $\frac{1}{|G|}$. Furthermore, since we only consider sets of size k, for any two distinct sets S and T we can find some i such that $i \in S$ and $i \notin T$. Since all the group elements are chosen independently,

$$\Pr[I_S = 0 \mid I_T] = \Pr\left[X_i = -\sum_{\substack{j \in S \\ j \neq i}} X_j \;\middle|\; I_T\right] = \Pr\left[X_i = -\sum_{\substack{j \in S \\ j \neq i}} X_j\right] = \Pr[I_S]. \tag{11}$$

This shows that the variables I_S are i.i.d. Bernoulli variables. Note that we can write $c(X) = \sum I_S$. We therefore have

$$\mathbb{E}\left[c(X)\right] = \mathbb{E}\left[\sum_{\substack{S \subset [r] \\ |S| = k}} I_S\right] = \sum_{\substack{S \subset [r] \\ |S| = k}} \mathbb{E}\left[I_S\right] = \sum_{\substack{S \subset [r] \\ |S| = k}} \frac{1}{|G|} = \frac{\binom{r}{k}}{|G|}. \tag{12}$$

$$\mathrm{Var}(c(X)) = \mathrm{Var}\left(\sum_{\substack{S\subset[r]\\|S|=k}} I_S\right) = \sum_{\substack{S\subset[r]\\|S|=k}} \mathrm{Var}(I_S) = \sum_{\substack{S\subset[r]\\|S|=k}} \frac{1}{|G|}\left(1-\frac{1}{|G|}\right) = \frac{\binom{r}{k}}{|G|}\left(1-\frac{1}{|G|}\right). \tag{13}$$

Now let us consider the sampled distribution; $X \sim D_1$. We denote by K the k-tuple where the solution has been planted. By linearity of expectation, we can still calculate the expected number of solutions quite simply.

$$\mathbb{E}\left[c(X)\right] = \mathbb{E}\left[\sum_{\substack{S\subset[r]\\|S|=k}} I_S\right] = \sum_{\substack{S\subset[r]\\|S|=k}} \mathbb{E}\left[I_S\right] = \mathbb{E}\left[I_K\right] + \sum_{\substack{S\subset[r]\\|S|=k\\S\neq K}} \frac{1}{|G|} = 1 + \frac{\binom{r}{k}-1}{|G|}. \tag{14}$$

However, we can no longer calculate the total variance in the same way as before since these variables may not be independent anymore. We will bound $\mathrm{Var}(c(X))$ by arguing that I_S and I_T are independent for *most* S, T pairs.

Let S and T be any two distinct subsets of $[r]$, both of which are different from K. Since the indicator variables are binary, independence can be shown by proving $\Pr[I_S = 1 | I_T = 1] = \Pr[I_S = 1]$. Observe that if $S \nsubseteq T \cup K$, there exists some index $i \in S$ such that $i \notin T \cup K$. In this case, we can just repeat the argument in Eq. (11) to establish independence. Similarly, if $T \nsubseteq S \cup K$, we are done as well by symmetry. If $K \nsubseteq S \cup T$, there must be some index $l \in K$ such that $l \notin S \cup T$. Observe that in the definition of the planted distribution, it does not matter which of the k elements in the planted solution is chosen to be replaced. Therefore, without loss of generality, we can assume that X_l was the element replaced during the planting process. However, this implies that all the elements in $S \cup T$ are independent and chosen uniformly at random; this implies the independence of I_S and I_T as before.

In the following calculations, S and T are always size-k subsets of $[r]$. Note that we can write the variance as follows.

$$\mathrm{Var}\left(c(X)\right) = \mathrm{Var}\left(\sum_{S} I_S\right)$$

Since I_K is always 1, subtracting it from $c(X)$ does not change the variance.

$$\begin{aligned}
&= \mathrm{Var}\left(\sum_{S\neq K} I_S\right) \\
&= \sum_{S\neq K} \mathrm{Var}(I_S) + \sum_{S\neq K} \sum_{\substack{T\neq K\\T\neq S}} \mathrm{Cov}(I_S, I_T)
\end{aligned}$$

As shown above, the covariance terms are zero unless any two of S, T and K contain the third.

$$= \left(\binom{r}{k} - 1\right) \cdot \frac{1}{|G|} \cdot \left(1 - \frac{1}{|G|}\right) + \sum_{\substack{S,T,K \text{ distinct}\\S\subset T\cup K\\T\subset S\cup K\\K\subset S\cup T}} \mathrm{Cov}(I_S, I_T)$$

Using the inequalities $\mathrm{Cov}(X,Y) \leq \sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}$, and $\mathrm{Var}(I_S) = \mathrm{Var}(I_T) < 1/|G|$, we can rewrite this as,

$$< \frac{\binom{r}{k}}{|G|} + \sum_{\substack{S,T,K \text{ distinct}\\S\subset T\cup K\\T\subset S\cup K\\K\subset S\cup T}} \frac{1}{|G|}$$

We now count the number of S, T pairs satisfying these constraints. Since $|S| = |T| = |K| = k$, we must have $|K \setminus S| = |K \setminus T| = |S \setminus K| = l$ for some $l \in [1, k/2]$. For a fixed l, we can choose $K \setminus S$ and $K \setminus T$ in $\binom{k}{l}$ ways. We have $\binom{r-k}{l}$ ways of choosing $S \setminus K$.

$$\begin{aligned}
&\leq \frac{\binom{r}{k}}{|G|} + \sum_{l=1}^{k/2} \binom{k}{l}^2 \binom{r-k}{l} \cdot \frac{1}{|G|} \\
&< \frac{\binom{r}{k}}{|G|} + \frac{k}{2|G|} \binom{k}{k/2}^2 \binom{r-k}{k/2} \\
&< \frac{\binom{r}{k}}{|G|} + \frac{k2^k}{2|G|} \binom{r-1}{k-1} \\
&= \frac{\binom{r}{k}}{|G|} \cdot \left(1 + 2^k k^2 / r\right)
\end{aligned}$$

□

4 Basic Complexity of k-SUM

In this section, we provide various results about the basic complexity of the k-SUM problem and its variants in the sparse regime. We first show an equivalence of planted and non-planted k-SUM at densities $\Delta \geq 1$. We then show two conditional lower bounds on the runtime of an algorithm that solves planted k-SUM at any density $\Delta \in \left(\frac{1}{2}, 1\right]$. We then show a reduction from search to decision at densities $\Delta < 1$. Finally, we show how to solve k-XOR efficiently at very low densities.

4.1 Relating 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 (Equivalence at Density 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 $\epsilon(r)^{3/2}/(21k^k)$.*

This theorem implies that planting is a fine-grained one-way function assuming the (non-planted) average-case k-SUM conjecture over G holds. To illustrate this, consider the case of k-XOR. Let,

$$f : \mathbb{F}_2^{\lceil k \log r \rceil \times r} \times \binom{r}{k} \to \mathbb{F}_2^{\lceil k \log r \rceil \times r},$$

be the 'planting function' that takes as input a matrix at density 1 – such that $m = k \log r$ – and plants a solution in the locations specified by the second input, where these locations are ordered lexicographically among all subsets of $[r]$ of size k. For instance, the output $f(1)$ is a random matrix that has a solution in the set $[k]$. Then, assuming the average-case k-XOR conjecture, it follows immediately from Theorem 4.1 that f is a fine-grained one-way function that takes $\widetilde{\mathcal{O}}(r)$ time to compute, and cannot be inverted by algorithms running in $r^{k/2-\Omega(1)}$ time. 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 their work, the one-way function relies implicitly on the hardness of planted k-SUM, whereas ours can rely on the hardness of non-planted search k-SUM.

We observe that in the dense regime, the two distributions are equivalent in a stronger sense.

Theorem 4.2 (Statistical Closeness in Dense Regime)**.** *Fix some admissible group ensemble* $\mathsf{G} = \left\{G^{(r)}\right\}_{r\in\mathbb{N}}$ *and let* $D_0^{(r)}$ *(resp.* $D_1^{(r)}$*) be the non-planted (resp. planted) distribution on* r *group elements. If for some* $k > 0$*, it holds that the density* $\Delta = \Delta_k(\mathsf{G}) > 1$*, then,*

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

This means that if we modify f above to have $k = \Theta(\log r)$ and $\Delta > 1$ be some constant, then this 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 at density $\Delta = 1$, 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 ϵ'. 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. $D_1(X) = \frac{|G|}{\binom{r}{k}}\, c(X)\, D_0(X)$.
2. $D^\ell(X) = \left(\mathbb{1}[c(X) \le \ell] \cdot \frac{|G|}{\binom{r}{k}}\, c(X) + \Pr_{X\sim D_1}[c(X) > \ell]\right) D_0(X)$.

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

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

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 $c(X) \geq 1$ 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 $|G|$ such vectors corresponding to each possible group element as Y_i. Starting from X, we can therefore make $|G| \cdot c(X)$ 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 $\frac{1}{|G|^r}\frac{1}{\binom{r}{k}}$. Multiplying the two expressions, we get

$$D_1(X) = \frac{|G|}{\binom{r}{k}} \cdot c(X) \cdot \frac{1}{|G|^r} = \frac{|G|}{\binom{r}{k}} c(X) D_0(X)$$

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 $D_1(X) = \left(|G|/\binom{r}{k}\right) c(X) D_0(X)$. □

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. $R(D^\ell \| D_0) = \left(\frac{|G|}{\binom{r}{k}} \cdot \ell + \Pr_{X \sim D_1}[c(X) > \ell] \right)$.

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. *For any $\ell > 3$, the following two inequalities hold at density $\Delta = 1$:*

$$SD(D^\ell, D_1) \leq \Pr_{X \sim D_1}[c(X) > \ell] \leq \frac{2}{(\ell - 3)^2} \tag{15}$$

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. Note that as density Δ is 1, we have $|G| = r^k > \binom{r}{k}$ if $k \geq 3$.

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

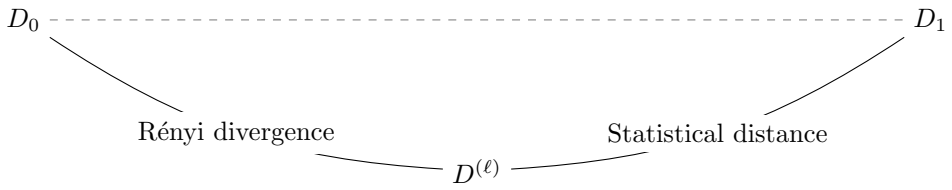

Figure 3: Depiction of the argument for statistical closeness of D_0 and D_1. We define a class of hybrid distributions $D^{(\ell)}$ for which we can bound the distance to D_0 and D_1, specifically we bound the Rényi divergence between D_0 and $D^{(\ell)}$ and the statistical distance between $D^{(\ell)}$ 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.

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.

$$\begin{aligned} SD(D^\ell, D_1) &= \sum_{\substack{X \in G^r \\ c(X) \leq \ell}} \Pr_{X \sim D_1}[c(X) > \ell]\, D_0(X) \\ &= \Pr_{X \sim D_1}[c(X) > \ell] \cdot \frac{|\{X \in G^r \mid c(X) \leq \ell\}|}{|G|^r} \\ &= \Pr_{X \sim D_1}[c(X) > \ell] \cdot \Pr_{X \sim D_0}[c(X) \leq \ell] \\ &\leq \Pr_{X \sim D_1}[c(X) > \ell] \end{aligned} \tag{16}$$

Note that the standard deviation of $c(X)$ when $X \sim D_1$ is less than $\sqrt{2\binom{r}{k}/|G|}$ for large enough r (Lemma 3.11, Eq. (10)). Furthermore, Eq. (3) at density 1 implies $|G^r| \geq r^k/2$. Therefore,

$$\mathop{\mathbb{E}}_{X \sim D_1}[c(X)] < 1 + \frac{\binom{r}{k}}{|G|} \leq 1 + \frac{2\binom{r}{k}}{r^k} < 3, \quad \text{using Lemma 3.11, Eq. (9).}$$

We can now apply Chebyshev's inequality (Lemma 2.3) to get,

$$\begin{aligned} &SD(D^\ell, D_1) \\ &\quad \leq \Pr_{X \sim D_1}[|c(x) - 3| > \ell - 3] \\ &\quad \leq \Pr_{X \sim D_1}\left[|c(X) - \mathbb{E}[c(X)]| > \frac{\ell - 3}{\sqrt{2\binom{r}{k}/|G|}} \mathrm{Std}(c(X))\right] \\ &\quad \leq \frac{2\binom{r}{k}}{|G|(\ell-3)^2} \\ &\quad \leq \frac{2}{(\ell-3)^2} \cdot \left(\frac{er}{k}\right)^k \cdot \frac{1}{r^k} \\ &\quad \leq \frac{2}{(\ell-3)^2} \end{aligned}$$

□

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. 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 success probability. We shall prove that

$\mathcal{A}$ solves non-planted k-SUM with probability $\geq \epsilon'$ for some other constant $\epsilon' > 0$. Since $\mathcal{A}$ succeeds with probability $\geq \epsilon$ on inputs from D_1, it must have a success probability at least $\epsilon - SD(D^\ell, D_1)$ on inputs from D^ℓ. Lemma 2.1 now implies that the success probability of $\mathcal{A}$ on inputs from D_0 must satisfy

$$\epsilon' \geq \frac{\epsilon - SD(D^\ell, D_1)}{R(D^\ell \| D_0)}$$

Plugging in values from Lemma 4.5 and Corollary 4.4, we get that,

$$\begin{aligned} &\geq \frac{\epsilon - \frac{2}{(\ell-3)^2}}{\left(\frac{|G|}{\binom{r}{k}} \cdot \ell + \Pr_{X \sim D_1}[c(X) > \ell]\right)} \\ &\geq \frac{\epsilon - \frac{2}{(\ell-3)^2}}{\frac{\ell|G|}{\binom{r}{k}} + \frac{2}{(\ell-3)^2}} \end{aligned}$$

To ensure that $\epsilon' > 0$, solving for ℓ, we get that $\ell > 3 + \sqrt{\frac{2}{\epsilon}}$. 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 $\ell = 3 + \frac{2}{\sqrt{\epsilon}}$ which gives a bound of,

$$\epsilon' \geq \frac{\epsilon/2}{\frac{(3\sqrt{\epsilon}+2)|G|}{\binom{r}{k}\sqrt{\epsilon}} + \frac{\epsilon}{2}}$$

By Eq. (3), we have $\frac{|G|}{\binom{r}{k}} < \frac{2r^k}{(r/k)^k} = 2k^k$. So the above inequality simplifies to

$$\begin{aligned} &> \frac{\epsilon^{3/2}}{(12\sqrt{\epsilon} + 8)k^k + \epsilon^{3/2}} \\ &> \frac{\epsilon^{3/2}}{21k^k} = \Omega_k\left(\epsilon^{3/2}\right) \end{aligned}$$

□

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,

$$SD\left(D_0, D_1\right) = \sum_{X \in M} D_0(X) = \frac{|M|}{|G|^r} = \Pr_{X \sim D_0}[c(X) = 0] = 1 - \Pr_{X \sim D_0}[c(X) > 0]$$

We can now use the Paley-Zygmund inequality (Lemma 2.4) to get

$$\Pr_{X \sim D_0}[c(X) > 0] \geq \frac{\mathop{\mathbb{E}}_{X \sim D_0}[c(X)]^2}{\mathop{\text{Var}}_{X \sim D_0}[c(X)] + \mathop{\mathbb{E}}_{X \sim D_0}[c(X)]^2}$$

Substituting the values from Lemma 3.11, Eqs. (7) and (8), we get

$$\begin{aligned}
&= \frac{\left(\binom{r}{k}/|G|\right)^2}{\frac{\binom{r}{k}}{|G|}\left(1-\frac{1}{|G|}\right)+\left(\binom{r}{k}/|G|\right)^2} \\
&= \frac{\binom{r}{k}}{|G|-1+\binom{r}{k}} \\
&> 1-\frac{|G|}{|G|+\binom{r}{k}}
\end{aligned}$$

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

$$SD\left(D_0, D_1\right) < \frac{|G|}{|G|+\binom{r}{k}} < \frac{|G|}{\binom{r}{k}} < \frac{2r^{k/\Delta}}{(r/k)^k} = \mathcal{O}\left(k^k r^{k\left[\frac{1}{\Delta}-1\right]}\right) = \mathcal{O}_k\left(r^{k\left[\frac{1}{\Delta}-1\right]}\right) \qquad \square$$

4.2 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 4.2.1, 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 4.2.2, 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.[8] 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 $2^m / r^k = r^{k\left[\frac{1}{\Delta}-1\right]}$ 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 3.5[9] 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 4.6 (Dinur, Keller, Klein [DKK21])**.** *Suppose Conjecture 3.5 (resp. Conjecture 3.6) 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

[8] See Footnote 7 in Section 3 discussing the slightly different definition of the k-SUM problem as considered by [DKK21].

[9] Throughout this section, we use a weaker version of Conjectures 3.5 and 3.6 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.

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 3.5 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 4.7 (Follows from techniques in [DKK21])**.** *Suppose Conjecture 3.6 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 $\Omega\left(r^{\left[k\left(\frac{3}{2}-\frac{1}{\Delta}\right)\right]-o(1)}\right)$.*

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

In this section, we show how to generalize Theorem 4.7 to k-SUM in arbitrary groups. Specifically, we will prove the following theorem.

Theorem 4.8. *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 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 $\mathcal{O}\left(r^{k\,(1-\Delta)}\,T\right)$, as desired. □

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

Corollary 4.9 (Conditional Lower Bound)**.** *Suppose Conjecture 3.5 (resp. Conjecture 3.6) 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 $\Omega\left(r^{\left[k\left(1-\frac{1}{2\Delta}\right)\right]-o(1)}\right)$.*

4.2.2 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 4.10. *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 $\mathcal{O}\left(r^{k_2-k_1}T(r)\right)$ 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,

$$1-\left(1-\frac{1}{3^{2k_2}r^{k_2-k_1}}\right)^{3^{2k_2}r^{k_2-k_1}} \geq 1-\frac{1}{e} = \Omega(1)$$

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 $\mathcal{O}\left(r^{k_2-k_1}T(r)\right)$, as required. □

Corollary 4.11 (Conditional Lower Bound for Different k's)**.** *Suppose Conjecture 3.5 (resp. Conjecture 3.6) 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 $\Omega\left(r^{\left[k\left(1-\frac{1}{2\Delta}\right)\right]-o(1)}\right)$.*

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 4.10 implies that we can solve k'-SUM at density 1 in time $\mathcal{O}\left(T\, r^{k'-k}\right)$. By assumption, this is at least $\Omega\left(r^{\lceil k'/2 \rceil - o(1)}\right)$. This, along with the definition of Δ, implies $T = \Omega\left(r^{\left[k\left(1-\frac{1}{2\Delta}\right)\right]-o(1)}\right)$, as needed. □

4.3 Search to Decision Reduction

A search-to-decision reduction for k-SUM is implied by the work of Impagliazzo and Naor [IN89]. They show a similar reduction for the Subset Sum problem modulo prime numbers or powers of 2, and their proof can be extended – using an efficient instantiation of the Goldreich-Levin algorithm [GL89, Tre04] and some minor optimizations – to obtain the following theorem.

Theorem 4.12 (Search-to-Decision Reduction, implied by [IN89])**.** *For $k \in \mathbb{N}$ and $\Delta < 1$, suppose there is an algorithm that runs in time $T(r)$, and solves the* decision *k-SUM problem over $\mathsf{G}^{(\Delta)}_{k\text{-}SUM}$ (resp. $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$) with success probability $(1/2 + \epsilon)$. Then, there is an algorithm that runs in time $O(T(r) \cdot r \log(r) \cdot (k/\epsilon)^4)$, and solves the* planted search *k-SUM problem over $\mathsf{G}^{(\Delta)}_{k\text{-}SUM}$ (resp. $\mathsf{G}^{(\Delta)}_{k\text{-}XOR}$) with success probability at least $3/4$.*

We show a different search-to-decision reduction for the k-SUM problem over general group ensembles that is incomparable to the one above. Whereas the above reduction can work with any decision algorithm that has success probability more than $1/2$, our reduction requires this success probability to be close to 1. On the other hand, it avoids the factor of r loss in the running time of the above reduction. Our proof is also more elementary, using an algorithm reminiscent of binary search. The precise statement we show is the following.

Theorem 4.13 (Search-to-Decision Reduction)**.** *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 $\widetilde{\mathcal{O}}(T(r))$, 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

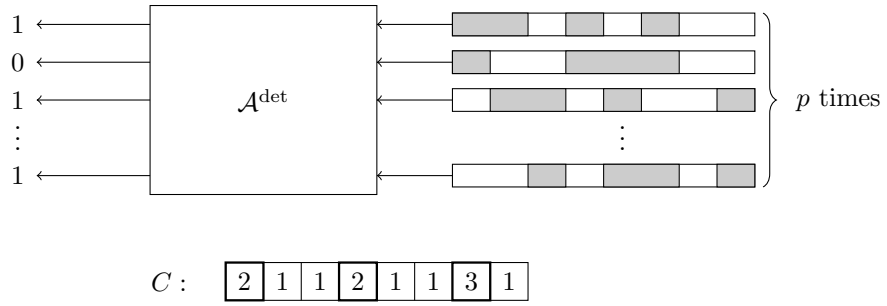

Figure 4: Illustration of the structure of the algorithm $\mathcal{A}^{\text{rec}}$ shown for an instance X of size $r = 8$ and $k = 3$. It chooses a random subset S of the elements of size $r/2 = 4$ and replaces all elements in X from S with fresh samples from the group and gives the resulting instance 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.

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

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.

Lemma 4.14. *Let $K \subseteq [r]$ denote the planted set. Then for sufficiently large r it holds that in any given iteration of $\mathcal{A}^{\text{rec}}$, 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. Observe that the $R(X)$ used in a given iteration is effectively sampled from D_1 if all the elements in K were preserved, and it is sampled from D_0 otherwise. In either case, we can upper bound the probability of a solution distinct from K existing by $\Pr_{Y \sim D_1}[c(Y) > 1]$.

Note that the standard deviation of $c(Y)$ when $Y \sim D_1$ is less than $\sqrt{2\binom{r}{k}/|G|}$ for large enough r (Lemma 3.11, Eq. (10)). Furthermore, Eq. (3) implies $|G| \geq r^{k/\Delta}/2$. Therefore by using Lemma 3.11, Eq. (9),

$$\mathop{\mathbb{E}}_{Y \sim D_1}[c(Y)] < 1 + \frac{\binom{r}{k}}{|G|} \leq 1 + \frac{2\binom{r}{k}}{r^{k/\Delta}} < 1 + r^{k\,(1-\frac{1}{\Delta})}.$$

We can now apply Chebyshev's inequality (Lemma 2.3) to get,

$$\begin{aligned}
\Pr_{Y \sim D_1}[c(Y) > 1] &= \Pr_{Y \sim D_1}[c(Y) \geq 2] \\
&\leq \Pr_{Y \sim D_1}\left[|c(Y) - \mathbb{E}[c(Y)]| > 1 - r^{k\,(1-\frac{1}{\Delta})}\right] \\
&\leq \Pr_{Y \sim D_1}\left[|c(Y) - \mathbb{E}[c(Y)]| > \frac{1 - r^{k\,(1-\frac{1}{\Delta})}}{\sqrt{2\binom{r}{k}/|G|}}\,\mathrm{Std}(c(Y))\right] \\
&\leq \frac{2\binom{r}{k}}{|G|\left(1 - r^{k\,(1-\frac{1}{\Delta})}\right)^2}
\end{aligned}$$

Substituting $\binom{r}{k} < (r/k)^k < r^k/2$ and $|G| \geq r^{k/\Delta}/2$, we get,

$$< \frac{2r^{k\,(1-\frac{1}{\Delta})}}{\left(1 - r^{k\,(1-\frac{1}{\Delta})}\right)^2}$$

The denominator is a monotonically increasing function in r with limit 1. Therefore, for sufficiently large r, it will exceed $1/2$, and hence we get,

$$< 4r^{k\,(1-\frac{1}{\Delta})}$$ □

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 4.15. *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.

$$\Pr_{X \sim D_0}[\mathrm{Det}(X) \neq 0] = \Pr_{X \sim D_0}[c(X) \geq 1]$$

Applying Markov's inequality (Lemma 2.2), we get

$$\begin{aligned}
&\leq \mathbb{E}[c(X)] = \binom{r}{k}/|G| && \text{Lemma 3.11, Eq. (7)} \\
&< r^k/|G| \leq 2r^{k\,(1-\frac{1}{\Delta})} && \text{Eq. (3)}
\end{aligned}$$

which as remarked is subconstant. □

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

Lemma 4.16. *If $\mathcal{A}^{\text{det}}$ 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 $\mathcal{A}^{\text{det}}$ 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 4.15 that except with probability $o(1)$, $X \sim D_{\text{Det}(X)}$. 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 $Z(X) = \Pr_{Y \leftarrow R(X)}[\mathcal{A}(Y) \neq \text{Det}(Y)]$, 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 4.13. 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.

Fix an input X. Now, in the reduction, for each choice of $R(X)$, the counter will be incremented by some 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 4.16, 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,

$$\mathbb{E}[C_i] \geq p\left(\frac{1}{2^k} - \epsilon\right), \qquad \text{when } i \text{ is solution.}$$

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

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).

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

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. □

4.4 Reduction from k-SUM to Subset 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 4.17 (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 4.18 (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 4.19. *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 4.20. *In fact, the specific algorithms described in [LO85, Ben22] can be applied directly to solve* k*-SUM at the densities in Corollaries 4.23 and 4.24 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.

4.4.1 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 4.21 (Worst-Case Reduction to Subset Sum)**.** *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. □

4.4.2 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 4.22 (Average-Case Reduction to Subset Sum)**.** *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 constant density $\Delta < 1$. Then, for any constant k, 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 $1 \le i \le r$.
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 will 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 $Y_t \leftarrow -\sum_{\substack{j \in T \\ j \neq t}} Y_j$
4. Choose α uniformly at random from $\mathbb{Z}_p$.
5. Increment Y_i by α for all $1 \le i \le r$.

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 $Y_t \leftarrow k\alpha - \sum_{\substack{j \in T \\ j \neq t}} Y_j$. 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 $Y_t \leftarrow k\alpha - \sum_{\substack{j \in T \\ j \neq t}} Y_j$

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, for any setting of the Y_i's, $k\alpha - \sum_{\substack{j \in T \\ j \neq t}} Y_j$ is a uniformly random member of $\mathbb{Z}_p$, since so is α and thus $k\alpha$ as k is co-prime to p. 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 $\mathbb{Z}_p^r$. Further, due to symmetry, the set T is also independent of Y and distributed uniformly over all subsets of $[r]$ of size k.

Observe that since the density of the provided k-SUM is $\frac{k\Delta \log r}{r}$, we have $\frac{k \log r}{\log p} = \frac{k\Delta \log r}{r}$, and so $\frac{r}{\log p} = \Delta$. 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. We can now upper bound the probability of there being a different subset of Y having the same sum as follows:

$$
\begin{aligned}
&\Pr\left[\exists \mathbf{a} \in \{0,1\}^r \text{ such that } \langle Y, \mathbf{a} \rangle = \langle Y, \mathbf{b} \rangle \text{ and } \mathbf{a} \neq \mathbf{b}\right], && \mathbf{b} \text{ is the indicator bit vector for } S' \\
&\leq \sum_{\substack{\mathbf{a} \in \{0,1\}^r \\ \mathbf{a} \neq \mathbf{b}}} \Pr\left[\langle Y, \mathbf{a} \rangle = \langle Y, \mathbf{b} \rangle\right], && \text{by the union bound} \\
&= \sum_{\substack{\mathbf{a} \in \{0,1\}^r \\ \mathbf{a} \neq \mathbf{b}}} \frac{1}{p} && \text{Since } Y \xleftarrow{\$} \mathbb{Z}_p^r \\
&< \frac{2^r}{p} = 2^{r - \log p} \\
&= 2^{r - \frac{r}{\Delta}} = 2^{r(1 - \frac{1}{\Delta})} \\
&= \text{negl}(r) && \text{Since } \Delta \text{ is a constant} < 1
\end{aligned}
$$

So if A succeeds on the problem instance (Y, t) constructed in line 5 with some constant probability q, we can conclude that B successfully solves the k-SUM instance with probability at least $\frac{q}{2^{k+1}}$.

$$
\begin{aligned}
&\Pr\left[B \text{ succeeds}\right] \geq \Pr\left[B \text{ succeeds} \mid S \cap T = \phi\right] \cdot \Pr\left[S \cap T = \phi\right] \\
=&\frac{1}{2^k} \Pr\left[B \text{ succeeds} \mid S \cap T = \phi\right] \\
=&\frac{1}{2^k} \Pr\left[A \text{ succeeds and } A(Y, t) \text{ passes the checks in line 6} \mid S \cap T = \phi\right]
\end{aligned}
$$

If we now condition on Y not having a second solution, A succeeding would imply it finds the set $S \cup T$ which passes the checks in line 6

$$
\geq \frac{1}{2^k}(1 - \text{negl}(r)) \Pr\left[A \text{ succeeds} \mid S \cap T = \phi \text{ and } (Y, t) \text{ has a unique subset sum solution}\right]
$$

Note that for any 3 events P, Q, R, we have $\Pr[P \mid Q \ \& \ R]\Pr[R]+\Pr[P \mid Q \ \& \ \texttt{not}R]\Pr[\texttt{not}R] = \Pr[P \mid Q]$ which implies $\Pr[P \mid Q \ \& \ R] \geq \Pr[P \mid Q] - \Pr[\texttt{not}R]$

$$\geq \frac{1-\text{negl}(r)}{2^k}\left(\Pr[A \text{ succeeds} \mid S \cap T = \phi] - \Pr[Y \text{ has a second subset sum solution}]\right)$$
$$= \frac{1-\text{negl}(r)}{2^k}\left(q - \text{negl}(r)\right) \geq \frac{q}{2^{k+1}} \qquad \text{for large } r$$

where q is the probability that A succeeds conditioned on $S \cap T = \phi$.

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 $\mathbb{Z}_p^r$. 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 $S \bigcap T = \emptyset$. Since T was independent from Y, so is S' even after the above conditioning. Therefore, since k is a constant, use Lemma 4.25 to conclude that the probability that A solves the subset sum problem we construct conditioned on $S \cap T = \phi$ (that is, q) is at least a constant. Since k is a constant, this coupled with the inequality above implies that B has constant success probability. □

Corollary 4.23 (k-SUM in $\mathbb{Z}_p$ is Easy at Very Low Density)**.** *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 $1/r$ with constant success probability (see Remark 4.19). The statement then follows from Theorem 4.22. □

We can use a reduction by [DKK21] to lift this result to k-SUM over the integers.

Corollary 4.24 (k-SUM Over Integers is Easy at Very Low Density)**.** *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 does not depend on k.*

Proof. Let p be a prime between N and $2N$. Corollary 4.23 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 4.25. *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 $\gamma_0 = \Omega_k(1)$ on inputs from D_0 and γ_1 on inputs from D_1, then $\gamma_1 \geq \gamma_0 \, 2^{-(2k+1)}$.

Proof. Let D_2 be the uniform distribution on subsets of $[r]$ with size at least $\frac{r}{4}$, and let γ_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 $|\gamma_0 - \gamma_2|$ as well as $\frac{\gamma_1}{\gamma_2}$, 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,

$$\begin{aligned}
&\Delta(D_0, D_2) \\
&= \sum_{S \in \mathrm{supp}(D_2)} (D_2(S) - D_0(S)) \\
&= \sum_{S \in \mathrm{supp}(D_0) - \mathrm{supp}(D_2)} D_0(S) \\
&= \sum_{S \subset [r], |S| < \frac{r}{4}} D_0(S) \\
&= \Pr_{S \sim D_0}\left[|S| < \frac{r}{4}\right] \\
&\leq \exp\left(-2r\left(\frac{1}{2} - \frac{1}{4}\right)^2\right)
\end{aligned}$$

Using Hoeffding's inequality (Lemma 2.6), we get,

$$\begin{aligned}
&= \exp\left(-\frac{r}{8}\right) \\
&= o(1).
\end{aligned}$$

This implies that $\gamma_2 \geq \gamma_0 - o(1) > \frac{\gamma_0}{2}$. 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} (note that the sampling process for D_1 implicitly conditions on $S \cap T = \emptyset$). 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.

$$\begin{aligned}
&R(D_2 \| D_1) \\
&= \max_{Q \in \mathrm{supp}(D_2)} \frac{D_2(Q)}{D_1(Q)} \\
&= \max_{Q \subseteq [r], |Q| \geq \frac{r}{4}} \left(D_2(Q) \cdot \frac{\binom{r}{k}}{\binom{|Q|}{k}} \cdot 2^{r-k}\right) \\
&= \max_{Q \subseteq [r], |Q| \geq \frac{r}{4}} \left(\frac{1}{|S \subseteq [r] \mid |S| \geq r/4|} \cdot \frac{\binom{r}{k}}{\binom{|Q|}{k}} \cdot 2^{r-k}\right)
\end{aligned}$$

We may bound this as follows.

$$\leq \left(\frac{1}{2^{r-1}} \cdot \frac{\binom{r}{k}}{\binom{r/4}{k}} \cdot 2^{r-k}\right).$$

This can be written as follows.

$$
\begin{aligned}
&= \frac{1}{2^{k-1}} \cdot \frac{r\,(r-1)\cdots(r-k+1)}{(r/4)(r/4-1)\cdots(r/4-k+1)} \\
&< \frac{2}{2^k} \cdot \left(\frac{r-k+1}{r/4-k+1}\right)^k \\
&< \frac{1}{2^k} \cdot \left(\frac{2r-8k+8}{r/4-k+1}\right)^k = 4^k.
\end{aligned}
$$

Where the last inequality holds for large enough values of r. Applying the Rényi divergence lemma (Lemma 2.1) on the event that algorithm A succeeds, we get that,

$$\gamma_1 \geq \gamma_2 / R(D_2 \| D_1) > \gamma_2 / 4^k.$$

Combining the two above relations, we get $\gamma_1 \geq 2^{-(2k+1)}\gamma_0$. □

4.5 Algorithm for k-XOR at Low Densities

In this section, we will show that the k-XOR problem becomes easy to solve at densities $\Delta \leq \frac{1}{r^{0.5+\epsilon}}$. To start with, observe that the k-XOR problem becomes very easy to solve when the input matrix is square (i.e. $m = r$). This is because a random boolean square matrix is full rank with constant probability, in which case the planted k-XOR solution is the only linear dependence, and we can find it using Gaussian elimination in $\mathcal{O}(r^3)$ time. This algorithm also works when $m > r$; we can simply ignore the bottom $m - r$ rows. Below, we will provide an algorithm for k-XOR that works by repeatedly trying to reduce a more general instance to this case.

Theorem 4.26 (Algorithm for k-XOR)**.** *For any $k \in \mathbb{N}$, there is an algorithm that runs in time $\widetilde{\mathcal{O}}\left(r^k m^{3-k}\right)$ and solves the planted search k-XOR problem at any density with success probability $1 - o(1)$.*

Proof. Consider the following algorithm that is given input $A \in \mathbb{F}_2^{m \times r}$. It becomes trivial when $r < m/2$, but in that case the Gaussian elimination part can be applied directly to the instance to get the same results.

Algorithm $\mathcal{P}(A)$

1. Repeat the following steps $\left(\frac{4r}{m}\right)^k \log r$ times:
   1.1. Randomly choose $\frac{m}{2}$ columns of A to create a new $m \times \frac{m}{2}$ matrix B.
   1.2. Run Gaussian elimination on B to find any linear relationships between its columns.
   1.3. If the previous step returns a linear dependence of size exactly k, return it.
2. Return $\perp$.

Observe that the planted solution is preserved in B in any given repetition with probability

$$\frac{\binom{r-k}{m/2-k}}{\binom{r}{m/2}} = \frac{(r-k)!\,(m/2)!}{(m/2-k)!\,r!} = \frac{(m/2)(m/2-1)\cdots(m/2-k+1)}{r\,(r-1)\cdots(r-k+1)} > \left(\frac{m/2-k+1}{r-k+1}\right)^k > \left(\frac{m}{4r}\right)^k$$

Since we run $(\frac{4r}{m})^k \log r$ repetitions, the probability that B contains the original k-XOR solution at least once is greater than

$$1 - \left(1 - \left(\frac{m}{4r}\right)^k\right)^{\left(\frac{4r}{m}\right)^k \log r} > 1 - \frac{1}{r}$$

If we condition on B having the k-XOR solution preserved, it is easy to see that the induced distribution on B is the planted distribution on $\mathbb{F}_2^{m \times \frac{m}{2}}$. Since the number of rows is much larger than the number of columns, the planted solution is the only linear dependence in such a matrix with probability $1 - \mathrm{negl}(m)$. Therefore, Gaussian elimination returns a k-XOR solution and we return the correct answer 1 with probability greater than $1 - o(1)$.

The runtime of Gaussian elimination is $\mathcal{O}(m^3)$. The total runtime of the above algorithm is therefore $\mathcal{O}\left(\left(\frac{r}{m}\right)^k m^3 \log r\right) = \widetilde{\mathcal{O}}(r^k\, m^{3-k})$. □

Corollary 4.27. *For any $k \in \mathbb{N}$ and constant $\epsilon > \dfrac{3}{k-3}$, there is an algorithm for the planted search k-XOR problem that runs in time $r^{\lceil k/2 \rceil - \Omega(1)}$.*

Proof. Since $m = \dfrac{k \log r}{\Delta}$, the runtime of $\mathcal{P}$ is $\widetilde{\mathcal{O}}\left(\dfrac{r^k\, \Delta^{k-3}}{k^{k-3}}\right) = \widetilde{\mathcal{O}}_k(r^{(k-3)/2}) = o(r^{\lceil k/2 \rceil - \Omega(1)})$. □

5 Hardness Amplification

In this section, we propose a success amplification procedure for search k-SUM outside the dense regime. The procedure amplifies the success probability of any algorithm that solves planted search k-SUM with probability $\Omega(1/\mathrm{polylog}(r))$ to $1 - o(1/\log r)$, at the cost of increasing its runtime by a polylogarithmic factor. In Section 5.1, we prove this result for any abelian group up to density $(1-\epsilon)$ for some $\epsilon = o(1)$. In Sections 5.2 and 5.3, we show how to extend our result up to density 1 for the special case of k-SUM over integers modulo a power of 2, and vector k-SUM (of which k-XOR is a special case). Note that while G is the group ensemble, and G is the group specific to the instance size r, for simplicity of notation, we will use G to denote also the specific group and trust that this will not cause confusion.

5.1 Hardness Amplification for General Groups

We will consider the k-SUM problem over some arbitrary abelian group ensemble $\mathsf{G} = \left\{G^{(r)}\right\}$. Our input will be an array of length r containing elements of $G^{(r)}$. At density Δ, we have the relation $\Delta \log |G^{(r)}| = k \log r$. As long as $1 > \Delta = \Omega\left(\frac{1}{\mathrm{polylog}(r)}\right)$ and $k = O(1)$, this implies that $\log |G^{(r)}| = \Theta(\mathrm{polylog}(r))$. Below, for any instance M and set $S \subseteq [r]$, we will denote by $M[S]$ the sub-array of M indexed by the elements in S.

Recall the distributions D_0 and D_1 defined in Section 3 for any r (specified by context) as follows:

1. To sample from D_0, we simply choose each element of the array independently and uniformly at random from $G^{(r)}$.
2. To sample from D_1, we do the same thing and then replace a random entry with the negated sum of $k-1$ randomly chosen other entries[10].

Given an arbitrary instance M sampled from D_1, the planted search k-SUM problem asks for a k-tuple of indices S such that:

$$\sum_{i \in S} M_i = 0 \tag{17}$$

Theorem 5.1 (Hardness Amplification for General Groups)**.** *Consider any $k \geq 3$, $\alpha \geq 1$, and a group ensemble G with density $\Delta_k(\mathsf{G}) = \Delta$ such that:*

$$\Omega\left(\frac{1}{\mathrm{polylog}(r)}\right) \leq \Delta \leq \frac{k \log r}{k \log r + (\alpha+1) \log\log r} < 1.$$

[10] This is equivalent to replacing the smallest entry with the negated sum of the other entries, as defined in Section 3.

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 G with success probability $\gamma = \Omega\left(\frac{1}{\log^{\alpha} r}\right)$. Then, there is an algorithm that runs in time $\widetilde{\mathcal{O}}\left(T \cdot (k/\gamma^2)^k\right)$, and solves the planted search k-SUM problem over G with success probability $\left(1 - o\left(\frac{1}{\log r}\right)\right)$.

The proof of this statement is postponed to first build an appropriate framework. 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 $\gamma = \Theta\left(\frac{1}{\log^{\alpha} r}\right)$ 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.

Fix a group ensemble G satisfying the conditions in the theorem. Fix some r, and denote the corresponding group $G^{(r)} \in \mathsf{G}$ simply by G. For our construction, we will restrict our attention to arrays which have exactly one k-SUM solution.

Definition 5.2 (Permissible Array)**.** *We call an array that having exactly one k-*SUM *solution* permissible*. Formally, an array $M \in \mathsf{G}^r$ is permissible if and only if there exists a unique subset $\kappa \subset [r]$ such that $|\kappa| = k$ and $\sum_{i \in \kappa} M[i] = 0$. Define D_{perm} to be the distribution D_1 conditioned on the sampled array being permissible.*

Remark 5.3. *Since it is nontrivial to check if a given array is permissible, there is no obvious efficient sampling process for D_{perm}. We only use this distribution to make theoretical arguments.*

Lemma 5.4. *The following is true.*

1. $\Pr_{M \sim D_1}[M \text{ is permissible}] = 1 - o(\gamma)$.
2. *D_{perm} is uniform on its support.*

Proof. Since M is sampled from D_1, there is always the solution we explicitly planted – call this set S. The probability of another k-SUM solution among the $r-1$ original entries is at most $\binom{r-1}{k}/|\mathsf{G}| < r^{k\,(1-1/\Delta)}$. The new entry creates an extraneous solution exactly when there are some other $k-1$ entries which have the same sum as the unchanged entries in S; this happens with probability at most $\frac{\binom{r-1}{k-1}}{|\mathsf{G}|} < r^{k\,(1-1/\Delta)-1} < \frac{1}{r} = o(\gamma)$. Therefore, we conclude that M is permissible except with probability $\mathcal{O}(r^{k\,(1-1/\Delta)})$. The following calculation shows that our upper bound on Δ implies that the probability of multiple solutions is $o(\gamma)$.

$$r^{k\,(1-\frac{1}{\Delta})} \leq r^{k\left(1-\frac{k\log r + (\alpha+1)\log\log r}{k\log r}\right)} = 2^{k\log r\left(\frac{-(\alpha+1)\log\log r}{k\log r}\right)} = \left(\frac{1}{\log r}\right)^{\alpha+1} = o(\gamma).$$

The second statement follows from Lemma 4.3 and the fact that each permissible array has exactly one k-SUM solution. □

5.1.1 Solution Obfuscation

Our first step is creating an intermediate algorithm $\mathcal{A}^{\text{obf}}$ that runs in $\widetilde{\mathcal{O}}(T)$ 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 5.1; it takes as input an array sampled from D_1 and returns an array of size k which contains the 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 $r/2$ entries and always fails in the $1 - 1/2^k$ fraction of the inputs where the solution involves the other entries). To deal with this issue, we *anonymize* the solution by applying an arbitrary permutation to M.

The original oracle can also only look for solutions where the entries have some particular property (e.g. it can only find solutions where all the elements in S belong to some subgroup of G and always fails for the

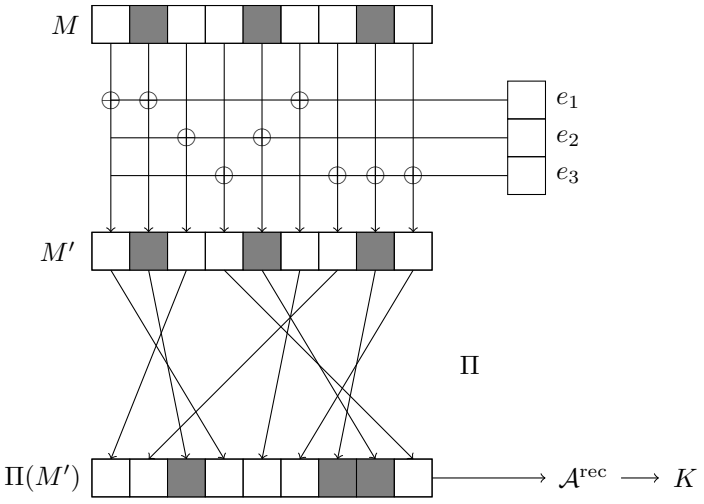

Figure 5: Illustration of the obfuscation procedure $\mathcal{A}^{\text{obf}}$. The shaded cells indicate the position of the solution. We first add a random e_i to each element (represented by $\oplus$) of the input, and permute it before giving it to $\mathcal{A}^{\text{rec}}$.

rest of the input space). To fix this issue, we *randomize* the solution by adding to it k random elements from G that sum to 0. This transforms the solution into a random set of k elements adding up to 0. We cannot do this directly because we do not know where the solutions are. Instead, we pick a set of k random elements that sum to 0, and add a random element from this set to each element in M. With some small constant probability, we will then end up with the desired result of a different one of these added to each element of the solution.

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

1. Sample a random permutation Π of $[r]$.
2. Sample $e_i \xleftarrow{\$} G$ uniformly at random for $1 \leq i < k$.
3. Set $e_k \leftarrow -\sum_{i=1}^{k-1} e_i$.
4. Repeat $k^k \cdot \log r$ times :
   - 4.1. Create a copy of the array $M' \leftarrow M$.
   - 4.2. For each index $j \in \{1, 2, \ldots, r\}$:
     - 4.2.1. Sample $i \xleftarrow{\$} [k]$ uniformly at random
     - 4.2.2. Set $M'[j] \leftarrow M[j] + e_i$.
   - 4.3. Let $K \leftarrow \mathcal{A}^{\text{rec}}(\Pi(M'))$ and $K' \leftarrow \Pi^{-1}(K)$
   - 4.4. If $\sum_{i \in K'} M[i] = 0$, return K'.
5. If a solution has not been found yet, return $\perp$.

Assuming we can sample elements of the underlying group and perform group operations in $\mathcal{O}(\log |\mathsf{G}|)$ time, the runtime of $\mathcal{A}^{\text{obf}}$ is $\mathcal{O}(k^k \log r(r \log |\mathsf{G}| + T)) = \widetilde{\mathcal{O}}(k^k \cdot T)$, since $\log |\mathsf{G}| = \Theta(\text{polylog}(r))$ and $T = \Omega(r)$.

Lemma 5.5. *Let S be any set of indices in $[r]$ of size k. Let V be any array of elements of G of length k such that $\sum_{j=1}^{k} V[j] = 0$. Recall that γ is the success probability of $\mathcal{A}^{\text{rec}}$ Then, for large enough r,*

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

Proof. Note that the distribution D_1 may be equivalently sampled as follows:

Equivalent Sampling Algorithm for D_1

1. Sample $M \leftarrow D_0$
2. Sample uniformly at random a set S' of indices in $[r]$ of size k, and an array V' of elements of G of size k such that $\sum_{j=1}^{k} V'[j] = 0$
3. Replace the entries of M at the indices given by S' with the respective elements of V'
4. Output M

For any S and V, denote by $D_1^{S,V}$ the distribution that follows from fixing these to be the S' and V', respectively, in the sampling process above (rather than sampling them at random). Note that $D_1^{S,V}$ may not be the same as sampling an M from D_1 conditioned on $M[S] = V$. Nevertheless, we have the following.

Claim 5.5.1. *For any S and V as in the statement of the lemma, the following two distributions on M are identical:*

1. *$M \sim D_1$ conditioned on $((M \text{ is permissible}) \wedge (M[S] = V))$.*
2. *$M \sim D_1^{S,V}$ conditioned on $(M \text{ is permissible})$.*

Proof of Claim 5.5.1. M being permissible implies that M has exactly one k-SUM solution. By definition of S and V, the set S is a k-SUM solution. Further, in the above sampling procedure for D_1, the set S' chosen there is also always a solution. So conditioning D_1 on having exactly one solution and also satisfying $M[S] = V$ implies that $S' = S$ and $V' = V$ in the sampling process. So the sampling process for this distribution is to set $S' = S$ and $V' = V$, and then condition on the resulting M having exactly one solution. But this is also exactly the procedure of sampling $D_1^{S,V}$ conditioned on it having exactly one solution. This proves the claim. □

Using Claim 5.5.1 and the definition of D_{perm}, we can write the quantity we want to bound for the lemma as follows:

$$\begin{aligned}\Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S] = V\right] &= \Pr_{M \sim D_1}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S] = V \wedge M \text{ is permissible}\right] \\ &= \Pr_{M \sim D_1^{S,V}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M \text{ is permissible}\right] \qquad (18)\end{aligned}$$

Next we show that the probability that M sampled from $D_1^{S,V}$ is not permissible is very small. So it is sufficient to bound the above probability without the conditioning on M's permissibility.

Claim 5.5.2. *For any S and V as in the statement of the lemma,*

$$\Pr_{M \sim D_1^{S,V}}\left[M \text{ is not permissible}\right] \leq o(\gamma).$$

Proof of Claim 5.5.2. M is not permissible iff there is an additional k-SUM solution apart from S. The probability that there exists another solution is at most $\binom{r}{k} \cdot |G|^{-1} = o(\gamma)$, as argued for Lemma 5.4. □

Claim 5.5.3. *For any S and V as in the statement of the lemma,*

$$\Pr_{M \sim D_1^{S,V}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds}\right] \geq \gamma - o(\gamma).$$

Proof of Claim 5.5.3. Recall that A^{obf} proceeds by selecting a random permutation Π and a random array E of k elements $(e_1, \ldots, e_k)$ of G that sum to 0. Denote by $\Pi(S)$ the set that results from applying Π to each index contained in S. $A^{\text{obf}}(M)$ definitely succeeds if both of the following events happen in at least one of the iterations of step 4 of $A^{\text{obf}}(M)$:

1. The element e_i of E added to $M[j]$ for each $j \in S$ is distinct.

2. $\mathcal{A}_{\text{rec}}(\Pi(M'))$ returns $\Pi(S)$

In any iteration, the first event above happens with probability $(k!)/k^k > 1/k^k$, independently of M, S, and V. So the probability that it never happens in all $k^k \cdot \log(r)$ iterations is $\mathcal{O}\left(\exp\left(-\log(r)\right)\right) = o(\gamma)$.

Suppose the first event does happen. Denote by $\widehat{M} = \Pi(M')$ the input provided to $\mathcal{A}_{rec}$ in the first iteration in which this event happens. For simplicity, suppose that the array E gets added elementwise to V in M – that is, for the least $j \in S$, the element added to $M[j]$ is e_1, and so on. Observe that the overall sampling procedure for $\widehat{M}$ can be equivalently described as follows:

1. Sample $M \leftarrow D_1^{S,V}$. M is a uniformly random array subject to the condition $M[S] = V$
2. Add random elements of E to elements of M to get M', subject to the above condition. M' is now a uniformly random array subject to the condition $M[S] = V + E$
3. Apply Π to M' to get $\widehat{M}$. $\widehat{M}$ is a uniformly random array subject to the condition $\widehat{M}[\Pi(S)] = \Pi_S(V + E)$, where Π_S permutes the elements of the array $(V + E)$ according to match the change in the relative ordering of the indices in S following the application of Π.

As E was chosen as a uniformly random array of elements that sum to 0, and V is an array of elements that sum to 0, the sum $(V + E)$ is also a uniformly random array of elements that sum to 0. As Π is a uniformly random permutation, $\Pi(S)$ is a uniformly random set of indices of size k. Thus, if M is sample from $D_1^{S,V}$ for any S and V as in the statement of the lemma, $\widehat{M}$ is distributed according to D_1.

By our hypothesis, given a sample $\widehat{M}$ from D_1, the algorithm $\mathcal{A}_{rec}(\widehat{M})$ outputs a k-SUM solution with probability γ. $\Pi(S)$ is always a solution of $\widehat{M}$ (under our current conditioning). The probability that there exists another solution is at most $o(\gamma)$, as argued earlier. Thus, the probability, when M is sampled from $D_1^{S,V}$ and conditioning on the first event above happening, that $\mathcal{A}_{rec}(\widehat{M})$ outputs $\Pi(S)$ is at least $\gamma - o(\gamma)$.

By the union bound, both the events above together happen with probability at least $(\gamma - o(\gamma))$, which proves the claim. □

Now, putting together Claims 5.5.2 and 5.5.3 and (18) gives us the following statement,

$$\Pr_{M \sim D_{perm}}\left[\mathcal{A}^{\text{obf}}(M) \text{ succeeds} \mid M[S] = V\right] \geq \gamma - o(\gamma) \qquad \square$$

5.1.2 Success Amplification

Our next step is amplifying the success probability to almost 1. The main idea is replacing many of the entries of M with random elements from G, and running $\mathcal{A}^{\text{obf}}$ on the result. If none of the entries in the solution get replaced, we end up with an almost random array sampled from $D_1^{S,V}$. Repeating this procedure enough times and utilizing the guarantees provided by $\mathcal{A}^{\text{obf}}$, we can amplify the success probability significantly. 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 G 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.

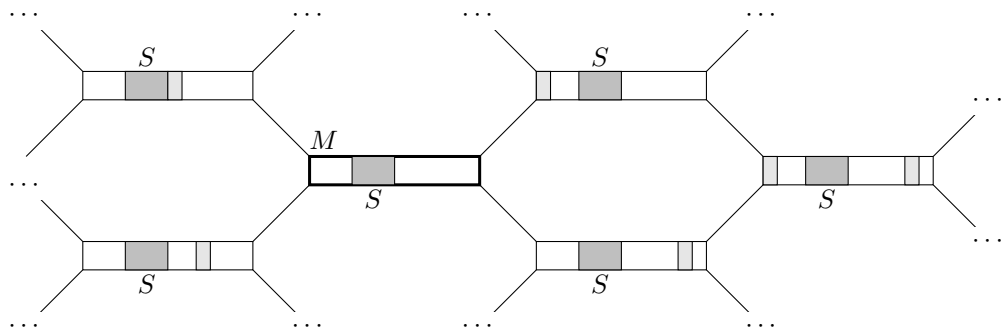

Figure 6: The Hamming graph representing the inner loop of $\mathcal{A}^{\text{amp}}(M)$. Each iteration resamples a random entry of M and corresponds to taking a random step in the graph, assuming this does not destroy the solution S. Intuitively, the oracle cannot consistently fail on all the vertices 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.

It is easy to see that the algorithm never returns a wrong answer. Note that the inner loop takes $\mathcal{O}(\log|\mathsf{G}| + \log r)$ time if we assume that sampling from a set takes time logarithmic in the size of that set. The outer loop takes $\widetilde{\mathcal{O}}(k^k \cdot T) + \widetilde{\mathcal{O}}\left(r \log|\mathsf{G}| \log\frac{1}{\gamma} + r\log|\mathsf{G}|\right)$ time since the runtime of $\mathcal{A}^{\text{obf}}$ is $\widetilde{\mathcal{O}}(k^k \cdot T)$ (recall that T was defined as the runtime of $\mathcal{A}^{\text{rec}}$). Since $\log|\mathsf{G}| = \mathcal{O}(\text{polylog}(r))$, $\frac{1}{\gamma} = \mathcal{O}(\text{polylog}(r))$ and $T = \Omega(r)$, the expression simplifies to $\widetilde{\mathcal{O}}(k^k T)$. The runtime of the whole algorithm is therefore $\widetilde{\mathcal{O}}\left(\frac{\log r}{\gamma^{2k+2}}\right) \cdot \widetilde{\mathcal{O}}(k^k T) = \widetilde{\mathcal{O}}\left(T \cdot (k/\gamma^2)^k\right)$, as desired. To analyze the success probability of $\mathcal{A}^{\text{amp}}$, we start with a couple of definitions.

Definition 5.6 (Correspondence Graph)**.** *For a permissible array M with the k-SUM solution $S \subset [r]$, we define its* correspondence graph G_M *as follows: create a vertex for every array $M' \in \mathsf{G}^r$ such that $M[S] = M'[S]$. Two vertices are connected by an edge if and only if the corresponding arrays only differ in one index.*

Lemma 5.7. *G_M is isomorphic to the Hamming graph $H(r-k, |\mathsf{G}|)$ (see Definition 2.8).*

Proof. We are allowed to change $r-k$ entries of M, and there are $|\mathsf{G}|$ possibilities for each of those indices. The isomorphism follows from definition. □

Lemma 5.8. *The fraction of vertices corresponding to arrays which are not permissible is $o(\gamma)$ in any correspondence graph G_M.*

Proof. We will show that for any permissible M, if we choose the $r-k$ entries outside its k-SUM solution uniformly at random from G, we end up with a permissible array with probability $1 - o(\gamma)$. This is equivalent to the lemma statement.

The proof of Lemma 5.4 establishes this result almost directly. With very high probability, there will be no new solutions other than the k fixed entries. We can bound the probability of a new solution by

$$\frac{1}{|\mathsf{G}|}\left(\binom{r-k}{k}\binom{k}{0} + \binom{r-k}{k-1}\binom{k}{1} + \cdots \binom{r-k}{1}\binom{k}{k-1}\right)$$

Since k is a constant, this sum is $\mathcal{O}\left(r^{k(1-1/\Delta)}\right) = o(\gamma)$ assuming $\Delta \le k\log r/(k\log r + (\alpha+1)\log\log r)$. □

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 5.9. *For a permissible array 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 5.10. *The algebraic expansion of G_M^* is $((r-k)(|\mathsf{G}|-1)-|\mathsf{G}|)^{r\log\frac{1}{\gamma}}$.*

Proof. By Lemmas 2.9 and 5.7, we know that for G_M, the highest eigenvalue is $(r-k)(|\mathsf{G}|-1)$, the second highest eigenvalue is $(r-k)(|\mathsf{G}|-1)-|\mathsf{G}|$, 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 $(r-k)(|\mathsf{G}|-1)-|\mathsf{G}| > r-k$ for any $r > k$, the lemma follows. □

Lemma 5.11. *Each vertex of G_M^* has degree $((r-k)(|\mathsf{G}|-1))^{r\log\frac{1}{\gamma}}$*

Proof. Lemma 5.7 implies that G_M is a regular graph of degree $d=(r-k)(|\mathsf{G}|-1)$. Therefore, starting from any vertex v, there are exactly d^l paths of length l. By definition, this implies G_M^* is $((r-k)(|\mathsf{G}|-1))^{r\log\frac{1}{\gamma}}$-regular. □

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

Definition 5.12 (Good Vertices)**.** *We call an array X* good *if it is permissible and $\mathcal{A}^{\mathrm{obf}}$ has at least a $\frac{\gamma}{4}$ success probability on it.*

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

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

Lemma 5.13. *Let M be a large enough permissible array. 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 does not exceed $\frac{\gamma}{8}$. The rest of the vertices exactly correspond to the support of D_{perm} which has the same k-SUM solution in the same position as M. Since D_{perm} is uniform on its support (Lemma 5.4), 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-SUM solution in the same place as M. Lemma 5.5 now yields

$$\mathop{\mathbb{E}}_{M'\sim G_M}[\Pr[\mathcal{A}^{\mathrm{obf}}(M')\,\text{succeeds}] \mid M' \text{ is permissible}] \geq \frac{\gamma}{2} \tag{19}$$

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 have 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:

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

This expected value is maximized when all the good vertices have success probability 1 and all the other permissible arrays 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. (19), and hence we are done. □

Definition 5.14 (Bad Vertices)**.** *Let M be a permissible array. 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 5.15. *Let M be a large enough permissible array. The fraction of bad vertices in G^*_M is $o\left(\frac{1}{\log r}\right)$.*

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 5.11 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 2.7) now implies

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

The number of edges between S and T is at most $\frac{d\gamma|S|}{16}$ since each bad vertex, by Definition 5.14, has at most $\frac{d\gamma}{16}$ edges connecting to a good vertex. Therefore,

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

Since T is the set of good vertices, Lemma 5.13 implies $\frac{|T|}{|V|} \ge \frac{\gamma}{8}$. Therefore,

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

We now plug in the values of λ and d from Lemmas 5.10 and 5.11 respectively to get

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

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

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

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

We are now ready to conclude our second step.

Lemma 5.16. *The average-case success probability of $\mathcal{A}^{\text{amp}}$ is $1 - o\left(\frac{1}{\log r}\right)$. This algorithm works at any $\Omega\left(\frac{1}{\text{polylog}(r)}\right)$ density $\Delta \le \frac{k\log r}{k\log r + (\alpha+1)\log\log r}$ for large enough r.*

Proof. With probability $1-o(\gamma) = 1-o(1/\log r)$, the input array M is permissible (see Lemma 5.4). In that case, M has a unique solution. Lemma 5.15 tells us that among all arrays that contain this exact solution at this exact position, only a $o(1/\log r)$ fraction can be *bad*. So we can assume with probability $1-o(1/\log r)$ 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 $r\log\frac{1}{\gamma}$ replacements before calling $\mathcal{A}^{\text{obf}}$. The original solution is preserved if we choose one of the $r-k$ entries not in the solution at every step. This happens with probability $(1-\frac{k}{r})^{r\log\frac{1}{\gamma}} > \exp\left(-2k\log\frac{1}{\gamma}\right) = \gamma^{2k}$, 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 which is permissible and not a bad vertex. Observe that different iterations of the outer loop are independent once we condition on the input array. 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

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

5.2 Hardness Amplification for Vector k-SUM at Density $\Delta \le 1$

The statement of Theorem 5.1 can be further strengthened if the underlying group ensemble has some extra structure. In some special cases, we can show an analogous results for all densities $\Delta \le 1$. In this subsection, we will handle the special case where G is $\mathbb{Z}_q^m$ for some fixed integer q, and addition is defined as pointwise addition modulo q. Note that any element of $\mathbb{Z}_q^m$ can be represented by a vector of size m, hence we will represent the input as an $m \times r$ matrix whose columns represent the elements. Each entry of this matrix will be in $\mathbb{Z}_q$. At density Δ, we have the relation $m\Delta\log q = k\log r$. As long as $1 \ge \Delta = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$, this implies that $m = \frac{k\log r}{\Delta\log q} = \Theta(\text{polylog}(r))$. The k-XOR problem is the special case of this with $q=2$.

Theorem 5.17 (Hardness Amplification for Vector k-SUM)**.** *For $k \ge 3$ and density $\Omega\left(\frac{1}{\text{polylog}(r)}\right) \le \Delta \le 1$, suppose there exists an algorithm $\mathcal{A}^{\text{rec}}$ that runs in time $T(r) = \Omega(r)$, and solves the planted search vector k-SUM problem at density Δ (k-SUM over $\mathsf{G}^{(\Delta)}_{\text{VECTOR-}(q,k)\text{-SUM}}$) with success probability $\gamma = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$. Then, there is an algorithm that runs in time $\widetilde{\mathcal{O}}\left(T\cdot(k/\gamma^2)^k\right)$, and solves the same problem with success probability $\left(1-o\left(\frac{1}{\log r}\right)\right)$.*

Proof. As before, we will assume w.l.o.g. that $\gamma = \Theta\left(\frac{1}{\log^\alpha r}\right)$ for some constant $\alpha > 1$. Let us define,

$$\Delta_0 := \frac{k\log r}{k\log r + (\alpha+1)\log\log r}$$

We have already proven this result for densities $\Delta \le \Delta_0$ in Section 5.1. To complete the proof, we will now consider the other case and assume that $\Delta \in (\Delta_0, 1]$.

Note that this implies that permissible matrices may no longer take up a $1-o(\gamma)$ fraction of the input space, and we can no longer pretend the input is permissible without a significant loss in success probability anymore.

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 vector k-SUM, we can argue by

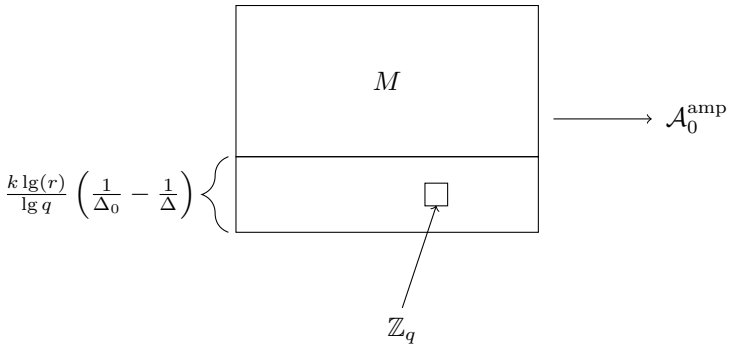

Figure 7: Illustration of how to lift the hardness amplification to all densities $\Delta \leq 1$ for the case of vector k-SUM, with k-XOR being a special case of $q = 2$. Here, a total of $\frac{k\lg(r)}{\lg q}\left(\frac{1}{\Delta_0}-\frac{1}{\Delta}\right)$ rows are added to the solution, where each entry is sampled i.i.d. from $\mathbb{Z}_q$, and the result is given to $\mathcal{A}_0^{\text{amp}}$.

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,

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

We can therefore apply Markov's inequality (Lemma 2.2) to conclude that with probability at least $1/2$, there are less than 6 vector k-SUM 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 $\mathcal{O}(mr) + T = \widetilde{\mathcal{O}}(T)$.

We now use our success amplification procedure described in Section 5.1 to obtain an average-case recovery algorithm $\mathcal{A}_0^{\text{amp}}$ with runtime $\widetilde{\mathcal{O}}(T)$ that solves density Δ_0 instances with probability $1 - o(1/\log r)$.

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 (see also Fig. 7 for an illustration).

Algorithm $\mathcal{B}(M)$

1. Define $t := (\log r)^{\alpha+2}$
2. Repeat t times:
   - 2.1. Add $\frac{k\log(r)}{\log q}\left(\frac{1}{\Delta_0}-\frac{1}{\Delta}\right)$ rows to M. Each new entry is chosen uniformly at random from $\mathbb{Z}_q$.
   - 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 $q^{\frac{k\log(r)}{\log q}\left(\frac{1}{\Delta}-\frac{1}{\Delta_0}\right)}$ since the probability of the original solution being preserved drops by a factor of q for each additional row added. 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.

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

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

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

We now substitute in $\Delta_0 = \frac{k \log r}{k \log r + (\alpha+1) \log\log r}$ to get,

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

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

$$\begin{aligned} &= 1 - \mathsf{exp}\left(-(\log r)^{\alpha+2} \cdot (\log r)^{-(\alpha+1)}\right) \\ &= 1 - \mathsf{exp}\left(-\log(r)\right) = 1 - o\left(\frac{1}{\log r}\right) \end{aligned}$$

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 $1 - o(1/\log r)$. Taking a union bound, we conclude that with probability $1 - o(1/\log r)$, we call $\mathcal{A}_0^{\text{amp}}$ on a matrix with a vector k-SUM 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 $\widetilde{\mathcal{O}}(T)$ and $t = \mathcal{O}(\text{polylog}(r))$, our algorithm also runs in $\widetilde{\mathcal{O}}(T)$ time. □

Corollary 5.18. *If the planted search k-XOR problem at density $\Omega\left(\frac{1}{\text{polylog}(r)}\right) \leq \Delta \leq 1$ is hard to solve with probability $1 - o\left(\frac{1}{\log r}\right)$ in time $\widetilde{\mathcal{O}}(T)$, it is also hard to solve with probability $\gamma = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$ in time $\widetilde{\mathcal{O}}(T \cdot (\gamma^2/k)^k)$.*

Proof. This follows from setting $q = 2$ in Theorem 5.17 and taking the contrapositive. □

Corollary 5.19 (Strong Hardness of k-XOR)**.** *For any constant $k \geq 3$, the average-case k-XOR conjecture (Conjecture 3.6) implies that any algorithm that, given r uniformly random vectors from $\mathbb{F}_2^m$ for $m = k \log r$, can find a set of k that sum to 0 with probability $\Omega(1/\text{polylog}(r))$ takes time at least $r^{\lceil k/2 \rceil - o(1)}$.*

Proof. This follows from setting $\Delta = 1$ and $k = O(1)$ in Corollary 5.18. □

5.3 Hardness Amplification for Modular k-SUM at Density $\Delta \leq 1$

In this subsection, we will prove an analogue of Theorem 5.17 for the special case where G is $\mathbb{Z}_{2^m}$ (see Eq. (4)). At density Δ, we have the relation $m\Delta = k \log r$. Note that the input will now be an array M of size r where each entry is between 0 and $2^m - 1$.

Theorem 5.20 (Hardness Amplification for Modular k-SUM)**.** *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 $\mathsf{G}_{k\text{-}SUM}^{(\Delta)}$ with success probability $\gamma = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$. Then, there is an algorithm that runs in time $\widetilde{\mathcal{O}}\left(T \cdot (k/\gamma^2)^k\right)$, and solves the same problem with success probability $\left(1 - o\left(\frac{1}{\log r}\right)\right)$.*

Proof. We follow the proof of Theorem 5.17 very closely. We will again assume w.l.o.g. that $\gamma = \Theta\left(\frac{1}{\log^\alpha r}\right)$ for some constant $\alpha > 1$. We define,

$$\Delta_0 := \frac{k \log r}{k \log r + (\alpha + 1) \log \log r}$$

Since we already proved this result for densities $\Delta \leq \Delta_0$ in Section 5.1, we only consider the case $\Delta \in (\Delta_0, 1]$.

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 reduces each entry modulo $2^{k \log r/\Delta}$ 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 entries of M_0 are independently sampled uniformly from $\mathbb{Z}_{2^{k \log r/\Delta_0}}$ and the latter modulus is a multiple of the new modulus, 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, since (in the following $a \mid b$ means that "a divides b"),

$$\begin{aligned}
&\sum_{i \in \kappa} M_0[i] = 0 \\
&\quad \implies 2^{k \log r/\Delta_0} \mid \sum_{i \in \kappa} M_0[i] && \text{By definition of } + \text{ in } M_0 \\
&\quad \implies 2^{k \log r/\Delta} \mid \sum_{i \in \kappa} M_0[i] && \text{Since } \Delta_0 < \Delta \\
&\quad \implies 2^{k \log r/\Delta} \mid \sum_{i \in \kappa} (M_0[i] \mod 2^{k \log r/\Delta}) \\
&\quad \implies 2^{k \log r/\Delta} \mid \sum_{i \in \kappa} M[i] && \text{By design of } \mathcal{A}_0 \\
&\quad \implies \sum_{i \in \kappa} M[i] = 0 && \text{By definition of } + \text{ in } M
\end{aligned}$$

If M has c solutions of k-SUM, we can argue by symmetry that they are all equally likely to be a solution for M_0. Since M has density ≤ 1, the expected value of c is less than 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-SUM solutions for M. As before, this implies that the success probability of $\mathcal{A}_0$ is at least $\frac{\gamma}{10} = \Omega\left(\frac{1}{\text{polylog}(r)}\right)$. The runtime of $\mathcal{A}_0$ is clearly $\mathcal{O}(mr) + T = \widetilde{\mathcal{O}}(T)$.

We now use our success amplification procedure described in Section 5.1 to obtain an average-case recovery algorithm $\mathcal{A}_0^{\text{amp}}$ with runtime $\widetilde{\mathcal{O}}(T)$ that solves density Δ_0 instances with probability $1 - o(1/\log r)$.

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 (see also Fig. 8 for an illustration).

Algorithm $\mathcal{B}(M)$

1. Define $t := (\log r)^{\alpha+2}$
2. Repeat t times:
   - 2.1. Initialize a new empty array M_0 of size r.
   - 2.2. For each $1 \leq i \leq r$:
     - 2.2.1. Sample β_i uniformly from $[2^{k \log r(1/\Delta_0 - 1/\Delta)}]$
     - 2.2.2. Set $M_0[i] := M[i] + \beta_i 2^{k \log r/\Delta}$
   - 2.3. Call $\mathcal{A}_0^{\text{amp}}$ on the new matrix M_0.
   - 2.4. If a solution was returned in the last step, check if that is also a solution for M. If so, return it.

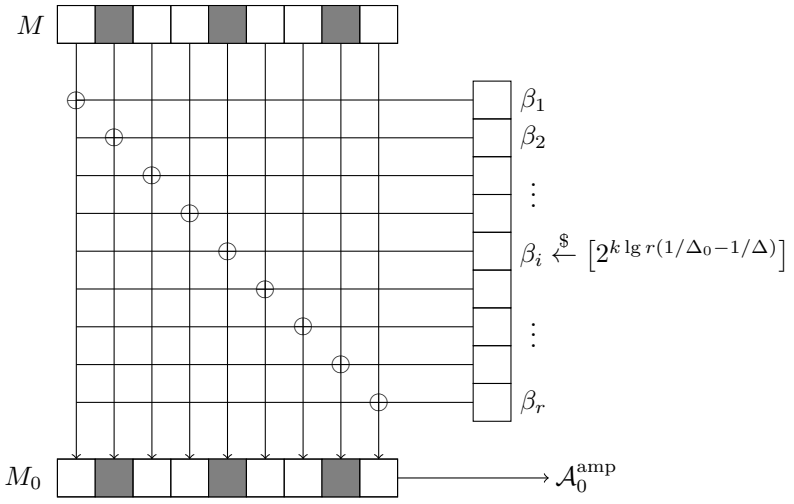

Figure 8: Illustration of how to lift the hardness amplification to all densitites $\Delta \leq 1$ for the case of modular k-SUM. We sample random β_i from an appropriate range and add to each entry $M[i]$ of the original array, and give the result is given to $\mathcal{A}_0^{\text{amp}}$.

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-SUM solution for M_0 is

$$\begin{aligned}
\Pr\left[\sum_{i\in S} M_0[i] = 0\right] &= \Pr\left[2^{k\log r/\Delta_0} \Big| \sum_{i\in S} M_0[i]\right] \\
&= \Pr\left[2^{k\log r/\Delta_0} \Big| \sum_{i\in S} \left(M[i] + \beta_i 2^{k\log r/\Delta}\right)\right] \\
&= \Pr\left[2^{k\log r/\Delta_0} \Big| \sum_{i\in S} M[i] + 2^{k\log r/\Delta} \sum_{i\in S} \beta_i\right]
\end{aligned}$$

Since S is a k-SUM solution in M, the first term in the above sum is divisible by $2^{k\log r/\Delta}$. We can therefore cancel out $2^{k\log r/\Delta}$ on both sides to get

$$\begin{aligned}
&= \Pr\left[2^{k\log r(1/\Delta_0 - 1/\Delta)} \Big| \left(\sum_{i\in S} M[i]\right) \Big/ 2^{k\log r/\Delta} + \sum_{i\in S} \beta_i\right] \\
&= \Pr\left[\sum_{i\in S} \beta_i \equiv -\left(\sum_{i\in S} M[i]\right) \Big/ 2^{k\log r/\Delta} \mod 2^{k\log r(1/\Delta_0 - 1/\Delta)}\right]
\end{aligned}$$

Since each β_i is uniformly random, so is their sum $\beta := \sum_{i\in S} \beta_i$. Denoting the right hand side of the above congruence by t and letting $x := 2^{k\log r(1/\Delta_0 - 1/\Delta)}$, we have

$$= \Pr_{\beta \xleftarrow{\$} [x]} [\beta \equiv t \mod x] = \frac{1}{x} = 2^{k\log r(1/\Delta - 1/\Delta_0)}.$$

As in the proof of Theorem 5.17, we can bound the probability p that at least one iteration of the inner loop ran $\mathcal{A}_0^{\text{amp}}$ on an array where S was a solution by

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

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 $1 - o(1/\log r)$. Taking a union bound, we conclude that with probability $1 - o(1/\log r)$, we call $\mathcal{A}_0^{\text{amp}}$ on an array with a k-SUM 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 $\widetilde{\mathcal{O}}(T)$ and $t = \mathcal{O}(\text{polylog}(r))$, our algorithm also runs in $\widetilde{\mathcal{O}}(T)$ time. □

Corollary 5.21 (Strong Hardness of k-SUM)**.** *For any constant $k \geq 3$, the average-case k-SUM conjecture (Conjecture 1.1) implies that any algorithm that, given r uniformly random integers from $[-r^k, r^k]$ can find a set of k that sum to 0 with probability $\Omega(1/\text{polylog}(r))$ takes time at least $r^{\lceil k/2 \rceil - o(1)}$.*

Proof. This follows from setting $\Delta = 1$ and $k = O(1)$ in Theorem 5.20, using the reduction of k-SUM to modular k-SUM presented in Theorem 4.5 in [DKK21], and taking the contrapositive. □

6 Implications to Public-Key Encryption

In this section, we propose a class of public-key bit encryption schemes based on the planted search 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 planted search k-XOR is hard.

6.1 Preliminaries

Definition 6.1 (Public Key Encryption)**.** *A* Public Key Encryption (PKE) *scheme for a message space $\mathcal{M}$ consists of PPT algorithms $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:*

- $\mathsf{KeyGen}(1^r) \to (pk, sk)$*: on input the unary representation of the security parameter r, generates a public key pk and a secret key sk.*
- $\mathsf{Enc}(pk, m) \to ct$*: on input a public key pk and a message $m \in \mathcal{M}$, outputs a ciphertext ct.*
- $\mathsf{Dec}(sk, ct) \to m$*: on input a secret key sk and a ciphertext ct, outputs a message $m \in \mathcal{M}$.*

The scheme should satisfy the following properties:

Correctness*. A scheme* PKE *is* correct *if there exists a negligible function* $\text{negl}(.)$ *such that for every security parameter r and message $\mu \in \mathcal{M}$:*

$$\Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, \mu)) = \mu] \geq 1 - \text{negl}(r)$$

where $(pk, sk) \leftarrow \mathsf{KeyGen}(1^r)$. The scheme is weakly *correct if the probability of correct decryption is bounded by $1 - o(1)$ instead of $1 - \text{negl}(r)$ above.*

CPA Security*. A scheme* PKE *is* IND-CPA secure *if for any PPT adversary* A *there exists a negligible function* negl *such that:*

$$\left| \Pr[\mathsf{PKEGame}^0_{\mathsf{A}(r)} = 1] - \Pr[\mathsf{PKEGame}^1_{\mathsf{A}(r)} = 1] \right| \leq \text{negl}(r)$$

where $\mathsf{PKEGame}^b_{\mathsf{A}(r)}$ is a game between an adversary A and a challenger C with a challenge bit b defined as follows:

- C *samples $(pk, sk) \leftarrow \mathsf{KeyGen}(1^r)$, and sends pk to* A.
- A *chooses $\mu_0, \mu_1 \in \mathcal{M}$ and sends them to $\mathcal{C}$.*

- C *computes* $ct \leftarrow \mathsf{Enc}(pk, \mu_b)$*, and sends* ct *to* A*.*
- *The adversary* A *outputs a bit* b' *which we define as the output of the game.*

The scheme is said to be weakly *IND-CPA secure if we replace* $\mathrm{negl}(r)$ *with* $o(1)$ *in the two conditions above.*

Remark 6.2. *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]) to get a full-fledged PKE scheme.*

Definition 6.3 (Search LPN Problem)**.** *An algorithm* $\mathcal{A}$ *is said to solve the* search Learning Parity with Noise (LPN) 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 $2^{\mathcal{O}(m/\log(m))}$. There is another algorithm by Esser, Kübler and May [EKM17] that runs in time $2^{\mathcal{O}(\eta m)}$ 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 6.4 (Decision LPN Problem)**.** *The* Decision Learning Parity with Noise (LPN) problem 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. $\langle X, y\rangle$*, where* $X \leftarrow \mathbb{F}_2^{r\times m}$*, and* $y \leftarrow \mathbb{F}_2^r$*.*

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

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

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 6.5 (Search-to-Decision LPN [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* $\mathcal{O}\left(\frac{T\, m \log m}{\epsilon^2}\right)$ *and solves the search LPN problem with probability* $\epsilon/4$*.*

Definition 6.6 (Hardness of LPN)**.** *We say that* η-noise search LPN is $T(m)$-hard *if any algorithm* $\mathcal{A}$ *that runs in time at most* $T(m)$ *has success probability at most* $o(1/T(m))$ *in solving the search LPN problem for secrets of size* m *with noise rate* $\Omega(\eta)$*.*

We can leverage the equivalence of Lemma 6.5 to translate hardness of LPN into a bound on the advantage of an algorithm that solves decision LPN.

Corollary 6.7. *If* η*-noise Search LPN is* $(T(m)^3 \cdot m \log m)$*-hard, then there is no algorithm that runs in time* $\mathcal{O}(T(m))$ *and solves the decision LPN problem with advantage* $\Omega(1/T(m))$*.*

Definition 6.8 (Hardness of Planted Search k-XOR)**.** *We say that planted search* k-XOR is $T(r)$-hard at density Δ *if any algorithm that runs in time at most* $T(r)$ *has a success probability at most* $(1 - \Omega(1/\log r))$ *in solving the planted search* k*-XOR problem at density* Δ*.*

Similarly, we put together Theorem 4.12 and Corollary 5.18 to translate mild hardness of planted search k-XOR (Definition 6.8) into a bound on the advantage of an algorithm that solves decision k-XOR with any constant advantage.

Corollary 6.9. *For any* $k = o(\log r/\log\log r)$*, if planted search* k*-XOR is* $T(r)$*-hard at density* Δ*, then any algorithm that runs in time* $\left(T(r)/r^2\right)$ *solves the decision-*k*-XOR problem at density* Δ *with advantage at most* $o(1)$*.*

6.2 Construction

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 planted k-XOR solution, $m(r)$ is the dimension of the vectors, and $\ell(r)$ is the number of repetitions in the cryptosystem we describe (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 planted search k-XOR is assumed hard. We can also alternatively trade off the density for the size of the public key. This allows us to obtain public keys of size $r^{1+o(1)}$ at noise rate $\eta = \Theta(1)$ from $2^{m^{0.5}}$-hardness, while the previous best-known construction of PKE from the same assumptions used a public key of size r^2. For given η, k, ℓ, m, the cryptosystem is defined as follows.

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 6.10. *Decryption succeeds with probability at least* $(1-\epsilon)$ *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 $i = 1 \dots \ell$. 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,

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

This gives a decryption error of at most ε whenever $\ell \ge 32\,(1-\eta)^{-2k}\ln(1/\varepsilon)$. 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, as per the usual definition of semantic security [GM82].

Lemma 6.11 (Indistinguishability of Cryptosystem)**.** *Suppose the following conditions are satisfied for some choice of η, k, ℓ, m (as functions of r), and T (as a function of m):*

(1) $k = o(\log(r)/\log\log(r))$ and $\ell = o(T(m))$.

(2) η-noise Search LPN is $(T(m)^3 \cdot m\log m)$-hard (Definition 6.6) for secrets of size m.

(3) Planted search k-XOR is $r^{k/2-o(1)}$-hard (Definition 6.8) at density $\frac{k\log r}{m}$.

Then any adversary that runs in time $\min\left(T(m), r^{k/2-2}\right)$ has an advantage $o(1)$ in distinguishing an encryption of 0 from an encryption of 1 in $PKE_{\eta,k,\ell,m}$.

Proof. We proceed using a hybrid argument (see Fig. 9) 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 our assumption about the hardness of planted search k-XOR and Corollary 6.9, any adversary that runs in time $r^{k/2-2}$ has advantage $o(1)$ in distinguishing $H_1^{(0)}$ from $H_0^{(0)}$ (respectively, $H_1^{(\ell)}$ from $H_0^{(\ell)}$).

Next, note that $H_0^{(i)}$ differs from $H_0^{(i+1)}$ only by having replaced one random vector with a vector of the form $s^\top pk + e^\top$ – this is exactly an instance of decision LPN. Thus, by our assumption about the hardness of LPN and Corollary 6.7 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 $\min\left(T(m), r^{k/2-2}\right)$, which proves the lemma. □

Using Lemma 6.11, we can identify various tradeoffs between the hardness assumed for LPN and the densities at which planted search k-XOR is assumed to be hard.

Theorem 6.12 (PKE from k-XOR and LPN)**.** *There are values of $\eta = \Theta(1)$, $k = \mathcal{O}(\log r)$, $m = r^{o(1)}$, 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 any 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 planted search k-XOR is $r^{k/2-o(1)}$-hard at every density $\frac{1}{\mathrm{polylog}(r)}$; or,

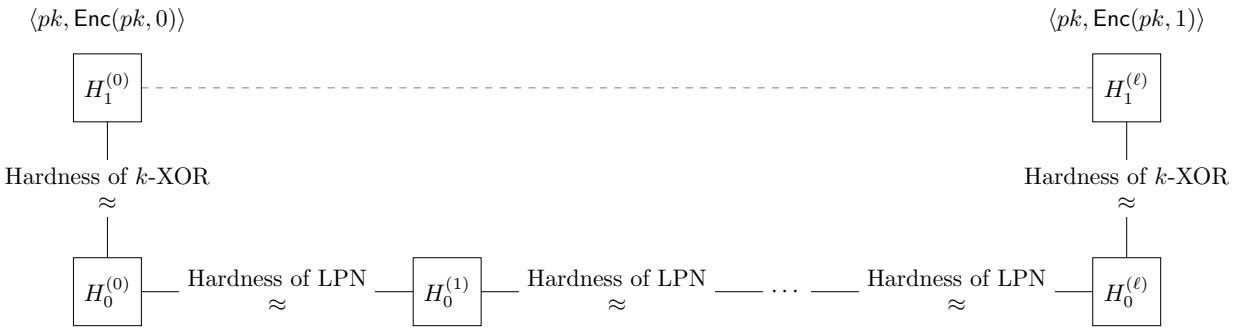

Figure 9: Structure of the hybrid argument. The distribution $H_b^{(i)}$ generates a public key pk using the distribution D_b, generates i vectors of the form $s^\top pk + e^\top$ and $\ell - i$ random vectors from $\mathbb{F}_2^r$. The distribution $H_1^{(0)}$ thus corresponds to an encryption of 0, while $H_1^{(\ell)}$ is an encryption of 1.

(3) Constant-noise LPN is $\mathrm{poly}(m)$*-hard and planted search* k*-XOR is* $r^{k/2-o(1)}$*-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 will always set $\alpha > 0$ so that k is super-constant. The constraint $\ell = \mathsf{exp}(k)\,\omega(\log r)$ is needed by Lemma 6.10 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 $\frac{k \log r}{m} = \epsilon\,(\log r)^{\alpha+1-\beta}$.

Now to show conclusion (2), 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 $T(m) \approx 2^{m^c/3} = 2^{(\log r)^{\beta c}/3}$ 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 $c \ge \frac{1}{2}$. Setting $c = \frac{1}{2}$, $\alpha = 1 - \delta$, and $\beta = 2 - \delta$ for a small constant $\delta > 0$ shows conclusion (1) of the theorem.

Suppose instead we allow the density to be sub-constant such that we may circumvent this lower bound. Again with $T(m) \approx 2^{m^c/3} = 2^{(\log r)^{\beta c}/3}$, set β such that $T(m) = r^{k/2}$. That is, $(k)\log(r)/2 \approx m^c/3$. By Lemma 6.11, the cryptosystem satisfies indistinguishability against $r^{k/2-2}$-time adversaries if we assume planted search-k-XOR is hard at density $\Delta = \frac{k \log r}{m} \approx \frac{2m^{c-1}}{3}$. Using the fact that $m = k\log(r)/\Delta$, this is:

$$\Delta = \left(\frac{2}{3\,(k\log(r))^{1-c}}\right)^{\frac{1}{c}} = \frac{1}{\mathrm{polylog}(r)},$$

for any constant α and β chosen to satisfy the above conditions, say $\alpha = 1$ and $\beta \approx 2/c$. Applying Lemma 6.11 with these parameters now gives conclusion (2).

Finally, to show conclusion (3), suppose instead that $T(m) = m^c$ for some $c = \omega(1)$ but $c \ll m^{0.01}$. In this case, we pick m such that $m^c = r^{k/2}$, and hence using $m = k\log(r)/\Delta$, we need hardness of k-XOR at some $\Delta \approx \frac{k \lg r}{r^{\frac{k}{2c}}}$. Now let $k = \min(\sqrt{c}, \sqrt{\log r})$, then we can choose $m = r^{o(1)}$ such that $\Delta = \frac{k \lg r}{r^{o(1)}}$, for which again, we know planted search k-XOR is hard by assumption. Applying Lemma 6.11 now gives the conclusion.

□

Observe that the size of the public key size is $m \cdot r$ bits. The encryption time is $\mathcal{O}(\ell \cdot m \cdot r)$, while decryption takes time $\mathcal{O}(\ell \cdot r)$. In all of the instantiations in Theorem 6.12, the above quantities are at most $r^{1+o(1)}$.

We note that Theorem 6.12(1) was also 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. Their cryptosystem has a public key of size of r^2, while the above cryptosystem has a public key of size $mr = r^{1+o(1)}$, which is substantially smaller for large r.

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.

[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.

[BDK+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Annual Cryptology Conference*, pages 1–20. Springer, 2011.

[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.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[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.

[Bon82] John Adrian Bondy. *Graph theory with applications*. 1982.

[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.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. 2010.

[GL89] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, 1989.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery.

[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.

[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.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, mar 1999.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[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.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147. IEEE, 1995.

[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.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–755. Springer, 2012.

[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.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[Pei15] Chris Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Paper 2015/939, 2015. https://eprint.iacr.org/2015/939.

[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.

[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.

[Tre04] Luca Trevisan. Some applications of coding theory in computational complexity. *arXiv preprint cs/0409044*, 2004.

[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.

A PKE from LWE and k-SUM

In this section, we show how to build a public key encryption scheme from hardness of vector k-SUM and the Learning With Errors (LWE) problem with super-constant modulus to noise ratio.

A.1 Reduction from k-SUM to Vector k-SUM

We first show via a reduction that the hardness of decision vector k-SUM may be based on the hardness of decision k-SUM.

Lemma A.1. *If there is an algorithm that solves decision vector k-SUM at density Δ in time $T(r)$ with advantage ϵ, then there is an algorithm that solves decision k-SUM at density Δ in time $O(T(r) \cdot k^m)$ with advantage ϵ.*

Proof. Assume that we have an oracle access to an adversary $\mathcal{A}$ that solves decision vector k-SUM . It follows from definition of decision vector k-SUM that $\mathcal{A}$ takes as input r vectors of length m each of whose elements are members of $\mathbb{Z}_q$.

For sake of simplicity, we assume that we are trying to solve decision k-SUM modulo q^m, and that q is a prime. It is easy to verify that a density Δ instance consists of r elements.

Our first step involves expressing each element in base q. This transforms an element of $\mathbb{Z}_{q^m}$ into a vector of size m with each element belonging to $\mathbb{Z}_q$. Observe that if k elements added up to zero, each element of the sum of their image vectors must be within $[-(k-1), 0] \mod q$ since the carry at each position must be less than the number of summands. This suggests the following algorithm.

Algorithm $\mathcal{B}_1(x_1, x_2, \ldots, x_r)$

1. Create matrix Y of size $m \times r$ where $Y[i][j]$ is the ith digit of the base q representation of x_j.
2. For each vector $v \in [0, k-1]^m \mod q$:
   2.1. Create a new matrix Y_v where $Y_v[i][j] = Y[i][j] + v_i/k$
   2.2. Call $\mathcal{A}$ on Y_v. Say it returns b'
   2.3. If $b' = 1$ return 1
3. Return 0

Let us assume that the input X is a planted instance of decision k-SUM at density Δ. Let S be the set of planted indices. Observe that there exists some particular $v \in [0, k-1]^m \mod q$ such that $\sum_{j \in S} Y[i][j] + v[i] = 0 \mod q$ for all $i \in [m]$ (v can be thought of as the carry vector). Since $|S| = k$, the above equation implies $\sum_{j \in S} Y_v[i][j] = 0 \mod q \ \forall i \in [m]$. Furthermore, it is easy to see that Y_v is sampled from the planted distribution on $\mathbb{Z}_q^m$. Let us further assume that the input X is a non-planted instance of decision k-SUM at density Δ. This means that all the elements $\{x_i\}_{i \in r]}$ are sampled uniformly at random $\implies$ Y_v is identical to being sampled uniformly.

By the above, we have shown that the distribution from which Y_v is sampled is identical to the distribution from which an instance of decision k-SUM is sampled in both planted and non-planted cases respectively. Hence, if $\mathcal{A}$ solves decision vector k-SUM with advantage ϵ then $\mathcal{B}_1$ solves decision k-SUM with success probability ϵ.

The runtime of this algorithm is k^m times the runtime of $\mathcal{A}$ which simplifies to $O(k^m T)$. □

Note: At low densities ($\Delta \leq \frac{1}{r^{0.5+\epsilon} \cdot \log q}$) where $\epsilon \geq \frac{3}{k-3}$, decision vector k-SUM can be solved in time $o(r^{\lceil \frac{k}{2} \rceil})$ by the algorithm described in Lemma 4.26.

Corollary A.2. *If planted search k-SUM problem is hard at density Δ, then there is no algorithm that runs in time $\mathcal{O}\left(r^{k/2\,(1-\Omega(1))}\, k^{-m}\right)$ and solves the decision vector k-SUM problem at density Δ with advantage $o(1)$.*

Note that this follows from conjectured hardness of the planted k-SUM problem, Theorem 4.12, and the above reduction.

A.2 Reduction from Vector k-SUM to Targeted Vector k-SUM

Definition A.3 (Targeted Vector k-SUM)**.** *For any $r \in \mathbb{N}$, $k \in \mathbb{N}$, $q \in \mathbb{N}$, $\Delta \in \mathbb{R}$, an algorithm $\mathcal{A}$ is said to solve the* targeted vector k-SUM *with success probability ϵ if for both $b \in \{0,1\}$,*

$$\Pr_{X \sim \mathsf{D}_b^{(r)}}[\mathcal{A}(X) = b] \geq \epsilon.$$

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

where the distributions are defined below:

Distribution $\mathsf{D}_0^{(r)}$

1. Sample r group elements $X_1, X_2, \ldots, X_r$ i.i.d. uniformly at random from $\mathsf{G}^{(\Delta)}_{\text{VECTOR-}(q,\,k)\text{-SUM}}$
2. Return (X_1, X)

Distribution $\mathsf{D}_1^{(r)}$

1. Sample $r-1$ group elements $X_2, \ldots, X_r$ i.i.d. uniformly at random from $\mathsf{G}^{(\Delta)}_{\text{VECTOR-}(q,\,k)\text{-SUM}}$
2. Choose a random set $S \subseteq [2, r]$ with $|S| = k - 1$.
3. Compute $X_1 := \sum_{j \in S} X_j$
4. Return $(X_1,\ X)$

Note that targeted vector k-SUM as defined above is a decision problem. We do not prepend the word 'decision' to it because we only use the decision version of targeted vector k-SUM in this paper and it's use is restricted to the appendix.

Lemma A.4. *If there is an algorithm that solves the targeted vector k-SUM problem at density Δ in time T with advantage ϵ, then there exists an algorithm that solves decision vector k-SUM at density Δ in expected time $r\Theta(T)$ with advantage ϵ.*

Proof. Below, we give a reduction (algorithm $\mathcal{B}_2$) from targeted vector k-SUM to vector k-SUM .

Algorithm $\mathcal{B}_2(x_1, x_2, \ldots, x_r)$

1. Repeat atmost r times:
   (a) Pick a permutation $\pi : [r] \to [r]$ uniformly. Let $x' := [x_{\pi(1)}, \ldots, x_{\pi(r)}]$.
   (b) Call $\mathcal{A}$ on the vector $[x'_2, \ldots, x'_r]$ and the target x'_1. Say it returns value b.
   (c) If $b = 1$ then return 1 (Indicating a planted set)
2. Return 0

We can re-interpret the decision vector k-SUM problem defined at density Δ as follows:

1. Pick the (planted) index $i \xleftarrow{\$} [r - k + 1]$.
2. Pick a set S uniformly such that from $S \subseteq [r] \setminus [i]$ and $|S| = k - 1$.

3. Pick $a_1, \ldots a_r$ for all indices uniformly at random except i.
4. Let $x_i = -\sum_{j \in S} x_j$ (or) $x_i \xleftarrow{\$} \mathbb{Z}_q^m$
5. Output: $[x_1, \ldots, x_r]$.

The targeted vector k-SUM problem at density Δ can be re-interpreted as follows:

1. Pick a set S uniformly such that from $S \subseteq [2, r]$ and $|S| = k - 1$.
2. Pick $x_2, \ldots x_r$ for all indices uniformly.
3. Let $x_1 = \sum_{j \in S} x_j$ (or) $x_1 \xleftarrow{\$} \mathbb{Z}_q^m$
4. Output: $([x_2, \ldots, x_n], x_1)$.

Notice that the distributions of decision vector k-SUM and targeted vector k-SUM are equivalent at density Δ, conditioned on the fact that (planted) index i sampled in vector k-SUM is equal to 1. And by re-permuting the indices after every iteration in our reduction, we ensure that the planted index gets permuted to the location 1 (this happens in expected no.of iterations $= r$). □

Corollary A.5. *If planted search k-SUM is hard at density Δ, then there is no algorithm that runs in time $\mathcal{O}\left(r^{k/2\,(1-\Omega(1))}\, k^{-m}\right)$ and solves the targeted vector k-SUM problem at density Δ with advantage $o(1)$.*

Note that the Corollary A.5 directly follows from Corollary A.2 and the above reduction.

A.3 Construction of Public Key Encryption

Preliminaries. Below we define some preliminaries needed for our construction.

Lemma A.6. *(Leftover Hash Lemma)([HILL99, GKPV10]) Let $(X, Z) \in \mathcal{X} \times \mathcal{Z}$ be any joint random variable over $\mathbb{Z}_q^n$ with min-entropy $H_\infty(X|Z) \geq k$. For any $\epsilon > 0$ and $m \leq \frac{k - 2\log(\frac{1}{\epsilon}) - O(1)}{\log q}$, the joint distribution of $(Z, C, C \cdot s)$ where $C \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$ is uniformly random and $s \in \mathcal{X}$ is ϵ-close to the uniform distribution over $(Z, \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m)$.*

Regev [Reg09] defined a natural distribution over lattices called the discrete Gaussian distribution, parametrized by a scalar $\alpha > 0$. We additionally need a bound on samples drawn from the discrete Gaussian distribution.

Lemma A.7. *[Lyu12] Let χ_σ be the discrete Gaussian distribution on $\mathbb{Z}$ with parameter σ. Then, for any $k > 0$, $\Pr[|z| > k\sigma; z \leftarrow \chi_\sigma] \leq 2e^{(-k^2/2)}$.*

Definition A.8 (LWE Problem)**.** *Let r be the security parameter, let $m = m(r)$, $n = n(r)$ and $q = q(r) > 2$ be integers, and $\chi = \chi(r)$ be a distribution over $\mathbb{Z}_q$. The $\mathsf{LWE}(m, n, q, \chi)$ problem over $\mathbb{Z}_q$ is to distinguish between the following distributions:*

- $(A, s^T A + e^T)$, *where $A \leftarrow \mathbb{Z}_q^{m \times n}$, $s \leftarrow \mathbb{Z}_q^m$, and $e \leftarrow \chi^n$*
- (A, v^T) *where $A \leftarrow \mathbb{Z}_q^{m \times n}$, and $v \leftarrow \mathbb{Z}_q^n$*

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

$$\left| \Pr[\mathcal{A}(A, s^T A + e^T) = 1] - \Pr[\mathcal{A}(A, v^T) = 1] \right| \geq \epsilon,$$

where the randomness is taken over the instance and the random coins used by $\mathcal{A}$. We say that the $\mathsf{LWE}(m, n, q, \chi)$ problem is T-hard if no adversary A running in time $T(r)$ can distinguish between the above distributions with advantage greater than $1/T(r)$.

It is known [Reg09, BLP+13] that if we set χ to be the discrete Gaussian distribution with parameter αq, the $\mathsf{LWE}(m, n, q, \chi)$ problem is as hard as solving worst-case lattice problems such as gapSVP and SIVP with approximation factor $p(m)/\alpha$ for some polynomial p.

Theorem A.9. *[Reg09, Pei09, BLP+13] For any $n = \text{poly}(m)$, any modulus $q \leq 2^{\text{poly}(m)}$, and any (discretized) gaussian error distribution χ of parameter $\alpha q \geq 2\sqrt{m}$ where $0 < \alpha < 1$, solving the $\mathsf{LWE}(m, n, q, \chi)$ problem is at least as hard as solving gapSVP_γ on arbitrary m-dimensional lattices, for some $\gamma = \tilde{O}(\frac{m}{\alpha})$*

Since the best known algorithms for 2^k-approximation of gapSVP and SIVP run in time $2^{\tilde{O}(m/k)}$[Pei15], Theorem A.9 implies that the best known algorithms that solve LWE run in time $\Omega(2^{\frac{m}{\log \frac{m}{\alpha}}})$.

Construction. Our public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for message space $\mathcal{M} = \{0, 1\}$, is described as follows:

$\mathsf{PKE.KeyGen}(1^r)$: Upon input the unary representation of the security parameter r, do the following:

1. $A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$
2. Sample $x \in \{0, 1\}^n$ uniformly such that $\|x\|_0 = k$
3. Let $u := Ax$
4. Output $\mathsf{pk} := (u, A)$, $\mathsf{sk} := x$

$\mathsf{PKE.Enc}(\mathsf{pk} := (u, A), \mu \in \{0, 1\})$: Upon input the public key pk and the message μ, do the following:

1. Compute $c_1 := A^T s + e$ where $s \xleftarrow{\$} \mathbb{Z}_q^m, e \leftarrow \chi^n$
2. Compute $c_2 := u^T s + e' + \lfloor \frac{q}{2} \rfloor \mu$ where $e' \leftarrow \chi$
3. Output $\mathsf{ct} := (c_1, c_2)$

$\mathsf{PKE.Dec}(\mathsf{pk} := (u, A), \mathsf{sk} := x, \mathsf{ct} := (c_1, c_2))$: Upon input the public key pk, the secret key sk and the ciphertext ct, do the following:

1. Compute $\mu' := c_2 - x^T c_1$.
2. If $\|\mu'\|_2 \leq \frac{q}{4}$ then output 0 else output 1.

Lemma A.10 (Correctness)**.** *For any r, given parameters of the PKE construction are set as mentioned in (Appendix A.3). Then, the public key encryption scheme PKE described above is correct.*

Proof. Follows from a straight-forward calculation.

$$\begin{aligned}
\mu' &= c_2 - x^T c_1 \\
&= u^T s + e' + \left\lfloor \frac{q}{2} \right\rfloor \mu - x^T A^T s - x^T e. \\
&= e' + \left\lfloor \frac{q}{2} \right\rfloor \mu - x^T e. \\
&= \left\lfloor \frac{q}{2} \right\rfloor \mu + (e' - x^T e).
\end{aligned}$$

Hence, when $\mu = 0$, the value is $e' - x^T e$, if we show that $\left\|e' - x^T e\right\|_2 \leq \frac{q}{4}$ then that would be sufficient.

Note that $\left\|x^T e\right\|_2 = k(\alpha q)$ with probability $1 - o(1)$, since at most k of the entries in the summation are from bounded distribution and the rest of them are zero, we have that $|e'| \leq \alpha \cdot q$ with probability $1 - o(1)$(Lemma A.7), and so,

$$\left\|e' + x^T e\right\|_2 \leq (k + 1)\alpha q$$

Therefore, the correctness holds for the parameters mentioned in Appendix A.3. □

Next, we prove the security of our scheme under the hardness of LWE and k-SUM.

Lemma A.11 (Security)**.** *Assuming the hardness of planted-search-k-SUM at density Δ, and that $\mathsf{LWE}(m, n+1, q, \chi)$ (Theorem A.9) is $2^{\frac{m}{\log^{\frac{m}{\alpha}}}}$-hard where the parameters are chosen as described in Appendix A.3, the public key encryption scheme PKE satisfies weak $\mathsf{IND\text{-}CPA}$ security (Definition 6.1).*

Proof. We prove the theorem via a sequence of hybrids between the challenger and a PPT adversary $\mathcal{A}$.

$\mathsf{Hybrid\ 0}$: This is the real world with challenge bit 0. This is same as $\mathsf{PKEGame}^0_{\mathcal{A}(r)}$

$\mathsf{Hybrid\ 1}$: This world is same as $\mathsf{Hybrid\ 0}$ except we sample public key pk as $(u', A) \xleftarrow{\$} (\mathbb{Z}_q^{n\times 1}, \mathbb{Z}_q^{m\times n})$.

$\mathsf{Hybrid\ 2}$: This world is same as $\mathsf{Hybrid\ 1}$ except we sample the cipher-text, i.e, $\mathsf{ct} := (c_1', c_2')$ as $c_1' \xleftarrow{\$} \mathbb{Z}_q^n, c_2' \xleftarrow{\$} \mathbb{Z}_q$ respectively.

In $\mathsf{Hybrid\ 2}$, the distribution seen by the adversary is independent of the challenge bit b hence the advantage of the adversary in this world is negligible.

Indistinguishability of Hybrids. We now show that consecutive hybrids are indistinguishable.

Claim A.11.1. *Assume that planted search k-SUM is hard at density Δ (Corollary A.5) for the parameters described in Appendix A.3. Then, $\mathsf{Hybrid\ 0}$ and $\mathsf{Hybrid\ 1}$ are indistinguishable for any PPT adversary.*

Proof. Let $\mathcal{D}$ be a PPT adversary that distinguishes $\mathsf{Hybrid\ 0}$ from $\mathsf{Hybrid\ 1}$ with advantage ϵ. Let $\mathcal{C}$ be the targeted vector k-SUM challenger at density Δ. Then we give a reduction $\mathcal{B}$ that solves targeted vector k-SUM with advantage ϵ as follows:

1. $\mathcal{C}$ outputs $(A, u^*) \in (\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$.
2. $\mathcal{B}$ samples $\mu_0, \mu_1 \leftarrow \mathcal{M}, s \xleftarrow{\$} \mathbb{Z}_q^m$, $e \leftarrow \chi^n$, $e' \leftarrow \chi$
3. $\mathcal{B}$ then computes $(\mu_0, \mu_1, A, u^*, c_1 := A^T s + e, c_2 := u^{*T} s + e' + \lfloor \frac{q}{2} \rfloor \mu)$
4. If $\mathcal{D}$ outputs b then $\mathcal{B}$ outputs b

Notice that when $\mathcal{C}$ outputs $u' \xleftarrow{\$} \mathbb{Z}_q^m$ then $\mathcal{B}$ simulates $\mathsf{Hybrid\ 1}$ else, it simulates $\mathsf{Hybrid\ 0}$. Hence, $\mathcal{B}$ solves targeted vector k-SUM at density Δ with probability equal to the advantage of $\mathcal{D}$ i.e, ϵ.

From Corollary A.5, $\epsilon = o(1)$. Therefore, by the hardness of targeted vector k-SUM at density Δ, we have $\mathsf{Hybrid\ 0} \approx_c \mathsf{Hybrid\ 1}$ □

Claim A.11.2. *Assume that $\mathsf{LWE}(m, n+1, q, \chi)$ is $2^{\frac{m}{\log^{\frac{m}{\alpha}}}}$ hard (Theorem A.9) for the parameters described in Section A.3. Then, $\mathsf{Hybrid\ 1}$ and $\mathsf{Hybrid\ 2}$ are indistinguishable for any PPT adversary.*

Proof. Let $\mathcal{D}$ be a PPT adversary that distinguishes $\mathsf{Hybrid\ 1}$ from $\mathsf{Hybrid\ 2}$ with advantage ϵ. Let $\mathcal{C}$ be the $\mathsf{LWE}(m, n+1, q, \chi)$ challenger. Then we give a reduction $\mathcal{B}$ that solves $\mathsf{LWE}(m, n+1, q, \chi)$ problem with advantage ϵ as follows:

1. $\mathcal{C}$ outputs $\begin{bmatrix} c_1^* \\ c_2^* \end{bmatrix} \in \mathbb{Z}_q^{n+1}$ such that $c_1^* \in \mathbb{Z}_q^n$ and $c_2^* \in \mathbb{Z}_q$
2. $\mathcal{B}$ samples $\mu_0, \mu_1 \leftarrow \mathcal{M}, u' \xleftarrow{\$} \mathbb{Z}_q^m$, $e \leftarrow \chi^n$, $e' \leftarrow \chi$
3. $\mathcal{B}$ then computes $(\mu_0, \mu_1, A, u', c_1^*, c_2^*)$
4. If $\mathcal{D}$ outputs b then $\mathcal{B}$ outputs b

Notice that when $\mathcal{C}$ outputs such that $c_1^* \xleftarrow{\$} \mathbb{Z}_q^n$ and $c_2^* \xleftarrow{\$} \mathbb{Z}_q$ then $\mathcal{B}$ simulates Hybrid 2 else, it simulates Hybrid 1. Hence, $\mathcal{B}$ solves $\mathsf{LWE}(m, n+1, q, \chi)$ with probability equal to the advantage of $\mathcal{D}$ i.e, ϵ.

By the hardness of $\mathsf{LWE}(m, n+1, q, \chi)$ (Theorem A.9) where χ is parameterised by αq, $\epsilon = \mathrm{negl}(r)$. □

From the claims (Claims A.11.1 and A.11.2) we have that for any PPT adversary $\mathcal{A}$:

$$\left|\Pr[\mathsf{PKEGame}^0_{\mathcal{A}(r)} = 1] - \Pr[\mathsf{PKEGame}^1_{\mathcal{A}(r)} = 1]\right| \leq o(1) \qquad \square$$

Parameters. We now wish to give instantiations of parameters for the PKE construction in Appendix A.3, and observe the improvements in these parameters due to replacing targeted vector k-SUM as a computational analogue of "LHL" (Leftover Hash Lemma). Replacing LHL with targeted vector k-SUM is equivalent to assuming mild planted-search-k-SUM (Corollary A.5).

First, we enumerate the constraints that need to be satisfied for the correctness and security of PKE construction in Appendix A.3. We then provide two instantiations of parameters along with the analysis of how assuming hardness of targeted vector k-SUM assumption helps us improve the parameters of underlying PKE scheme. For correctness to hold, from Theorem A.10 we need $(k+1)\alpha q \leq \frac{q}{4}$ which implies that,

$$\alpha \leq \frac{1}{4(k+1)}.$$

For security wrt .Claim A.11.2, we need that the minimum time taken by an adversary to solve $\mathsf{LWE}(m, n, q, \chi)$ to be super-polynomial in r, this implies that,

$$2^{\frac{m}{\log \frac{m}{\alpha}}} \geq r^{\omega(1)}.$$

In order to use Theorem A.9 we also need to satisfy the condition $q \geq 2\sqrt{m}k$. For security wrt. Claim A.11.1, we need that minimum time taken by an adversary to solve targeted vector k-SUM at density Δ is super-polynomial in r. Thus, from Lemma A.5, we require that $n^{\frac{k-2}{2}}\, k^{-m} \geq r^{\omega(1)}$.

Parameter Analysis. In the construction of PKE in Appendix A.3, the $|\mathsf{pk}| = n\,m\,\log q$, $|\mathsf{sk}| = m\,\log q$, encryption time is $m\,k\,\log q$, decryption time is $m\,\log q$. Below we provide parameters for two settings, one which emphasizes the improvement in the public key size and one which emphasizes the improvement of approximation factor (α) of $\mathsf{LWE}(m, n, q, \chi)$. Note that the first three constraints are to be satisfied irrespective of whether we assume the hardness of targeted vector k-SUM (or) not. Therefore, we first set the parameters such that they satisfy the first three constraints then focus only on the constraint that assuming targeted vector k-SUM (or) using LHL enforces to analyze the improvement. Note that the constraint enforced by LHL is $m \leq \frac{k\,\log n}{\log q}$ (Lemma A.6).

Parameters for Reducing Public Key Size. We basically fix all the values except n and see how it is affected by assuming targeted vector k-SUM hardness in place of LHL. As the discussion is about efficiency in terms of pk-size, sk-size, encryption and decryption time, when given a choice we picked smaller values of m and q.

- $c = \omega(1)$
- $k = c\,(\log\log r)$
- $m = 2c\,\log r\,(\log\log r)$
- $q = 8\sqrt{c\,(\log r)\,(\log\log r)}\;(k+1)$
- $\frac{1}{\alpha} = 4\,(k+1)$

If we invoke LHL i.e, try to satisfy the constraint $m \leq \frac{k \log n}{\log q}$ to argue security then $n \geq r^2$. On the other hand if we assume hardness of targeted vector k-SUM at density Δ then we have to satisfy the constraint that $\Delta = \frac{k \log n}{m \log q}$ instead and, assuming $\Delta = \frac{1}{\text{polylog}(r)}$ means $n \geq (r)^{1/\log\log r}$. This means that the construction using targeted vector k-SUM instead of LHL has considerable gain improvement in terms of pk-size, i.e.,

Targeted vector k-SUM construction	LHL-based construction
$\lvert\mathsf{pk}\rvert = r^{1/\log\log r}\,(2c \log r \log\log r)$	$\lvert\mathsf{pk}\rvert = r^2\,(2c \log r \log\log r)$

Similarly, we now try to reduce k value.

Parameters to Improve the Approximation Factor (α) of $\mathsf{LWE}(m, n, q, \chi)$. Setting $r = n$, would ease our analysis. As all the other parameters except k are same, this assumption would not affect our qualitative analysis of how k is affected by the computational LHL. Therefore, we retain the values of q and m from the above analysis and, the parameters are as follows:

- $c = \omega(1)$
- $m = 2c \log r\,(\log\log r)$
- $q = 8\sqrt{c \log r\,(\log\log r)}\;(k+1)$
- $\frac{1}{\alpha} = 4\,(k+1)$
- $n = r$

Note that in LHL, one has to satisfy the condition of $m \leq \frac{k \log n}{\log q} \implies k \geq 2\,c(\log\log r)\,\log q$ whereas when we instantiate our scheme using targeted vector k-SUM assumption, the condition to satisfy is $\Delta = \frac{k \log r}{m \log q}$. Notice that, in this case we can set k as any super-constant because $\Delta = \frac{1}{\text{polylog}(r)}$ which is fine. Thereby providing us with a small improvement in terms of the value of k. Note that, by trading off n and k value, one can improve both approximation factor, $|\mathsf{pk}|$ and encryption time.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il