Fine-Grained Complexity in a World without Cryptography

Josh Alman [1], Yizhi Huang [1], and Kevin Yeo [1,2]

[1]Columbia University
[2]Google

Abstract

The study of fine-grained cryptography has proliferated in recent years due to its allure of potentially relying on weaker assumptions compared to standard cryptography. As fine-grained cryptography only requires polynomial gaps between the adversary and honest parties, it seems plausible to build primitives relying upon popular hardness assumptions about problems in **P** such as k-SUM or Zero-k-Clique. The ultimate hope is that fine-grained cryptography could still be viable even if all current cryptographic assumptions are false, such as if $\mathbf{P} = \mathbf{NP}$ or if we live in Pessiland where one-way functions do not exist.

In our work, we consider whether this approach is viable by studying fine-grained complexity when all standard cryptographic assumptions are false. As our main result, we show that many popular fine-grained complexity problems are easy to solve in the average-case when one-way functions do not exist. In other words, many candidate hardness assumptions for building fine-grained cryptography are no longer options in Pessiland. As an example, we prove that the average-case k-SUM and Zero-k-Clique conjectures are false for sufficiently large constant k when no one-way functions exist. The average-case Zero-k-Clique assumption was used to build fine-grained key-exchange by Lavigne *et al.* [CRYPTO'19]. One can also view the contrapositive of our result as providing an explicit construction of one-way functions assuming $n^{\omega_k(1)}$ average-case hardness of k-SUM or Zero-k-Clique for all constant k.

We also show that barriers for reductions in fine-grained complexity may be explained by problems in cryptography. First, we show that finding faster algorithms for computing discrete logarithms is equivalent to designing average-case equivalence between k-SUM and k-CYC (an extension of k-SUM to cyclic groups). In particular, finding such a reduction from k-CYC to k-SUM could potentially lead to breakthrough algorithms for the discrete logarithm, factoring, RSA and quadratic residuosity problems. Finally, we show that discrete logarithms with preprocessing may be reduced to the k-CYC-Index problem, and we present faster algorithms for average-case k-SUM-Index and k-CYC-Index.

1 Introduction

In modern cryptography, cryptographic primitives that aim for security against super-polynomial time adversaries are typically built from a wide-range of standard assumptions including one-way functions (OWF), structured algebraic problems from number theory such as factoring, discrete logarithm, RSA and quadratic residuosity and geometric problems from lattices including learning

This is the full version of the paper that appears in the proceedings of Eurocrypt 2025 [7].

parity with noise (LPN) and shortest vector problems (SVP). Even though they are all standard assumptions, they remain unproven, and proving them appears far beyond current techniques, since it requires proving $\mathbf{P} \neq \mathbf{NP}$ and much more including that $\mathbf{NP} \not\subseteq \mathbf{BPP}$, and it must be possible to efficiently sample hard instances with known solutions in polynomial time. These requirements correspond to Impagliazzo's worlds [53] of Algorithmica where $\mathbf{NP} \subseteq \mathbf{BPP}$, Heuristica where $\mathbf{NP}$ problems are easy in the average-case and Pessiland where hard instances with known solutions cannot be efficiently sampled and OWFs do not exist. In other words, modern cryptography requires assumptions that we are very far from proving unconditionally (and could be false for all we know).

This has motivated the study of fine-grained cryptography where the goal is to only obtain polynomial time gaps between adversaries and honest parties, with the weaker goal to only obtain security against adversaries with polynomial running time $O(n^c)$ for a fixed constant c. This difference is substantial as it enables plausible construction of cryptographic primitives that are secure even if $\mathbf{P} = \mathbf{NP}$, since it could potentially rely only upon the (average-case) polynomial time hardness of problems in $\mathbf{P}$. It is conceivable the fine-grained approach could lead to secure protocols even if all standard cryptographic assumptions are false.

Even though fine-grained cryptography no longer requires assuming $\mathbf{P} \neq \mathbf{NP}$, it still requires the polynomial-time hardness of problems in $\mathbf{P}$. This is exactly where the area of fine-grained complexity comes into play: substantial work has been able to prove a large web of reductions starting from a core set of problems with well-founded hardness assumptions including the orthogonal vectors (OV), k-SUM, Zero-k-Clique and all-pairs shortest path (APSP) problems and all of their variants. For these core problems, there is a simple algorithm in time $\tilde{O}(n^c)$ for some constant $c > 1$. Yet, there has been no progress in developing $n^{c-\Omega(1)}$ time algorithms even after decades of research. So, we assume that such faster algorithms do not exist.

The first work to build fine-grained cryptography from fine-grained complexity assumptions is Ball *et al.* [11, 12] that developed proofs of work from the worst-case hardness of OV, 3-SUM or APSP. Recently, it was shown that permissionless consensus may also be built assuming worst-case hardness of OV [10]. Unfortunately, these are the only primitives that are known to be built from worst-case hardness of fine-grained problems. In fact, Ball *et al.* [11] presented barriers to constructing fine-grained OWFs from worst-case assumptions using current techniques. To circumvent this obstacle, recent works [63, 35] showed that fine-grained OWFs and key-exchange may be constructed using the average-case hardness of k-SUM and Zero-k-Clique. Somewhat tangentially, it was shown that public-key encryption may be built using weaker LPN variants assuming average-case hardness of k-XOR [6]. Nevertheless, the current state-of-the-art heavily relies upon the average-case hardness of standard fine-grained problems to obtain fine-grained constructions of important cryptographic primitives.

Going back to the high-level goal, we recall that the ultimate hope of fine-grained cryptography is to build secure protocols even if all standard cryptographic assumptions are false. In particular, we want secure protocols even if we live in Pessiland where OWFs do not exist. Therefore, we can ask the following:

What happens to popular fine-grained complexity problems in the average-case if one-way functions do not exist?

This question is of particular importance since the fine-grained constructions of fundamental primitives including one-way functions and key exchange critically rely upon average-case hardness.

Interestingly, we present strong evidence that many of the popular fine-grained complexity problems are easy on average when no OWFs exist. This includes the k-SUM and Zero-k-Clique problems that were used in the construction of fine-grained OWFs and key-exchange [63].

We can also consider the question in the reverse direction. Even though there has been extraordinary progress in understanding the relationships between problems in fine-grained complexity, there remain many prominent problems whose relationships remain elusive. For example, there is a class of problems generalizing k-SUM which we call $k\text{-LIST}^{\mathsf{G}}$, where one fixes a group ensemble $\mathsf{G} = \{\mathbb{G}^{(n)}\}$ (i.e., a group $\mathbb{G}^{(n)}$ for each input size n), and given k lists of n elements from $\mathbb{G}^{(n)}$, one would like to determine whether one can pick an element from each list which sum (under the $\mathbb{G}^{(n)}$ operation) to a given target $t \in \mathbb{G}^{(n)}$. When $\mathbb{G}^{(n)}$ is the additive group of integers modulo $m(n)$, this is the k-SUM problem, but when $\mathbb{G}^{(n)}$ is the group $\mathbb{Z}_2^{m(n)}$ it is the k-XOR problem, when $\mathbb{G}^{(n)}$ is the multiplicative group modulo $m(n)$, we call this the k-PROD problem, and when $\mathbb{G}^{(n)}$ is another cyclic group (such as a cyclic subgroup of an elliptic curve group which arises frequently in cryptosystems) we call this the $k\text{-CYC}^{\mathsf{G}}$ problem. In general, the best known algorithms for all these problems run in time $\Theta(n^{\lceil k/2 \rceil})$ as long as $|\mathbb{G}^{(n)}|$ is not too small, and many known fine-grained results can be proved in the same way for all these problems (e.g., [81, 36, 1, 78, 41]), but there are no known nontrivial fine-grained reductions showing that the hardness of one implies the hardness of another [54]. There is a limited number of known approaches to proving barriers against fine-grained reductions, which all do not appear to apply here [27, 17, 18]. One may wonder whether the lack of progress can be explained through the cryptographic lens. This leads to the following:

Are there any potential reductions between fine-grained problems that would have major implications in cryptography?

This would act as a barrier for showing such reductions. We show one manifestation of this phenomenon: we prove that efficient, average-case reductions from either k-PROD or $k\text{-CYC}^{\mathsf{G}}$ to k-SUM could potentially lead to surprising algorithmic speed-ups for a wide-range of number-theoretic cryptographic assumptions including discrete logarithm, factoring, RSA and quadratic residuosity.

1.1 Our Contributions

Limits of FGC without Crypto. In view of the construction of fine-grained cryptography in [63] based on average-case fine-grained assumptions, we consider whether the assumptions still hold if we are in a world without cryptography. Indeed, in a world with cryptography, there is no need for fine-grained cryptography which only gives weaker guarantees.

We focus particularly on the following two conjectures about average-case k-SUM and average-case Zero-k-Clique (for definitions of the two problems, see Section 2.2), as these problems were considered by [63].

Conjecture 1 (AVG-k-SUM **Conjecture, Informal).** *Any $n^{\lceil k/2 \rceil - \Theta(1)}$-time algorithm does not solve the* k-SUM *search problem on average with inverse polynomial failure probability.*

Conjecture 2 (AVG-Zero-k-Clique **Conjecture, Informal).** *Any $n^{k-\Theta(1)}$-time algorithm does not solve the* Zero-k-Clique *search problem on average with inverse polynomial failure probability.*

The AVG-Zero-k-Clique Conjecture, in particular, is widely cited and applied in prior work on average-case fine-grained hardness [74, 63, 41, 6]. We show that:

Theorem 1.1 (Informal)**.** *If one-way functions do not exist, then the* AVG-k-SUM *Conjecture and the* AVG-Zero-k-Clique *Conjecture are false.*

In fact, we prove an even stronger refutation assuming no one-way functions, showing that as k grows larger, one can achieve a very fast running time whose exponent does not grow with k at all:

Theorem 1.2 (Informal)**.** *If one-way functions do not exist, then there exists a universal constant c, such that for all $k > 0$, there is an $O(n^c)$-time algorithm that solves k-*SUM *on average with inverse polynomial failure probability, for infinitely many n.*[1]

Theorem 1.3 (Informal)**.** *If one-way functions do not exist, then there exists a universal constant c, such that for all $k > 0$, there is an $O(n^c)$-time algorithm that solves* Zero-k-Clique *on average with inverse polynomial failure probability, for infinitely many n.*

Thus, in a world without cryptography, one should not aim to design fine-grained cryptosystems based on the AVG-k-SUM Conjecture or the AVG-Zero-k-Clique Conjecture for large k. We note that our results can also be viewed in the contrapositive: if the AVG-k-SUM Conjecture (or the AVG-Zero-k-Clique Conjecture) is true, then there exist one-way functions. Thus, progress on proving these conjectures in fine-grained complexity can be seen as progress on establishing the foundations of cryptography. As an example, a worst-case to average-case reduction for either k-SUM or Zero-k-Clique would allow us to build one-way functions from worst-case assumptions.

The prior work [63] constructs both fine-grained one-way functions and fine-grained key exchange protocols based on the hardness of k-SUM and Zero-k-Clique. Our results imply limitations on how secure these constructions can be if there are no one-way functions. The fine-grained one-way function construction in [63] is actually based on weaker conjectures than the AVG-k-SUM Conjecture (or the AVG-Zero-k-Clique Conjecture). Namely, if for some k, k-SUM (or Zero-k-Clique) is hard on average for $O(n^{1+\gamma})$-time algorithms (or $O(n^{2+\gamma})$, resp.) for some constant $\gamma > 0$, then their fine-grained one-way function is also secure against $O(n^{1+\gamma})$-time (or $O(n^{2+\gamma})$-time, resp.) adversaries. Thus, the conclusion of our result implies that there exists a constant $c > 0$ such that the fine-grained one-way function construction of [63] cannot be secure against n^c-time adversaries.

As for the fine-grained key exchange construction in [63], their construction is based on the AVG-Zero-k-Clique Conjecture. Namely, if for some k, Zero-k-Clique is hard on average for $O(n^{k-\delta})$-time algorithms for any $\delta > 0$, then their key exchange construction takes $O(n^{k+2})$ time and is secure against $O(n^{2k-\delta})$-time adversaries. They take k to be arbitrarily large to achieve a gap arbitrarily close to 2 in the running time exponent of the key-exchanging parties and that of the adversary. However, our results show that one cannot pick arbitrarily large k, and the gap achievable by their construction must be bounded below 2.

One can also view our results in the contrapositive direction, where they provide another characterization (and construction) of one-way functions based on average-case k-SUM hardness:

Theorem 1.4 (Informal, contrapositive of Theorem 1.2)**.** *Suppose solving k-*SUM *on average with inverse polynomial failure probability requires $n^{\omega_k(1)}$ time. Then, one-way functions exist.*

It is often conjectured that k-SUM requires time $\Omega(n^{\lceil k/2 \rceil})$ in the average case (Conjecture 1 above); Theorem 1.4 shows that a much weaker form of this conjecture suffices to imply the existence of one-way functions.

[1] It seems necessary for our consequences of one-way functions not existing to hold only infinitely often, since "one-way functions" and "infinitely often one-way functions" are not known to be equivalent.

Our results here generalize to all problems that have “planting algorithms”. Roughly speaking, a planting algorithm is an algorithm f that takes a uniformly random input r and outputs an instance $f(r)$ that always has a solution, such that from r we can also efficiently compute a solution to $f(r)$. This is similar to “plantability” defined in [63], which also requires the output distribution of f to be close to the uniform distribution over all instances with exactly one solution.

In the informal statements of Theorems 1.2 and 1.3 above, we didn't mention the exact formation of k-SUM. Actually, in this work, for k-SUM, we mainly consider the variant where there are k lists of numbers, instead of one list, and where the numbers are elements in a finite cyclic group $\mathbb{Z}_m$, instead of integers. We are also focusing on the search version, as there is a simple reduction from decision to search. For the exact definition we use for k-SUM, and a discussion of different variants, please see Section 2.2. Similarly, for Zero-k-Clique, we consider the variant where the underlying graph is a complete k-partite graph, instead of a complete graph, and where the edge weights are elements in $\mathbb{Z}_m$, instead of integers. For the exact definition we use for Zero-k-Clique, please see Section 2.4.

There are also different variants of average-case search k-SUM or Zero-k-Clique with respect to the modulus and the input distribution. We are actually able to extend Theorems 1.2 and 1.3 for different distributions and modulus regimes:

- The uniform distribution over all instances. Note that when $m(n) = \Theta(n^{k+\Omega(1)})$, it becomes trivial since almost every instance does not have a solution.
- The planted distribution, which is the distribution obtained from “planting” a solution in a uniformly sampled instance.
- The uniform distribution over all instances with a solution.
- The uniform distibution over all instances with exactly one solution, for $m(n) \geq n^k$.

Note that we always consider the modulus m as a function on n, because the properties of k-SUM and Zero-k-Clique, including the probability that a uniform instance has a solution, depends on the relation between m and n^k. See Section 2.2 for a detailed discussion.

We remark that for the case where the input distribution is uniform over all instances and $m \ll n^k$, there is Wagner's algorithm [80], which is incomparable to our result: Wagner's algorithm is unconditional and has running time $n^{o(k)}$ where the exponent grows with k, while ours is based on the non-existence of one-way functions and has running time n^c for constant c independent of k. For details of Wagner's algorithm, see Appendix A.1.

Cryptographic Barriers to Fine-Grained Reductions. Even though there has been substantial progress in relating problems in fine-grained complexity (see, for instance, the survey [82] for a list of many problems in diverse areas of algorithms which are known to be hard assuming common assumptions like SETH or the hardness of OV, k-SUM, or APSP), there remain many open questions about relations between certain problems of interest. One could also wonder whether the lack of these reductions relate to potential questions in cryptography. In our work, we present one instance of this phenomenon where we show that the DLog problem in cryptography is exactly the barrier for a fine-grained reductions between either the k-CYC or k-PROD and k-SUM problems. This is somewhat surprising as k-SUM, k-PROD and k-CYC seem to be nearly identical problems: they all work over different representations of the cyclic group.

Theorem 1.5 (Informal)**.** *The* $\mathsf{DLog}^{\mathsf{G}}$ *problem for group ensemble* G *of order* m *can be solved in* $m^{1/2-\Omega(1)}$ *time if and only if there exists an efficient reduction from average-case* k*-*$\mathsf{CYC}^{\mathsf{G}}$ *to average-case* k*-*SUM *over* $\mathbb{Z}_m$ *in time* $n^{k/2-\Omega(1)}$*. If the reduction runs in* $\mathsf{poly}(n)$ *time for* $k = \omega(1)$*, then there exists sub-exponential time algorithms for* $\mathsf{DLog}^{\mathsf{G}}$*.*

Notably, $\mathsf{DLog}^{\mathsf{G}}$ is known to be solvable in $O(m^{1/2})$ time in generic groups, and Theorem 1.5 says that whether one can improve on this for a particular G is *equivalent* to whether one can give an efficient average-case fine-grained reduction from k-$\mathsf{CYC}^{\mathsf{G}}$ to k-SUM. We will observe below that a reduction in the other direction is not too difficult to achieve. Perhaps most strikingly, the interpretation of Theorem 1.5 is very different depending on the ensemble G.

Consider first when G is the multiplicative group $\mathbb{Z}^{\times}_{m(n)}$. In this case, k-$\mathsf{CYC}^{\mathsf{G}}$ is the problem k-PROD we defined earlier. DLog over the multiplicative group is very well-studied, and although there is no known algorithm which provably runs in $m^{1/2-\Omega(1)}$ time, a line of work has given candidate algorithms which heuristically run in sub-exponential time (see Section 9.2 of [59]). Our Theorem 1.5 thus yields a candidate fine-grained equivalence between k-PROD and k-SUM, which would show that each can be solved in time $n^{\lceil k/2\rceil-\Omega(1)}$ in the average case if and only if the other can, but proving the correctness of this reduction requires proving the running time of the candidate DLog algorithms.

By contrast, consider when G is a cyclic group where we believe $\mathsf{DLog}^{\mathsf{G}}$ is hard and requires $m^{1/2-o(1)}$ time, such as popular elliptic curve groups used in practice (see [45, 60]). In this case, Theorem 1.5 gives a barrier to reducing from k-$\mathsf{CYC}^{\mathsf{G}}$ to k-SUM. This is, to our knowledge, the first barrier to a fine-grained reduction based on cryptographic assumptions. Notably, using known connections with DLog over $\mathbb{Z}^{\times}_{m}$ (see [9] for example), we can extend this to show that an efficient reduction from k-PROD to k-SUM would yield sub-exponential time algorithms for several cryptographic number-theoretic assumptions.

Theorem 1.6 (Informal)**.** *If there exists a* $\mathsf{poly}(n)$ *time reduction from average-case* k*-*PROD^m *to average-case* k*-*SUM^m *for* $k = \omega(1)$*, then there exists sub-exponential time algorithms for factoring, RSA and quadratic residuosity.*

Next, we also consider reductions in the preprocessing setting. Since practical cryptography often works over particular well-studied groups, it is natural to study preprocessing versions of problems over those groups. We show that prior reductions from DLog to k-CYC shown in [34, 80] may be extended to the preprocessing setting as well. In particular, we show that DLog with preprocessing may be reduced to the average-case k-CYC-Index problem that extends the k-SUM-Index problem [38, 49, 51, 62] to cyclic groups. Finally, we provide evidence that the average-case k-SUM-Index and k-CYC-Index problems are easier than their worst-case counterparts by presenting improved algorithms.

Theorem 1.7 (Informal)**.** *There exists average-case algorithms for* k*-*SUM*-*Index *and* k*-*CYC*-*Index *using space* S *and query time* T *such that* $S^2T = \tilde{O}(n^{2(k-1)})$*. For space* $S = \tilde{O}(n^{k-1-\delta})$*, the algorithm uses query time* $T = \tilde{O}(n^{2\delta})$ *for* $\delta \geq 0$*.*

This improves upon previous worst-case constructions for k-SUM-Index [51, 62] that required query time $T = \tilde{O}(n^{3\delta})$ for space $S = \tilde{O}(n^{k-1-\delta})$.

1.2 Technical Overview

k-SUM parameter regimes. In this section, we give an overview of the techniques we use to prove our results. We will focus here particularly on the k-SUM problem, as this case already illustrates the main ideas.

Before getting into it, we briefly discuss the importance of the modulus m in the k-SUM problem. The case when $m \ll n^k$ is referred to as the "dense regime", since this is the regime where a random instance will likely have many solutions. Similarly, the case when $m \gg n^k$ is referred to as the "sparse regime", since this is the regime where a random instance is unlikely to have any solutions. The intermediate case $m = \Theta(n^k)$ is perhaps the most well-studied, since it is when the decision version of k-SUM is most interesting. Many prior works have developed techniques which focus on only one of the regimes; for instance, Wagner's algorithm [80] can solve k-SUM in $n^{\lceil k/2 \rceil - \Omega(1)}$ time in the dense regime $m < n^{k(1-\Omega(1))}$, and recent work [6] has shown how to amplify error probabilities in the sparse regime. Our results hold for all modulus choices, and we will need to handle separate intricacies which arise in each regime.

k-SUM and Zero-k-Clique algorithms when one-way functions don't exist. We focus in this overview on k-SUM; the more general proofs for Zero-k-Clique and more general problems with planting algorithms are similar.

Consider the following "planting algorithm" that always outputs an instance of k-SUM with a solution: the algorithm takes as input a choice of indices corresponding to a solution $(y_1, y_2, \ldots, y_k) \in [n]^k$ and a k-SUM instance $(L_1, L_2, \ldots, L_{k-1}, L'_k)$ where each L_i is a list of n elements in $\mathbb{Z}_m$, while L'_k is a list of $n-1$ elements that can be viewed as removing $L_k[y_k]$ from a list L_k of n elements. The algorithm then outputs $(L_1, L_2, \ldots, L_{k-1}, L^*_k)$ where L^*_k is obtained by inserting $L^*_k[y_k] \coloneqq -(L_1[y_1] + L_2[y_2] + \cdots + L_{k-1}[y_{k-1}])$ at the y_k-th position of L'_k. We can see that this algorithm outputs a k-SUM instance with solution $(y_1, y_2, \ldots, y_k)$, and moreover, every k-SUM instance with a solution is a possible output of the algorithm (when the input is a solution and the instance with the solution element in the k-th list removed). We note that the fine-grained one-way function construction in [63] is also based on planting algorithms, but we will discuss shortly that their proof technique does not suffice for our result.

Our main observation is that, when the modulus $m \geq n^k$, the length of input of the planting algorithm, $k \log n + (kn-1) \log m$, is less than the length of output, $kn \log m$. In this case, we call the planting algorithm "expanding". Since every k-SUM instance with a solution is a possible output of the algorithm, the planting algorithm can be viewed as a way to "compress" any k-SUM instance with a solution. On the other hand, a uniformly sampled instance, which is just a random string, should not be compressible. Therefore, intuitively, if we were able to compute the Kolmogorov complexity of a string, which measures how compressible the string is, then we would be able to distinguish between an instance with a solution and a uniformly sampled instance. Unfortunately, it is well known that Kolmogorov complexity is uncomputable.

To get around this, we use the fact that our planting algorithm is efficient. We instead use time-bounded Kolmogorov complexity K^t, which is defined as the length of the shortest program that outputs a string x in at most $t(|x|)$ time. Since the planting algorithm is efficient (actually linear-time), the argument above still holds for K^t when $t(N)$ is a super-linear polynomial. So, if we are able to compute K^t-complexity, we will be able to distinguish between an instance with a solution and a uniformly sampled instance.

Liu and Pass [66] showed that the existence of one-way functions is equivalent to the average-case hardness of K^t-complexity for any (large enough) polynomial t. Therefore, when one-way functions do not exist, this approach gives an algorithm that solves k-SUM for modulus $m \geq \alpha n^k$ for some constant $\alpha > 1$ for infinitely many n, and the algorithms have running time close to that of the algorithm computing K^t-complexity, which is independent of k.

By delving deeper into the proof of equivalence between the hardness of K^t-complexity and the existence of one-way functions, we are able to streamline this proof by using universal one-way functions. For a fixed polynomial time bound $T(\cdot)$, the universal one-way function is defined as

$$u_T(M, x) \coloneqq (M, U_T(M, x))$$

where M is the description of a Turing machine and $U_T(M, x)$ is the output of a universal Turing machine simulating M on input x for time $T(|x|)$. It can be shown that if there is a weak inverter for u_T that runs in $(\cdot)^c$ time for some constant c, then for any function f that runs in $T(\cdot)$ time, there is a weak inverter for f that runs in $(\cdot)^c$ time. For more details on universal one-way functions, we refer the reader to Appendix H or [47, Section 2.4.1].

Since there is no machine that runs in polynomial time that can simulate every polynomial-time Turing machine, universal one-way functions can only be constructed for machines that have running time bounded by a fixed polynomial, and this is one of the reasons why universal one-way functions are not used very often in cryptography. However, they are good for our purposes since the planting algorithms for k-SUM run in linear time. Then, by using an inverter for universal one-way functions on the planting algorithm, we obtain an algorithm for k-SUM for infinitely many n over the planted distribution, which is the output distribution of the planting algorithm when the input is from the uniform distribution.

When the modulus satisfies $m \gg n^k$, the planted distribution is very close to the uniform distribution, so we automatically get an algorithm for k-SUM over uniform distribution in that case. However, when m is around n^k, the total variation distance between the two distributions are constant. In this case, we make use of the "expanding" property of the planting algorithm, that is, the output length is greater than the input length, and the surjective property, that is, every instance with a solution is a possible output of the planting algorithm, to bound the error probability of the inverter on the uniform distribution.

When $m \ll n^k$, the planting algorithm is no longer expanding, so the inverter does not work on the uniform distribution. However, in this case, we are able to show an average-case fine-grained reduction to the case where m is around n^k, by dividing the instance into $n/m^{1/k}$ sub-instances of k-SUM each of size $m^{1/k}$ and uses the algorithm in the previous case (when m is around n^k) on each sub-instance.

We emphasize that the details here, and particularly the error analysis, require particular care. For instance, the construction of fine-grained one-way functions in [63] is also based on planting algorithms for k-SUM. However, their error analysis only uses the fact that the distance between the planted distribution and the uniform distribution is constant when $m \geq n^k$, and so they use a stronger version of the AVG-k-SUM conjecture with constant failure probability. Furthermore, to the best of our knowledge, the known hardness amplification [6] for AVG-k-SUM only reduces from failure probability of $o(1/\log n)$ to $1 - \Omega(1/\mathsf{poly}\log(n))$, falling short of $1/\mathsf{poly}(n)$. It therefore seems difficult to use the proof techniques in [63] to achieve our desired inverse polynomial error probability. We get around this and achieve our desired error probability by giving a better analysis with the expanding property of the planting algorithm when $m \geq n^k$.

DLog Barrier to k-SUM and k-CYC Reductions. The core intuition comes from the existence of Wagner's k-SUM algorithm [80] that solves k-SUM in $n^{\lceil k/2 \rceil - \Omega(1)}$ time for the dense setting where

$n^k \gg m$ and $k \geq 4$. In contrast, these algorithms do not easily extend to the k-PROD or k-CYC problem and there do not exist algorithms that beat the trivial $O(n^{\lceil k/2 \rceil})$ algorithm even in the dense setting. In fact, we show that if one could find such an algorithm, it would lead to the fastest known algorithms for DLog in many popular elliptic curve groups used in practice (see Appendix A). Using the above as guidance, we can formulate the $\mathsf{DLog}^{\mathsf{G}}$ conjecture that there exists some group ensemble $\mathsf{G} = \{\mathbb{G}^{(n)}\}$ with groups of order $m = \mathsf{poly}(n)$ such that there is no algorithm solving $\mathsf{DLog}^{\mathsf{G}}$ in time $m^{1/2-\Omega(1)}$.[2] This conjecture matches known lower bounds for computing discrete logarithms in the generic group model [79].

We use the above intuition for the first direction. Towards a contradiction, suppose there existed an efficient, average-case reduction from k-$\mathsf{CYC}^{\mathsf{G}}$ to k-SUM. Then, we show that there exists faster $n^{\lceil k/2 \rceil - \Omega(1)}$ time algorithms for DLog contradicting the above conjecture by relying upon on Wagner's algorithm. It is not hard to see that DLog over group $\mathbb{G}$ of size m with generator g and target t can be reduced to a random, dense k-CYC instance over in the following way (also done in [34, 80]). We generate k random lists of $n \gg m^{1/k}$ where each element of the lists are generated by first picking a random exponent x from $[m]$ and computing g^x. It is not hard to see that there exists some k-tuple whose product equals t except with small probability since $n^k \gg m$. Afterwards, we can reduce the k-CYC instance to a k-SUM instance and apply Wagner's algorithm resulting in a $n^{\lceil k/2 \rceil - \Omega(1)} = m^{1/2-\Omega(1)}$ algorithm that contradicts the DLog conjecture. It turns out that the above reduction works even for groups with composite order. Therefore, we can use DLog over composite order groups to extend the result to the factoring, RSA and quadratic residuosity problems (see [9]). In other words, this is a barrier to reductions between k-SUM and k-CYC (k-PROD) as it would result in breakthrough algorithms for DLog and other cryptographic problems.

For the other direction, we suppose DLog is easy in some group $\mathbb{G}$. Then, we can construct an efficient, average-case reduction from k-CYC to k-SUM. The reduction executes the DLog algorithm for all nk elements in the random k-CYC instance that immediately results in a random k-SUM instance.

Average-Case k-SUM-Index and k-CYC-Index Algorithms. For our improved algorithms for both k-SUM-Index and k-CYC-Index, we extend the prior worst-case constructions in [62, 51]. Both problems receive $k-1$ lists of size n as input and the goal is to output a $(k-1)$-tuple whose sum (product) equals a query target. The prior works relied upon the Fiat-Naor inversion algorithm [44] that use space S and time T to invert functions $f : \mathcal{X} \to \mathcal{X}$ such that $S^3 T = \tilde{O}(|\mathcal{X}|^3)$. Prior works [62, 51] showed there exists algorithms with space $S = \tilde{O}(n^{k-1-\delta})$ and query time $T = \tilde{O}(n^{3\delta})$ for any $\delta \geq 0$.

For the average-case, we note the $k-1$ input lists consisting of n uniformly random group elements. Our idea is to, instead, use Hellman's algorithm [52] for inverting random functions that only requires space S and T such that $S^2 T = \tilde{O}(|\mathcal{X}|^2)$. Even though the resulting function is not fully random, we show that the collision probability is sufficiently small that Hellman's algorithm is still applicable and efficient (using the analysis in [44]). As a result, we obtain better average-case algorithms for both k-SUM-Index and k-CYC-Index that use space $S = \tilde{O}(n^{k-1-\delta})$ and query time $T = \tilde{O}(n^{2\delta})$ for any $\delta \geq 0$. We note that our algorithm requires less query time for the same space usage.

[2]The parameter n for $\mathsf{DLog}^{\mathsf{G}}$ can be viewed as the security parameter λ that allows a cryptographic protocol to pick a group that is secure against $\mathsf{poly}(\lambda)$ adversaries.

1.3 Related Works

Fine-Grained Complexity. The worst-case complexity of many problems has been studied in fine-grained complexity (see [46, 13, 2, 3, 61, 62, 51] and references therein). We point readers to this survey [82] for more details.

The average-case hardness of various fine-grained problems have also been studied. Starting from Wagner's algorithm [80], there has been a line of work trying to construct faster algorithms for k-SUM and/or k-XOR in the dense setting such as [70, 73, 71, 40, 65]. Several works have shown that these kind of algorithms are optimal for k-SUM using reductions from worst-case lattice problems in the dense case [25] as well as showing lower bounds for higher density variants of k-SUM assuming average-case hardness of lower density variants of k-SUM [41].

Another line of work has attempted to prove that average-case hardness of problems may be based on weaker assumptions. The first set of problems whose average-case hardness could be based on worst-case assumptions was presented in [11]. Factored versions of popular fine-grained problems were introduced in [35] whose average-case hardness may be based on the worst-case hardness of standard fine-grained problems. The same work also showed average-case reductions from the search to the decision variants of Zero-k-Clique. Relatedly, prior works have also shown worst-case to average-case reductions for counting k-cliques [48, 21]. Finally, a recent work [6] studied the average-case hardness of k-SUM and k-XOR in the sparse setting and presented reductions from planted to non-planted distributions, search to decision as well as hardness amplification.

There are many pairs of problems, particularly problems at the core of hardness assumptions, for which researchers have been unable to give fine-grained reductions. In some cases, there are known barrier results explaining why fine-grained reductions may be difficult. [27] showed that, assuming a nondeterministic analogue of the Strong Exponential Time Hypothesis (SETH), one cannot reduce from k-SAT to APSP or k-SUM in a way that shows that SETH implies the APSP or k-SUM conjectures. [17, 18] used the existence and conjectured non-existence of efficient polynomial formulations of problems to prove it's unlikely that SETH could even imply a super-linear lower bound for k-SUM. These known techniques do not seem to give barriers for reducing between k-CYC problems like we do in Theorem 1.5 above, since those problems have similar nondeterministic and polynomial formulation complexity.

Fine-Grained Cryptography. The idea of trying to build fine-grained cryptographic protocols with polynomial gaps between the adversaries and honest parties relying on fine-grained conjectures has been studied previously. Prior works have shown that it is possible to construct proof of works [11, 12] from worst-case assumptions. More recently, it was shown that fine-grained permissionless consensus could be constructed in the random beacon model from the worst-case orthogonal vectors conjecture [10]. Other works have also considered building fine-grained primitives from average-case conjectures. For example, it was shown that fine-grained constructions for one-way functions and key-exchange from average-case fine-grained complexity assumptions [63, 35]. More recent work [26] presents fine-grained one-way functions from strong one-way average-case hardness as well as impossibility results for fine-grained one-way functions from weaker average-case hardness assumptions. Prior works also studied one-way functions and pseudorandom generators against NC^1 and AC^0 circuit adversaries [37] as well as fine-grained key exchange where security is proven in the generic group model [4, 15] that do not utilize fine-grained assumptions. In a somewhat different direction, it was shown that certain average-case assumptions about k-XOR enables building public-key encryption from weaker variants of LPN [6]. Beyond constructions, we note that fine-grained complexity problems have often appeared in various applications of cryptography such as k-SUM and k-XOR in cryptanalysis [29, 56, 16, 73, 24, 65, 42]. Finally, cryptographic primitives have been also built using problems similar to planted distributions studied in average-case fine-grained conjectures such as planted cliques [58, 8] as well as planted subset sum [67].

2 Preliminaries and Definitions

2.1 Notation

For a finite set S we use $\mathcal{U}(S)$ to denote the uniform distribution over S. For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. Given a list of L of length n, we denote the i-th entry by $L[i]$ for any $i \in [n]$. For two strings x, y, we use $x \| y$ to denote their concatenation.

2.2 k-SUM, k-PROD and k-CYC

We start with preliminary definitions to define problems and input distributions. In many cases, our notation follow similarly to prior work such as [6].

Group Ensembles. In the majority of our problems, we will consider an infinite sequence of finite abelian groups $\mathsf{G} = \{\mathbb{G}^{(n)}\}_{n\in\mathbb{N}^+}$ that we denote as the group ensemble. One can view a group ensemble as defining the underlying group $\mathbb{G}^{(n)}$ drawing n elements each uniformly at random from $\mathbb{G}$. For convenience, we may drop the subscript and denote group ensembles by $\mathsf{G} = \{\mathbb{G}^{(n)}\}$. We may also drop the superscript and refer to the group as $\mathbb{G}$ when n can be inferred. At times, we may denote the group ensemble along with an operation as $(\mathsf{G}, \odot)$. When it is clear from context, we will typically use $+$ for additive groups and $\cdot$ for multiplicative groups. In our work, we will exclusively consider group ensembles with efficient algorithms for sampling and performing group operations.

Definition 2.1 (Efficient Group Ensembles)**.** A group ensemble $\mathsf{G} = \{\mathbb{G}^{(n)}\}$ with group operation $\odot$ is efficient if it satisfies the following properties:

- There exists an algorithm that given as input $n \in \mathbb{N}^+$ samples a random element from $\mathbb{G}^{(n)}$ according to $\mathcal{D}^{(n)}$ in time $O(\mathsf{polylog}|\mathbb{G}^{(n)}|)$.
- There exists an algorithm that given as inputs $n \in \mathbb{N}^+$ and $x, y \in \mathbb{G}^{(n)}$ is able to compute the output of the group operation $x \odot y$ in time $O(\mathsf{polylog}|\mathbb{G}^{(n)}|)$.

Groups Ensembles Instantiations. In our work, we will especially interested in two specific group ensembles that will directly correlate to k-SUM, k-PROD and k-CYC. Each group ensemble is parameterized by a function $m(n)$ that denotes the size of the group with respect to the input size n.

- The group ensemble for k-SUM will defined by the additive group of integers modulo $m(n)$ denoted by $\mathbb{Z}_{m(n)}$ as follows:
$$\mathsf{G}^{(m)}_{k\text{-}\mathsf{SUM}} = \{\mathbb{Z}_{m(n)}\}_{n\in\mathbb{N}^+}.$$
- The group ensemble for k-PROD will defined by the multiplicative group of integers modulo $m(n)$ denoted by $\mathbb{Z}^\times_{m(n)}$ as follows:
$$\mathsf{G}^{(m)}_{k\text{-}\mathsf{PROD}} = \{\mathbb{Z}^\times_{m(n)}\}_{n\in\mathbb{N}^+}.$$

- The group ensemble for k-CYC will defined using any infinite sequence of cyclic groups of order $m(n)$ denoted by $\mathbb{G}_{m(n)}$ as follows:

$$\mathsf{G}^{(m)}_{k\text{-}\mathsf{CYC}} = \{\mathbb{G}_{m(n)}\}_{n\in\mathbb{N}^+}.$$

- The group ensemble for k-LIST will defined using any infinite sequence of abelian groups of order $m(n)$ denoted by $\mathbb{G}_{m(n)}$ as follows:

$$\mathsf{G}^{(m)}_{k\text{-}\mathsf{LIST}} = \{\mathbb{G}_{m(n)}\}_{n\in\mathbb{N}^+}.$$

We choose to be specific for k-SUM and k-PROD to work over the integers modulo $m(n)$ whereas we are generic for k-CYC and consider arbitrary cyclic groups of size $m(n)$ with an identity element 1. We do this to allow readers to easily infer whether we are working with k-SUM, k-PROD or k-CYC throughout the work. We choose more generality of k-CYC as we will consider relations with discrete logarithms and Diffie-Hellman problems where we wish to cover all possible types of cyclic groups used currently in practice such as NIST P-256, Curve25519 and DSA. For convenience, we will drop the function notation and consider the group size by m in many places.

Finally, we note our definition of group ensembles is general enough that it can be extended to other problems such as k-XOR where we would define the group ensembles as Galois fields of size $2^{m(n)}$ where $m(n)$ is the number of bits.

k-LIST Search Problems. In our work, we will consider the average-case variants of k-SUM, k-PROD and k-CYC. Therefore, we will only consider inputs that are drawn from the distribution equipped with each group ensemble (and not define a worst-case version of these problems). For the input distribution, we will consider the case where the inputs are k lists, $L_1, \dots, L_k$ each of size n and where each element is drawn uniformly at random $\mathbb{G}^{(n)}$ that we denote by $\mathcal{D}^{(n)}_U$.

We start with the search version of the problems where the goal is to output a sequence of k indices, $y_1, \dots, y_n \in [n]^k$ corresponding to a solution. Suppose the group operation is denoted by $\odot$ and the identity of the group is denoted 1. Then, a solution means that $L[y_1] \odot \dots \odot L[y_k] = 1$.

Definition 2.2 (k-$\mathsf{LIST}^{\mathsf{G}}$ Search Problem)**.** For $k \geq 0$ and group ensemble $(\mathsf{G}, \odot)$, an algorithm A correctly solves the k-$\mathsf{LIST}^{\mathsf{G}}$ search problem if it outputs the following in each corresponding scenario:

- If there exists a solution $(y_1, \dots, y_k) \in [n]^k$ such that $L_1[y_1] \odot \dots \odot L_k[y_k] = 1$, then A outputs any such solution.
- If no solution exists, A outputs $\perp$.

We say algorithm A has error probability $\gamma(n)$ if

$$\Pr[A(L_1, \dots, L_k) \text{ is correct}] \geq 1 - \gamma(n)$$

where the randomness is over the internal coin tosses of A and random choice of $L_1, \dots, L_k$ from $\mathcal{D}^{(n)}_U$.

For convenience and simplicity, we will consider error probability as γ dropping the function notation in many places.

Naturally, we can now define the k-SUM, k-PROD and k-CYC problems where we plug in the correct group ensemble along with the group identity.

Definition 2.3 (k-SUM Search Problem)**.** The k-SUM^m search problem is the k-LIST search problem with group ensemble $(\mathsf{G}^{(m)}_{k\text{-}\mathsf{SUM}}, +)$.

Definition 2.4 (k-PROD Search Problem)**.** The k-PROD^m search problem is the k-LIST search problem with group ensemble $(\mathsf{G}^{(m)}_{k\text{-}\mathsf{PROD}}, \cdot)$.

Definition 2.5 (k-CYC Search Problem)**.** The k-$\mathsf{CYC}^{\mathsf{G}}$ search problem is the k-LIST search problem with cyclic group ensemble $(\mathsf{G}, \cdot)$.

We will omit the superscript m and G when it is obvious from context.

Existence of Solutions. Before we present our conjectures, we start by discussing the probability that there exists a solution. Note, one could consider a trivial algorithm to solve the search problem if the probability that there exists a solution is very small. The naive algorithm simply outputs $\perp$ and is incorrect with the same probability that a solution exists. Therefore, we must pick the group size $m = m(n)$ carefully to ensure that there exists a solution with some reasonable probability. We can compute the probability that there exists a solution for either k-SUM and k-CYC. For the case when $n^k/2 \leq m$, we show

$$\frac{n^k}{m} - \frac{\binom{n^k}{2}}{m^2} \leq \Pr[L_1, \ldots, L_k \text{ contains a solution}] \leq \frac{n^k}{m}$$

where the probability is over the random choice of $L_1, \ldots, L_k$ from the uniform distribution $\mathcal{D}^{(n)}_U$. When $m < n^k$, we are able to show that

$$\Pr[L_1, \ldots, L_k \text{ contains a solution}] \geq 1 - 2^{1-n/m^{1/k}}$$

where the probability is also over the uniform distribution of $L_1, \ldots, L_k$. For the proof of these bounds please see Appendix B.

We note that prior work [6] considered densities that approximated the number of expected solutions. For our work, we will only care whether solutions exist with probability larger than the corresponding target error probability to rule out the trivial algorithm. So, we will only be in error probability $\gamma < n^k/m$.

Our Conjectures. We now state our average-case fine-grained conjectures with respect to k-SUM, k-PROD and k-CYC. In general, we will consider error probabilities $1/\mathsf{poly}(n)$ and group sizes $m = \Theta(n^k)$. Note, this implies that solutions will exist with probability $\Omega(1)$ meaning that our choice of group size m and error γ rules out the trivial algorithm of outputting $\perp$ is not possible.

Conjecture 3 (AVG-k-SUM Conjecture)**.** *For any integer $k \geq 3$, group size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-SUM^m search problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.*

Note, we require that $m = O(n^k)$ to rule out the trivial algorithm of outputting $\perp$. It turns out that it is also important that $m = \Omega(n^k)$ due to known faster algorithms. For the case when $m = O(n^{k-\epsilon})$ for some constant $\epsilon > 0$, there are well-known algorithms [80] that solve the problem in $O(n^{\lceil k/2 \rceil - \epsilon'})$ time for some constant $\epsilon' > 0$ (see Appendix A). We note there also exists algorithms to solve the problem in $O(n + m \log m)$ time using fast Fourier transforms (see Chapter 30.1 in [28]) even in the worst-case. Note, this runs faster than the conjectured runtime above when $m = O(n^{\lceil k/2 \rceil - \epsilon})$ for any constant $\epsilon > 0$. Therefore, it is critical that we consider large enough modulus $m = \Omega(n^k)$. As a note, it is possible to expand our conjecture to encompass both slightly smaller and larger moduli. For example, we could consider $n^{k-o(1)} \leq m \leq n^{k+o(1)}$. To our knowledge, there are no algorithms beating our conjecture for $m = n^{k-o(1)}$. For $m = n^{k+o(1)}$, we note that a solution exists with probability $1/n^{o(1)} > 1/\mathsf{poly}(n)$ meaning the trivial algorithm of outputting $\perp$ is ruled out. To be passive, we only consider modulus of size $m = \Theta(n^k)$.

Conjecture 4 (AVG-k-PROD Conjecture)**.** *For any integer $k \geq 3$, group size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-PROD^m search problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.*

Conjecture 5 (AVG-k-CYC Conjecture)**.** *For any integer $k \geq 3$, cyclic group ensemble G with size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-$\mathsf{CYC}^{\mathsf{G}}$ search problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.*

Interestingly, unlike the k-SUM problem, we are unaware of any faster algorithms for k-PROD or k-CYC in the random setting where the group size m is small such as $m = O(n^{k-\epsilon})$ for some constant $\epsilon > 0$. In particular, the faster algorithms for k-SUM do not seem to directly translate k-CYC (see Appendix A for more details). Nevertheless, we use the same choice of $m = \Theta(n^k)$ to allow easily working with both k-SUM and k-CYC together.

Decision Variants. We can also consider the decision version of the problems where it simply suffices for a correct algorithm A to detect whether a solution or not and output the corresponding bit. Note, the decision problem is easier than the search problem, but there are prior reductions (such as [6]) that show the problems are equivalent in the sparse regime when $m \geq c \cdot n^k$ for some constant $c > 1$. In our work, we will mostly concern ourselves with the search problems. However, these reductions mean our results also apply to the decision variants. We formally present the decision problems in Appendix C as well as their relations with the search problem.

Planted Variants. We can also consider the search problem over a modified distribution where a random solution is planted to ensure a solution always exists (regardless of the choice of m). In more detail, the k lists are first drawn from $\mathcal{D}_U^{(n)}$. Afterwards, a uniformly random set of indices $y_1, \ldots, y_k \in [n]^k$ are chosen and the entry $L_1[y_1]$ is modified to ensure that $y_1, \ldots, y_k$ is a solution such that $L_1[y_1] \odot \ldots \odot L_k[y_k] = 1$. In this case, it only makes sense to consider the search problem as the decision problem is trivial. Once again, there are known relations between the search and planted search variants of k-SUM, k-PROD and k-CYC presented in [6] in the dense setting where $m \leq c \cdot n^k$ for some constant $c < 1$. We present formal definitions and their relations in Appendix D.

Single List Variants. In our problems, we consider the inputs to be k lists each of size n. However, we note that many prior works have considered a slightly different variant where the input is a single list of size n. The goal remains the same with the exception that all k elements of the solution come from the same list now. These are known to be equivalent in the worst-case, but it is unclear if they are equivalent in the average-case setting that we focus on in our work. In Appendix E, we show that any algorithm for the k list variant may be used to solve the single list variant in the sparse regime when $m \geq c \cdot n^k$ for some constant $c > 1$. Even though this is only one direction, it allows us to relate some of our results to the single list variant as well.

2.3 k-SUM, k-PROD and k-CYC with Preprocessing

Next, we consider variants in the indexing or preprocessing setting that are more akin to data structure problems. In these problems, the lists are given ahead of time and may be preprocessed by (computationally unbounded) algorithms into a data structure of bounded size S. Afterwards, the data structure is given some queries that wish to be answered quickly in time T.

The indexing problems for k-SUM were first studied by Demaine and Vadhan [38] with several follow-ups [49, 51, 62]. In this case, the input consists of $k-1$ lists of size n that may be preprocessed into some data structure **DS**. Afterwards, a query algorithm receives a target t and data structure **DS** with the goal of finding $k-1$ entries one from each list that sum to the target t. We generalize this to arbitrary groups as follows:

Definition 2.6 (k-LIST Indexing Problem)**.** For $k \geq 0$ and group ensemble $(\mathsf{G}, \odot)$, a tuple of algorithms (P, Q) solves the k-LIST indexing problem if the preprocessing algorithm $P(L_1, \ldots, L_k)$ outputs some data structure **DS** such that the query algorithm Q outputs the following in each corresponding scenario:

- Given target $t \in \mathsf{G}$, $Q(t, \mathbf{DS})$ outputs $(y_1, \ldots, y_{k-1}) \in [n]^{k-1}$ satisfying the equation $L_1[y_1] \odot \ldots \odot L_{k-1}[y_{k-1}] = t$ if such a solution exists.
- Given target $t \in \mathsf{G}$, $Q(t, \mathbf{DS})$ outputs $\perp$ if no solution exists.

We say algorithm A has error probability $\gamma(n)$ if

$$\Pr[Q(t, \mathbf{DS}) \text{ is correct} \mid \mathbf{DS} \leftarrow P(L_1, \ldots, L_{k-1})] \geq 1 - \gamma(n)$$

where the randomness is over the internal coin tosses of P and Q, the random choice of $L_1, \ldots, L_{k-1}$ from $\mathcal{D}_U^{(n)}$ and uniformly random t.

Once again, we can now define k-SUM, k-PROD and k-CYC indexing search problems by plugging in the according group ensemble.

Definition 2.7 (k-SUM-Index Problem)**.** The k-SUM-Index^m problem is the k-LIST indexing problem with group ensemble $(\mathsf{G}^{(m)}_{k\text{-SUM}}, +)$.

Definition 2.8 (k-PROD-Index Problem)**.** The k-PROD-Index^m problem is the k-LIST indexing problem with group ensemble $(\mathsf{G}^{(m)}_{k\text{-PROD}}, \cdot)$.

Definition 2.9 (k-CYC-Index Problem)**.** The k-CYC-Index$^{\mathsf{G}}$ problem is the k-LIST indexing problem with cyclic group ensemble G.

Note, there are a couple straightforward algorithms. The first completely foregoes any non-trivial usage of the space S and simply stores the $k-1$ lists using $S = O(nk) = O(n)$ space since we suppose k is constant. Afterwards, we run the trivial algorithm to solve k-SUM that runs in time $T = O(n^{\lceil (k-1)/2 \rceil})$. On the other extreme, one could use the space to store all possible k-tuples sorted using space $S = O(n^{k-1})$ and answer queries in $T = \tilde{O}(1)$ time. Lastly, prior works [51, 62] have shown algorithms obtaining algorithms with space S and query time T satisfying $S^3 T = \tilde{O}(n^{3(k-1)})$. For any $\delta > 0$, this means there exists algorithms $S = \tilde{O}(n^{k-1-\delta})$ space and query time $T = \tilde{O}(n^{3\delta})$. One can naturally conjecture that no algorithm can beat either of the above trivial algorithms. In our work, we will be interested in the average-case variant with $k-1$ lists each consisting of n random elements first studied in [51]. We present our k-SUM-Index conjecture below:

Conjecture 6 (AVG-k-SUM-Index Conjecture)**.** *For every constant $\epsilon > 0$, constant $k \geq 3$ and group size $m = \Theta(n^k)$, there exists no algorithm that solves the k-SUM-Index^m problem in space S and time T satisfying $ST^2 = O(n^{k-1-\epsilon})$ with error probability $1/\mathsf{poly}(n)$.*

We note our above conjecture is weaker than the 3SUM-Indexing conjectures from prior works [38] where it is assumed that no algorithm obtains $ST = \Omega(n^2)$ as we consider a quadratic relation with the running time T. For example, our conjecture states that there exists no algorithm with time $T = O(n^{1/2-\epsilon})$ when using linear space while prior works have conjectured that there exists no linear space algorithm with time $T = O(n^{1-\epsilon})$. Indeed, our conjecture is only tight with the algorithm that utilizes $S = O(n^{k-1})$ space and $T = \tilde{O}(1)$ time. The algorithm on the other end using linear space of $S = O(n)$ and time $T = O(n^{\lceil (k-1)/2 \rceil})$ would result in $ST^2 = O(n^k)$ for odd k. For even k, it would be $ST^2 = O(n^{k+1})$. In fact, one could also present another conjecture for this setting where for any linear space algorithm with $S = O(n)$ would require time $T = O(n^{\lceil (k-1)/2 \rceil})$. Prior works have done the opposite where the conjecture that algorithms with fixed time $T = \tilde{O}(1)$ must use space $S = \Omega(n^2)$ such as [38, 49].

Similar to our prior choices of m, we require sufficiently large $m = \Omega(n^{k-1})$ as, otherwise, one can run faster algorithms for k-SUM [80] to solve the problem in time $T = O(n^{(k-1)/2-\epsilon})$ for some $\epsilon > 0$ and using space $S = O(n)$ that would contradict the conjecture.

Naturally, we can also present a conjecture for k-PROD-Index and k-CYC-Index. We use the same parameters as our k-SUM-Index as the trivial algorithms are essentially the same in either case.

Conjecture 7 (AVG-k-PROD-Index Conjecture)**.** *For every constant $\epsilon > 0$, constant $k \geq 3$ and group size $m = \Theta(n^k)$, there exists no algorithm that solves the k-PROD-Index^m problem in space S and time T satisfying $ST^2 = O(n^{k-1-\epsilon})$ with error $1/\mathsf{poly}(n)$.*

Conjecture 8 (AVG-k-CYC-Index Conjecture)**.** *For every constant $\epsilon > 0$, constant $k \geq 3$ and cyclic group ensemble G with size $m = \Theta(n^k)$, there exists no algorithm that solves the k-CYC-Index$^{\mathsf{G}}$ problem in space S and time T satisfying $ST^2 = O(n^{k-1-\epsilon})$ with error $1/\mathsf{poly}(n)$.*

2.4 Zero-k-Clique

In the Zero-k-Clique problem, informally, we are given a complete k-partite graph in which each edge has a weight, and the goal is to find a k-clique such that the weights of all edges in the clique sum to 0, or declare that no such clique exists.

In our work, we will consider the average-case variants of Zero-k-Clique problem. For the input distribution, we will consider the complete k-partite graph where every edge weight is uniformly sampled from the additive group of integers modulo m denoted by $\mathbb{Z}_m$.

Definition 2.10 (Zero-k-Clique search problem)**.** For $k > 0$, an algorithm A correctly solves the Zero-k-Clique^m problem if, on input of complete k-partite graph G, it outputs the following in each corresponding scenario:

- If there exists a clique of size k whose edge weights sum to zero, A outputs such a zero clique.
- If no such solution exists, A outputs $\perp$.

We say algorithm A has error probability $\gamma(n)$ if

$$\Pr[A(G) \text{ is correct}] \geq 1 - \gamma(n)$$

over the internal coin tosses of A and random choice of G.

2.5 Discrete Logarithms

In our work, we will explore the relations between various fine-grained conjectures and discrete logarithm and other assumptions used in cryptography. Given a description of cyclic group $\mathbb{G}$ along with a generator g, the goal is to take an input of the form $y = g^x$ and computing x. The assumption that computing discrete logarithms is not computationally tractable lies at the foundation of many cryptography primitives including key exchange [39], public-key encryption [43, 31] and signatures [77, 55].

Definition 2.11 (DLog Problem)**.** For group ensemble $(\mathsf{G}, \cdot)$, an algorithm A solves the $\mathsf{DLog}^{\mathsf{G}}$ problem if, on input n and $y \in \mathbb{G}^{(n)}$, it outputs x such that $y = g^x$. An algorithm A has error probability $\gamma(m)$ if

$$\Pr[y = g^x \mid x \leftarrow A(n, g^x)] \geq 1 - \gamma(m)$$

where the randomness is over the choice of y and internal coin tosses of A.

Proving that computing discrete logarithms is computationally intractable remains elusive in general models. However, there is significant work in the generic group model [72, 79, 68] where algorithms can only use group operations as a blackbox. In some sense, algorithms in the generic group model are generic since they should apply to every group regardless of their underlying structure. More formally, the generic group model uses a random injective function σ mapping integers in $\mathbb{Z}_m$ representing the set of possible discrete logarithm answers to a set of labels $\mathcal{L}$ representing group elements in $\mathbb{G}$ with generator g. The group $\mathbb{G}$ can be described as $\{\sigma(0), \sigma(1), \ldots, \sigma(n-1)\}$ that is equivalent to $\{1, g, \ldots, g^{n-1}\}$.

In the generic group model, prior works have had success in proving lower bounds on the efficiency of algorithms computing discrete logarithms. Shoup [79] proved that any generic algorithm for solving discrete logarithms requires $\Omega(m^{1/2})$ time for cyclic groups $\mathbb{G}$ of order m. While the generic group model seems restrictive, it remains important as the best algorithms for popular elliptic curve groups used in practice are generic group model algorithms [45, 60]. Using the generic group as guidance, we can construct conjectures similar to fine-grained complexity with respect to discrete logarithms as follows:

Definition 2.12 (DLog Conjecture)**.** There exists an ensemble of cyclic groups $(\mathsf{G}, \cdot)$ of order $m = \mathsf{poly}(n)$ such that, for every constant $\epsilon > 0$, there exists no algorithm that solves the $\mathsf{DLog}^{\mathsf{G}}$ problem in time $O(m^{1/2-\epsilon})$ with error $1/\mathsf{poly}(m)$.

Similar to the indexing versions of k-SUM and k-CYC, we can consider the discrete logarithmic with preprocessing algorithm. In particular, a preprocessing algorithm can construct a data structure of size S for a specific group $\mathbb{G}$ and generator g. Afterwards, a query algorithm is given a target $y = g^x$ with the goal of outputting x efficiently. In the generic group model, Corrigan-Gibbs and Kogan [30] showed that no generic algorithms with S space and T time can achieve better than $ST^2 = \Omega(m)$ for groups of order m and constant error probability. Once again, the best known algorithms for popular groups used in practice are generic algorithms matching this bound (see [69, 64, 19]). Therefore, we can make a similar conjecture about the preprocessing case.

Definition 2.13 (DLog-Preprocess Conjecture)**.** There exists an ensemble of cyclic groups $(\mathsf{G}, \cdot)$ of order $m = \mathsf{poly}(n)$ such that, for every constant $\epsilon > 0$, there exists no algorithm that solves the $\mathsf{DLog\text{-}Preprocess}^{\mathsf{G}}$ problem in space S and time T satisfying $ST^2 = O(m^{1-\epsilon})$ with error probability $1/\mathsf{poly}(m)$.

Fixed vs. Random Generator. As a note, we consider the problem where the generator is fixed and, essentially, part of the cyclic group's description. Prior works have also studied the setting where the generator is randomly selected and also given as input. The fixed and random generator problems are known to be equivalent for the constant error probability case in the generic group model. However, prior work [14] showed that algorithms for the random generator problem are more inefficient in the sub-constant error probability. It turns out all our results and reductions are agnostic to this difference and may be proved in either setting. For example, it is easy to extend our result to the random generation setting if one wishes to consider the sub-constant error.

3 Algorithms when One-Way Functions Don't Exist

In this section, we discuss the algorithmic consequences for fine-grained problems when one-way functions do not exist. Our main goal is to prove Theorem 1.2 and Theorem 1.3.

Lemma from Universal One-Way Functions. We call a function f *length-regular* if for any two inputs x_1, x_2 such that $|x_1| = |x_2|$, it is also true that $|f(x_1)| = |f(x_2)|$. We call a length-regular function *length-increasing* if for any two inputs x_1, x_2 such that $|x_1| < |x_2|$, it is also true that $|f(x_1)| < |f(x_2)|$. Note that it thus holds that $|x| \le |f(x)|$ for any x.

We will use the following lemma that is proved via universal one-way functions. Basically, it states that, if there is no one-way function, then there exists a universal constant c such that every n^ℓ-time function can be inverted in $n^{\ell c}$ time for infinitely many n.

Lemma 3.1. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 2$, such that for any constant $\ell \ge 2$ and any n^ℓ-time (length-increasing) Turing machine M where n is the input length, there exists a (possibly randomized) algorithm A such that for infinitely many n,*

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[M(A(1^n, M(x))) = M(x)] \ge 1 - \frac{2^{2|M|}}{n^{\ell d}}$$

where $|M|$ is the length of the code of M, and the running time of $A(1^n, M(x))$ is $n^{\ell c}$ for $x \in \{0,1\}^n$.

3.1 Planting algorithms and general statements

In this section, we define planting algorithms for search problems. Roughly speaking, a planting algorithm efficiently generates an instance of the problem with a solution from some randomness. Moreover, we can efficiently find a solution using the randomness.

Definition 3.2 (Planting algorithm for search problems)**.** Let $P(1^n)$ be a search problem where n is a size parameter such that the length of instances is a strictly increasing polynomial of n to be denoted by $N(n)$. A pair of (deterministically) polynomial-time computable functions (f, g) is called a *planting algorithm* for P if the following hold:

- There is a function $N_I(n)$ that is a strictly increasing polynomial of n. We call N_I the input length of (f, g).
- f takes $x \in \{0,1\}^{N_I(n)}$ as input and outputs an instance $y \in \{0,1\}^{N(n)}$ of $P(1^n)$ that has a solution.
- g takes $x \in \{0,1\}^{N_I(n)}$ as input and outputs a solution to $f(x)$.

We call a planting algorithm (f, g) $T(\cdot)$-time if both $f(x)$ and $g(x)$ can be computable in time $T(|x|)$.

We also define properties for planting algorithms, which will be helpful when we analyse the failure probability for k-$\mathsf{LIST}^{\mathsf{G}}$ over uniform input distribution.

Definition 3.3. Let $P(1^n)$ be a search problem and (f, g) be a planting algorithm for P with input length N_I.

- (f, g) is *surjective* if for any instance $y \in \{0,1\}^{N(n)}$ of $P(1^n)$ that has a solution, there exists $x \in \{0,1\}^{N_I(n)}$ such that $f(x) = y$.
- (f, g) is $\beta(\cdot)$-*almost-expanding* if for all n that $N_I(n) \le N(n) + \beta(n)$.
- (f, g) is $\beta(\cdot)$-almost-good if it is $\beta(\cdot)$-almost-expanding and surjective.

Below we show that, for a generic problem P, if there exists a almost-good planting algorithm that runs in a fixed polynomial time, then there is an algorithm that solves P on average over uniform distribution in a fixed polynomial time with inverse polynomial failure probability.

Theorem 3.4. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 2$, such that for any search problem $P(1^n)$ of which n is a size parameter and $N(n) \ge n$ is the length of a instance, for any constant $\ell \ge 2$ and any function $\beta(\cdot)$, if there exists a $\beta(\cdot)$-almost-good $(\cdot)^\ell$-time planting algorithm (f, g) for P with input length $N_I(n)$, then there is an $O((N_I(n))^{\ell c})$-time algorithm A that solves P over $\mathcal{U}(\{0,1\}^{N(n)})$ for infinitely many n with failure probability at most $2^{2|f|+\beta(n)}/(N_I(n))^{\ell d}$, where $|f|$ is the code length of the $(\cdot)^\ell$-time algorithm for f.*

Proof. Suppose f is a function defined only over input length $N_I(n)$ where N_I is a polynomial, we use padding to extend f to a function f' over every input length as follows: for every input length N'_I such that $N_I(n) \le N'_I \le N_I(n+1)$, for any input $z \in \{0,1\}^{n'}$, let $z = x \| z'$ where $|x| = N_I(n)$ and define $f'(z) = f(x) \| z'$.

By Lemma 3.1, assuming one-way function does not exist, for any constant $d > 0$, there exists a constant $c > 2$ such that for any constant $\ell \ge 2$, if (f, g) is a $(\cdot)^\ell$-time planting algorithm for P, let f' be defined as in the previous paragraph, then there exists an $(N'_I)^{\ell c}$-time algorithm B' such that for infinitely many N'_I,

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N'_I})}[f'(B'(1^{N'_I}, f'(x))) = f'(x)] \ge 1 - \frac{2^{2|f|}}{(N'_I)^{\ell d}},$$

which implies that there exists an $O((N_I(n))^{\ell c})$-time algorithm B such that for infinitely many n,

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N_I(n)})}[f(B(1^{N_I(n)}, f(x))) = f(x)] \ge 1 - \frac{2^{2|f|}}{(N_I(n))^{\ell d}}.$$

Let A be the following algorithm:

1. On input y, find the unique n such that $N(n) = |y|$.
2. Run $B(1^{N_I(n)}, y)$, and suppose the output is x'.

3. If $f(x') = y$, output $g(x')$, otherwise output $\perp$, indicating there is no solution.

Since the running time of B is $(N_I(n))^{\ell c}$, the running time of A is $O((N_I(n))^{\ell c})$.

Note that by definition of planting algorithms, when y does not have a solution, y is not in the image of f, so $f(x') \neq y$ and the output of $A(y)$ is always correct in this case. We thus have

$$\begin{aligned}
&\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^{N(n)})}[\text{The output of } A(y) \text{ is not correct}] \\
&= \frac{1}{2^{N(n)}} \sum_{y \in \{0,1\}^{N(n)}} \Pr[\text{The output of } A(y) \text{ is not correct}] \\
&= \frac{1}{2^{N(n)}} \sum_{y \in f(\{0,1\}^{N_I(n)})} \Pr[\text{The output of } A(y) \text{ is not correct}] \\
&\leq \frac{1}{2^{N(n)}} \sum_{y \in f(\{0,1\}^{N_I(n)})} \Pr[f(B(1^{N_I(n)}, y)) \neq y] \qquad (\text{By definition } g(x') \text{ outputs a solution to } y \text{ if } f(x') = y) \\
&\leq \frac{1}{2^{N(n)}} \sum_{x \in \{0,1\}^{N_I(n)}} \Pr[f(B(1^{N_I(n)}, f(x))) \neq f(x)] \qquad (\text{Since } (f,g) \text{ is surjective}) \\
&= \frac{2^{N_I(n)}}{2^{N(n)}} \Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N_I(n)})}[f(B(1^{N_I(n)}, f(x))) \neq f(x)] \\
&\leq \frac{2^{N_I(n)}}{2^{N(n)}} \cdot \frac{2^{2|f|}}{(N_I(n))^{\ell d}} \\
&\leq \frac{2^{2|f| + \beta(n)}}{(N_I(n))^{\ell d}} \qquad (\text{Since } (f,g) \text{ is expanding, } N_I(n) \leq N(n) + \beta(n))
\end{aligned}$$

for infinitely many n. □

We also show a similar result for the planted distribution instead of the uniform distribution.

Theorem 3.5. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 2$, such that for any search problem $P(1^n)$ of which n is a size parameter and $N(n) \geq n$ is the length of a instance, for any constant $\ell \geq 2$, if there exists a $(\cdot)^\ell$-time planting algorithm (f, g) for P with input length $N_I(n)$, then there is an $O((N_I(n))^{\ell c})$-time algorithm A that solves P over $f(\mathcal{U}(\{0,1\}^{N_I(n)}))$ for infinitely many n with failure probability at most $2^{2|f|}/(N_I(n))^{\ell d}$, where $|f|$ is the code length of the $(\cdot)^\ell$-time algorithm for f.*

Proof. Similar to the proof of Theorem 3.4, we can pad a function f defined only over input length $N_I(n)$ into a function f' defined over every input length.

Then by Lemma 3.1, assuming one-way function does not exist, for any constant $d > 0$, there exists a constant $c > 2$ such that for any constant $\ell \geq 2$, if (f, g) is a $(\cdot)^\ell$-time planting algorithm for P, then there exists an $(N_I(n))^{\ell c}$-time algorithm B such that for infinitely many n,

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N_I(n)})}[f(B(1^{N_I(n)}, f(x))) = f(x)] \geq 1 - \frac{2^{2|f|}}{(N_I(n))^{\ell d}}.$$

Let A be the following algorithm:

1. On input y, find the unique n such that $N(n) = |y|$.

2. Run $B(1^{N_I(n)}, y)$, and suppose the output is x'.

3. If $f(x') = y$, output $g(x')$, otherwise output $\perp$, indicating there is no solution.

Since the running time of B is $(N_I(n))^{\ell c}$, the running time of A is $O((N_I(n))^{\ell c})$.

We have

$$\begin{aligned}
&\Pr_{y \leftarrow f(\mathcal{U}(\{0,1\}^{N_I(n)}))}[\text{The output of } A(y) \text{ is not correct}]\\
=\ &\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N_I(n)})}[\text{The output of } A(f(x)) \text{ is not correct}]\\
\leq\ &\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^{N_I(n)})}[f(B(1^{N_I(n)}, f(x))) = f(x)] \qquad \text{(By definition } g(x') \text{ outputs a solution to } y \text{ if } f(x') = y)\\
\leq\ &\frac{2^{2|f|}}{(N_I(n))^{\ell d}}
\end{aligned}$$

for infinitely many n. □

3.2 Results for k-LIST$^{\mathsf{G}}$

In this subsection, we will always consider the k-LIST$^{\mathsf{G}}$ problem where the group size $m(n) := |\mathbb{G}^{(n)}|$ strictly increases with n.

We first show that there is a quadratic-time $2k \log_2 \log_2 n$-almost-good planting algorithm for search k-LIST$^{\mathsf{G}}$.

Example 3.6 (Planting algorithm for search k-LIST$^{\mathsf{G}}$)**.** We define a function $f_{k,m}$ as follows:

1. Parse the input as $(y_1, y_2, \ldots, y_k, L_1, L_2, \ldots, L_{k-1}, L_k[1 : y_k - 1], L_k[y_k + 1 : n])$, where $y_1, \ldots, y_k \in [n]$, $L_1, \ldots, L_{k-1} \in (\mathbb{G}^{(m)})^n$, $L_k[1 : y_k - 1] \in (\mathbb{G}^{(m)})^{y_k - 1}$, and $L_k[y_k + 1 : n] \in (\mathbb{G}^{(m)})^{n - y_k - 1}$.

2. Let $L_k[y_k] = (L_1[y_1] \odot L_2[y_2] \odot \cdots \odot L_{k-1}[y_{k-1}])^{-1}$, and let L_k be $L_k[1 : y_k - 1]$, $L_k[y_k]$ and $L_k[y_k + 1, n]$ concatenated.

3. Output (the encoding of) $(L_1, L_2, \ldots, L_k)$.

We also define a function $g_{k,m}$ as follows:

1. Parse the input as $(y_1, y_2, \ldots, y_k, L_1, L_2, \ldots, L_{k-1}, L_k[1 : y_k - 1], L_k[y_k + 1 : n])$, where $y_1, \ldots, y_k \in [n]$, $L_1, \ldots, L_{k-1} \in (\mathbb{G}^{(m)})^n$, $L_k[1 : y_k - 1] \in (\mathbb{G}^{(m)})^{y_k - 1}$, and $L_k[y_k + 1 : n] \in (\mathbb{G}^{(m)})^{n - y_k - 1}$.

2. Output $(y_1, y_2, \ldots, y_k)$.

Note that $f_{k,m}(y_1, y_2, \ldots, y_k, L_1, L_2, \ldots, L_{k-1}, L_k[1 : y_k - 1], L_k[y_k + 1 : n])$ is always an instance of k-LIST$^{\mathsf{G}}$ that has a solution $(y_1, \ldots, y_k)$, which is also $g_{k,m}(y_1, y_2, \ldots, y_k, L_1, L_2, \ldots, L_{k-1}, L_k[1 : y_k - 1], L_k[y_k + 1 : n])$. Clearly the running time of $f_{k,m}$ and $g_{k,m}$ are quasilinear in the input length, so $(f_{k,m}, g_{k,m})$ is a $(\cdot)^2$-time planting algorithm.

Moreover, for any k-$\mathsf{LIST}^{\mathsf{G}}$ instance $(L_1, L_2, \dots, L_k)$ that has a solution, suppose $(y_1, y_2, \dots, y_k)$ is a solution. Let $L_k[1 : y_k - 1]$ be the first $y_k - 1$ numbers in the list L_k and $L_k[y_k + 1 : n]$ be the last $n - y_k + 1$ numbers in L_k, and then $f_{k,m}(y_1, y_2, \dots, y_k, L_1, L_2, \dots, L_{k-1}, L_k[1 : y_k - 1], L_k[y_k + 1 : n]) = (L_1, L_2, \dots, L_k)$. Therefore, $(f_{k,m}, g_{k,m})$ is surjective.

The input length of $f_{k,m}$ is $k \log_2 n + ((k-1)n + n - 1)\log_2 m$, and its output length is $kn \log_2 m$, so $(f_{k,m}, g_{k,m})$ is $(2k \log_2 \log_2 n)$-almost-expanding if and only if $k \log_2 n \le 2k \log_2 \log_2 n + \log_2 m$, i.e. $m \ge n^k/(\log_2 n)^{2k}$.

Search k-$\mathsf{LIST}^{\mathsf{G}}$ on uniform distribution. We first show that when one-way functions do not exist and the group size $m \ge n^k/(\log_2 n)^{2k}$, then k-$\mathsf{LIST}^{\mathsf{G}}$ can be solved in a fixed polynomial time with inverse polynomial failure probability.

Corollary 3.7. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ be a group ensemble such that $m(n) \ge n^k/(\log_2 n)^{2k}$ for sufficiently large n. Then, there exists an $O((n \log m)^c)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over uniform distribution for infinitely many n with failure probability $O((\log_2 n)^{2k}/(n \log m)^d)$.*

Proof. Apply Theorem 3.4 with the good quasi-linear-time planting algorithm $(f_{k,m}, g_{k,m})$ in Example 3.6. Note that for fixed function m, the code length of $f_{k,m}$ is $\log_2 k + O(1)$. □

We also state the contrapositive of Corollary 3.7 here.

Corollary 3.8 (Contrapositive of Corollary 3.7)**.** *If there exists a constant $d > 0$, such that for any constant $c > 4$, there exists a k such that there is no $O((n \log m)^c)$-time algorithm that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over uniform distribution with failure probability $O((\log_2 n)^{2k}/(n \log m)^d)$ where $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ is a group ensemble such that $m(n) \ge n^k/(\log_2 n)^{2k}$ for sufficiently large n, then one-way functions exist.*

We also show a result similar to above when the group size $m \le n^k/(\log_2 n)^{2k}$, by using a average-case fine-grained reduction stated as below.

Lemma 3.9. *There is an oracle algorithm A satisfying the following: there exists a constant $n_0 > 0$ such that for all k, for all $n \ge n_0$, for all $(2k)^k \le m \le n^k/(\log_2 n)^{2k}$, if there exists an (possibly randomized) algorithm B solving search k-$\mathsf{LIST}^{\mathsf{G}}$ for size $\lfloor m^{1/k} \rfloor$ over a group $\mathbb{G}^{(m)}$ of size m on the uniform distribution with failure probability at most $1/8$, then A solves search k-$\mathsf{LIST}^{\mathsf{G}}$ for size n over $\mathbb{G}^{(m)}$ on uniform distribution in $\tilde{O}(n \log m)$ time making $(\log_2 n)^2$ oracle calls, with failure probability at most $1/n^{\log n}$ if each oracle query of A is answered using a call to B with the query as the input and the output as the answer.*

Proof. Let $n' = \lfloor m^{1/k} \rfloor$ and $s = (\log_2 n)^2$, then $n' \le n/s$. The oracle algorithm A works as follows:

1. Parse the input as $(L_1, L_2, \dots, L_k)$ where $L_1, \dots, L_k \in (\mathbb{G}^{(m)})^n$.
2. For $j \in [s]$, let $L_i^{(j)} := L_i[(j-1)n' + 1 : jn']$, which is the sublist of L_i containing the $((j-1)n' + 1)$-th to (jn')-th elements.

3. For every $j \in [s]$, make the query $Q_j := (L_1^{(j)}, L_2^{(j)}, \dots, L_k^{(j)})$ to the oracle. If there exists $j \in [s]$ such that the oracle response is $(y_1, y_2, \dots, y_k) \in [n']^k$ such that

$$\bigodot_{i=1}^{k} L_i[(j-1)n' + y_i] = \bigodot_{i=1}^{k} L_i^{(j)}[y_i] = 0,$$

then output $((j-1)n' + y_1, (j-1)n' + y_2, \dots, (j-1)n' + y_k)$. If there is no such j satisfying the above, then output $\perp$.

Note that on uniform input distribution, the oracle queries Q_j are independent and uniform. For a fixed j, by Lemma B.1, the probability that Q_j as an instance of search k-$\mathsf{LIST}^{\mathsf{G}}$ of size n' over $\mathbb{G}^{(m)}$ has a solution is at least

$$\frac{(n')^k}{m} - \frac{\binom{(n')^k}{2}}{m^2} \geq \frac{m^2 - (m-(n')^k)^2}{2m^2} \geq \frac{1}{2} - \left(1 - \left(\frac{n'}{n'+1}\right)^k\right)^2 \geq \frac{1}{4}.$$

Since the distribution of Q_j is uniform, the oracle response is the correct solution to search k-$\mathsf{LIST}^{\mathsf{G}}$ of size n' over $\mathbb{G}^{(m)}$ with probability at least $7/8$. Therefore, algorithm A finds a solution with probability at least $1 - (1/4 - 1/8)^s = 1/n^{\log n}$ on the uniform distribution, because $Q_j, j \in [s]$ are independent.

On the other hand, when the input does not have a solution, it is easy to see that for any $j \in [s]$, Q_j does not have a solution, so in this case A outputs $\perp$, which is correct.

Therefore, A solves search k-$\mathsf{LIST}^{\mathsf{G}}$ of size n over $\mathbb{G}^{(m)}$ on uniform distribution with failure probability at most $1/n^{\log n}$. The running time of A is $\tilde{O}(n \log m)$, with s oracle queries. $\square$

Using the above reduction, we have:

Theorem 3.10. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ be a group ensemble such that $m(n) \leq n^k/(\log_2 n)^{2k}$ and $(m(n))^{1/k} \leq (m(n-1))^{1/k} + 1$ for sufficiently large n. Then, there exists an $\tilde{O}(m^{c/k} + n \log m)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over uniform distribution for infinitely many n with failure probability at most $1/n^{\log n}$.*

Proof. Define $n' : \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}, n \mapsto \lfloor (m(n))^{1/k} \rfloor$, then we have for all k, $n'(n) \leq n/(\log_2 n)^2$, $n'(n) \leq n'(n-1) + 1$, and $n'(n) \geq \lfloor n^{1/k} \rfloor$.

If one-way functions do not exist, then by Corollary 3.7, for any constant $d > 0$, there exists a constant $c > 0$ such that for all k, there exists an $O((n' \log m)^c)$-time algorithm B that solves k-$\mathsf{LIST}^{\mathsf{G}}$ of size n' over $\mathbb{G}^{(m)}$ over uniform distribution for infinitely many n with failure probability $O((\log_2 n')^{2k}/(n' \log m)^d) \leq 1/8$ (note that we get infinitely many n' from Corollary 3.7, but it implies infinitely many n since $n'(n) \leq n'(n-1) + 1$).

Therefore by Lemma 3.9, for all k, we have an algorithm A' that runs in $\tilde{O}((n \log m))$ time, makes $(\log_2 n)^2$ oracle calls that is answered with B, and solves k-$\mathsf{LIST}^{\mathsf{G}}$ of size n over $\mathbb{G}^{(m)}$ on uniform distribution with failure probability at most $1/n^{\log n}$ for infinitely many n. We replace the oracle calls to B with a simulation of B, and thus for all k we get an algorithm that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ of size n over $\mathbb{G}^{(m)}$ over uniform distribution for infinitely many n with failure probability at most $1/n^{\log n}$, and has a running time

$$\tilde{O}((m^{1/k} \cdot \log m)^c \cdot (\log_2 n)^2 + n \log m) = \tilde{O}(m^{c/k} + n \log m). \qquad \square$$

Search k-$\mathsf{LIST}^{\mathsf{G}}$ on the planted distribution. In the planted distribution, we first uniformly sample an instance and then "plant" a random solution into it.

Corollary 3.11. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds. Let m be a function. Let $(f_{k,m}, g_{k,m})$ be the planting algorithm from Example 3.6. Then, for all k there exists an $O((kn \log m)^c)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over the planted distribution (i.e. $f_{k,m}(\mathcal{U}(\mathbb{Z}_m \times [n]^k \times \mathbb{Z}_m^{kn-1}))$) for infinitely many n with failure probability $O(k^2/(kn \log m)^d)$.*

Proof. Apply Theorem 3.5 with the quasi-linear-time planting algorithm $(f_{k,m}, g_{k,m})$ in Example 3.6. For fixed function m, the code length of $f_{k,m}$ is $\log k + O(1)$. □

Search k-$\mathsf{LIST}^{\mathsf{G}}$ on uniform distribution over instances with a solution.

Theorem 3.12. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ be a group ensemble such that $m(n) \geq n^k/(\log_2 n)^{2k}$. Then, for all k there exists an $O((n \log m)^c)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over the uniform distribution over all instances that have a solution for infinitely many n with failure probability $O((\log n)^{2k}/(n \log m)^d)$.*

Proof. Let $(f_{k,m}, g_{k,m})$ be the planting algorithm for search k-$\mathsf{LIST}^{\mathsf{G}}$ from Example 3.6. Let A be the algorithm in Corollary 3.11 that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over the distribution $f_{k,m}(\mathcal{U}([n]^k \times (\mathbb{G}^{(m)})^{kn-1}))$ for infinitely many n with failure probability $O(1/(n \log m)^{2d})$.

Let S be the set of k-$\mathsf{LIST}^{\mathsf{G}}$ instances that have a solution, then the support of $f_{k,m}(\mathcal{U}([n]^k \times (\mathbb{G}^{(m)})^{kn-1}))$ is S since $(f_{k,m}, g_{k,m})$ is surjective. Therefore, for any $y \in S$,

$$\Pr_{x \leftarrow f_{k,m}(\mathcal{U}([n]^k \times (\mathbb{G}^{(m)})^{kn-1}))}[x = y] \geq \frac{1}{n^k m^{kn-1}}.$$

For a fixed n, if $m \geq e^{-1/2} n^k$, then by Lemma B.1,

$$|S| \geq m^{kn} \left(\frac{n^k}{m} - \frac{\binom{n^k}{2}}{m^2} \right) = n^k m^{kn-1} \cdot \frac{2m + 1 - n^k}{2m} \geq \frac{n^k m^{kn-1}}{6}.$$

If $m \leq e^{-1/2} n^k$, then by Lemma B.2,

$$|S| \geq m^{kn} \left(1 - \frac{1}{2^{n/m^{1/k}-1}} \right) \geq n^k m^{kn-1} \cdot \frac{m}{n^k} \cdot (1 - 2^{1-e^{1/(2k)}}) \geq \frac{n^k m^{kn-1}}{(3k+1)(\log_2 n)^{2k}}$$

since $m \geq n^k/(\log_2 n)^{2k}$.

We thus have, for infinitely many n,

$$
\begin{aligned}
&\Pr_{x \leftarrow \mathcal{U}(S)}[A(x) \text{ is not a solution to } x] \\
&= \sum_{y \in S} \Pr[A(x) \text{ is not a solution to } x \mid x = y] \Pr_{x \leftarrow \mathcal{U}(S)}[x = y] \\
&\leq \frac{n^k m^{kn-1}}{|S|} \sum_{y \in S} \Pr[A(x) \text{ is not a solution to } x \mid x = y] \Pr_{x \leftarrow f_{k,m}(\mathcal{U}([n]^k \times (\mathbb{G}^{(m)})^{kn-1}))}[x = y] \\
&= \frac{n^k m^{kn-1}}{|S|} \Pr_{x \leftarrow f_{k,m}(\mathcal{U}([n]^k \times (\mathbb{G}^{(m)})^{kn-1}))}[A(x) \text{ is not a solution to } x] \\
&\leq (3k+1)(\log_2 n)^{2k} \cdot O(1/(n \log m)^{2d}) \\
&\leq O(1/(n \log m)^d). \qquad \square
\end{aligned}
$$

Theorem 3.13. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ be a group ensemble such that $m(n) \leq n^k/(\log_2 n)^{2k}$ and $(m(n))^{1/k} \leq (m(n-1))^{1/k} + 1$ for sufficiently large n. Then, there exists an $\tilde{O}(m^{c/k} + n \log m)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over the uniform distribution over instances with a solution for infinitely many n with failure probability $1/n^{\log n}$.*

Proof. By Lemma B.2, when sampling from the uniform distribution over all instances, the probability that the instance has a solution is at least $1 - 1/n^{\log n} \geq 1/2$. So the statement above directly follows from Theorem 3.10, as the failure probability here is at most twice the failure probability in Theorem 3.10. $\square$

Search k-$\mathsf{LIST}^{\mathsf{G}}$ on the uniform distribution over all instances with exactly one solution. We also show the following, as the uniform distribution over solutions with exactly one solution is what [63] used.

Theorem 3.14. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $\mathsf{G} = \{\mathbb{G}^{(m)}\}$ be a group ensemble such that $m(n) \geq n^k$. Then, there exists an $O((n \log m)^c)$-time algorithm A that solves search k-$\mathsf{LIST}^{\mathsf{G}}$ over the uniform distribution over all instances that have exactly solution for infinitely many n with failure probability $O(1/(n \log m)^d)$.*

The proof is almost the same as the case where $m \geq e^{-1/2} n^k$ in the proof of Theorem 3.12, since the same lower bound in Lemma B.1 also works for instances with exactly one solution. The proof is thus omitted.

3.3 Results for Zero-k-Clique

We give a planting algorithm for Zero-k-Clique and show Zero-k-Clique results similar to those for k-$\mathsf{LIST}^{\mathsf{G}}$ below.

Example 3.15 (Planting algorithm for search Zero-k-Clique)**.** Consider the search problem Zero-k-Clique of size n over $\mathbb{Z}_m$, in which m is regarded as a function of (k, n). Let $J := \{(i_1, i_2) \mid 1 \leq i_1 < i_2 \leq k\} \times [n]^2$. We define a function $f^{\mathsf{zkc}}_{k,m}$ as follows:

1. Parse the input as $((y_i)_{i\in[k]}, (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J\setminus(k-1,k,y_{k-1},y_k)})$, where $y_i \in [n]$ and $e_{\{(i_1,j_1),(i_2,j_2)\}} \in \mathbb{Z}_m$.
2. Let
$$e_{\{(k-1,y_{k-1}),(k,y_k)\}} = -\sum_{1\le i_1<i_2\le k,(i_1,i_2)\ne(k-1,k)} e_{\{(i_1,y_{i_1}),(i_2,y_{i_2})\}}.$$
3. Output (the encoding of) $(e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J}$.

We also define a function $g^{\mathsf{zkc}}_{k,m}$ as follows:

1. Parse the input as $((y_i)_{i\in[k]}, (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J\setminus(k-1,k,y_{k-1},y_k)})$, where $y_i \in [n]$ and $e_{\{(i_1,j_1),(i_2,j_2)\}} \in \mathbb{Z}_m$.
2. Output $(y_i)_{i\in[k]}$.

Note that $f^{\mathsf{zkc}}_{k,m}((y_i)_{i\in[k]}, (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J\setminus(k-1,k,y_{k-1},y_k)})$ is always an instance of Zero-k-Clique of size n over $\mathbb{Z}_m$ that has a solution $(y_1,\dots,y_k) = g^{\mathsf{zkc}}_{k,m}((y_i)_{i\in[k]}, (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J\setminus(k-1,k,y_{k-1},y_k)})$. Clearly that the running time of $f^{\mathsf{zkc}}_{k,m}$ and $g^{\mathsf{zkc}}_{k,m}$ are quasilinear in the input length, so $(f^{\mathsf{zkc}}_{k,m}, g^{\mathsf{zkc}}_{k,m})$ is a $(\cdot)^2$-time planting algorithm.

Moreover, for any instance $(e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J}$ of Zero-k-Clique with size n over $\mathbb{Z}_m$ that has a solution, suppose $(y_1,y_2,\dots,y_k)$ is a solution. Then, $(f^{\mathsf{zkc}}_{k,m}, g^{\mathsf{zkc}}_{k,m})$ is surjective, since

$$f^{\mathsf{zkc}}_{k,m}((y_i)_{i\in[k]}, (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J\setminus(k-1,k,y_{k-1},y_k)}) = (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J}.$$

The input length of $f^{\mathsf{zkc}}_{k,m}$ is $k\log_2 n + (\frac{1}{2}k(k-1)n^2 - 1)\log_2 m$, and its output length is $\frac{1}{2}k(k-1)n^2\log_2 m$, so $(f^{\mathsf{zkc}}_{k,m}, g^{\mathsf{zkc}}_{k,m})$ is $(2k\log_2\log_2 n)$-almost-expanding if and only if $k\log_2 n \le 2k\log_2\log_2 n + \log_2 m$, i.e. $m \ge n^k/(\log_2 n)^{2k}$.

Corollary 3.16. *If one-way functions do not exist, then for any constant $d > 0$, there exists a constant $c > 4$ such that the following holds for all k. Let $m : \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}$ be any function on n strictly increasing with n and computable in* poly $\log(n)$ *time, such that $m(n) \ge n^k/(\log_2 n)^{2k}$ for sufficiently large n. Then, there exists an $O((n^2\log m)^c)$-time algorithm A that solves search* Zero-k-Clique *for size n over $\mathbb{Z}_m$ over uniform distribution for infinitely many n with failure probability $O((\log_2 n)^{2k}/(n^2\log m)^d)$.*

Proof. Apply Theorem 3.4 with the good quasi-linear-time planting algorithm $(f^{\mathsf{zkc}}_{k,m}, g^{\mathsf{zkc}}_{k,m})$ in Example 3.15. Note that for fixed function m, the code length of $f^{\mathsf{zkc}}_{k,m}$ is $\log_2 k + O(1)$. □

We can actually prove a similar result for Zero-k-Clique for every result about k-LIST$^{\mathsf{G}}$ stated in Section 3.2 in similar fashions, and we omit them for brevity.

4 Relations between k-SUM, k-CYC and DLog

In this section, we present relationships between k-SUM, k-CYC and DLog. First, we show that there is an inherent barrier in being able to prove equivalences between k-SUM and k-CYC. In particular, proving an equivalence would show the DLog conjecture is false. In other words, this provides a strong barrier in proving reductions from k-CYC to k-SUM. The above results remains true even if we replace k-CYC with k-PROD. Next, we extend relationships between DLog and k-CYC to the setting of preprocessing. Finally, we present faster algorithms for k-SUM-Index and related problems in the average-case setting improving upon previous worst-case algorithms [51, 62].

4.1 Barriers to k-CYC, k-PROD and k-SUM Reductions

First, we note that one direction between k-SUM and k-CYC is quite easy. In particular, it is quite straightforward to construct an algorithm for k-SUM if one is given an efficient algorithm for k-CYC. If one is given an instance of k-SUM, it can easily be converted into k-CYC in the following way. Any element in an input list $x \in \mathbb{Z}_m$ can be mapped to $g^x \in \mathbb{G}$ where g is a generator of the cyclic group for the k-CYC problem. Since the input to k-SUM is random over $\mathbb{Z}_m$, the input to k-CYC also remains uniformly random $\mathbb{G}$ assuming $|G| = m$. This is due to the fact that exponentiation in cyclic groups is efficient.

The reverse direction seems more challenging where we wish to solve k-CYC using an algorithm for k-SUM. If one wanted to extend the original approach, it would require essentially solving discrete logarithms to essentially map elements of the form $g^x \in \mathbb{G}$ for k-CYC to obtain $x \in \mathbb{Z}_m$ for k-SUM. In this section, we will show that solving discrete logarithms efficiently is indeed fundamental to enabling efficient reductions from k-CYC to k-SUM.

To do this, we start with relations between DLog and k-CYC. This was observed previously by Dai [34] and Wagner [80] that show algorithms for k-CYC may be used to solve the DLog problem (see Theorem G.1 in Appendix G). In other words, we know that efficient algorithms for k-CYC would end up leading to faster algorithms for DLog. As a note, while there are faster algorithms for k-SUM by Wagner [80] in certain dense settings, we are unaware of any such faster algorithms for k-CYC to date. See Appendix A for more details about Wagner's k-SUM algorithm [80] and why it cannot be used for k-CYC.

This is where the barrier for reductions between k-SUM and k-CYC come into play. If an efficient reduction existed that reduces random input instances of k-CYC to random input instances of k-SUM, then we could apply the faster algorithms of Wagner [80]. This would immediately translate into faster algorithms for DLog. The above intuition leads to a barrier for showing equivalences between k-SUM and k-CYC. It turns out that k-SUM and k-CYC are equivalent if and only if the DLog conjecture is false. To start, we prove the direction where the equivalence implies that the DLog conjecture is false.

Theorem 4.1. *For any constant $\epsilon > 0$ and $k \geq 4$, suppose that there exists an efficient reduction in time $O(n^{k/2-\epsilon})$ from a random instance of size n for k-CYC$^{\mathsf{G}}$ over group ensemble G of order m to a random instance of size $O(n)$ for k-SUM^m over group $\mathbb{Z}_m$. Then, the* DLog *conjecture for $\mathbb{G}$ is false.*

Proof. Suppose there exists an algorithm A that maps random instances of k-CYC$^{\mathsf{G}}$ to random instances of k-SUM^m in time $O(n^{k/2-\epsilon})$ for some constant $\epsilon > 0$ and $k \geq 4$. We use A to show that we can construct an algorithm A' that solves the DLog problem on input g^x in time $O(m^{1/2-\epsilon'})$ for some constant $\epsilon' > 0$. First, we map an instance of DLog$^{\mathsf{G}}$ to a random and dense instance of k-CYC$^{\mathsf{G}}$. For any constant $0 < c < 2\epsilon/k$, we will create k lists each consisting of $n = m^{(1+c)/k}$ uniformly random elements from $\mathbb{G}$. For the input g^x, we see the expected number of k-tuple solutions whose product is g^x is at least $(m^{(1+c)/k})^k/m = m^{1+c}$. Therefore, at least one solution for g^x will exist except with $1/\mathsf{poly}(n)$ probability using Lemma B.1.

Afterwards, we execute the algorithm A to map the random instance of k-$\mathsf{CYC}^{\mathsf{G}}$ into a random instance of k-SUM^m with k lists of size $O(n)$. Note, the execution of A takes time $O(n^{k/2-\epsilon}) = O(m^{(k/2-\epsilon)\cdot((1+c)/k)}) = O(m^{1/2+c/2-\epsilon(1+c)/k})$. We see that $c/2 < \epsilon/k < \epsilon(1+c)/k$ by our choice of constant c. Therefore, the algorithm A runs in time $O(n^{k/2-\epsilon'})$ for some constant $\epsilon' > 0$ depending only on k, c and ϵ. Finally, we run algorithm of Wagner [80] to solve random instances of k-SUM^m in the dense setting when $k \geq 4$. In particular, these algorithms are able to run in time $O(n^{k/2-\epsilon''})$ for some constant $\epsilon'' > 0$ assuming $n^k \geq m^{1+\delta}$ for some constant $\delta > 0$ (see Theorem A.1 in Appendix A). In our case, we note that $n^k = m^{k(1+c)/k} = m^{1+c}$ where $c > 0$ is a constant. Therefore, we can solve the random instance of k-SUM in time $O(n^{k/2-\epsilon''})$. As a result, we obtain an algorithm for $\mathsf{DLog}^{\mathsf{G}}$ in $m^{1/2-\Omega(1)}$ time. □

As a note, we point out that our reduction from DLog to k-CYC is similar to the one in Theorem G.1 with one main difference. In the prior reduction, it suffices to map an instance of DLog over group order m into k lists of size $n = \tilde{O}(m^{1/k})$. In our above reduction, it is critical that we actually map to a dense instance where $n = O(m^{(1+c)/k})$ for some constant $c > 0$ as this enables us to apply the algorithm of Wagner [80] to obtain faster algorithms.

Next, we prove that the opposite direction is also true. In particular, if the DLog may be solved efficiently, one can construct an efficient reduction from k-CYC to k-SUM. This essentially formalizes the trivial algorithm that we presented at the beginning of Section 4.1.

Theorem 4.2. *Suppose the $\mathsf{DLog}^{\mathsf{G}}$ conjecture is false for group ensemble G of order m. For every $n \geq 1$, there exists some constant $k \geq 4$ such that there exists an efficient reduction in time $O(n^{k/2-\epsilon})$ from a random instance of size n for k-$\mathsf{CYC}^{\mathsf{G}}$ to a random instance of size n for k-SUM^m for some constant $\epsilon > 0$.*

Proof. Since the DLog conjecture is false, there exists some algorithm A running in time $O(m^{1/2-\epsilon})$ for some constant $\epsilon > 0$. For any n, pick constant $k \geq 4$ such that $n = m^{1/k} < m^{\epsilon}$. Our reduction runs by running algorithm A on all $nk = O(n)$ elements in the k input lists for k-CYC to compute their discrete logarithms. This immediately results in a random instance of size n for k-SUM. The running time of the reduction is $O(nm^{1/2-\epsilon}) = O(m^{1/k} \cdot m^{1/2-\epsilon}) = O(m^{1/2-\epsilon'})$ where $\epsilon' = 1/k - \epsilon > 0$ is a constant since k and ϵ are constant. □

Implications to Other Number-Theoretic Assumptions. We note that Theorem 4.1 has implications to other cryptographic assumptions. First, we know that if discrete logarithms can be computed in $T(n)$ time, then both the computational and decisional Diffie-Hellman assumptions may be solved in $T(n)$ time as well. Therefore, we immediately get that an efficient reduction between k-CYC and k-SUM would immediately imply that the equivalent conjectures for computational and decisional Diffie-Hellman would be false.

Similar results can be obtained for the factoring, RSA and quadratic residuosity assumptions if we consider reductions from k-PROD over $\mathbb{Z}_m^\times$ where m is a composite number. Therefore, an efficient reduction between k-PROD^m and k-SUM^m over composite group orders m immediately implies faster algorithms for DLog over $\mathbb{Z}_m^\times$. It is known that computing DLog over composite group orders enables factoring [9]. So, we may extend Theorem 4.1 showing that an equivalence between k-SUM and k-PROD would also imply faster algorithms for the factoring, RSA and quadratic residuosity problems (see Appendix F).

Sub-Exponential Algorithms. We can extend Theorem 4.1 to also consider slightly more efficient reductions between from k-CYC to k-SUM. In particular, if the reductions run in $\mathsf{poly}(n)$ time for some super-constant $k = \omega(1)$, then we can obtain sub-exponential algorithms for DLog as well as the other related number-theoretic assumptions. See Appendix F.1 for further details.

4.2 Reductions between DLog-Preprocess and k-CYC-Index

As we stated earlier, prior works [34, 80] have shown relations between k-CYC and DLog. In this work, we show that the same relationship may be extended to the preprocessing variants of each problem and show a reduction between DLog-Preprocess and k-CYC-Index.

Theorem 4.3. *For any constant $k \geq 3$, if the AVG-k-PROD-Index$^{\mathsf{G}}$ conjecture is true for group ensemble G, then the DLog-Preprocess$^{\mathsf{G}}$ conjecture is true.*

Proof. Let m be the group orders for the ensemble G. Suppose we have the tuple of algorithms (P, Q) for preprocessing and querying that contradicts the AVG-k-PROD-Index conjecture for some constant $k \geq 3$ such that $ST^2 = O(n^{k-1-\epsilon})$ for some constant $\epsilon > 0$. We will pick some $n = \tilde{O}(m^{1/(k-1)})$ with the following guarantee. Instantiate $k-1$ lists, $L_1, \ldots, L_{k-1}$ consisting of n elements chosen uniformly at random in the following way. To generate an element, we pick a random integer z from $[m]$ and compute g^z as the uniformly random element from $\mathbb{G}$. We record both z and g^z. For sufficiently large $n = \tilde{O}(m^{1/(k-1)})$, we note there exists some set of indices $(y_1, \ldots, y_{k-1}) \in [n]^{k-1}$ such that $L_1[y_1] \cdots L_{k-1}[y_{k-1}] = t$ except with $1/\mathsf{poly}(n)$ probability for every $t \in \mathbb{G}$ using Lemma B.1. Then, we run the preprocessing algorithm $P(L_1, \ldots, L_{k-1})$ for k-CYC-Index on inputs of the $k-1$ lists $L_1, \ldots, L_{k-1}$ to obtain some data structure with space S.

We receive a query of the form $t = g^x$ that is a uniformly random element from $\mathbb{G}$. We execute querying algorithm Q with target t and the preprocessed data structure as input. The algorithm Q outputs some $(y_1, \ldots, y_{k-1}) \in [n]^{k-1}$ such that $L_1[y_1] \cdots L_{k-1}[y_{k-1}] = t$. By simply summing the corresponding $k-1$ exponents as elements in $\mathbb{Z}_m$, we compute x. Note, we solve the DLog-Preprocess problem with space S and query time T such that

$$ST^2 = O(n^{k-1-\epsilon}) = \tilde{O}(|\mathbb{G}|^{(k-1-\epsilon)/(k-1)}) = \tilde{O}(|\mathbb{G}|^{1-\epsilon/(k-1)})$$

that contradicts the DLog-Preprocess conjecture for $\mathbb{G}$. □

This implies that, if one wishes to find faster algorithms for k-CYC-Index, then one should not try and construct algorithms that apply to every group. In particular, the above reduction would end up producing a generic algorithm for DLog-Preprocess that would contradict known generic group lower bounds [30].

4.3 Average-Case k-SUM-Index and k-CYC-Index Algorithms

Finally, we present new algorithms for both k-SUM-Index, k-PROD-Index and k-CYC-Index with improved space-time trade-offs compared to prior works. As we discussed in Section 2.3, there are some trivial algorithms for these problems. One simply stores the lists in linear space $S = O(n)$ and runs the best algorithms without preprocessing in time $O(n^{\lceil (k-1)/2 \rceil})$. The other extreme stores all k-tuples using $S = \tilde{O}(n^k)$ space and answers queries in $T = \tilde{O}(1)$ time.

Recent works [62, 51] presented algorithms with better space-time trade-offs for arbitrary space usages achieving $S^3T = \tilde{O}(n^{3(k-1)})$ for k-SUM-Index. This means there exists algorithms using space $S = \tilde{O}(n^{k-1-\delta})$ and query time $T = \tilde{O}(n^{3\delta})$ for any $\delta > 0$. At a high level, these algorithms relied upon the Fiat-Naor function inversion algorithms [44]. The function of interest is defined as

$f(y_1, \ldots, y_{k-1}) = L_1[y_1] \odot \ldots \odot L_{k-1}[y_{k-1}]$ assuming any group $\mathbb{G}$ with group operation $\odot$. It can be easily seen that answering a query for target $t \in \mathbb{G}$ is equivalent to computing $f^{-1}(t)$. These worst-case algorithms may be extended to k-CYC-Index in a straightforward manner with the same efficiency guarantees.

We present a new algorithm for the average-case versions of k-SUM-Index and k-CYC-Index that we study in this paper. In particular, we note that the function $f(y_1, \ldots, y_{k-1}) = L_1[y_1] \odot \ldots \odot L_{k-1}[y_{k-1}]$ contains significant amounts of randomness as each of the n elements in the k lists are random. This leads us to the idea of trying to use Hellman's algorithm [52] for inverting random functions. However, we note that the function itself is not a random function as the outputs of a large subsets of elements in the domain may be highly correlated. Nevertheless, we can rely on the analysis of Hellman's algorithm by Fiat and Naor [44] that characterizes the efficiency of Hellman's algorithm with respect to collision probabilities. If q is the collision probability, then Hellman's algorithm satisfies $S^2T = \tilde{O}(q \cdot |\mathcal{X}|^3)$ (see Theorem G.2).

We construct our function f that we use for our data structure in the following way. We start with the easy case where $n^{k-1} = m$. In this case, we can immediately define $f : [n^{k-1}] \to [m]$ as $f(y_1, \ldots, y_{k-1}) = L_1[y_1] \odot \ldots \odot L_{k-1}[y_{k-1}]$. As a note, we will use the trivial bijection between $(k-1)$-tuples from $[n]^{k-1}$ and the set $[n^{k-1}]$. To expand to more applications, we modify f for the case where $m > n^{k-1}$. In this case, we define $f : [m] \to [m]$ as a piecewise function. For any $x \in [m]$, we define $f(x) = f(y+1)$ where $y = x \bmod n^{k-1}$. Note, $f(x)$ can be computed efficiently in $\tilde{O}(k) = \tilde{O}(1)$ time. Our algorithm is applying Hellman's algorithm [52] to this function f for inversion for both preprocessing and querying. For a query target t, we execute the query of Hellman's algorithm to compute $x = f^{-1}(t)$. Lastly, we compute $y = x \bmod n^{k-1}$ and return the $(k-1)$-tuple in $[n]^{k-1}$ corresponding to $y + 1 \in [n^{k-1}]$.

Theorem 4.4. *For any constant $k \geq 4$ and group ensemble G with group order m such that $n^{k-1} \leq m = \Theta(n^{k-1})$, there exists algorithms for k-SUM-Index$^{\mathsf{G}}$, k-PROD-Index$^{\mathsf{G}}$ and k-CYC-Index$^{\mathsf{G}}$ that uses space S and time T with error probability $1/\mathsf{poly}(n)$ such that $S^2T = \tilde{O}(n^{2(k-1)})$.*

Proof. We start by proving that the probability q that two random elements in the domain of f map to the same element in the image is $O(1/n^{k-1})$. The analysis applies all three problems k-SUM-Index, k-PROD-Index and k-CYC-Index since the elements are sampled uniformly at random from the respective groups (in fact, it applies to all groups). Pick two random elements $x, y \in [m]$ and compute the probability $f(x) = f(y)$ in two cases. First, if $x = y \bmod n^{k-1}$, we know that $f(x) = f(y)$. If this is not true, then x and y correspond to two different $(k-1)$-tuples that differ in at least one entry. So, the probability that $f(x) = f(y)$ is at most $1/m$. Altogether, we get the probability as follows:

$$\begin{aligned} q &\leq \Pr[f(x) = f(y) \mid x \neq y \bmod n^{k-1}] + \Pr[x = y \bmod n^{k-1}] \\ &\leq \frac{1}{m} + \frac{1}{n^{k-1}} = O\left(\frac{1}{n^{k-1}}\right). \end{aligned}$$

We can now apply Theorem G.2 and obtain an algorithm that inverts function f with space S and time T such that $S^2T = \tilde{O}(qn^{3(k-1)}) = \tilde{O}(n^{2(k-1)})$. □

In other words, there exists an algorithm with space $S = \tilde{O}(n^{k-1-\delta})$ and query time $T = \tilde{O}(n^{2\delta})$ for any $\delta > 0$. This beats the prior worst-case algorithms [62, 51] that obtain require $T = \tilde{O}(n^{3\delta})$ for the same space usage. One way to view this result is that this is evidence that k-SUM-Index, k-PROD-Index and k-CYC-Index are easier in the average-case compared to the worst-case.

Acknowledgements

The authors were supported in part by NSF Grant CCF-2238221 and a grant from the Simons Foundation (Grant Number 825870 JA). The research was partially supported by a Google CyberNYC grant, an Amazon Research Award and an NSF grant CCF-2312242. Yizhi Huang was also supported in part by NSF grants CCF-2106429 and CCF-2211238.

References

[1] A. Abboud, S. Feller, and O. Weimann. On the fine-grained complexity of parity problems. In *International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2020.

[2] A. Abboud and K. Lewi. Exact weight subgraphs and the k-sum conjecture. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *ICALP 2013, Part I*, volume 7965 of *LNCS*, pages 1–12. Springer, Heidelberg, July 2013.

[3] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th FOCS*, pages 434–443. IEEE Computer Society Press, Oct. 2014.

[4] A. Afshar, G. Couteau, M. Mahmoody, and E. Sadeghi. Fine-grained non-interactive key-exchange: constructions and lower bounds. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 55–85. Springer, 2023.

[5] D. Aggarwal and U. Maurer. Breaking RSA generically is equivalent to factoring. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 36–53. Springer, Heidelberg, Apr. 2009.

[6] S. Agrawal, S. Saha, N. I. Schwartzbach, A. Vanukuri, and P. N. Vasudevan. k-sum in the sparse regime: Complexity and applications. In *Annual International Cryptology Conference*, pages 315–351. Springer, 2024.

[7] J. Alman, Y. Huang, and K. Yeo. Fine-grained complexity in a world without cryptography. In *Eurocrypt 2025*, 2025.

[8] B. Applebaum, B. Barak, and A. Wigderson. Public-key cryptography from different assumptions. In L. J. Schulman, editor, *42nd ACM STOC*, pages 171–180. ACM Press, June 2010.

[9] E. Bach. *Discrete logarithms and factoring.* University of California at Berkeley, 1984.

[10] M. Ball, J. Garay, P. Hall, A. Kiayias, and G. Panagiotakos. Towards permissionless consensus in the standard model via fine-grained complexity. In *Annual International Cryptology Conference*, pages 113–146. Springer, 2024.

[11] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Average-case fine-grained hardness. In H. Hatami, P. McKenzie, and V. King, editors, *49th ACM STOC*, pages 483–496. ACM Press, June 2017.

[12] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of work from worst-case assumptions. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 789–819. Springer, Heidelberg, Aug. 2018.

[13] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008.

[14] J. Bartusek, F. Ma, and M. Zhandry. The distinction between fixed and random generators in group-based assumptions. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 801–830. Springer, Heidelberg, Aug. 2019.

[15] B. Bauer, G. Couteau, and E. Sadeghi. Fine-grained non-interactive key exchange, revisited. In *Annual International Cryptology Conference*, pages 286–312. Springer, 2024.

[16] A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 364–385. Springer, Heidelberg, May 2011.

[17] T. Belova, A. Golovnev, A. S. Kulikov, I. Mihajlin, and D. Sharipov. Polynomial formulations as a barrier for reduction-based hardness proofs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3245–3281. SIAM, 2023.

[18] T. Belova, A. S. Kulikov, I. Mihajlin, O. Ratseeva, G. Reznikov, and D. Sharipov. Computations with polynomial evaluation oracle: ruling out superlinear seth-based lower bounds. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1834–1853. SIAM, 2024.

[19] D. J. Bernstein and T. Lange. Non-uniform cracks in the concrete: The power of free precomputation. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 321–340. Springer, Heidelberg, Dec. 2013.

[20] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[21] E. Boix-Adserà, M. Brennan, and G. Bresler. The average-case complexity of counting cliques in Erdős-Rényi hypergraphs. In D. Zuckerman, editor, *60th FOCS*, pages 1256–1280. IEEE Computer Society Press, Nov. 2019.

[22] D. Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[23] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and cartier–manin operator. *SIAM Journal on Computing*, 36(6):1777–1806, 2007.

[24] C. Bouillaguet, C. Delaplace, and P.-A. Fouque. Revisiting and improving algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, pages 254–276, 2018.

[25] Z. Brakerski, N. Stephens-Davidowitz, and V. Vaikuntanathan. On the hardness of average-case k-sum. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2021.

[26] C. Brzuska and G. Couteau. On building fine-grained one-way functions from strong average-case hardness. In O. Dunkelman and S. Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 584–613. Springer, Heidelberg, May / June 2022.

[27] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. 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*, pages 261–270, 2016.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2022.

[29] J.-S. Coron and A. Joux. Cryptanalysis of a provably secure cryptographic hash function. *Cryptology ePrint Archive*, 2004.

[30] H. Corrigan-Gibbs and D. Kogan. The discrete-logarithm problem with preprocessing. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 415–447. Springer, Heidelberg, Apr. / May 2018.

[31] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, Aug. 1998.

[32] R. E. Crandall and C. Pomerance. *Prime numbers: a computational perspective*, volume 2. Springer, 2005.

[33] D. Dachman-Soled, J. Loss, and A. O'Neill. Breaking rsa generically is equivalent to factoring, with preprocessing. *Cryptology ePrint Archive*, 2022.

[34] W. Dai. Personal Communication with David Wagner.

[35] M. Dalirrooyfard, A. Lincoln, and V. V. Williams. New techniques for proving fine-grained average-case hardness. In *61st FOCS*, pages 774–785. IEEE Computer Society Press, Nov. 2020.

[36] M. Dalirrooyfard, A. Lincoln, and V. V. Williams. New techniques for proving fine-grained average-case hardness. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 774–785. IEEE, 2020.

[37] A. Degwekar, V. Vaikuntanathan, and P. N. Vasudevan. Fine-grained cryptography. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 533–562. Springer, Heidelberg, Aug. 2016.

[38] E. Demaine and S. Vadhan. Some notes on 3sum. Unpublished note, 2001.

[39] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[40] I. Dinur. An algorithmic framework for the generalized birthday problem. *Designs, Codes and Cryptography*, 87:1897–1926, 2019.

[41] I. Dinur, N. Keller, and O. Klein. Fine-grained cryptanalysis: Tight conditional bounds for dense k-sum and k-xor. *Journal of the ACM*, 71(3):1–41, 2024.

[42] Y. Dodis, D. Khovratovich, N. Mouha, and M. Nandi. T5: Hashing five inputs with three compression calls. In *2nd Conference on Information-Theoretic Cryptography, ITC 2021*, page 24. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2021.

[43] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, Aug. 1984.

[44] A. Fiat and M. Naor. Rigorous time/space trade-offs for inverting functions. *SIAM Journal on Computing*, 29(3):790–803, 2000.

[45] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, Apr. 2010.

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

[47] O. Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001.

[48] O. Goldreich and G. N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In M. Thorup, editor, *59th FOCS*, pages 77–88. IEEE Computer Society Press, Oct. 2018.

[49] I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures: 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31–August 2, 2017, Proceedings 15*, pages 421–436. Springer, 2017.

[50] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, pages 173–201. 2019.

[51] A. Golovnev, S. Guo, T. Horel, S. Park, and V. Vaikuntanathan. Data structures meet cryptography: 3SUM with preprocessing. In K. Makarychev, Y. Makarychev, M. Tulsiani, G. Kamath, and J. Chuzhoy, editors, *52nd ACM STOC*, pages 294–307. ACM Press, June 2020.

[52] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

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

[54] Z. Jafargholi and E. Viola. 3sum 3 sum, 3xor 3 xor, triangles. *Algorithmica*, 74:326–343, 2016.

[55] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[56] A. Joux. *Algorithmic cryptanalysis.* Chapman and Hall/CRC, 2009.

[57] A. Joux, D. Naccache, and E. Thomé. When e-th roots become easier than factoring. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 13–28. Springer, Heidelberg, Dec. 2007.

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

[59] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* CRC Press, 2020.

[60] N. Koblitz, A. Menezes, and S. Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography*, 19:173–193, 2000.

[61] T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. In R. Krauthgamer, editor, *27th SODA*, pages 1272–1287. ACM-SIAM, Jan. 2016.

[62] T. Kopelowitz and E. Porat. The strong 3sum-indexing conjecture is false. *arXiv preprint arXiv:1907.11206*, 2019.

[63] R. LaVigne, A. Lincoln, and V. V. Williams. Public-key cryptography in the fine-grained setting. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 605–635. Springer, Heidelberg, Aug. 2019.

[64] H. T. Lee, J. H. Cheon, and J. Hong. Accelerating ID-based encryption based on trapdoor DL using pre-computation. Cryptology ePrint Archive, Report 2011/187, 2011. https://eprint.iacr.org/2011/187.

[65] G. Leurent and F. Sibleyras. Low-memory attacks against two-round even-mansour using the 3-XOR problem. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 210–235. Springer, Heidelberg, Aug. 2019.

[66] Y. Liu and R. Pass. On one-way functions and kolmogorov complexity. In *FOCS*, pages 1243–1254. IEEE, 2020.

[67] V. Lyubashevsky, A. Palacio, and G. Segev. Public-key cryptographic primitives provably as secure as subset sum. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 382–400. Springer, Heidelberg, Feb. 2010.

[68] U. Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10*, pages 1–12. Springer, 2005.

[69] J. P. Mihalcik. *An analysis of algorithms for solving discrete logarithms in fixed groups.* PhD thesis, Citeseer, 2010.

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

[71] M. Nandi. Revisiting security claims of XLS and COPA. Cryptology ePrint Archive, Paper 2015/444, 2015.

[72] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes-New York*, 55(1):165–172, 1994.

[73] I. Nikolic and Y. Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 683–703. Springer, Heidelberg, Nov. / Dec. 2015.

[74] S. Pettie. Higher lower bounds from the 3sum conjecture, https://simons.berkeley.edu/talks/higher-lower-bounds-3sum-conjecture.

[75] J. M. Pollard. A monte carlo method for factorization. *BIT*, 15(3):331–334, Sept. 1975.

[76] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[77] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, Aug. 1990.

[78] A. Schrottenloher. Improved quantum algorithms for the k-xor problem. In *International Conference on Selected Areas in Cryptography*, pages 311–331. Springer, 2021.

[79] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[80] D. Wagner. A generalized birthday problem. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, Aug. 2002.

[81] J. R. Wang. Space-efficient randomized algorithms for k-sum. In *European Symposium on Algorithms*, pages 810–829. Springer, 2014.

[82] V. V. Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

A Algorithms for Average-Case k-SUM in Dense Setting

In this section, we revisit the algorithms of Wagner [80] that follows the approach of Blum, Kalai and Wassermann [20] for solving average-case k-SUM in the dense setting where n is sufficiently large that there are many solutions with high probability (that is, $n^k \gg m$). In particular, these algorithms outperform the trivial $n^{\lceil k/2 \rceil}$ algorithms for k-SUM. We note that there exists follow-up works (such as [73, 71, 40, 65]) presenting improvements of Wagner's algorithm.

A.1 k-SUM Algorithm of Wagner [80]

For simplicity, we start by presenting the $k\text{-SUM}^m$ algorithm for $k = 4$ over group $\mathbb{Z}_m$. We will generalize later to larger $k \geq 4$. We focus specifically on the setting where $n = m^{1/3}$ and consider the more general case of $n^k = m^{1+c}$ for some constant $c > 0$ later as done in [70]. In this case, we see that $n^k \gg m$ meaning that we should expect at least many solutions in expectation. By applying the algorithm of Wagner [80] with the correct parameters, $k\text{-SUM}^m$ may be solved in time

$O(n^{\lceil k/2 \rceil - \epsilon})$ for some constant $\epsilon > 0$ depending only on k and c that beats the naive algorithm for k-SUM^m.

At a high level, the algorithm works as follows for $k = 4$ where $n = m^{1/3}$ for constant $0 < c < 1$. We will pair the input lists into two groups of two. To process L_1 and L_2, we will essentially find at most n pairs whose sum is at most n. We note this can be computed in $\tilde{O}(n)$ time in the following way. First, we sort both lists in ascending order. Then, we keep two pointers in L_1 pointing to the smallest and largest entries that are at most ℓ. Then, we iterate through L_2 in ascending order and use as well as adjust the pointers in L_1 to help find pairs of solutions. So, this allows us to find solution pairs that we denote L_{12} in time $\tilde{O}(n)$. We can also do this as well for the lists L_3 and L_4 and get at most n pairs of solutions whose sum is at least $m - n$ modulo m that we denote L_{34}.

Lastly, we run the trivial 2-SUM algorithm to find a solution in L_{12} and L_{34} in time $\tilde{O}(n)$ time. Note that L_{12} contains at most n pairs whose sum is at most n modulo m. Similarly, L_{34} contains at most n pairs whose sum is at least $m - n$ modulo m. Therefore, the algorithm runs in time $\tilde{O}(n)$ time altogether.

Next, we analyze the probability that the algorithm finds a solution. Note that the number of expected pairs of solutions in L_{12} and L_{34} is $n^2/n = n$. Therefore, we expect the number of solutions in the last step to be

$$\frac{\min\{n, \mathsf{E}[|L_{12}|]\} \cdot \min\{n, \mathsf{E}[|L_{34}|]\}}{n} = \frac{\Omega(n^2)}{n} = \Omega(n).$$

Therefore, the last step executing the trivial 2-SUM algorithm will find a solution except with $1/\mathsf{poly}(n)$ probability by applying Lemma B.1.

The total runtime of this algorithm is $\tilde{O}(k \cdot n) = \tilde{O}(n)$ since each of the list merges requires $\tilde{O}(n)$ time. It is not hard to see that this algorithm may be generalized to arbitrary $k \geq 4$. At a high level, we create a binary tree of height $O(\log k)$ where the leaf nodes are each of the individual k lists. Afterwards, we essentially merge lists two at a time until reaching the root and obtaining the solution. As a note, we can relax the requirement to be simply that $n = m^{(1+c)/k}$ for any constant $c > 0$. Furthermore, we note that prior work [70] has considered smaller lists and presented time bounds for these settings.

Theorem A.1 ([80, 70]). *For any $k \geq 4$, constant $c > 0$ and $m \leq n^{k/(1+c)}$, there exists an algorithm for k-SUM^m in time $O(m^{1/2-\epsilon})$ with error probability $1/\mathsf{poly}(n)$ for some constant $\epsilon > 0$.*

We note that there are other choices of parameters that result in smaller running times if one is allowed to freely choose the size of the input lists n sufficiently large with respect to the group size m. More generally, it is always possible to run Wagner's algorithm on input lists of size $n = m^{1/(1+\lfloor \log k \rfloor)}$ and obtain algorithms that run in time $\tilde{O}(n) = \tilde{O}(m^{1/(1+\lfloor \log k \rfloor)})$ for any $k \geq 4$. For $k = 4$, this results in algorithms running in time $\tilde{O}(m^{1/3})$. This leads us to the following corollary that shows Wagner's algorithm runs in sub-exponential time for sufficiently large $k = \omega(1)$.

Theorem A.2 ([80]). *For any $k = \omega(1)$ and $m \leq n^{1+\log k}$, there exists an algorithm for k-SUM^m in time $m^{o(1)}$ with error probability $1/\mathsf{poly}(n)$.*

A.2 Inapplicability to k-PROD and k-CYC

In the prior section, we have shown that Wagner's algorithm [80] can beat the trivial k-SUM^m algorithm in the dense setting where $n^k \gg m$. One may wonder whether the same algorithm could also be used to solve k-PROD^m or k-$\mathsf{CYC}^{\mathsf{G}}$ in the similar dense setting. Somewhat surprisingly, we show that the algorithm does not seem easily adaptable to k-PROD or k-CYC. Additionally, we also show that if Wagner's algorithm were adaptable to k-PROD or k-CYC, it would end up speeding up algorithms known for DLog in their respective groups providing additional evidence that it may be quite challenging to modify Wagner's algorithm to work for k-PROD or k-CYC.

Obstacles with List Merging. Recalling the key step of list merging in the k-SUM algorithm, it first sorts the two lists L_1 and L_2, and then keep two pointers in L_1 while traversing the other L_2 in ascending order. When traversing the second list $y \in L_2$, we guarantee that all possible $x \in L_1$ such that $x + y \leq n$ modulo m are between the two pointers maintained in L_1. Immediately, we see that this algorithm is not possible for k-PROD or k-CYC. Generalizing the idea, suppose we have any subset S of a cyclic group $\mathbb{G}$ of size n and we wish to find pairs of solution $(x, y) \in L_1 \times L_2$ such that $x \cdot y \in S$. To our knowledge, there is no way to pick subset S such that a variant of the two pointer-style algorithm still succeeds at finding pairs of solutions. In general, we are unaware of any $\tilde{O}(n)$ algorithm that allows merging two lists of size n that allows us to find $\Omega(n)$ pairs of solutions whose product appears in some subset S of size n.

Relation to DLog. To give some additional evidence that extending Wagner's algorithm [80] to k-PROD or k-CYC will be challenging, we show the implications to DLog if such an extension were possible. As shown by prior works [34, 80] and discussed in Section 4.1, it is known that faster algorithms for k-CYC would immediately imply faster algorithms for DLog. In particular, suppose that one could essentially extend Wagner's algorithm and show something equivalent to Theorem A.1 for k-CYC for some group $\mathbb{G}$ of order m. Using these reductions, we could attempt to solve DLog by constructing k lists of random group elements that are chosen by picking a random exponent $x \in [m]$ and computing g^x. With knowledge of the all discrete logarithms of each input element, we can pick k lists of $n \geq m^{(1+c)/k}$ random elements for some constant $c > 1$. Finally, we can apply this new algorithm to obtain an algorithm for DLog that runs in time $O(m^{1/2-\epsilon})$ for some constant $\epsilon > 0$. This would be a breakthrough as it would result in better algorithms for solving DLog for a wide-range of popular elliptic curve groups used in practice such as NIST P-256 where the best algorithms remain $\tilde{O}(m^{1/2})$. We point readers to [45, 60] for more details about the state-of-the-art algorithms for DLog in these elliptic curve groups.

The same holds true even for k-PROD where we consider multiplicative groups over the integers $\mathbb{Z}_m^\times$ where m is prime. Even though there are sub-exponential algorithms for DLog over $\mathbb{Z}_m^\times$, these are heuristic algorithms that do not have provable running times. The best provable running time algorithms remain $\tilde{O}(m^{1/2})$. Therefore, if one could adapt Wagner's algorithm for k-PROD, it would also lead to a breakthrough in algorithms with provable running times of $O(m^{1/2-\epsilon})$ for constant $\epsilon > 0$ solving DLog over $\mathbb{Z}_m^\times$.

B Existence of Solution

Below we give estimations on the probability that a uniformly sampled k-SUM(m, n) instance has a solution. We use the fact that the sum (product) of uniformly sampled group elements is also uniformly distributed. First, we have the following lemma:

Lemma B.1. *For a k-SUM^m instance of size n $z = (L_1, L_2, \ldots, L_k)$, the probability that z has a solution is at most n^k/m, and the probability that z has exactly one solution is at least $n^k/m - \binom{n^k}{2}/m^2$ (so the probability that z has a solution is also at least $n^k/m - \binom{n^k}{2}/m^2$).*

Proof. For any $(y_1, y_2, \ldots, y_k) \in [n]^k$, the probability that $(y_1, y_2, \ldots, y_k)$ is a solution to z is $1/m$. Since there are n^k possible solutions, by union bound, the probability that z has a solution is upper-bounded by n^k/m.

For any distinct $(y_1, y_2, \ldots, y_k), (y'_1, y'_2, \ldots, y'_k) \in [n]^k$, the probability that both are solutions to z is $1/m^2$. The reason is as follows.

Let $A = \{i : y_i = y'_i\}$, then for any fixed $(L_i[y_i])_{i\in A}$, since $|A| < n$,

$$\Pr_{(L_i[y_i])_{i\notin A} \leftarrow \mathcal{U}([n]^{k-|A|})}[(y_1, y_2, \ldots, y_k) \text{ is a solution to } z] = 1/m,$$

$$\Pr_{(L_i[y'_i])_{i\notin A} \leftarrow \mathcal{U}([n]^{k-|A|})}[(y'_1, y'_2, \ldots, y'_k) \text{ is a solution to } z] = 1/m,$$

and $(L_i[y_i])_{i\notin A}$ and $(L_i[y'_i])_{j\notin A}$ are independent.

Since there are $\binom{n^k}{2}$ possible $((y_1, y_2, \ldots, y_k), (y'_1, y'_2, \ldots, y'_k)) \in ([n]^k)^2$, by inclusion-exclusion principle, the probability that z has exactly one solution is lower-bounded by $n^k/m - \binom{n^k}{2}/m^2$. $\square$

Lemma B.1 only gives a non-trivial bound when $m \geq n^k/2$. When this is not the case, we have the following bound, which is proved based on Lemma B.1.

Lemma B.2. *For a uniform k-$\mathsf{SUM}(m, n)$ instance $z = (L_1, L_2, \ldots, L_k)$ where $m < n^k$, the probability that z has a solution is at least $1 - 1/2^{n/m^{1/k}-1}$.*

Proof. Let $n' := \lceil m^{1/k} \rceil \leq n$, and for $1 \leq j \leq n/n'$, let z_j be the k-$\mathsf{SUM}(m, n')$ instance $(L_1[(j-1)n'+1 : jn'], L_2[(j-1)n'+1 : jn'], \ldots, L_k[(j-1)n'+1 : jn'])$ where $L_i[(j-1)n'+1 : jn']$ denotes the list consisting of the $((j-1)n'+1)$-th to (jn')-th elements of L_i.

It is easy to see that $z_1, z_2, \ldots, z_{\lfloor n/n' \rfloor}$ are independent uniform k-$\mathsf{SUM}(m, n')$ instance. Moreover, if there exists j such that z_j has a solution, then z also has a solution. By Lemma B.1, for a fixed j, the probability that z_j has a solution is at least $(n')^k/m - \binom{(n')^k}{2}/m^2 \geq 1/2$. Therefore, the probability that z has a solution is at least $1 - 1/2^{\lfloor n/n' \rfloor} \geq 1 - 1/2^{n/m^{1/k}-1}$. $\square$

Similar to Lemma B.1 for k-SUM, we also have the following for Zero-k-Clique.

Lemma B.3. *Let $J := \{(i_1, i_2) \mid 1 \leq i_1 < i_2 \leq k\} \times [n]^2$. For a uniform Zero-k-$\mathsf{Clique}(m, n)$ instance $z = (e_{\{(i_1,j_1),(i_2,j_2)\}})_{(i_1,i_2,j_1,j_2)\in J}$, the probability that z has a solution is at most n^k/m, and the probability that z has exactly one solution is at least $n^k/m - \binom{n^k}{2}/m^2$ (so the probability that z has a solution is also at least $n^k/m - \binom{n^k}{2}/m^2$).*

The proof is almost the same as the proof of Lemma B.1 and is thus omitted.

C Decision k-SUM and k-CYC

We present the formal definitions of the search problems where it suffices for a correct algorithm to simply determine whether there exists a solution or not. We only present the definitions for k-SUM and k-CYC, but the definition of k-PROD follows in a similar manner.

Definition C.1 (k-LIST Decision Problem)**.** For $k \geq 0$ and group ensemble $(\mathsf{G}, \odot)$, an algorithm A correctly solves the k-$\mathsf{LIST}^{\mathsf{G}}$ decision problem if it outputs the following in each corresponding scenario:

- If there exists a solution, A outputs 1.
- If no solution exists, A outputs 0.

We say algorithm A has failure probability $\epsilon(n)$ if

$$\Pr[A(L_1, \ldots, L_k) \text{ is correct}] \geq 1 - \epsilon(n)$$

where the randomness is over the internal coin tosses of A and random choice of $L_1, \ldots, L_k$ from $\mathcal{D}_U^{(n)}$.

Once again, by plugging in the correct group ensemble, we will obtain the corresponding decision problems for k-SUM and k-CYC.

Definition C.2 (k-SUM Decision Problem)**.** The k-SUM decision problem is the k-LIST decision problem with group ensemble $(\mathsf{G}^{(m)}_{k\text{-}\mathsf{SUM}}, +)$.

Definition C.3 (k-CYC Decision Problem)**.** The k-$\mathsf{CYC}^{\mathsf{G}}$ decision problem is the k-LIST decision problem with cyclic group ensemble G.

Next, we can present conjectures for the decisions variants of both k-SUM and k-CYC following similar ideas as the search variants. In general, we will use the same group sizes of $m = \Theta(n^k)$ to rule out trivial algorithms as well as faster algorithms for the dense setting.

Definition C.4 (AVG-k-SUM Decision Conjecture)**.** For any integer $k \geq 3$, group size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-SUM^m decision problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.

Definition C.5 (AVG-k-CYC Decision Conjecture)**.** For any integer $k \geq 3$, group size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-PROD^m decision problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.

Finally, we discuss the equivalences between the search and decision variants of k-SUM and k-CYC. First, it is clear that one direction is true. If one can solve the search variants of k-SUM and k-CYC, one can easily solve the decision k-SUM and k-CYC problems. For the opposite direction, we show an equivalence for the sparse regime where $m \geq c \cdot n^k$ for some constant $c > 1$. We note that prior work [6] has considered similar reductions but with two main differences. First, their reduction considers decisions algorithms with $o(1)$ error probability and converts it into algorithms for the planted search problems with $O(1)$ error probability. Secondly, they consider the case with a single list of n elements. We present a similar, but slightly different, algorithm that will maintain $1/\mathsf{poly}(n)$ error probability for the case of k input lists. Furthermore, we consider relations between the decision and (non-planted) search problems. We present the algorithm below that uses algorithm A for the decision problem:

```
$A'(L_1, \ldots, L_k)$ :
  1. For $i = 1, \ldots, O(\log n)$:
     (a) For $j = 1, \ldots, c_j = O(2^k \log n)$:
         i. Construct $L'_1, \ldots, L'_k$ by sampling a random half from $L_1, \ldots, L_k$.
```

ii. Initialize counter $\mathsf{cnt} \leftarrow 0$.

iii. For $k = 1, \ldots, O(\log n)$:

A. Construct $X_1, \ldots, X_k$ by starting with $L'_1, \ldots, L'_k$ and padding with random elements until each are size n.

B. If $A(L'_1, \ldots, L'_k) = 1$, increment cnt by 1.

iv. If cnt shows at least γ of trials are true, set $L_1 \leftarrow L'_1, \ldots, L_k \leftarrow L'_k$ and go to next iteration of Step 1

v. If $j = c_j$, then return $\perp$.

2. Run trivial algorithm on remaining lists $L_1, \ldots, L_k$ and return answer.

Theorem C.6. *For all $k \geq 3$ and (group ensemble G such that) $m \geq c \cdot n^k$ for some constant $c > 1$, suppose there exists an algorithm that runs in time in $T(n)$ and solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) decision problem with error probability $1/\mathsf{poly}(n)$. Then, for any constant $0 < \epsilon < 1$, there exists an algorithm that solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) search problem that runs in time $\tilde{O}(T(n) + n)$ with error probability $1/\mathsf{poly}(n)$.*

Proof. We will show that A' is an algorithm that solves the search problem assuming A is an algorithm for the decision problem. We show that A' maintains the following loop invariant for index i where it will only continue to the next iteration if $L_1, \ldots, L_k$ contains a solution. If so, the modified lists will also contain a solution.

Suppose the input lists $L_1, \ldots, L_k$ contain at least one solution. We note that the solution is sampled into $L'_1, \ldots, L'_k$ with probability $1/2^k$. So, there exists at least one loop that samples the solution except with probability n^{-100} since we run $c_j = O(2^k \log n)$ iterations for j. Assuming $L'_1, \ldots, L'_k$ contain a solution, we note that A returns 1 with probability $1 - 1/\mathsf{poly}(n)$ for any constant $0 < \gamma < 1$. By Chernoff's bound, we can immediately see that this will trigger halving the lists and going back to Step 1 except with probability n^{-100}. In contrast, suppose that $L'_1, \ldots, L'_k$ do not contain a solution. Since $m \geq c \cdot n^k$, we know that the input lists $X_1, \ldots, X_k$ contain a solution with probability at most $n^k/m = 1/c$ for some constant $c > 1$. We pick any constant $\gamma > 1/c$. Then, A returns 1 with probability $1/\mathsf{poly}(n)$ meaning that 2/3 of trials will succeed only with probability n^{-100}. As a result, it will never be that the input lists $L_1, \ldots, L_k$ are halved such that they do not include a solution except with probability $O(2^k \log^2 n) \cdot n^{-100} < n^{-99}$ if $L_1, \ldots, L_k$ originally contained a solution. Therefore, we see that all $O(\log n)$ iterations of i will halve lists and keep a solution except with probability at most $n \cdot n^{-99} = n^{-98}$. In that case, the algorithm will run the trivial algorithm and return a correct solution.

For the other case, suppose that $L_1, \ldots, L_k$ does contain a solution. Then, we can see that $L'_1, \ldots, L'_k$ never contains a solution too. Using the same argument, we see that A' will output $\perp$ except with probability $O(2^k \log n) \cdot n^{-100} < n^{-99}$.

Finally, for the runtime, we note that we run algorithm A on inputs lists of size n $O(2^k \log^3 n)$ times. Additionally, we run the trivial algorithm on $O(1)$ input lists that runs in $O(1)$ time. Therefore, if A runs in time $T(n)$, then A' runs in time $\tilde{O}(T(n))$ as claimed. $\square$

In other words, whenever the group size $m(n) > c \cdot n^k$ for some constant $c > 1$, we can essentially use the fact that the decision and search problems of k-SUM and k-CYC are equivalent. Additionally, our algorithm and analysis may easily adapted to the case of a single input list as well by adjusting m accordingly as studied in [6].

D Planted k-SUM and k-CYC

We formally present definitions of the planted problems. At a high level, they will take a random instance from $\mathcal{D}_U^{(n)}$ and embed a random solution into the instance. In this case, it is clear that only the search problem is interesting as a solution always exists. We start by defining the planted distribution $\mathcal{D}_P^{(n)}$ using the group ensemble $\mathsf{G} = \{\mathbb{G}^{(n)}\}$ with group operation $\odot$:

$\mathcal{D}_P^{(n)}$**:**

1. Generate k lists, $L_1, \ldots, L_k$ of size n where each element of all k lists is drawn uniformly at random from $\mathbb{G}^{(n)}$.
2. Pick uniformly random indices $y_1, \ldots, y_k$ from $[n]^k$.
3. Pick random $j \in [k]$.
4. Set $L_j[y_j] \leftarrow (L_1[y_1] \odot \ldots \odot L_{j-1}[y_{j-1}] \odot L_{j+1}[y_{j+1}] \odot \ldots \odot L_k[y_k])^{-1}$.
5. Return $L_1, \ldots, L_k$.

We are now ready to present the planted search problems using the above distribution $\mathcal{D}_P^{(n)}$.

Definition D.1 (k-List Planted Search Problem)**.** For $k \geq 0$ and group ensemble $(\mathsf{G}, \odot)$, an algorithm A correctly solves the k-list planted search problem if it outputs $(y_1, \ldots, y_k) \in [n]^k$ satisfying the equation $L_1[y_1] \odot \ldots \odot L_k[y_k] = 1$. We say algorithm A has failure probability $\epsilon(n)$ if

$$\Pr[A(L_1, \ldots, L_k) \text{ is correct}] \geq 1 - \epsilon(n)$$

where the randomness is over the internal coin tosses of A and random choice of $L_1, \ldots, L_k$ from $\mathcal{D}_P^{(n)}$.

By plugging in the correct ensemble, we obtain the planted search versions of k-SUM and k-CYC.

Definition D.2 (k-SUM Planted Search Problem)**.** The k-SUM^m planted search problem is the k-LIST planted search problem with group ensemble $(\mathsf{G}_{k\text{-SUM}}^{(m)}, +)$.

Definition D.3 (k-CYC Planted Search Problem)**.** The k-CYC$^{\mathsf{G}}$ planted search problem is the k-LIST planted search problem with cyclic group ensemble G.

Finally, we present the conjectures for the planted search versions of k-SUM and k-CYC.

Definition D.4 (AVG-k-SUM Planted Conjecture)**.** For any integer $k \geq 3$, group size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-SUM^m planted search problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.

Definition D.5 (AVG-k-CYC Decision Conjecture)**.** For any integer $k \geq 3$, cyclic group ensemble G with size $m = \Theta(n^k)$ and constant $\epsilon > 0$, there exists no algorithm that solves the k-CYC$^{\mathsf{G}}$ planted search problem with error $1/\mathsf{poly}(n)$ in time $O(n^{\lceil k/2 \rceil - \epsilon})$.

Finally, we consider relations between the planted search and search problems. We note that we can essentially treat the planted search and search problems as equivalent in the dense setting where $m(n) \leq c \cdot n^k$ for any constant $c < 1$ using the results in [6] where it was shown that the planted distribution $\mathcal{D}_P^{(n)}$ and non-planted distribution $\mathcal{D}_U^{(n)}$ have at most $1/\mathsf{poly}(n)$ statistical distance meaning that algorithms for the planted search problem will also work for the standard search problem. We note the prior work considered one single input list as opposed to k input lists, but the proof remains identical.

Theorem D.6 ([6])**.** *For all constant $k \geq 3$ and (group ensemble G such that) $m \leq c \cdot n^k$ for some constant $c < 1$, suppose there exists an algorithm that runs in time $T(n)$ and solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) planted search problem with error probability $1/\mathsf{poly}(n)$. Then, there exists an algorithm that solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) search problem in time $T(n)$ with error probability $1/\mathsf{poly}(n)$.*

Additionally, we note that equivalences were shown even when $m = n^k$ in [6]. However, these reductions incur larger error probability increases.

E Single-List k-SUM and k-CYC

We present another common variant of k-SUM and k-CYC where the input consists of a single list of size n. As a caveat, we will focus on the setting where we insist the k indices of the solution have to be unique. In the worst-case, these are equivalent. However, we are unaware of an equivalence in the average-case. We choose the unique index solution setting as we are able to present a reduction to the k-list search problem.

Definition E.1 (Single List Search Problem)**.** For $k \geq 0$ and group ensemble $(\mathsf{G}, \odot)$, an algorithm A correctly solves the k-LIST single list search problem if it outputs the following in each corresponding scenario:

- If there exists a solution, A outputs $y_1 \neq \ldots \neq y_k \in [n]^k$ satisfying the equation $L[y_1] \odot \ldots \odot L[y_k] = 1$.
- If no solution exists, A outputs $\perp$.

We say algorithm A has failure probability $\epsilon(n)$ if

$$\Pr[A(L) \text{ is correct}] \geq 1 - \epsilon(n)$$

where the randomness is over the internal coin tosses of A and random choice of L from $\mathcal{D}_U^{(n)}$.

Definition E.2 (k-SUM Single List Search Problem)**.** The k-SUM single list search problem is the single list search problem with group ensemble $(\mathsf{G}_{k\text{-}\mathsf{SUM}}^{(m)}, +)$.

Definition E.3 (k-CYC Single List Search Problem)**.** The k-CYC single list search problem is the single list search problem with cyclic group ensemble G.

Next, we show that there exists a way to reduce random input instances of the single list search problem into random input instances of the k-LIST search problem. We present the reduction algorithm A' below that uses an algorithm A that solves the k-LIST search problem.

$A'(L)$**:**

```
1. For $i = 1, \ldots, O(k^k \cdot \log n)$:
   (a) Initialize empty lists $L_1, \ldots, L_k$.
   (b) For each $x \in L$, pick uniformly random $i \in [k]$ and append $x$ to $L_i$.
   (c) For $j = 1, \ldots, O(\log n)$:
       i. Pad each $L_i$ to size $n$ by appending uniformly random elements.
       ii. Execute $r \leftarrow A(L_1, \ldots, L_k)$.
       iii. If $r \neq \perp$:
            A. If $r = (y_1, \ldots, y_k) \in [n]^k$ is a valid solution in $L$, return $r$.
2. Return $\perp$.
```

Theorem E.4. *For all constant $k \geq 3$ and (group ensemble G such that) $m \geq c \cdot n^k$ for some constant $c > 1$, suppose there exists an algorithm that runs in time $T(n)$ and solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) search problem with error $1/\mathsf{poly}(n)$. Then, there exists an algorithm that solves the k-SUM^m (k-$\mathsf{CYC}^{\mathsf{G}}$) single list search problem in time $\tilde{O}(T(n) + n)$ with error $1/\mathsf{poly}(n)$.*

Proof. Suppose that L contains a solution. For any iteration of i, we know the probability that the k different summands appear in different lists. This occurs with probability at least $1/k^k = \Theta(1)$ since k is constant. Since there are $O(k^k \cdot \log n)$ iterations, we know that there exists at least one iteration of i where the each entry of the solution appears in different lists except with probability $n^{-c'}$ for any constant $c' > 0$. Now, consider any iteration of j assuming the solution appears in different lists. We note the probability that an additional solution that contains one of the padding random elements is at most $n^k/m = 1/c$. As j iterates $O(\log n)$, we know there exists one iteration of j where no additional solution contained a random padded element exists. In this case, we know that A' will return the correct solution with probability at least $1/\mathsf{poly}(n)$. Additionally, we note that A' never returns an incorrect non-$\perp$ solution (such as one including a padded random element). So, if A' breaks out during any of the iterations for j, it will return a correct solution. Therefore, if there is a solution in L, A' returns it with probability $1/\mathsf{poly}(n) - 1/n^c = 1/\mathsf{poly}(n)$ for sufficiently large c. Similarly, if no solution exists in L, then A will always return $\perp$.

Suppose A runs in time $T(n)$. Note, that we run A at most $O(k^k \log^2 n) = O(\log^2 n)$ times. Additionally, there is at most $\tilde{O}(n)$ running time for preparing the inputs to A as well as checking solutions. Altogether, A' has running time $\tilde{O}(T(n) + n)$ time. □

F Extensions of Theorem 4.1

We present some extensions of the DLog barrier for k-CYC to k-SUM reductions in Theorem 4.1. In particular, we show that one can construct sub-exponential algorithms when assuming slightly more efficient reductions and larger $k = \omega(1)$. Finally, we show the implications of the results towards other number-theoretic assumptions and compare with the current state-of-the-art algorithms.

F.1 Sub-Exponential Algorithms using $k = \omega(1)$ Reductions

In Theorem 4.1, we showed any reduction running in time $n^{k/2-\Omega(1)}$ would result in faster algorithms for DLog running in time $m^{1/2-\Omega(1)}$ for groups $\mathbb{G}$ of order m. This would result in polynomial time speed-ups in m for DLog compared to the best known algorithms that run in time $\tilde{O}(m^{1/2})$ for popular elliptic curve groups (see [45, 60] for details). We show that one can essentially consider a variant of Theorem 4.1 where we consider $\mathsf{poly}(n)$ time reductions for super-contant choices of $k = \omega(1)$. This would immediately result in sub-exponential time algorithms for the DLog problem as well as the related number-theoretic problems (such as the factoring, RSA and quadratic residuosity problems). These algorithms run in time $m^{o(1)}$ that is sub-exponential in the input size of $O(\log m)$.

Theorem F.1. *For any $k = \omega(1)$ and cyclic group ensemble G with size m, suppose that there exists an efficient reduction in time $\mathsf{poly}(n)$ from a random instance of size n for k-$\mathsf{CYC}^{\mathsf{G}}$ to a random instance of size $O(n)$ for k-SUM^m. Then, there exists a sub-exponential $m^{o(1)}$ time algorithm solving $\mathsf{DLog}^{\mathsf{G}}$.*

Proof. The proof follows identically to Theorem 4.1 where we first map a random instance of $\mathsf{DLog}^{\mathsf{G}}$ to a random dense instance of k-$\mathsf{CYC}^{\mathsf{G}}$ where $n = m^{(1+c)/k}$ for some constant $c > 0$. Afterwards, we use the reduction to obtain a random k-SUM^m instance with input lists of size $O(n)$. Finally, we execute Wagner's algorithm (see Theorem A.2) that solves the random k-SUM^m instance in time $m^{o(1)}$ for some constant $\epsilon > 0$. As a result, we obtain an algorithm for DLog that runs in $\tilde{O}(n) + m^{o(1)} = m^{o(1)}$ time as $n = m^{(1+c)/k} = m^{o(1)}$ since $k = \omega(1)$. □

As an immediate corollary, we also obtain sub-exponential time algorithms for related number-theoretic algorithms using prior works [9] showing that DLog over composite order groups enables factoring.

Corollary F.2. *For any $k = \omega(1)$, suppose that there exists an efficient reduction in time $\mathsf{poly}(n)$ from a random instance of size n for k-PROD^m to a random instance of size $O(n)$ for k-SUM^m. Then, there exists a sub-exponential $m^{o(1)}$ time algorithms for the factoring, RSA and quadratic residuosity problems.*

F.2 Factoring

We show the implications of Corollary F.2 for factoring z-bit integers. The current, state-of-the-art factoring algorithms can be divided into two categories: algorithms with provable running time guarantees and algorithms with heuristic running time guarantees. The fastest algorithms for factoring used in practice today typically fall from the latter group with only heuristic running time guarantees such as the quadratic sieve and the general number field sieve (see Chapter 6 in [32] for more details). Given an z-bit integer input, both of these algorithms are conjectured to run in sub-exponential $2^{o(z)}$ time.

For the category of algorithms with provable running times, the best algorithm remains variants of the Pollard-Strassen method [75] that runs in time $2^{z/4+o(1)}$ when given a z-bit integer input. There are also algorithms that run in time $2^{z/5+o(1)}$ time assuming the generalized Riemann hypothesis [23]. It remains a major barrier to obtain provably secure algorithms in time $2^{z/4-\Omega(1)}$ without relying on other number-theoretic conjectures. Relating back to Corollary F.2, we immediately see that an average-case reduction from k-PROD to k-SUM in $\mathsf{poly}(n)$ time for $k = \omega(1)$ would result in a provably sub-exponential algorithm for factoring that would be a huge breakthrough in the area. In fact, any $n^{k/5+o(1)}$ reduction for $k \geq 16$ from k-PROD to k-SUM would result in a provably $2^{z/5+o(1)}$ time algorithm for factoring that would also be a substantial breakthrough.

F.3 RSA and Quadratic Residuosity

Next, we show implications of reductions from k-PROD to k-SUM with respect to the state-of-the-art algorithms for the RSA problem [76]. Recall that the RSA problem considers a modulus of the form $m = pq$ where p and q are prime numbers. The public parameters of the RSA problem are some modulus m and public exponent $e \in \mathbb{Z}^*_{\phi(m)}$. For a random target $t \in \mathbb{Z}_m$, the goal is to output some element $x \in \mathbb{Z}_m$ such that $x^e = t$. In other words, the output x is the e-th root of the target t in $\mathbb{Z}_m$. It is well known that factoring the modulus m into its factors p and q is sufficient to solve the RSA problem (see [22] for example). Prior works have also shown some evidence that solving the RSA problem seems equivalent to factoring. For example, any algorithms that solve the RSA problem in the generic ring model (analogous to the generic group model) would result in a factoring algorithm as well [5]. This ends up being true even in the case of preprocessing [33]. There is prior work [57] that have obtained algorithms for computing e-th roots faster than factoring that modify the quadratic sieve and general number field sieve algorithms. Once again, these rely upon algorithms with only heuristic running time guarantees. Using Corollary F.2, we obtain provably sub-exponential time algorithms for RSA (via factoring) that would also constitute a major breakthrough in RSA attacks.

It turns out that the landscape is similar for the quadratic residuosity problem [50]. Again, the problem consists of a public modulus $m = pq$ where the prime factors p and q are hidden. Given an integer a such that $\left(\frac{a}{m}\right) = 1$ using the Legendre symbol, the goal is to determine whether a is a quadratic residue modulo m. That is, determine if there exists b such that $a = b^2 \mod m$. To our knowledge, the best attacks on the quadratic residuosity assumption with provable time guarantees are the factoring algorithms. Therefore, Corollary F.2 would again constitute a huge breakthrough in attacks with provable running times for the quadratic residuosity problem.

F.4 Computational and Decisional Diffie-Hellman

We can also consider relations with the computational Diffie-Hellman (CDH) and decisional Diffie-Hellman (DDH) problems. It is clear that any algorithms that successfully solves the DLog problem may be immediately used to solve either the CDH or DDH problems. It turns out that the best algorithms for either CDH or DDH on popular elliptic curve groups remain the best generic algorithms for solving DLog [45, 60] that run in time $\tilde{O}(m^{1/2})$ for groups of order m. In the generic group model, prior work [30] has been shown that the efficiencies for all three of DLog, CDH and DDH are the same (in the constant error regime). Therefore, Theorem 4.1 and Corollary F.2 would result in faster algorithms for both CDH and DDH constituting another breakthrough in cryptographic attacks.

G Supplementary Material for Section 4

The first result from prior works by Dai [34] and Wagner [80] essentially shows that faster algorithms for k-CYC would result in faster algorithms for DLog.

Theorem G.1 ([34, 80])**.** *For all $k \geq 3$ and group ensemble G such that $m = \Theta(n^k)$, suppose there exists an algorithm that runs in time $T(n)$ for the k-$\mathsf{CYC}^{\mathsf{G}}$ search problem with error probability $1/\mathsf{poly}(n)$. Then, there exists an algorithm that solves the $\mathsf{DLog}^{\mathsf{G}}$ problem in time $\tilde{O}(T(n))$ with error probability $1/\mathsf{poly}(n)$.*

The other result is Hellman's algorithm [52] as well as the analysis by Fiat and Naor [44] for inverting random functions.

Theorem G.2 ([52, 44]). *Given a function $f : \mathcal{X} \to \mathcal{X}$ where the probability that any two random elements map to the same image of f is q. There exists an algorithm that uses space S and time T to compute inverses of f with error probability $1/\mathsf{poly}(|\mathcal{X}|)$ such that $S^2T = \tilde{O}(q \cdot |\mathcal{X}|^3)$.*

H Universal One-way Function

This appendix section gives a self-contained discussion of what we need for universal one-way functions, adapted from [47, Section 2.4.1]

Recall that we call a function f *length-regular* if for any two inputs x_1, x_2 such that $|x_1| = |x_2|$, it is also true that $|f(x_1)| = |f(x_2)|$, and we call a length-regular function *length-increasing* if for any two inputs x_1, x_2 such that $|x_1| < |x_2|$, it is also true that $|f(x_1)| < |f(x_2)|$.

We assume every function we discuss is length-increasing, unless otherwise stated. This is without loss of generality, because for every function f computable in $p(\cdot)$ time where p is an increasing function, we can instead consider the function f' defined as $f'(x) = f(x)\|10^{p(|x|)-|f(x)|}$. It is easy to see that f' can be computed in $\tilde{O}(p(\cdot))$ time, and any algorithm inverting f' can be easily converted into an algorithm inverting f and vice versa, up to a logarithmic factor time overhead.

Lemma H.1. *Suppose $\ell \geq 2$ is a constant and $f : \{0,1\}^* \to \{0,1\}^*$ is a function computable in n^ℓ time where n is the input length. Then there exists a function $g : \{0,1\}^* \to \{0,1\}^*$ computable in N^2 time where N is the input length, such that, if there exists a (possibly randomized) algorithm B, constant $d > 0$, constant $c \geq 2$, a function $\varepsilon : \mathbb{Z}_{>0} \to [0,1]$ and infinitely many N such that*

$$\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^N)}[g(B(1^N, g(y))) = g(y)] \geq 1 - \varepsilon(N)$$

and B runs in N^c time when the input is $g(y)$ and $y \in \{0,1\}^N$, then there exists a (possibly randomized) algorithm A and infinitely many n such that

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[f(A(1^n, f(x))) = f(x)] \geq 1 - \varepsilon(n^\ell)$$

and $A(1^n, f(x))$ runs in $n^{\ell(c+1)}$ time for $x \in \{0,1\}^n$.

Proof. Define g as follows: for any $n > 0$, $x \in \{0,1\}^n$, $n^\ell - n \leq t \leq (n+1)^\ell - (n+1)$ and $x' \in \{0,1\}^t$, let $g(x\|x') := f(x)\|x'$. (Note that if f is length-increasing, then g is also length-increasing.)

Suppose there is an algorithm B, $d > 0$, $c \geq 2$ and infinitely many N as in the lemma statement. Let A be the following algorithm:

1. Suppose the input is $(1^n, z)$.
2. For every $n^\ell - n \leq t \leq (n+1)^\ell - (n+1)$, uniformly sample $z_i \leftarrow \{0,1\}^t$.
3. For every $n^\ell - n \leq t \leq (n+1)^\ell - (n+1)$, run $B(1^{n+t}, z\|z_t)$ for $(n+t)^c$ time, and if B halts and outputs an $(n+t)$-bit string, let the output be denoted by x_t, otherwise let $x_t = \perp$.

4. If any t satisfies $g(x_t) = z\|z_t$, output the n-bit prefix of x_t, otherwise output $\perp$ (if there are more than one t that satisfies this, choose one of them arbitrarily).

Since there are $O(n^{\ell-1})$ possibilities fo t and the running time of B is $(n+t)^c$, the running time of A is $O(n^{\ell-1}) \cdot \tilde{O}((n+t)^c)$, which is at most $n^{\ell(c+1)}$ when n is sufficiently large.

For any N that satisfies the lemma statement, let (n,t) be the unique pair such that $n^\ell \leq N < (n+1)^\ell$ and $n+t = N$. Note that if $g(x_t) = z\|z_t$, then $f(n\text{-bit prefix of } x_t) = z$ and t-bit suffix of $x_t = z_t$. We thus have

$$\begin{aligned}
&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[f(A(1^n, f(x))) = f(x)] \\
\geq &\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n), z_t \leftarrow \mathcal{U}(\{0,1\}^t)}[f(n\text{-bit prefix of } B(1^N, f(x)\|z_t)) = f(x)] \\
\geq &\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n), z_t \leftarrow \mathcal{U}(\{0,1\}^t)}[f(n\text{-bit prefix of } B(1^N, f(x)\|z_t))\|(t\text{-bit suffix of } B(1^N, f(x)\|z_t)) = f(x)\|z_t] \\
= &\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n), z_t \leftarrow \mathcal{U}(\{0,1\}^t)}[g(B(1^N, g(x\|z))) = g(x\|z) \wedge |B(1^N, g(x\|z))| = N] \\
= &\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^N)}[g(B(1^N, g(y))) = g(y)]. \qquad \text{(Since } g \text{ is length-increasing)} \\
\geq &\, 1 - \varepsilon(N) \geq 1 - \varepsilon(n^\ell).
\end{aligned}$$

Since there are infinitely many N that satisfy the lemma statement, there is also infinitely many n that satisfy the above. $\square$

Let $Q : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*, (M,x) \mapsto 0^{|M|-1}1\|M\|x$. It is easy to see that given a string y, there is either at most one pair (M,x) such that $Q(M,x) = y$. Q can be understood as an encoding of two strings into one string. Note that $|Q(M,x)| = 2|M| + |x|$.

Lemma H.2. *Let U be a universal Turing machine with logarithmic overhead in the running time. Consider the following function $u : \{0,1\}^* \to \{0,1\}^*$ defined as follows:*

$$u(Q(M,x)) \coloneqq Q(M, U^{|x|^2}(M,x)).$$

That is, u parses the input as (M,x) and simulates M on input x with at most $|x|^2$ time, and then outputs M the output of M (encoded with Q). If the input cannot be parsed as (M,x), let u output $0^{|Q(M,x)|^2+1}$, which u never outputs if the input can be parsed as (M,x).

If there exists a (possibly randomized) algorithm B, constant $d > 0$, constant $c \geq 2$, a function $\varepsilon : \mathbb{Z}_{>0} \to [0,1]$ and infinitely many N such that

$$\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^N)}[u(B(1^N, u(y))) = u(y)] \geq 1 - \varepsilon(N)$$

and B runs in N^c time when the input is $u(y)$ and $y \in \{0,1\}^N$, then for any quadratic-time length-increasing Turing machine M, there exists a (possibly randomized) algorithm A and infinitely many n such that

$$\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[M(A(1^n, M(x))) = M(x)] \geq 1 - 2^{2|M|}\varepsilon(n)$$

and $A(1^n, M(x))$ runs in n^{c+1} time for $x \in \{0,1\}^n$.

Proof. Suppose there is an algorithm B, constant $d > 0$, constant $c \geq 2$ and infinitely many N that satisfies the lemma statement. Let A be the following algorithm:

1. Suppose the input is $(1^n, z)$.
2. Run $B(1^{2|M|+n}, Q(M, z))$ for $(2|M| + n)^c$ time. If B halts and outputs an $(2|M| + n)$-bit string, parse the output as $Q(M', x')$. Otherwise let $M' = \perp$ and $x' = \perp$.
3. If $M' = M$ and $M(x') = z$, output x', otherwise output $\perp$.

Since the running time of B is $(2|M| + n)^c$, the running time of A is at most n^{c+1} when n is sufficiently large.

For any N that satisfies the lemma statement, let $n = N - 2|M|$. We have

$$\begin{aligned}
&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[M(A(1^n, M(x))) = M(x)] \\
=&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[M' = M \wedge M(A(1^n, M(x))) = M(x)] \qquad (A \text{ does not output } \perp \text{ only if } M = M') \\
=&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[(M', M'(A(1^n, M(x)))) = (M, M(x))] \\
=&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[u(B(1^N, Q(M, M(x)))) = Q(M, M(x))] \\
=&\Pr_{x \leftarrow \mathcal{U}(\{0,1\}^n)}[u(B(u(Q(M, x)))) = u(Q(M, x))] \\
=&\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^{2|M|+n})}[u(B(u(y))) = u(y) \mid \text{the } (2|M|)\text{-bit prefix of } y \text{ is } 0^{|M|-1}1\|M] \\
\geq&\frac{\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^{2|M|+n})}[u(B(u(y))) = u(y)] - \Pr_{y \leftarrow \mathcal{U}(\{0,1\}^{2|M|+n})}[\text{the } (2|M|)\text{-bit prefix of } y \text{ is not } 0^{|M|-1}1\|M]}{\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^{2|M|+n})}[\text{the } (2|M|)\text{-bit prefix of } y \text{ is } 0^{|M|-1}1\|M]} \\
=&\, 1 - \frac{2^{2|M|}}{n^d}.
\end{aligned}$$

Since there are infinitely many N that satisfy the lemma statement, there is also infinitely many n that satisfy the above. □

Now we prove Lemma 3.1.

Proof of Lemma 3.1. Let u be the function as defined in Lemma H.2. Since there is no one-way function, for any constant $d > 0$, there exists an N^d-time algorithm B such that for infinitely many N,

$$\Pr_{y \leftarrow \mathcal{U}(\{0,1\}^N)}[u(B(1^N, u(y))) = u(y)] \geq 1 - \frac{1}{N^d}.$$

We apply Lemma H.2 and then Lemma H.1 to reach the conclusion. □