

Data Structures Meet Cryptography: 3SUM with Preprocessing

Alexander Golovnev
Harvard
alexgolovnev@gmail.com

Siyao Guo
NYU Shanghai
sg191@nyu.edu

Thibaut Horel
MIT
thibauth@mit.edu

Sunoo Park
MIT & Harvard
sunoo@csail.mit.edu

Vinod Vaikuntanathan
MIT
vinodv@csail.mit.edu

Abstract

This paper shows several connections between data structure problems and cryptography against preprocessing attacks. Our results span data structure upper bounds, cryptographic applications, and data structure lower bounds, as summarized next.

First, we apply Fiat–Naor inversion, a technique with cryptographic origins, to obtain a data structure upper bound. In particular, our technique yields a suite of algorithms with space S and (online) time T for a preprocessing version of the N -input 3SUM problem where $S^3 \cdot T = \tilde{O}(N^6)$. This disproves a strong conjecture (Goldstein *et al.*, WADS 2017) that there is no data structure that solves this problem for $S = N^{2-\delta}$ and $T = N^{1-\delta}$ for any constant $\delta > 0$.

Secondly, we show equivalence between lower bounds for a broad class of (static) data structure problems and one-way functions in the random oracle model that resist a very strong form of preprocessing attack. Concretely, given a random function $F : [N] \rightarrow [N]$ (accessed as an oracle) we show how to *compile* it into a function $G^F : [N^2] \rightarrow [N^2]$ which resists S -bit preprocessing attacks that run in query time T where $ST = O(N^{2-\varepsilon})$ (assuming a corresponding data structure lower bound on 3SUM). In contrast, a classical result of Hellman tells us that F itself can be more easily inverted, say with $N^{2/3}$ -bit preprocessing in $N^{2/3}$ time. We also show that much stronger lower bounds follow from the hardness of kSUM. Our results can be equivalently interpreted as security against adversaries that are very non-uniform, or have large auxiliary input, or as security in the face of a powerfully backdoored random oracle.

Thirdly, we give non-adaptive lower bounds for 3SUM which match the best known lower bounds for static data structure problems. Moreover, we show that our lower bound generalizes to a range of geometric problems, such as three points on a line, polygon containment, and others.

Contents

1	Introduction	1
1.1	3SUM and 3SUM-Indexing	1
1.2	Our Results	2
1.2.1	Upper bound for 3SUM-Indexing	2
1.2.2	Lower bound for 3SUM-Indexing and beyond	3
1.2.3	Cryptography against massive preprocessing attacks	3
1.3	Other related work	6
2	Preliminaries	7
2.1	Notation	7
2.2	kSUM-Indexing	7
2.2.1	Average-case hardness	8
3	Upper bound	9
4	Lower bound	11
4.1	3SUM-Indexing-hardness	15
5	Cryptography against massive preprocessing attacks	17
5.1	Background on random oracles and preprocessing	17
5.2	Constructing one-way functions from kSUM-Indexing	18
5.3	Cryptography with preprocessing and data structures	21
5.3.1	Cryptography with preprocessing and circuit lower bounds	24

1 Introduction

Cryptography and data structures have long enjoyed a productive relationship [Hel80, FN00, LN93, NSW08, BHKN13, BN16, ANSS16, NY15, LN18, JLN19]: indeed, the relationship has been referred to as a “match made in heaven” [Nao13]. In this paper, we initiate the study of a new connection between the two fields, which allows us to construct novel cryptographic objects starting from data structure lower bounds, and vice versa. Our results are three-fold. Our first result is a new upper bound for a data structure version of the classical 3SUM problem (called 3SUM-Indexing) using Fiat–Naor inversion [FN00], a technique with cryptographic origins. This result refutes a strong conjecture due to Goldstein, Kopelowitz, Lewenstein and Porat [GKLP17]. In our second and main result, we turn this connection around, and show a framework for constructing *one-way functions* in the random oracle model whose security bypasses known time/space tradeoffs, relying on any of a broad spectrum of (conjectured) data structure lower bounds (including for 3SUM-Indexing). As a third result, we show new lower bounds for a variety of data structure problems (including for 3SUM-Indexing) which match the state of the art in the field of static data structure lower bounds.

Next, we describe our results, focusing on the important special case of 3SUM-Indexing; all of our results and methods extend to the more general k SUM-Indexing problem where pairwise sums are replaced with $(k - 1)$ -wise sums for an arbitrary constant integer k independent of the input length. Section 1.1 gives background on 3SUM-Indexing, then Section 1.2 discusses our contributions.

1.1 3SUM and 3SUM-Indexing

One of the many equivalent formulations of the 3SUM problem is the following: given a set A of N integers, output $a_1, a_2, a_3 \in A$ such that $a_1 + a_2 = a_3$. There is an easy $O(N^2)$ time deterministic algorithm for 3SUM. Conversely, the popular 3SUM conjecture states that there are no sub-quadratic algorithms for this problem [GO95, Eri99].

Conjecture 1 (The “Modern 3SUM conjecture”). *3SUM cannot be solved in time $O(N^{2-\delta})$ for any constant $\delta > 0$.*

Conjecture 1 has been helpful for understanding the precise hardness of many geometric problems [GO95, dBdGO97, BVKT98, ACH⁺98, Eri99, BHP01, AHI⁺01, SEO03, AEK05, EHPM06, CEHP07, AHP08, AAD⁺12]. Furthermore, starting with the works of [VW09, Pät10], the 3SUM conjecture has also been used for conditional lower bounds for many combinatorial [AV14, GKLP16, KPP16] and string search problems [CHC09, BCC⁺13, AVW14, ACLL14, AKL⁺16, KPP16].

Our main results relate to a preprocessing variant of 3SUM known as 3SUM-Indexing, which was first defined by Demaine and Vadhan [DV01] in an unpublished note and then by Goldstein, Kopelowitz, Lewenstein and Porat [GKLP17]. In 3SUM-Indexing, there is an offline phase where a computationally *unbounded* algorithm receives $A = \{a_1, \dots, a_N\}$ and produces a data structure with S words of w bits each; and an online phase which is given the target b and needs to find a pair (a_i, a_j) such that $a_i + a_j = b$ by probing only T memory cells of the data structure (*i.e.*, taking “query time” T). The online phase does not receive the set A directly, and there is no bound on the computational complexity of the online phase, only the number of queries it makes.

There are two simple algorithms that solve 3SUM-Indexing. The first stores a sorted version of A as the data structure (so $S = N$) and in the online phase, solves 3SUM-Indexing in $T = O(N)$ time using the standard two-finger algorithm for 3SUM. The second stores all pairwise sums of A , sorted, as the data structure (so $S = O(N^2)$) and in the online phase, looks up the target b in $T = \tilde{O}(1)$ time.¹ There were no other algorithms known prior to this work. This led [DV01, GKLP17] to

¹The notation $\tilde{O}(f(N))$ suppresses poly-logarithmic factors in $f(N)$.

formulate the following three conjectures.

Conjecture 2 ([GKLP17]). *If there exists an algorithm which solves 3SUM-Indexing with preprocessing space S and $T = \tilde{O}(1)$ probes then $S = \tilde{\Omega}(N^2)$.*

Conjecture 3 ([DV01]). *If there exists an algorithm which solves 3SUM-Indexing with preprocessing space S and T probes, then $ST = \tilde{\Omega}(N^2)$.*

Conjecture 4 ([GKLP17]). *If there exists an algorithm which solves 3SUM-Indexing with $T = \tilde{O}(N^{1-\delta})$ probes for some $\delta > 0$ then $S = \tilde{\Omega}(N^2)$.*

These conjectures are in ascending order of strength:

$$\text{Conjecture 4} \Rightarrow \text{Conjecture 3} \Rightarrow \text{Conjecture 2}.$$

In terms of lower bounds, Demaine and Vadhan [DV01] showed that any 1-probe data structure for 3SUM-Indexing requires space $S = \tilde{\Omega}(N^2)$. They leave the case of $T > 1$ open. Goldstein *et al.* [GKLP17] established connections between Conjectures 2 and 4 and the hardness of Set Disjointness, Set Intersection, Histogram Indexing and Forbidden Pattern Document Retrieval.

1.2 Our Results

Our contributions are three-fold. First, we show better algorithms for 3SUM-Indexing, refuting Conjecture 4. Our construction relies on combining the classical Fiat–Naor inversion algorithm, originally designed for cryptographic applications, with hashing. Secondly, we improve the lower bound of [DV01] to arbitrary T . Moreover, we generalize this lower bound to a range of geometric problems, such as 3 points on a line, polygon containment, and others. As we argue later, any asymptotic improvement to our lower bound will result in a major breakthrough in static data structure lower bounds.

Finally, we show how to use the conjectured hardness of 3SUM-Indexing for a new cryptographic application: namely, designing cryptographic functions that remain secure with massive amounts of preprocessing. We show how to construct one-way functions in this model assuming the hardness of a natural average-case variant of 3SUM-Indexing. Furthermore, we prove that this construction generalizes to an *explicit equivalence* between certain types of hard data structure problems and OWFs in this preprocessing model. This setting can also be interpreted as security against backdoored random oracles, a problem of grave concern in the modern world.

We describe these results in more detail below.

1.2.1 Upper bound for 3SUM-Indexing

Theorem 1. *For every $0 \leq \delta \leq 1$, there is an adaptive data structure for 3SUM-Indexing with space $S = \tilde{O}(N^{2-\delta})$ and query time $T = \tilde{O}(N^{3\delta})$.*

In particular, Theorem 1 implies that by setting $\delta = 0.1$, we get a data structure that solves 3SUM-Indexing in space $S = \tilde{O}(N^{1.9})$ and $T = \tilde{O}(N^{0.3})$ probes, and thus refutes Conjecture 4.

In a nutshell, the upper bound starts by considering the function $f(i, j) = a_i + a_j$. This function has a domain of size N^2 but a potentially much larger range. In a preprocessing step, we design a hashing procedure to convert f into a function g with a range of size $O(N^2)$ as well, such that inverting g lets us invert f . Once we have such a function, we use Fiat and Naor [FN00]’s general space-time tradeoff for inverting functions, which gives non-trivial data structures for function *inversion* as long as function *evaluation* can be done efficiently. Due to our definitions of the

functions f and g , we can efficiently compute them at every input, which leads to efficient inversion of f , and, therefore, an efficient solution to 3SUM-Indexing. For more details, see Section 3. We note that prior to this work, the result of Fiat and Naor [FN00] was recently used by Corrigan-Gibbs and Kogan [CK19] for other algorithmic and complexity applications. In a concurrent work, Kopelowitz and Porat [KP19] obtain a similar upper bound for 3SUM-Indexing.

1.2.2 Lower bound for 3SUM-Indexing and beyond

We show that any algorithm for 3SUM-Indexing that uses a small number of probes requires large space, as expressed formally in Theorem 2.

Theorem 2. *For every non-adaptive algorithm that uses space S and query time T and solves 3SUM-Indexing, it holds that $S = \tilde{\Omega}(N^{1+1/T})$.*

The lower bound gives us meaningful (super-linear) space bounds for nearly logarithmic T . Showing super-linear space bounds for static data structures for $T = \omega(\log N)$ probes is a major open question with significant implications [Sie04, Păt11, PTW10, Lar12, DGW19].

The standard way to prove super-linear space lower bounds for $T = O(\log N)$ is the so-called *cell-sampling* technique. Applying this technique amounts to showing that one can recover a fraction of the input by storing a subset of data structure cells and then using an incompressibility argument. This technique applies to data structure problems which have the property that one can recover some fraction of the input given the answers to *any* sufficiently large subset of queries.

Unfortunately, the 3SUM-Indexing problem does not have this property and the cell-sampling technique does not readily apply. Instead we use a different incompressibility argument, closer to the one introduced by Gennaro and Trevisan in [GT00] and later developed in [DTT10, DGK17]. We argue that given a sufficiently large random subset of cells, with high probability over a random choice of input, it is possible to recover a constant fraction of the input. It is crucial for our proof that the input is chosen at random after the subset of data structure cells, yielding a lower bound only for *non-adaptive* algorithms.

Next, we show how to extend our lower bound to other data structure problems. For this, we define 3SUM-Indexing-hardness, the data structure analogue of 3SUM-hardness. In a nutshell, a data structure problem is 3SUM-Indexing-hard if there exists an efficient data structure reduction from 3SUM-Indexing to it. We then show how to adapt known reductions from 3SUM to many problems in computational geometry and obtain efficient reductions from 3SUM-Indexing to their data structure counterparts. This in turns implies that the lower bound in Theorem 2 carries over to these problems as well.

1.2.3 Cryptography against massive preprocessing attacks

In a seminal 1980 paper, Hellman [Hel80] initiated the study of algorithms for inverting (cryptographic) functions with preprocessing. In particular, given a function $F : [N] \rightarrow [N]$ (accessed as an oracle), an adversary can run in unbounded time and produce a data structure of S bits.² Later, given access to this data structure and (a possibly uniformly random) $y \in [N]$ as input, the goal of the adversary is to spend T units of time and invert y , namely output an $x \in [N]$ such that $F(x) = y$. It is easy to see that bijective functions F can be inverted at all points y with space S and time T where $ST = O(N)$. Hellman showed that a random function F can be inverted in space S and time T where $S^2T = O(N^2)$, giving in particular a

²The unbounded preprocessing time is amortized over a large number of function inversions. Furthermore, typically the preprocessing time is $\tilde{O}(N)$.

solution with $S = T = O(N^{2/3})$. Fiat and Naor [FN00] provided a rigorous analysis of Hellman’s tradeoff and additionally showed that a worst-case function can be inverted on a worst-case input in space S and time T where $S^3T = O(N^3)$, giving in particular a solution with $S = T = O(N^{3/4})$. A series of follow-up works [BBS06, DTT10, AAC⁺17] studied time-space tradeoffs for inverting one-way permutations, one-way functions and pseudorandom generators. In terms of lower bounds, Yao [Yao90] showed that for random functions (and permutations) $ST = \Omega(N)$. Sharper lower bounds, which also quantify over the success probability and work for other primitives such as pseudorandom generators and hash functions, are known from recent work [GGKT05, Umr07, DGK17, CDGS18, CDG18, AAC⁺17].

Hellman’s method and followups have been extensively used in practical cryptanalysis, for example in the form of so-called “rainbow tables” [Oec03]. With the increase in storage and available computing power (especially to large organizations and nation states), even functions that have no inherent weakness could succumb to preprocessing attacks. In particular, when massive amounts of (possibly distributed) storage is at the adversary’s disposal, S could be $\Omega(N)$, and the preprocessed string could simply be the function table of the inverse function F^{-1} which allows the adversary to invert F by making a single access to the S bits of preprocessed string.

One way out of this scenario is to re-design a new function F with a larger domain. This is a time-consuming and complex process [NIS01, NIS15], taking several years, and is fraught with the danger that the new function, if it does not undergo sufficient cryptanalysis, has inherent weaknesses, taking us out of the frying pan and into the fire.

We consider an alternative method that *immunizes* the function F against large amounts of preprocessing. In particular, we consider an adversary that can utilize $S \gg N$ bits of preprocessed advice, but can only access this advice by making a limited number of queries, in particular $T \ll N$. This restriction is reasonable when accessing the adversary’s storage is expensive, for example when the storage consists of slow but massive memory, or when the storage is distributed across the internet, or when the adversary fields a stream of inversion requests. (We note that while we restrict the number of queries, we do not place any restrictions on the runtime.)

In particular, we seek to design an *immunizing compiler* that uses oracle access to F to compute a function $G(x) = G^F(x)$. We want G to remain secure (for example, hard to invert) even against an adversary that can make T queries to a preprocessed string of length S bits. Both the preprocessing and the queries can depend on the design of the compiler G . Let $G : [N'] \rightarrow [N']$. To prevent the inverse table attack (mentioned above), we require that $N' > S$.

From Data Structure Lower Bounds to Immunizing Compilers. We show how to use data structure lower bounds to construct immunizing compilers. We illustrate such a compiler here assuming the hardness of the 3SUM-Indexing problem. The compiler proceeds in two steps.

1. First, given oracle access to a random function $F : [2N] \rightarrow [2N]$, construct a new (random) function $F' : [N] \rightarrow [N^2]$ by letting $F'(x) = F(0, x) \| F(1, x)$.
2. Second, let $G^F(x, y) = F'(x) + F'(y)$ (where the addition is interpreted, e.g., over the integers).

Assuming the hardness of 3SUM-Indexing for space S and T queries, we show that this construction is one-way against adversaries with S bits of preprocessed advice and T online queries. (As stated before, our result is actually stronger: the function remains uninvertible even if the adversary could run for unbounded time in the online phase, as long as it can make only T queries.) Conjecture 3 of Demaine and Vadhan, for example, tells us that this function is uninvertible as long as $ST = N^{2-\epsilon}$ for any constant $\epsilon > 0$. In other words, assuming (the average-case version of)

the 3SUM-Indexing conjecture of [DV01], this function is as uninvertible as a *random* function with the same domain and range.

This highlights another advantage of the immunization approach: assume that we have several functions (modeled as independent random oracles) F_1, F_2, \dots, F_ℓ all of which are about to be obsolete because of the increase in the adversary’s space resources. Instead of designing ℓ independent new functions F'_1, \dots, F'_ℓ , one could use our *immunizer* G to obtain, in one shot, $F'_i = G^{F_i}$ that are as uninvertible as ℓ new random functions.

A General Connection. In fact, we show a much more general connection between (average-case) data structure lower bounds and immunizing compilers. In more detail, we formalize a data structure problem by a function g that takes as input the data d and a “target” y and outputs a “solution” q . In the case of 3SUM-Indexing, d is the array of n numbers a_1, \dots, a_n , and q is a pair of indices i and j such that $a_i + a_j = y$. We identify a key property of the data structure problem, namely *efficient query generation*. The data structure problem has an efficient query generator if there is a function that, given i and j , makes a few queries to d and outputs y such that $g(d, y) = (i, j)$. In the case of 3SUM-Indexing, this is just the function that looks up a_i and a_j and outputs their sum.

We then show that any (appropriately hard) data structure problem with an efficient query generator gives us a one-way function in the preprocessing model. In fact, in Section 5.3, we show an equivalence between the two problems.

The Necessity of Unproven Assumptions. The one-wayness of our compiled functions rely on an unproven assumption, namely the hardness of the 3SUM-Indexing problem with relatively large space and time (or more generally, the hardness of a data structure problem with an efficient query generator). We show that unconditional constructions are likely hard to come by in that they would result in significant implications in circuit complexity.

In particular, a long-standing open problem in computational complexity is to find a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ which cannot be computed by binary circuits of linear size $O(n)$ and depth $O(\log n)$ [Val77, AB09, Frontier 3]. We show that even a weak one-way function in the random oracle model with preprocessing (for specific settings of parameters) implies a super-linear circuit lower bound. Our proof in Section 5.3.1 employs the approach used in several recent works [DGW19, Vio19, CK19, RR20].

Relation to Immunizing Against Cryptographic Backdoors. Backdoors in cryptographic algorithms pose a grave concern [CMG⁺16, CNE⁺14, Gre13], and a natural question is whether one can modify an entropic but imperfect (unkeyed) function, which a powerful adversary may have tampered with, into a function which is provably hard to invert even to such an adversary. In other words, can we use a “backdoored” random oracle to build secure cryptography? One possible formalization of a backdoor is one where an unbounded offline adversary may arbitrarily preprocess the random oracle into an exponentially large lookup table to which the (polynomial-time) online adversary has oracle access. It is easy to see that this formalization is simply an alternative interpretation of (massive) preprocessing attacks. Thus, our result shows how to construct one-way functions in this model assuming the hardness of a natural average-case variant of 3SUM-Indexing.

On immunizing against backdoors, a series of recent works [DGG⁺15, BFM18, RTYZ18, FJM18] studied backdoored primitives including pseudorandom generators and hash functions. In this setting, the attacker might be given some space-bounded backdoor related to a primitive, which could allow him to break the system more easily. In particular, backdoored hash functions and

random oracles are studied in [BFM18, FJM18]. Both of them observe that immunizing against a backdoor for a single unkeyed hash function might be hard. For this reason, [BFM18] considers the problem of combining two random oracles (with two independent backdoors). Instead, we look at the case of a single random oracle but add a restriction on the size of the advice. [FJM18] considers the setting of keyed functions such as (weak) pseudorandom functions, which are easier to immunize than unkeyed functions of the type we consider in this work.

The BRO model and an alternative immunization strategy. As mentioned just above, the recent work of [BFM18] circumvents the problem of massive preprocessing in a different way, by assuming the existence of at least two independent (backdoored) random oracles. This allows them to use techniques from two-source extraction and communication complexity to come up with an (unconditionally secure) immunization strategy. A feature of their approach is that they can tolerate unbounded preprocessing that is *separately* performed on the two (independent) random oracles.

Domain extension and indifferenciability. Our immunization algorithm is effectively a domain extender for the function (random oracle) F . While it is too much to hope that G^F is indifferenciability from a random oracle [DGHM13], we show that it could still have interesting cryptographic properties such as one-wayness. We leave it as an interesting open question to show that our compiler preserves other cryptographic properties such as pseudorandomness, or alternatively, to come up with other compilers that preserve such properties.

1.3 Other related work

Non-uniform security, leakage, and immunizing backdoors. A range of work on non-uniform security, preprocessing attacks, leakage, and immunizing backdoors can all be seen as addressing the common goal of achieving security against powerful adversaries that attack a cryptographic primitive given access to some “advice” (or “leakage” or “backdoor information”) that was computed in advance during an unbounded preprocessing phase.

On non-uniform security of hash functions, recent works [Unr07, DGK17, CDGS18] studied the *auxiliary-input random-oracle model* in which an attacker can compute arbitrary S bits of leakage before attacking the system and make T additional queries to the random oracle. Although our model is similar in that it allows preprocessed leakage of a random oracle, we differ significantly in two ways: the size of the leakage is larger, and the attacker only has oracle access to the leakage. Specifically, their results and technical tools only apply to the setting where the leakage is smaller than the random oracle truth table, whereas our model deals with larger leakage. Furthermore, the random oracle model with auxiliary input allows the online adversary to access and depend on the leakage arbitrarily while our model only allows a bounded number of oracle queries to the leakage; our model is more realistic for online adversaries with bounded time and which cannot read the entire leakage at query time.

Kleptography. The study of backdoored primitives is also related to — and sometimes falls within the field of — *kleptography*, originally introduced by Young and Yung [YY97, YY96b, YY96a]. A *kleptographic attack* “uses cryptography against cryptography” [YY97], by changing the behavior of a cryptographic system in a fashion undetectable to an honest user with black-box access to the cryptosystem, such that the use of the modified system leaks some secret information (*e.g.*, plaintexts or key material) to the attacker who performed the modification. An example of such an attack might be to modify the key generation algorithm of an encryption scheme such that

an adversary in possession of a “backdoor” can derive the private key from the public key, yet an honest user finds the generated key pairs to be indistinguishable from correctly produced ones.

Data-structure versions of problems in fine-grained complexity. While the standard conjectures about the hardness of CNF-SAT, 3SUM, OV and APSP concern *algorithms*, the OMV conjecture claims a *data structure* lower bound for the Matrix-Vector Multiplication problem. While algorithmic conjectures help to understand *time* complexity of the problems, it is also natural to consider data structure analogues of the fine-grained conjectures in order to understand *space* complexity of the corresponding problems. Recently, Goldstein *et al.* [GKLP17, GLP17] proposed data structure variants of many classical hardness assumptions (including 3SUM and OV). Other data structure variants of the 3SUM problem have also been studied in [DV01, BW09, CL15, CCI⁺19]. In particular, Chan and Lewenstein [CL15] use techniques from additive combinatorics to give efficient data structures for solving 3SUM on *subsets* of the preprocessed sets.

2 Preliminaries

2.1 Notation

When an uppercase letter represents an integer, we use the convention that the associated lowercase letter represents its base-2 logarithm: $N = 2^n, S = 2^s$, etc. $[N]$ denotes the set $\{1, \dots, N\}$ that we identify with $\{0, 1\}^n$. $x||y$ denotes the concatenation of bit strings x and y . PPT stands for probabilistic polynomial time.

We do not consistently distinguish between random variables and their realizations, but when the distinction is necessary or useful for clarity, we denote random variables in boldface.

2.2 kSUM-Indexing

This paper focuses on the variant of 3SUM known as 3SUM-Indexing, formally defined in [GKLP17], which can be thought of as a preprocessing or data structure variant of 3SUM. In fact, all our results extend to the more general kSUM and kSUM-Indexing problems which consider $(k - 1)$ -wise sums instead of pairwise sums. We also generalize the definition of [GKLP17] by allowing the input to be elements of an arbitrary abelian³ group. We use $+$ to denote the group operation.

Definition 3. The problem kSUM-Indexing(G, N), parametrized by an integer $N \geq k - 1$ and an abelian group G , is defined to be solved by a two-part algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows.

- **Preprocessing phase.** \mathcal{A}_1 receives as input a tuple $A = (a_1, \dots, a_N)$ of N elements from G and outputs a data structure D_A of size⁴ at most S . \mathcal{A}_1 is computationally unbounded.
- **Query phase.** Denote by Z the set of $(k - 1)$ -wise sums of elements from A : $Z = \{\sum_{i \in I} a_i : I \subseteq [N] \wedge |I| = k - 1\}$. Given an arbitrary query $b \in Z$, \mathcal{A}_2 makes at most T oracle queries to D_A and must output $I \subseteq [N]$ with $|I| = k - 1$ such that $\sum_{i \in I} a_i = b$.⁵

³This is for convenience and because our applications only involve abelian groups; our results and techniques easily generalize to the non-abelian case.

⁴The model of computation in this paper is the word RAM model where we assume that the word length is $\Theta(\log N)$. Furthermore we assume that words are large enough to contain description of elements of G , *i.e.*, $|G| \leq N^c$ for some $c > 0$. The size of a data structure is the number of words (or cells) it contains.

⁵Without loss of generality, we can assume that D_A contains a copy of A and in this case \mathcal{A}_2 could return the tuple $(a_i)_{i \in I}$ at the cost of $(k - 1)$ additional queries.

We say that \mathcal{A} is an (S, T) algorithm for $\text{kSUM-Indexing}(G, N)$. Furthermore, we say that \mathcal{A} is *non-adaptive* if the T queries made by \mathcal{A}_2 are non-adaptive (*i.e.*, the indices of the queried cells are only a function of b).

Remark 1. An alternative definition would have the query b be an arbitrary element of G (instead of being restricted to Z) and \mathcal{A}_2 return the special symbol \perp when $b \in G \setminus Z$. Any algorithm conforming to Definition 3 — with undefined behavior for $b \in G \setminus Z$ — can be turned into an algorithm for this seemingly more general problem at the cost of $(k-1)$ extra queries: given output $I \subseteq [N]$ on query b , return I if $\sum_{i \in I} a_i = b$ and return \perp otherwise.

Remark 2. The fact that kSUM-Indexing is defined in terms of $(k-1)$ -wise sums of *distinct* elements from G is without loss of generality for integers, but prevents the occurrence of degenerate cases in some groups. For example, consider the case of 3SUM-Indexing for a group G such that all elements are of order 2 (*e.g.*, $(\mathbb{Z}/2\mathbb{Z})^{cn}$) then finding (i_1, i_2) such that $a_{i_1} + a_{i_2} = 0$ has the trivial solution (i, i) for any $i \in [N]$.

Remark 3. In order to preprocess the elements of some group G , we assume an efficient way to enumerate its elements. More specifically, we assume a time- and space-efficient algorithm for evaluating an injective function $\text{Index}: G \rightarrow [N^c]$ for a constant c . For simplicity, we also assume that the word length is at least $c \log N$ so that we can store $\text{Index}(g)$ for every $g \in G$ in a memory cell. For example, for the standard 3SUM-Indexing problem over the integers from 0 to N^c , one can consider the group $G = (\mathbb{Z}/m\mathbb{Z}, +)$ for $m = 2N^c + 1$, and the trivial function $\text{Index}(a + m\mathbb{Z}) = a$ for $0 \leq a < m$. For ease of exposition, we abuse notation and write g instead of $\text{Index}(g)$ for an element of the group $g \in G$. For example, $g \bmod p$ for an integer p will always mean $\text{Index}(g) \bmod p$.

The standard 3SUM-Indexing problem (formally introduced in [GKLP17]) corresponds to the case where $G = (\mathbb{Z}, +)$ and $k = 3$. In fact, it is usually assumed that the integers are upper-bounded by some polynomial in N , which is easily shown to be equivalent to the case where $G = (\mathbb{Z}/N^c\mathbb{Z}, +)$ for some $c > 0$, and is sometimes referred to as *modular 3SUM* when there is no preprocessing.

Another important special case is when $G = ((\mathbb{Z}/2\mathbb{Z})^{cn}, +)$ for some $c > 0$ and $k = 3$. In this case, G can be thought of as the group of binary strings of length cn where the group operation is the bitwise XOR (exclusive or). This problem is usually referred to as 3XOR when there is no preprocessing, and we refer to its preprocessing variant as 3XOR-Indexing . In [JV16], the authors provide some evidence that the hardnesses of 3XOR and 3SUM are related and conjecture that Conjecture 1 generalizes to 3XOR . We similarly conjecture that in the presence of preprocessing, Conjecture 3 generalizes to 3XOR-Indexing .

Following Definition 3, the results and techniques in this paper hold for arbitrary abelian groups and thus provide a unified treatment of the 3SUM-Indexing and 3XOR-Indexing problems. It is an interesting open question for future research to better understand the influence of the group G on the hardness of the problem.

Open Question 1. *For which groups is $k\text{SUM-Indexing}$ significantly easier to solve, and for which groups does Conjecture 3 not hold?*

2.2.1 Average-case hardness

This paper moreover introduces a new *average-case* variant of kSUM-Indexing (Definition 4 below) that, to the authors' knowledge, has not been stated in prior literature. Definition 4 includes an error parameter ε , as for the cryptographic applications it is useful to consider solvers for average-case kSUM-Indexing that only output correct answers with probability $\varepsilon < 1$.

Definition 4. The *average-case* $\text{kSUM-Indexing}(G, N)$ problem, parametrized by an abelian group G and integer $N \geq k - 1$, is defined to be solved by a two-part algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows.

- **Preprocessing phase.** Let A be a tuple of N elements from G drawn uniformly at random and with replacement⁶. $\mathcal{A}_1(A)$ outputs a data structure D_A of size at most S . \mathcal{A}_1 has unbounded computational power.
- **Query phase.** Given a query b drawn uniformly at random in $Z = \{\sum_{i \in I} a_i : I \subseteq [N] \wedge |I| = k - 1\}$, and given up to T oracle queries to D_A , $\mathcal{A}_2(b)$ outputs $I \subseteq [N]$ with $|I| = k - 1$ such that $\sum_{i \in I} a_i = b$.

We say that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an (S, T, ε) solver for kSUM-Indexing if it answers the query correctly with probability ε over the randomness of \mathcal{A} , A , and the random query b . When $\varepsilon = 1$, we leave it implicit and write simply (S, T) .

Remark 4. In the query phase of Definition 4, the query b is chosen uniformly at random in Z and not in G . As observed in Remark 1, this is without loss of generality for $\varepsilon = 1$. When $\varepsilon < 1$, the meaningful way to measure \mathcal{A} 's success probability is as in Definition 4, since otherwise, if Z had negligible density in G , \mathcal{A} could succeed with overwhelming probability by always outputting \perp .

3 Upper bound

We will use the following data structure first suggested by Hellman [Hel80] and then rigorously studied by Fiat and Naor [FN00].

Theorem 5 ([FN00]). *For any function $F: \mathcal{X} \rightarrow \mathcal{X}$, and for any choice of values S and T such that $S^3 T \geq |\mathcal{X}|^3$, there is a deterministic data structure with space $\tilde{O}(S)$ which allows inverting F at every point making $\tilde{O}(T)$ queries to the memory cells and evaluations of F .*⁷

We demonstrate the idea of our upper bound for the case of 3SUM . Since we are only interested in the pairwise sums of the N input elements $a_1, \dots, a_N \in G$, we can hash down their sums to a set of size $O(N^2)$. Now we define the function $f(i, j) = a_i + a_j$ for $i, j \in [N]$, and note that its domain and range are both of size $O(N^2)$. We apply the generic inversion algorithm of Fiat and Naor to f with $|\mathcal{X}| = O(N^2)$, and obtain a data structure for 3SUM-Indexing .

First, in Lemma 6 we give an efficient data structure for the “modular” version of $\text{kSUM-Indexing}(G, N)$ where the input is an integer $p = \tilde{O}(N^{k-1})$ and N group elements $a_1, \dots, a_N \in G$. Given query $b \in G$ the goal is to find $(i_1, \dots, i_{k-1}) \in [N]^{k-1}$ such that $a_{i_1} + \dots + a_{i_{k-1}} \equiv b \pmod{p}$.⁸ Then, in Theorem 7 we reduce the general case of $\text{kSUM-Indexing}(G, N)$ to the modular case.

Lemma 6. *For every integer $k \geq 3$, real $0 \leq \delta \leq k - 2$, and every integer $p = \tilde{O}(N^{k-1})$, there is an adaptive data structure which uses space $S = \tilde{O}(N^{k-1-\delta})$ and query time $T = \tilde{O}(N^{3\delta})$ and solves modular $k\text{SUM-Indexing}(G, N)$: for input $a_1, \dots, a_N \in G$ and a query $b \in G$, it outputs $(i_1, \dots, i_{k-1}) \in [N]^{k-1}$ such that $a_{i_1} + \dots + a_{i_{k-1}} \equiv b \pmod{p}$, if such a tuple exists.*

⁶We remark that for the classical version of kSUM , the uniform random distribution of the inputs is believed to be the hardest (see, e.g., [Pet15]).

⁷While the result in Theorem 1.1 in [FN00] is stated for a randomized preprocessing procedure, we remark that a less efficient *deterministic* procedure which brute forces the probability space can be used instead.

⁸Recall from Remark 3 that this notation actually means $\text{Index}(a_{i_1} + \dots + a_{i_{k-1}}) \equiv \text{Index}(b) \pmod{p}$.

Proof. Let the N input elements be $a_1, \dots, a_N \in G$. The data structure stores all a_i (this takes only N memory cells) along with the information needed to efficiently invert the function $f: [N]^{k-1} \rightarrow G$ defined below. For $(i_1, \dots, i_{k-1}) \in [N]^{k-1}$, let

$$f(i_1, \dots, i_{k-1}) = a_{i_1} + \dots + a_{i_{k-1}} \bmod p.$$

Note that:

1. f is easy to compute. Indeed, given the input, one can compute f by looking at only $k-1$ input elements.
2. The domain of f is of size N^{k-1} , and the range of f is of size $p = \tilde{O}(N^{k-1})$.
3. Inverting f at a point $b \in G$ finds a tuple $(i_1, \dots, i_{k-1}) \in [N]^{k-1}$ such that $a_{i_1} + \dots + a_{i_{k-1}} \equiv b \bmod p$, which essentially solves the modular **kSUM-Indexing**(G, N) problem.

Now we use the data structure from Theorem 5 with $|\mathcal{X}| = \tilde{O}(N^{k-1})$ to invert f . This gives us a data structure with space $\tilde{O}(S + N) = \tilde{O}(S)$ and query time $\tilde{O}(T)$ for every $S^3 T \geq |\mathcal{X}|^3 = \tilde{O}(N^{3(k-1)})$, which finishes the proof. \square

It remains to show that the input of **kSUM-Indexing** can always be hashed to a set of integers $[p]$ for some $p = \tilde{O}(N^{k-1})$. While many standard hashing functions will work here, we remark that it is important for our application that the hash function of choice has a time- and space-efficient implementation (for example, the data structure in [FN00] requires non-trivial implementations of hash functions). Below, we present a simple hashing procedure which suffices for **kSUM-Indexing**; a more general reduction can be found in Lemma 17 in [CK19].

Theorem 7. *For every integer $k \geq 3$ and real $0 \leq \delta \leq k-2$, there is an adaptive data structure for **kSUM-Indexing**(G, N) with space $S = \tilde{O}(N^{k-1-\delta})$ and query time $T = \tilde{O}(N^{3\delta})$.*

In particular, by taking $k = 3$ and $\delta = 0.1$, we get a data structure which solves **3SUM-Indexing** in space $S = \tilde{O}(N^{1.9})$ and query time $T = \tilde{O}(N^{0.3})$, and, thus, refutes Conjecture 4.

Proof. Let the N inputs be $a_1, \dots, a_N \in G$. Let $Z \subseteq [N^c]$, $|Z| < \binom{N}{k-1}$ be the set of $(k-1)$ -wise sums of the inputs: $Z = \{a_{i_1} + \dots + a_{i_{k-1}} : 1 \leq i_1 < \dots < i_{k-1} \leq N\}$.

Let $I = \{N^{k-1}, \dots, 3kN^{k-1} \log N\}$ be an interval of integers. By the prime number theorem, for large enough N , I contains at least $2cN^{k-1}$ primes. Let us pick $n = \log N$ random primes p_1, \dots, p_n from I . For two distinct numbers $z_1, z_2 \in Z$, we say that they have a collision modulo p if $z_1 \equiv z_2 \bmod p$.

Let $g \in G$ be a positive query of **kSUM-Indexing**(G, N), that is, $b = \text{Index}(g) \in Z$. First, we show that with high probability (over the choices of n random primes) there exists an $i \in [n]$ such that for every $z \in Z \setminus \{b\}$, $z \not\equiv b \bmod p_i$. Indeed, for every $z \in Z \setminus \{b\}$, we have that $(z - b)$ has at most $\log_{N^{k-1}}(N^c) = c/(k-1)$ prime factors from I . Since $|Z| < \binom{N}{k-1}$, at most $c \binom{N}{k-1} / (k-1)$ primes from I divide $(z - b)$ for some $z \in Z$. Therefore, a random prime from I gives a collision between b and some $z \in Z \setminus \{b\}$ with probability at most

$$\frac{c \binom{N}{k-1}}{k-1} \cdot \frac{1}{2cN^{k-1}} \leq \frac{cN^{k-1}}{(k-1)(k-1)!} \cdot \frac{1}{2cN^{k-1}} = \frac{1}{2(k-1)(k-1)!} \leq \frac{1}{2^k}.$$

Now we have that for every $b \in Z$, the probability that there exists an $i \in [n]$ such that b does not collide with any $z \in Z \setminus \{b\}$ modulo p_i , is at least $1 - (2^{-k})^n = 1 - N^{-k}$. Therefore, with probability at least $1 - 1/N$, a random set of n primes has the following property: for every $b \in Z$

there exists an $i \in [n]$ such that b does not collide with any $z \in Z \setminus \{b\}$ modulo p_i . Since such a set of n primes exists, the preprocessing stage of the data structure can find it deterministically.

Now we construct $n = \log N$ modular **kSUM-Indexing**(G, N) data structures (one for each p_i), and separately solve the problem for each of the n primes. This results in a data structure as guaranteed by Lemma 6 with a $\log N$ overhead in space and time. The data structure also stores the inputs a_1, \dots, a_N . Once it sees a solution modulo p_i , it checks whether it corresponds to a solution to the original problem. Now correctness follows from two observations. Since the data structure checks whether a solution modulo p_i gives a solution to the original problem, the data structure never reports false positives. Second, the above observation that for every $b \in Z$ there is a prime p_i such that b does not collide with other $z \in Z$, a solution modulo p_i will correspond to a solution of the original problem (thus, no false negatives can be reported either). \square

Remark 5. A few extensions of Theorem 7 are in order.

1. The result of Fiat and Naor [FN00] also gives an efficient *randomized* data structure. Namely, there is a randomized data structure with preprocessing running time $\tilde{O}(|\mathcal{X}|)$, which allows inverting F at *every* point with probability at least $1 - 1/|\mathcal{X}|$ over the randomness of the preprocessing stage. Thus, the preprocessing phase of the randomized version of Theorem 5 runs in quasilinear time $\tilde{O}(|\mathcal{X}|) = \tilde{O}(N^{k-1})$ (since sampling $n = \log N$ random primes from a given interval can also be done in randomized time $\tilde{O}(1)$). This, in particular, implies that the preprocessing time of the presented data structure for **3SUM-Indexing** is optimal under the **3SUM** Conjecture (Conjecture 1). Indeed, if for $k = 3$, the preprocessing time was improved to $N^{2-\varepsilon}$, then one could solve **3SUM** by querying the N input numbers in (randomized or expected) time $N^{2-\varepsilon}$.
2. For the case of random inputs (for example, for inputs sampled as in Definition 4), one can achieve a better time-space trade-off. Namely, if the inputs a_1, \dots, a_N are uniformly random numbers from a range of size at least $\Omega(N^{k-1})$, then for every $0 \leq \delta \leq k - 2$ there is a data structure with space $S = \tilde{O}(N^{k-1-\delta})$ and query time $T = \tilde{O}(N^{2\delta})$ (with high probability over the randomness of the input instances). This is an immediate generalization of Theorem 7 equipped with the analogue of Theorem 5 for a function [FN00] with low collision probability, which achieves the trade-off of $S^2T = |\mathcal{X}|^2$.
3. For polynomially small $\varepsilon = 1/|\mathcal{X}|^\alpha$ (for constant α), the trade-off between S and T can be further improved for the ε -approximate solution of **kSUM-Indexing**, using approximate function inversion by De *et al.* [DTT10].

We have shown how to refute the strong **3SUM-Indexing** conjecture of [GKLP17] using techniques from space-time tradeoffs for function inversion [Hel80, FN00], specifically the general function inversion algorithm of Fiat and Naor [FN00]. A natural open question is whether a more specific function inversion algorithm could be designed.

Open Question 2. *Can the space-time trade-off achieved in Theorem 7 be improved by exploiting the specific structure of the **3SUM-Indexing** problem?*

4 Lower bound

We now present our lower bound: we prove a space-time trade-off of $S = \tilde{\Omega}(N^{1+1/T})$ for any non-adaptive (S, T) algorithm. While it is weaker than Conjecture 3, any improvement on this result would break a long-standing barrier in static data structure lower bounds: no bounds better than

$T \geq \Omega(\frac{\log N}{\log(S/N)})$ are known, even for the non-adaptive cell-probe and linear models [Sie04, Păt11, PTW10, Lar12, DGW19].

Our main lower bound (Theorem 9) is proven with respect to a slight variant on Definition 3 to which our proof techniques more readily lend themselves, defined next.

Definition 8. The *element version of the k SUM-Indexing(G, N) problem* is exactly like the k SUM-Indexing(G, N) problem (Definition 8), except that a solution is a set $\{a_i\}_{i \in I}$ of group elements summing to a given input instead of simply a set I of indices.

Remark 6. It follows from the observation in Footnote 5 that a lower bound for (S, T) algorithms for the element version of the k SUM-Indexing(G, N) problem implies a lower bound for $(S, T - k + 1)$ algorithms for the k SUM-Indexing(G, N) problem as defined in Definition 8.

Theorem 9. Let $k \geq 3$ and N be integers, and let G be an abelian group with $|G| \geq N^{k-1}$. Then any non-adaptive (S, T) algorithm for the element version of k SUM-Indexing(G, N) satisfies $S = \tilde{\Omega}(N^{1+1/T})$.

Our proof relies on a compressibility argument similar to [GT00, DTT10], also known as cell-sampling in the data structure literature [PTW10]. Roughly speaking, we show that given an (S, T) algorithm $(\mathcal{A}_1, \mathcal{A}_2)$, we can recover a subset of the input A by storing a randomly sampled subset V of the preprocessed data structure D_A and simulating \mathcal{A}_2 on all possible queries: the simulation succeeds whenever the queries made by \mathcal{A}_2 fall inside V . Thus, by storing V along with the remaining part of the input, we obtain an encoding of the entire input. This implies that the length of the encoding must be at least the entropy of a randomly chosen input.

Proof. Consider an (S, T) algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ for the element version of k SUM-Indexing(G, N). For conciseness, in the rest of this proof, we write simply k SUM-Indexing(G, N) to denote the element version of k SUM-Indexing(G, N).

We use \mathcal{A} to design encoding and decoding procedures for inputs of k SUM-Indexing(G, N): we first sample a subset V of the data structure cells which allows us to answer many queries, then we argue that we can recover a constant fraction of the input from this set, which yields a succinct encoding of the input.

Sampling a subset V of cells. For a query $b \in G$, $\text{Query}(b) \subseteq [S]$ denotes the set of probes made by \mathcal{A}_2 on input b (with $|\text{Query}(b)| \leq T$, since \mathcal{A}_2 makes at most T probes to the data structure). Given a subset $V \subseteq [S]$ of cells, we denote by G_V the set of queries in G which can be answered by \mathcal{A}_2 by only making probes within V : $G_V = \{b \in G : \text{Query}(b) \subseteq V\}$. Observe that for a uniformly random set V of size v :

$$\mathbb{E}[|G_V|] = \sum_{b \in G} \Pr[\text{Query}(b) \subseteq V] \geq |G| \frac{\binom{S-T}{v-T}}{\binom{S}{v}} = |G| \prod_{i=0}^{T-1} \frac{v-i}{S-i} \geq |G| \left(\frac{v-T}{S-T} \right)^T,$$

where the last inequality uses that $a/b \geq (a-1)/(b-1)$ for $a \leq b$. Hence, there exists a subset V of size v , such that:

$$|G_V| \geq |G| \left(\frac{v-T}{S-T} \right)^T,$$

and we will henceforth consider such a set V . The size v of V will be set later so that $|G_V| \geq |G|/N$.

Using V to recover the input. Consider some input $A = (a_1, \dots, a_N)$ for $\text{kSUM-Indexing}(G, N)$. We say that $i \in [N]$ is *good* if a_i is output by \mathcal{A}_2 given some query in G_V . Since queries in G_V can be answered by only storing the subset of cells of the data structure indexed by V , our decoding procedure will retrieve from these cells all the good elements from A .

For a set of indices $I \subseteq [N]$, let $a_I = \sum_{i \in I} a_i$ be the sum of input elements with indices in I . Also, for a fixed set G_V and $i \in [N]$, let $g(i) \in G_V$ by some element from G_V which can be written as a $(k-1)$ -sum of the inputs including a_i . If there is no such element in G_V , then let $g(i) = \perp$. Formally,

$$g(i) = \min\{g \in G_V : \exists I \subseteq [N] \setminus \{i\}, |I| = k-2 : a_i + a_I = g\}$$

with the convention that if the minimum is taken over an empty set, then $g(i) = \perp$.

Note that $i \in [N]$ is good if:

$$(g(i) \neq \perp) \wedge (\forall J \subseteq [N] \setminus \{i\}, |J| = k-1, a_J \neq g(i)). \quad (1)$$

Indeed, observe that:

1. The first part of the conjunction guarantees that there exists $b \in G_V$ which can be decomposed as $b = a_i + a_I$ for $I \subseteq [N] \setminus \{i\}$.
2. The second part of the conjunction guarantees that every decomposition $b = a_J, |J| = k-1$ contains the elements a_i .

By correctness of \mathcal{A} , \mathcal{A}_2 outputs a decomposition of its input as a sum of $(k-1)$ elements in A if one exists. For i as in (1), every decomposition $b = a_I$ contains the input a_i , and, therefore, $\mathcal{A}_2(a_I) = (a_{i_1}, \dots, a_{i_{k-1}})$, where $i \in \{i_1, \dots, i_{k-1}\}$.

We denote by $N_V \subseteq [N]$ the set of good indices, and compute its expected size when A is chosen at random according to the distribution in Definition 4, *i.e.*, for each $i \in [N]$, a_i is chosen independently and uniformly in G .

$$\mathbb{E}[|N_V|] \geq \sum_{i=1}^N \Pr[g(i) \neq \perp] \Pr[\forall J \subseteq [N] \setminus \{i\}, |J| = k-1, a_J \neq g(i) \mid g(i) \neq \perp] \quad (2)$$

Let $L \subseteq [N] \setminus \{i\}$ be a fixed set of indices of size $|L| = k-3$. Then:

$$\begin{aligned} \Pr[g(i) \neq \perp] &= \Pr[\exists I \subseteq [N] \setminus \{i\}, |I| = k-2 : a_i + a_I \in G_V] \\ &= 1 - \Pr[\forall I \subseteq [N] \setminus \{i\}, |I| = k-2 : a_i + a_I \notin G_V] \\ &= 1 - \Pr[\forall I' \subseteq [N] \setminus \{i\}, |I'| = k-3, \forall i' \in [N] \setminus (I' \cup \{i\}) : a_i + a_{I'} + a_{i'} \notin G_V] \\ &\geq 1 - \Pr[\forall i' \in [N] \setminus (L \cup \{i\}) : a_i + a_L + a_{i'} \notin G_V] \\ &\geq 1 - \left(1 - \frac{|G_V|}{|G|}\right)^{N-(k-2)}, \end{aligned}$$

where the first inequality follows from setting $I' = L$, the second inequality holds because for every $i' \in [N] \setminus (L \cup \{i\})$, $a_{i'}$ needs to be distinct from the $|G_V|$ elements $-a_i - a_L + g$ for $g \in G_V$.

Furthermore:

$$\begin{aligned}
& \Pr[\forall J \subseteq [N] \setminus \{i\}, |J| = k-1, a_J \neq g(i) \mid g(i) \neq \perp] \\
&= 1 - \Pr[\exists J \subseteq [N] \setminus \{i\}, |J| = k-1, a_J = g(i) \mid g(i) \neq \perp] \\
&\geq 1 - \sum_{\substack{J \subseteq [N] \setminus \{i\} \\ |J|=k-1}} \Pr[a_J = g(i) \mid g(i) \neq \perp] \\
&\geq 1 - \binom{N-1}{k-1} \cdot \frac{1}{|G|} \geq \frac{1}{2},
\end{aligned}$$

where the first inequality uses the union bound and the last inequality uses that $|G| \geq N^{k-1}$. Using the previous two derivations in (2), we get:

$$\mathbb{E}[|N_V|] \geq \frac{N}{2} \left(1 - \left(1 - \frac{|G_V|}{|G|} \right)^{N-(k-2)} \right) \geq \frac{N}{4}, \quad (3)$$

where the last inequality uses that $|G_V| \geq |G|/N$ and $(1 - 1/N)^{N-(k-2)} \leq 1/2$ for large enough N .

Encoding and decoding. It follows from (3) and a simple averaging argument that with probability at least $1/16$ over the random choice of A , N_V is of size at least $N/5$. We will henceforth focus on providing encoding and decoding procedures for such inputs A . Specifically, consider the following pair of encoding/decoding algorithms for A :

- **Enc(A):** given input $A = (a_1, \dots, a_N)$.
 1. use \mathcal{A}_2 to compute the set $N_V \subseteq [N]$ of good indices.
 2. store $(\mathcal{A}_1(A)_j)_{j \in V}$ and $(a_i)_{i \notin N_V}$.
- **Dec(Enc(A)):** for each $b \in G$, simulate \mathcal{A}_2 on input b :
 1. If $\text{Query}(b) \subseteq V$, use $(\mathcal{A}_1(A)_i)_{i \in V}$ (which was stored in Enc(A)) to simulate \mathcal{A}_2 and get $\mathcal{A}_2(b)$. By definition of N_V , when b ranges over the queries such that $\text{Query}(b) \subseteq V$, this step recovers $(a_i)_{i \in N_V}$.
 2. Then recover $(a_i)_{i \notin N_V}$ directly from Enc(A).

Note that the bit length of the encoding is:

$$|\text{Enc}(A)| \leq v \cdot w + (N - |N_V|) \log |G| \leq v \cdot w + \frac{4N}{5} \log |G|$$

where w is the word length and where the second inequality holds because we restrict ourselves to inputs A such that $|N_V| \geq N/5$. By a standard incompressibility argument (see for example Fact 8.1 in [DTT10]), since our encoding and decoding succeeds with probability at least $1/16$ over the random choice of A , we need to be able to encode at least $|G|^N/16$ distinct values, hence:

$$v \cdot w + \frac{4N}{5} \log |G| \geq N \log |G| + O(1) \quad (4)$$

Finally, as discussed before, we set v such that $|G_V|/|G| \geq 1/N$. For this, by the computation performed at the beginning of this proof, it is sufficient to have:

$$\left(\frac{v - T}{S - T} \right)^T \geq \frac{1}{N}.$$

Hence, we set $v = T + (S - T)/N^{1/T}$ and since $T \leq N \leq S$ (otherwise the result is trivial), (4) implies:

$$S = \tilde{\Omega}(N^{1+1/T}) \quad \square$$

Remark 7. $\text{kSUM-Indexing}(\mathbb{Z}/N^c\mathbb{Z}, N)$ reduces to kSUM-Indexing over the integers, so our lower bound extends to $\text{kSUM-Indexing}(\mathbb{Z}, N)$, too. Specifically, the reduction works as follows: we choose $\{\bar{0}, \dots, \overline{N^c - 1}\}$ as the set of representatives of $\mathbb{Z}/N^c\mathbb{Z}$. Given some input $A \subseteq \mathbb{Z}/N^c\mathbb{Z}$ for $\text{kSUM-Indexing}(\mathbb{Z}/N^c\mathbb{Z}, N)$, we treat it as a list of integers and build a data structure using our algorithm for $\text{kSUM-Indexing}(\mathbb{Z}, N)$. Now, given a query $\bar{b} \in \mathbb{Z}/N^c\mathbb{Z}$, we again treat it as an integer and query the data structure at $b, b + N^c, \dots, b + (k-2)N^c$. The correctness of the reduction follows from the observation that $\bar{b} = \bar{a}_{i_1} + \dots + \bar{a}_{i_{k-1}}$ if and only if $a_{i_1} + \dots + a_{i_{k-1}} \in \{b, b + N^c, \dots, b + (k-2)N^c\}$.

As we already mentioned, no lower bound better than $T \geq \Omega(\frac{\log N}{\log(S/N)})$ is known even for the non-adaptive cell-probe and linear models, so Theorem 9 matches the best known lower bounds for static data structures. An ambitious goal for future research would naturally be to prove Conjecture 3. A first step in this direction would be to extend Theorem 9 to adaptive strategies that may err with some probability.

Open Question 3. *Must any (possibly adaptive) (S, T, ε) algorithm for $3\text{SUM-Indexing}(G, N)$ require $S = \tilde{\Omega}(\varepsilon N^{1+1/T})$?*

4.1 3SUM-Indexing-hardness

Gajentaan and Overmars introduced the notion of 3SUM-hardness and showed that a large class of problems in computational geometry were 3SUM-hard [GO95]. Informally, a problem is 3SUM-hard if 3SUM reduces to it with $o(N^2)$ computational overhead. These fine-grained reductions have the nice corollary that the 3SUM conjecture immediately implies a $\tilde{\Omega}(N^2)$ lower bound for all 3SUM-hard problems. In this section, we consider a similar paradigm of efficient reductions between data structure problems, leading to the following definition of 3SUM-Indexing-hardness.

Definition 10. A (static) *data structure problem* is defined by a function $g : D \times Q \rightarrow Y$ where D is the set of data (the input to the data structure problem), Q is the set of queries and Y is the set of answers. (See, e.g., [Mil99].)

Definition 11 (3SUM-Indexing-hardness). A data structure problem g is *3SUM-Indexing-hard* if, given a data structure for g using space S and time T on inputs of size N , it is possible to construct a data structure for 3SUM-Indexing using space $O(S)$ and time T on inputs of size N .

As an immediate consequence of Definition 11, we get that all 3SUM-Indexing-hard problems admit the lower bound of Theorem 9: i.e., the same lower bound as 3SUM-Indexing. This is stated concretely in Corollary 12.

Corollary 12. *Let g be a 3SUM-Indexing-hard data structure problem. Any non-adaptive data structure for g using space S and time T on inputs of size N must satisfy $S = \tilde{\Omega}(N^{1+1/T})$.*

We now give two examples⁹ of how to adapt known reductions from 3SUM to 3SUM-hard problems and obtain efficient reductions between the analogous data structure problems. Corollary 12 then implies a lower bound of $S = \tilde{\Omega}(N^{1+1/T})$ for these problems as well, matching the best known lower bound for static data structure problems.

⁹This is far from exhaustive. All the problems from [GO95, BHP01] which inspired the two examples listed here similarly admit efficient data structure reductions from 3SUM-Indexing.

3 points on a line (3POL). Consider the following data-structure variant of the 3POL problem, referred to as 3POL-Indexing. The input is a set $\mathcal{X} = \{x_1, \dots, x_N\}$ of N distinct points in \mathbb{R}^2 . Given a query $q \in \mathbb{R}^2$, the goal is to find $\{i, j\} \subseteq [N]$ such that x_i, x_j , and q are collinear (or report \perp if no such $\{i, j\}$ exists).

The following observation was made in [GO95] and used to reduce 3SUM to 3POL: for distinct reals a, b and c , it holds that $a + b + c = 0$ iff $(a, a^3), (b, b^3), (c, c^3)$ are collinear. We obtain an efficient data-structure reduction from 3SUM-Indexing to 3POL-Indexing by leveraging the same idea, as follows. Given input $A = (a_1, \dots, a_N)$ for 3SUM-Indexing, construct $\mathcal{X} = \{(a_i, a_i^3) : i \in [N]\}$ and use it as input to a data structure for 3POL-Indexing. Then, given a query b for 3SUM-Indexing, construct the query $(-b, -b^3)$ for 3POL-Indexing. Finally, observe that an answer $\{i, j\}$ such that $(-b, -b^3)$ is collinear with $(a_i, a_i^3), (a_j, a_j^3)$ is also a correct answer for 3SUM-Indexing, by the previous observation. The resulting data structure for 3SUM-Indexing uses the same space and time as the original data structure and hence 3POL-Indexing is 3SUM-Indexing-hard.

Polygon containment (PC). The problem and reduction described here are adapted from [BHP01]. Consider the following data-structure variant of the polygon containment problem, denoted by PC-Indexing: the input is a polygon P in \mathbb{R}^2 with N vertices. The query is a polygon Q with $O(1)$ vertices and the goal is to find a translation $t \in \mathbb{R}^2$ such that $Q + t \subseteq P$.

We now give a reduction from 3SUM-Indexing to PC-Indexing. Consider input $A = \{a_1, \dots, a_N\}$ for 3SUM-Indexing and assume without loss of generality that it is sorted: $a_1 < \dots < a_N$. Let $0 < \varepsilon < 1$. We now define the following “comb-like” polygon P : start from the base rectangle defined by opposite corners $(0, 0)$ and $(3\bar{a}, 1)$, where \bar{a} is an upper bound on the elements of the 3SUM input (*i.e.*, $\forall a \in A, a < \bar{a}$).¹⁰ For each $i \in [N]$, add two rectangle “teeth” defined by corners $(a_i, 1), (a_i + \varepsilon, 2)$ and $(3\bar{a} - a_i - \varepsilon, 1), (3\bar{a} - a_i, 2)$ respectively. Note that for each $i \in [N]$ we have one tooth with abscissa in $[0, \bar{a}]$ and one tooth with abscissa in $[2\bar{a}, 3\bar{a}]$, and there are no teeth in the interval $[\bar{a}, 2\bar{a}]$. We then give P as input to a data structure for PC-Indexing.

Consider a query b for 3SUM-Indexing. If $b \geq 2\bar{a}$ we can immediately answer \perp , since a pairwise sum of elements in A is necessarily less than $2\bar{a}$. We henceforth assume that $b < 2\bar{a}$. Define the comb Q with base rectangle defined by corners $(0, 0)$ and $(3\bar{a} - b, 1)$ and with two rectangle teeth defined by corners $(0, 1), (\varepsilon, 2)$ and $(3\bar{a} - b - \varepsilon, 1), (3\bar{a} - b, 2)$ respectively. It is easy to see that there exists a translation t such that $Q + t \subseteq P$ iff it is possible to align the teeth of Q with two teeth of P . Furthermore, the two teeth of Q are at least \bar{a} apart along the x -axis, because $b < 2\bar{a}$ by assumption, which implies $3\bar{a} - b > \bar{a}$. Hence, the leftmost tooth of Q needs to be aligned with a tooth of P with abscissa in $[0, \bar{a}]$, the rightmost tooth of Q needs to be aligned with a tooth of P with abscissa in $[2\bar{a}, 3\bar{a}]$, and the distance between the two teeth needs to be exactly $3\bar{a} - b$. In other words, there exists a translation t such that $Q + t \subseteq P$ iff there exists $\{i, j\} \subseteq [N]$ such that $(3\bar{a} - a_j) - a_i = 3\bar{a} - b$, *i.e.*, $a_i + a_j = b$. The resulting data structure for 3SUM-Indexing uses the same space and time as the data structure for PC-Indexing. This concludes the proof that PC-Indexing is 3SUM-Indexing-hard.

¹⁰Our model assumes such a bound is known; see footnote 4. The reduction can also be adapted to work even if the upper bound is not explicitly known.

5 Cryptography against massive preprocessing attacks

5.1 Background on random oracles and preprocessing

A line of work initiated by Impagliazzo and Rudich [IR89] studies the hardness of a random oracle as a one-way function. In [IR89] it was shown that a random oracle is an exponentially hard one-way function against uniform adversaries. The case of non-uniform adversaries was later studied in [Imp96, Zim98]. Specifically we have the following result.

Proposition 13 ([Zim98]). *With probability at least $1 - \frac{1}{N}$ over the choice of a random oracle $R : \{0, 1\}^n \rightarrow \{0, 1\}^n$, for all oracle circuits C of size at most T :*

$$\Pr_{x \leftarrow \{0, 1\}^n} \left[C^R(R(x)) \in R^{-1}(R(x)) \right] \in \tilde{O} \left(\frac{T^2}{N} \right).$$

In Proposition 13, the choice of the circuit occurs *after* the random draw of the oracle: in other words, the description of the circuit can be seen as a non-uniform advice which depends on the random oracle. Proposition 14 is a slight generalization where the adversary is a uniform Turing machine independent of the random oracle, with oracle access to an advice of length at most S depending on the random oracle. While the two formulations are equivalent in the regime $S \leq T$, one advantage of this reformulation is that S can be larger than the running time T of the adversary.

Proposition 14 (Implicit in [DTT10]). *Let \mathcal{A} be a uniform oracle Turing machine whose number of oracle queries is $T : \{0, 1\}^n \rightarrow \mathbb{N}$. For all $n \in \mathbb{N}$ and $S \in \mathbb{N}$, with probability at least $1 - \frac{1}{N}$ over the choice of a random oracle $R : \{0, 1\}^n \rightarrow \{0, 1\}^n$:*

$$\forall P \in \{0, 1\}^S, \Pr_{x \leftarrow \{0, 1\}^n} \left[\mathcal{A}^{R, P}(R(x)) \in R^{-1}(R(x)) \right] \in O \left(\frac{T(S + n)}{N} \right).$$

In Proposition 14, the advice P can be thought of as the result of a preprocessing phase involving the random oracle. Also, no assumption is made on the computational power of the preprocessing adversary but it is simply assumed that the length of the advice is bounded.

Remark 8. Propositions 13 and 14 assume a deterministic adversary. For the regime of $S > T$ (which is the focus of this work), this assumption is without loss of generality since a standard averaging argument shows that for a randomized adversary, there exists a choice of “good” randomness for which the adversary achieves at least its expected success probability. This choice of randomness can be hard-coded in the non-uniform advice, yielding a deterministic adversary.

Note, however, that Proposition 14 provides no guarantee when $S \geq N$. In fact, in this case, defining P to be any inverse mapping R^{-1} of R allows an adversary to invert R with probability one by making a single query to P . So, R itself can no longer be used as a one-way function when $S \geq N$ — but one can still hope to use R to define a new function f^R that is one-way against an adversary with advice of size $S \geq N$. This idea motivates the following definition.

Definition 15. Let $R : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a random oracle. A *one-way function in the random oracle model with S preprocessing* is an efficiently computable oracle function $f^R : \{0, 1\}^{n'} \rightarrow \{0, 1\}^{m'}$ such that for any two-part adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ satisfying $|\mathcal{A}_1(\cdot)| \leq S$ and where \mathcal{A}_2 is PPT, the following probability is negligible in n :¹¹

$$\Pr_{R, x \leftarrow \{0, 1\}^n} \left[f^R \left(\mathcal{A}_2^{R, \mathcal{A}_1(R)}(f^R(x)) \right) = f^R(x) \right]. \quad (5)$$

¹¹A negligible function is one that is in $o(n^{-c})$ for all constants c .

We say that f is an (S, T, ε) -one-way function if the probability in (5) is less than ε and \mathcal{A}_2 makes at most T random oracle queries.

The adversary model in Definition 15 is very similar to the 1-BRO model of [BFM18], differing only in having a restriction on the output size of \mathcal{A}_1 . As was noted in [BFM18], without this restriction (and in fact, as soon as $S \geq 2^{n'}$ by the same argument as above), no function f^R can achieve the property given in Definition 15. [BFM18] bypasses this impossibility by considering the restricted case of two independent oracles with two independent preprocessed advices (of unrestricted sizes). Our work bypasses it in a different and incomparable way, by considering the case of a single random oracle with bounded advice.

5.2 Constructing one-way functions from kSUM-Indexing

Our main candidate construction of a OWF (Construction 17) relies on the hardness of average-case kSUM-Indexing. First, we define what hardness means, then give the constructions and proofs.

Definition 16. Average-case kSUM-Indexing is $(G, N, S, T, \varepsilon)$ -hard if the success probability¹² of any (S, T) algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in answering average-case kSUM-Indexing (G, N) queries is at most ε .

Construction 17. For $N \in \mathbb{N}$, let $(G, +)$ be an abelian group and let $R : [N] \rightarrow G$ be a random oracle. Our candidate OWF construction has two components:

- the function $f^R : [N]^{k-1} \rightarrow G$ defined by $f^R(x) = \sum_{i=1}^{k-1} R(x_i)$ for $x \in [N]^{k-1}$; and
- the input distribution, uniform over $\{x \in [N]^{k-1} : x_1 \neq \dots \neq x_{k-1}\}$.

Remark 9 (Approximate sampling). We depart from the standard definition of a OWF by using a nonuniform input distribution in our candidate construction. This makes it easier to relate its security to the hardness of kSUM-Indexing. As long as the input distribution is efficiently samplable, a standard construction can be used to transform any OWF with nonuniform input into a OWF which operates on uniformly random bit strings. Specifically, one simply defines a new OWF equal to the composition of the sampling algorithm and the original OWF, (see [Gol01, Section 2.4.2]).

In our case, since $N!/(N-k+1)!$ is not guaranteed to be a power of 2, the input distribution in Construction 17 cannot be sampled exactly in time polynomial in $\log N$. However, using rejection sampling, it is easy to construct a sampler taking as input $O([\log N]^2)$ random bits and whose output distribution is $1/N$ -close in statistical distance to the input distribution. It is easy to propagate this exponentially¹³ small sampling error without affecting the conclusion of Theorem 18 below. A similar approximate sampling occurs when considering OWFs based on the hardness of number theoretic problems, which require sampling integers uniformly in a range whose length is not necessarily a power of two.

Remark 10. Similarly, the random oracle R used in the construction is not a random oracle in the traditional sense since its domain and co-domain are not bit strings. If $|G|$ and N are powers of two, then R can be implemented exactly by a standard random oracle $\{0, 1\}^{\log N} \rightarrow \{0, 1\}^{\log |G|}$. If not, using a random oracle $\{0, 1\}^{\text{poly}(\lceil \log |G| \rceil)} \rightarrow \{0, 1\}^{\text{poly}(\lceil \log |G| \rceil)}$, and rejection sampling, it is possible to implement an oracle R' which is $1/N$ close to R in statistical distance. We can similarly propagate this $1/N$ sampling error without affecting the conclusion of Theorem 18.

¹²Over the randomness of \mathcal{A} , A , and the average-case kSUM-Indexing query. (Recall: A is kSUM-Indexing's input.)

¹³Recall that $N = 2^n$ and that following Definition 15, n is the security parameter. Terms like “exponential” or “negligible” are thus defined with respect to n .

Theorem 18. Consider a sequence of abelian groups $(G_N)_{N \geq 1}$ such that $|G_N| \geq N^{k-1+c}$ for some $c > 0$ and all $N \geq k-1$, and a function $S : \mathbb{N} \rightarrow \mathbb{R}$. Assume that for all polynomial T there exists a negligible function ε such that average-case **kSUM-Indexing** is $(G_N, N, S(n), T(n), \varepsilon(n))$ -hard for all $N \geq 1$ (recall that $n = \log N$). Then the function f defined in Construction 17 is a one-way function in the random oracle model with S preprocessing.

The function f^R in Construction 17 is designed precisely so that inverting f^R on input x is equivalent to solving **kSUM-Indexing** for the input $A = (R(1), \dots, R(N))$ and query $\sum_{i=1}^{k-1} a_{x_i}$. However, observe that the success probability of a OWF inverter is defined for a random input distributed as $\sum_{i \in I} a_i$ where $I \subseteq [N]$ is a uniformly random set of indices of size $k-1$. In contrast, in average-case **kSUM-Indexing**, the query distribution is uniform over $\{\sum_{i \in I} a_i : I \subseteq [N], |I| = k-1\}$. These two distributions are not identical whenever there is a collision: two sets I and I' such that $\sum_{i \in I} a_i = \sum_{i \in I'} a_i$. The following two lemmas show that whenever $|G| \geq N^{k-1+c}$ for some $c > 0$, there are few enough collisions that the two distributions are negligibly close in statistical distance, which is sufficient to prove Theorem 18.

Lemma 19. Let $N \geq k-1$ be an integer and let G be an abelian group with $|G| \geq N^{k-1+c}$ for some $c > 0$. Let $\mathbf{A} = (\mathbf{a}_1, \dots, \mathbf{a}_N)$ be a tuple of N elements drawn with replacement from G . Define the following two random variables:

- $\mathbf{X}_1 = \sum_{i \in \mathbf{I}} \mathbf{a}_i$ where $\mathbf{I} \subseteq [N]$ is a uniformly random set of size $k-1$.
- \mathbf{X}_2 : uniformly random over $\{\sum_{i \in I} \mathbf{a}_i : I \subseteq [N], |I| = k-1\}$.

Then the statistical distance is $\|(\mathbf{A}, \mathbf{X}_1) - (\mathbf{A}, \mathbf{X}_2)\|_s = O(1/\sqrt{N^c})$.

Proof. First, by conditioning on the realization of \mathbf{A} :

$$\|(\mathbf{A}, \mathbf{X}_1) - (\mathbf{A}, \mathbf{X}_2)\|_s = \sum_{A \in G^N} \Pr[\mathbf{A} = A] \|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s, \quad (6)$$

where $\mathbf{X}_{i|A}$ denotes the distribution of \mathbf{X}_i conditioned on the event $\mathbf{A} = A$ for $i \in \{1, 2\}$.

We now focus on a single summand from (6) corresponding to the realization $\mathbf{A} = A$ and define $Z = \{\sum_{i \in I} a_i : I \subseteq [N], |I| = k-1\}$, the set of $(k-1)$ -sums and for $g \in G$, $c_g = |\{I \subseteq [N] : |I| = k-1 \wedge \sum_{i \in I} a_i = g\}|$ is the number of $(k-1)$ -sets of indices whose corresponding sum equals g . Then we have:

$$\|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s = \frac{1}{2} \sum_{g \in Z} \left| \frac{1}{|Z|} - \frac{c_g}{\binom{N}{k-1}} \right|.$$

Observe that $c_g \geq 1$ whenever $g \in Z$. We now assume that $|Z| \geq \frac{1}{2} \binom{N}{k-1}$ (we will later only use the following derivation under this assumption). Splitting the sum on $c_g > 1$:

$$\|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s = \frac{1}{2} \sum_{g: c_g=1} \left(\frac{1}{|Z|} - \frac{1}{\binom{N}{k-1}} \right) + \frac{1}{2} \sum_{g: c_g>1} \left(\frac{c_g}{\binom{N}{k-1}} - \frac{1}{|Z|} \right),$$

where we used the trivial upper bound $|Z| \leq \binom{N}{k-1}$ and the assumption that $|Z| \geq \frac{1}{2} \binom{N}{k-1}$ to

determine the sign of the quantity inside the absolute value. We then write:

$$\begin{aligned}
\|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s &= \frac{1}{2} \sum_{g: c_g=1} \left(\frac{1}{|Z|} - \frac{1}{\binom{N}{k-1}} \right) + \frac{1}{2} \sum_{g: c_g>1} \left(\frac{c_g-1}{\binom{N}{k-1}} + \frac{1}{\binom{N}{k-1}} - \frac{1}{|Z|} \right) \\
&\leq \frac{1}{2} \sum_{g: c_g \geq 1} \left(\frac{1}{|Z|} - \frac{1}{\binom{N}{k-1}} \right) + \frac{1}{2} \sum_{g: c_g>1} \frac{c_g-1}{\binom{N}{k-1}} \\
&= \frac{1}{2} \sum_{g: c_g \geq 1} \left(\frac{1}{|Z|} - \frac{1}{\binom{N}{k-1}} \right) + \frac{1}{2} \sum_{g: c_g \geq 1} \frac{c_g-1}{\binom{N}{k-1}} = \left(1 - \frac{|Z|}{\binom{N}{k-1}} \right),
\end{aligned}$$

where the inequality uses again that $|Z| \leq \binom{N}{k-1}$, and the last equality uses that $\sum_{g: c_g \geq 1} c_g = \binom{N}{k-1}$ and that $Z = \{g : c_g \geq 1\}$.

We now consider some $\delta \leq 1/2$ which will be set at the end of the proof and split the sum in (6) on $|Z| \leq (1-\delta)\binom{N}{k-1}$:

$$\begin{aligned}
\|(\mathbf{A}, \mathbf{X}_1) - (\mathbf{A}, \mathbf{X}_2)\|_s &\leq \Pr \left[|Z| \leq \binom{N}{k-1} (1-\delta) \right] + \delta \cdot \Pr \left[|Z| > \binom{N}{k-1} (1-\delta) \right] \\
&\leq \Pr \left[|Z| \leq \binom{N}{k-1} (1-\delta) \right] + \delta,
\end{aligned}$$

where we used the trivial upper bound $\|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s \leq 1$ when $|Z| \leq (1-\delta)\binom{N}{k-1}$ and the upper bound $\|\mathbf{X}_{1|A} - \mathbf{X}_{2|A}\|_s < \delta$ when $|Z| > (1-\delta)\binom{N}{k-1}$ by the previous derivation.

We now use Markov's inequality and Lemma 20 below to upper bound the first summand:

$$\begin{aligned}
\|(\mathbf{A}, \mathbf{X}_1) - (\mathbf{A}, \mathbf{X}_2)\|_s &\leq \frac{1}{\delta \binom{N}{k-1}} \left(\binom{N}{k-1} - \mathbb{E}[|Z|] \right) + \delta \\
&\leq \frac{1}{\delta |G|} \binom{N}{k-1} + \delta \leq \frac{1}{\delta (k-1)! N^c} + \delta.
\end{aligned}$$

where the last inequality uses that $|G| \geq N^{k-1+c}$ by assumption. Finally, we set $\delta = 1/\sqrt{N^c}$ to get the desired conclusion. \square

Lemma 20. *Let $N \geq k-1$ be an integer and let G be an abelian group of size at least N . Let $\mathbf{A} = (\mathbf{a}_1, \dots, \mathbf{a}_N)$ be a tuple of N elements drawn with replacement from G . Define $Z = \{ \sum_{i \in I} \mathbf{a}_i : I \subseteq [N] \wedge |I| = k-1 \}$ to be the set of $(k-1)$ -sums of coordinates of \mathbf{A} , then:*

$$\binom{N}{k-1} - \mathbb{E}[|Z|] \leq \frac{1}{|G|} \binom{N}{k-1}^2.$$

Proof. For each $(k-1)$ -set of indices $I \subseteq [N]$, we define the random variable X_I to be the indicator that the sum $\sum_{i \in I} \mathbf{a}_i$ collides with $\sum_{i \in I'} \mathbf{a}_i$ for some $(k-1)$ -set of indices $I' \neq I$:

$$X_I = \mathbf{1} \left\{ \exists I' \subseteq [N] : |I'| = k-1 \wedge I' \neq I \wedge \sum_{i \in I} \mathbf{a}_i = \sum_{i \in I'} \mathbf{a}_i \right\}.$$

Then, using a union bound and since the probability of a collision is $1/|G|$:

$$\mathbb{E}[X_I] \leq \sum_{I' \neq I} \Pr \left[\sum_{i \in I} \mathbf{a}_i = \sum_{i \in I'} \mathbf{a}_i \right] \leq \frac{\binom{N}{k-1}}{|G|}.$$

On the other hand, there are at least as many elements in Z as $(k-1)$ -sets of indices $I \subseteq [N]$ which do not collide with any other $(k-1)$ -set:

$$|Z| \geq \sum_{\substack{I \subseteq [N] \\ |I|=k-1}} (1 - X_I) = \binom{N}{k-1} - \sum_{\substack{I \subseteq [N] \\ |I|=k-1}} X_I.$$

Combining the previous two inequalities concludes the proof. \square

We are now ready to prove Theorem 18.

Proof (Theorem 18). Throughout the proof, we fix N and write G, S, T to denote $G_N, S(n), T(n)$ respectively, leaving the parameter n implicit. Suppose, for contradiction, that f is not a one-way function in the random oracle model with S preprocessing. Then there exists $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $|\mathcal{A}_1(\cdot)| \leq S$ and \mathcal{A}_2 is PPT, which inverts f with probability at least δ for some non-negligible δ :

$$\Pr_{R, \mathbf{x}} \left[f^R \left(\mathcal{A}_2^{R, \mathcal{A}_1(R)} (f^R(\mathbf{x})) \right) = f^R(\mathbf{x}) \right] \geq \delta. \quad (7)$$

where $R : [N] \rightarrow G$ is a random oracle and $\mathbf{x} \in [N]^{k-1}$ is a random input to f^R distributed as defined in Construction 17. Then, we use \mathcal{A} to build an (S, T) solver $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ for $\text{kSUM-Indexing}(G, N)$ as follows. Given input $A = (a_1, \dots, a_N)$ for $\text{kSUM-Indexing}(G, N)$, \mathcal{A}'_1 defines random oracle $R : [N] \rightarrow G$ such that $R(i) = a_i$ for $i \in [N]$ and outputs $\mathcal{A}_1(R)$ — this amounts to interpreting the tuple A as a function mapping indices to coordinates. \mathcal{A}'_2 is identical to \mathcal{A}_2 . By construction, whenever \mathcal{A}_2 successfully inverts f^R (i.e., outputs $x \in [N]^{k-1}$ such that $f^R(x) = b$ for input b), then the output of \mathcal{A}'_2 satisfies $\sum_{i=1}^{k-1} a_{x_i} = b$.

It follows from (7) that \mathcal{A}' as described thus far solves average-case $\text{kSUM-Indexing}(G, N)$ with success probability δ when given as input a query distributed as $f^R(\mathbf{x})$. By construction, the distribution of $f^R(\mathbf{x})$ is identical to the distribution of $\sum_{i \in I} a_i$ for a uniformly random set $I \subseteq [N]$ of size $k-1$, let X_1 denote this distribution. However, average-case $\text{kSUM-Indexing}(G, N)$ is defined with respect to a distribution of queries which is uniform over $\{\sum_{i \in I} a_i : I \subseteq [N] \wedge |I| = k-1\}$, let us denote this distribution by X_2 . By Lemma 19, we have that $\|(A, X_1) - (A, X_2)\|_s = O(1/\sqrt{N^c})$, hence \mathcal{A}_2 solves $\text{kSUM-Indexing}(G, N)$ for the correct query distribution X_2 with probability at least $\delta - O(1/\sqrt{N^c})$ which is non-negligible since δ is non-negligible. Denoting by T the running time of \mathcal{A}_2 , we just proved that \mathcal{A}' is an $(S, T, \delta - O(1/\sqrt{N^c}))$ adversary for average-case $\text{kSUM-Indexing}(G, N)$, which is a contradiction. \square

We conjecture that 3SUM-Indexing is $(G, N, S, T, \varepsilon)$ -hard with $\varepsilon = \frac{ST}{N^2}$ when $G = (\mathbb{Z}/N^c\mathbb{Z}, +)$ (the standard 3SUM-Indexing problem) and $G = ((\mathbb{Z}/2\mathbb{Z})^{cn}, \oplus)$ (the 3XOR-Indexing problem) for $c > 2$. If this conjecture is true, the previous theorem implies the existence of (exponentially strong) one-way functions in the random oracle model as long the preprocessing satisfies $S \leq N^{2-\delta}$ for $\delta > 0$. As per the discussion below Definition 15, Theorem 18 is vacuous in the regime where $S = \tilde{\Omega}(N^2)$.

5.3 Cryptography with preprocessing and data structures

In this section we show that the construction in Section 5.2 is a specific case of a more general phenomenon. Specifically, Theorem 23 below states that the existence of one-way functions in the random oracle model with preprocessing is equivalent to the existence of a certain class of hard-on-average data structure problems. The next two definitions formalize the definitions of a data structure problem and a solver for a data structure problem.

Definition 21. An (S, T, ε) -solver for a data structure problem $g : D \times Q \rightarrow Y$ is a two-part algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that:

- \mathcal{B}_1 takes as input $d \in D$ and computes a data structure $\phi(d)$ such that $|\phi(d)| \leq S$; and
- \mathcal{B}_2 takes as input query $q \in Q$, makes at most T queries to $\phi(d)$, and outputs $y \in Y$.

We say that a given execution of \mathcal{B} *succeeds* if \mathcal{B}_2 outputs $y = g(d, q)$.

Theorem 23 considers a special class of data structure problems for which a query can be efficiently generated given its answer, as defined next.

Definition 22. Let $g : D \times Q \rightarrow Y$ be a static data structure problem and let $h : D \times Y \rightarrow Q$. Then h is an *efficient query generator* for g if h is computable in time $\text{poly}(\log |Q|, \log |Y|)$ and

$$\forall d \in D, y \in Y, g(d, h(d, y)) = y. \quad (8)$$

For any h which is an efficient query generator for g , we say that (g, h) is (S, T, ε) -hard if for query distribution $q = h(d, y)$ where $d \in D, y \in Y$ are uniformly random, no (S, T) -solver succeeds with probability more than ε .¹⁴

Remark 11. For the 3SUM-Indexing problem, h is the function that takes $d = (a_1, \dots, a_n)$ and a pair of indices $y = (i, j)$ and outputs $a_i + a_j$. Constructing a corresponding function g for this h is equivalent to solving the 3SUM-Indexing problem.

Remark 12. Let g, h be defined as in Definition 22. Then because g is a function and h satisfies (8), it holds that for any given $d \in D$, the function $h(d, \cdot)$ is injective. That is, for any $d \in D, y, y' \in Y$,

$$h(d, y) = h(d, y') \Rightarrow y = y'. \quad (9)$$

Theorem 23. *There exists a (S, T, ε) -hard data structure with efficient query generation iff there exists a (S, T, ε) -hard OWF in the random oracle model with preprocessing.*

*More specifically, there is an efficient explicit transformation: (1) from any (S, T, ε) -hard data structure with efficient query generation to a (S, T, ε) -hard OWF in the random oracle model with preprocessing; and (2) from any (S, T, ε) -hard OWF in the random oracle model with preprocessing to an explicit construction of a (S, T, ε) -hard data structure. For the second transformation, the resulting data structure is always in **QuasiP** (with respect to its input size), and is in fact in **P** whenever the input/output size of the underlying OWF is linear in the input/output size of the random oracle.*

Proof. We show the two implications in turn.¹⁵

- **DS \Rightarrow OWF.** Let $g : \{0, 1\}^{\tilde{N}} \times \{0, 1\}^{m'} \rightarrow \{0, 1\}^{n'}$ be a data structure problem, and let $h : \{0, 1\}^{\tilde{N}} \times \{0, 1\}^{n'} \rightarrow \{0, 1\}^{m'}$ be an efficient query generator for g such that (g, h) is (S, T, ε) -hard. Let $R : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a random oracle, such that $\tilde{N} = n2^n$. We define an oracle function $f^R : \{0, 1\}^{n'} \rightarrow \{0, 1\}^{m'}$ as follows:

$$f^R(x) = h(\hat{R}, x),$$

¹⁴For simplicity we consider the uniform distributions on D and Y , but all definitions and results easily generalize to arbitrary distributions.

¹⁵Throughout this proof, we assume the domain and range of the data structure problem and OWF are bitstrings. The proof generalizes to arbitrary domains and ranges.

where \hat{R} denotes the binary representation of R .

f is a (S, T, ε) -hard OWF in the random oracle model with preprocessing, because it is efficiently computable and hard to invert, as proven next. Since h is efficiently computable, f runs in time $\text{poly}(n', m')$.

It remains to show that f is (S, T, ε) -hard to invert. Suppose, for contradiction, that this is not the case: namely, that there is a two-part adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that

$$\Pr_{x \leftarrow \{0,1\}^{n'}} \left[h \left(R, \mathcal{A}_2^{\mathcal{A}_1(R)}(h(R, x)) \right) = h(R, x) \right] > \varepsilon, \quad (10)$$

and \mathcal{A}_1 's output size is at most S , \mathcal{A}_2 makes at most T queries to $\mathcal{A}_1(R)$, and the probability is also over the sampling of the random oracle R .

We use \mathcal{A} to build $(\mathcal{B}_1, \mathcal{B}_2)$, an (S, T) -solver for g , as follows. On input $d \in \{0, 1\}^{\tilde{N}}$, \mathcal{B}_1 simply outputs $\phi(d) = \mathcal{A}_1(d)$. On input $q \in \{0, 1\}^{m'}$, \mathcal{B}_2 runs $\mathcal{A}_2^{\mathcal{A}_1(R)}(q)$; for each query ζ that \mathcal{A}_2 's makes to $\mathcal{A}_1(R)$, \mathcal{B}_2 simply queries $\phi(d)$ on ζ and returns the response to \mathcal{A}_2 .

It follows from (9) and (10) that

$$\Pr_{\substack{d \leftarrow \{0,1\}^{\tilde{N}} \\ y \leftarrow \{0,1\}^{n'}}} \left[\mathcal{B}_2^{\phi(d)}(h(d, y)) = y \right] \geq \varepsilon.$$

This contradicts the (S, T, ε) -hardness of (g, h) .

- **OWF \Rightarrow DS.** Let $f^R : \{0, 1\}^{n'} \rightarrow \{0, 1\}^{m'}$ be a (S, T, ε) -hard OWF in the random oracle model with preprocessing, for a random oracle mapping n bits to n bits. We design a data structure problem $g : \{0, 1\}^{\tilde{N}} \times \{0, 1\}^{m'} \rightarrow \{0, 1\}^{n'}$ and an efficient query generator h for g such that $\tilde{N} = n2^n$ and (g, h) is (S, T, ε) -hard, as follows.

- $h(d, y) = f^d(y)$.
- $g(d, q) = \min\{y \in Y : f^d(y) = q\}$.¹⁶

h is computable in time $\text{poly}(n', m')$, as required by Definition 22, because f^d is efficiently computable (in its input size). Furthermore, h satisfies (8) since g is, by construction, an inverse of h .

Next, we show that (g, h) is (S, T, ε) -hard. Suppose the contrary, for contradiction. Then there exists an (S, T) -solver $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for g that succeeds with probability greater than ε on query distribution $q = h(d, y) = f^d(y)$ where d, y are uniformly random. Then \mathcal{B} is quite literally an inverter for the OWF f , where d corresponds to the random oracle and q corresponds to the challenge value to be inverted: by assumption, \mathcal{B} satisfies

$$\Pr_{\substack{d \leftarrow \{0,1\}^n \rightarrow \{0,1\}^n \\ y \leftarrow \{0,1\}^{n'}}} \left[f^d \left(\mathcal{B}_2^{\mathcal{B}_1(d)}(f^d(y)) \right) = f^d(y) \right] > \varepsilon.$$

This contradicts the (S, T, ε) -hardness of f .

Finally, g is computable in $\mathbf{DTIME}[2^{n'} \cdot \text{poly}(n')]$, since it can be solved by exhaustively searching all $y \in \{0, 1\}^{n'}$ and outputting the first (i.e., minimum) such that $f^d(y) = q$. Note

¹⁶For the purpose of this proof, $g(d, \cdot)$ can be any inverse of f^d that is computable in time $O(2^{n'})$. We use the concrete example of $g(d, q) = \min\{y \in Y : f^d(y) = q\}$ for ease of exposition.

that $n', m' \in \text{poly}(n)$ since n', m' are the input and output sizes of a OWF with oracle access to a random oracle mapping n bits to n bits. Hence, g is computable in time quasipolynomial in $|d| = \tilde{N} = n2^n$, i.e., the size of g 's first input. In particular, g is computable in time $\text{poly}(\tilde{N})$ whenever $n', m' \in O(n)$. \square

Remark 13. As an example, a one-way function $f^R: \{0, 1\}^{5n} \rightarrow \{0, 1\}^{5n}$ in the random oracle model with preprocessing $S = 2^{3n}$ would give an *adaptive* data structure lower bound for a function with N inputs, N^5 outputs, space $S = \Omega(N^3 / \text{poly log}(N))$ and query time $T = \text{poly log}(N)$. Finding such a function is a big open problem in the area of static data structures [Sie04, Păt11, PTW10, Lar12, DGW19].

5.3.1 Cryptography with preprocessing and circuit lower bounds

Although the existence of cryptography in the random oracle model with preprocessing does not have such strong implications in complexity as the existence of regular cryptography, in Theorem 26 we show that it still has significant implications in circuit complexity.

A long-standing open problem in computational complexity is to find a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ which cannot be computed by binary circuits of linear size $O(n)$ and logarithmic depth $O(\log n)$ [Val77, AB09, Frontier 3].¹⁷ We now show that a weak one-way function with preprocessing would resolve this question.

First we recall the classical result of Valiant [Val77] asserting that every linear-size circuit of logarithmic depth can also be efficiently computed in the common bits model.

Definition 24. A function $f = (f_1, \dots, f_m): \{0, 1\}^n \rightarrow \{0, 1\}^m$ has an (s, t) -solution in the common bits model if there exist s functions $h_1, \dots, h_s: \{0, 1\}^n \rightarrow \{0, 1\}$, such that each f_i can be computed from t inputs and t functions h_i .

Theorem 25 ([EGS75, Val77, Cal08, Vio09]). *Let $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$. For every $c, \varepsilon > 0$ there exists $\delta > 0$ such that*

1. *If f can be computed by a circuit of size cn and depth $c \log n$, then f has an $(\delta n / \log \log n, n^\varepsilon)$ -solution in the common bits model.*
2. *If f can be computed by a circuit of size cn and depth $c \log n$, then f has an $(\varepsilon n, 2^{\log n^{1-\delta}})$ -solution in the common bits model.*
3. *If f can be computed by a series-parallel circuit of size cn (and unbounded depth), then f has an $(\varepsilon n, \delta)$ -solution in the common bits model.*

Now we show that a weak OWF in the random oracle model with preprocessing (for certain settings of parameters) implies a super-linear circuit lower bound. This proof employs the approach used in [DGW19, Vio19, CK19, RR20]. For ease of exposition, in the next theorem we assume that the preprocessing is measured in bits (i.e., the word size w is a constant number of bits). For this reason, a trivial inverter for a function $f^R: \{0, 1\}^{n'} \rightarrow \{0, 1\}^{n'}$ requires space $n'2^{n'}$. This assumption is not crucial, and the result easily generalizes to any w , in which case the amount of preprocessing is decreased by a factor of w .

¹⁷The same question is open even for series-parallel circuits [Val77]. A circuit is called series-parallel if there exists a numbering ℓ of the circuit's nodes s.t. for every wire (u, v) , $\ell(u) < \ell(v)$, and no pair of arcs $(u, v), (u', v')$ satisfies $\ell(u) < \ell(u') < \ell(v) < \ell(v')$.

Theorem 26. Let $f^R : \{0, 1\}^{n'} \rightarrow \{0, 1\}^{n'}$ be a (S, T, ε) -hard OWF in the random oracle model with preprocessing, for a random oracle $R : \{0, 1\}^n \rightarrow \{0, 1\}^n$, where $n' = O(n)$. We construct a function $G \in \mathbf{P}$ such that:

1. If $S \geq \omega\left(\frac{n'2^{n'}}{\log n'}\right)$, $T \geq 2^{\delta n}$ and $\varepsilon = 1$ for a constant $\delta > 0$, then G cannot be computed by a circuit of linear size and logarithmic depth.
2. If $S \geq \delta n'2^{n'}$, $T \geq 2^{n^{1-o(1)}}$ and $\varepsilon = 1$ for a constant $\delta > 0$, then G cannot be computed by a circuit of linear size and logarithmic depth.
3. If $S \geq \delta n'2^{n'}$, $T \geq \omega(1)$ and $\varepsilon = 1$ for a constant $\delta > 0$, then G cannot be computed by a series-parallel circuit of linear size.

Proof. Let $\tilde{N} = n2^n$, and let $g : \{0, 1\}^{\tilde{N}} \times \{0, 1\}^{n'} \rightarrow \{0, 1\}^{n'}$ be defined as

$$g(d, q) = \min\{y \in \{0, 1\}^{n'} : f^d(y) = q\}.$$

Let $\ell := \frac{n'2^{n'}}{\tilde{N}}$, and let us define ℓ data structure problems $g_i : \{0, 1\}^{\tilde{N}} \times [\tilde{N}/n'] \rightarrow \{0, 1\}^{n'}$ for $i \in [\ell]$ as follows:

$$g_i(d, q) = g(d, q + (i - 1) \cdot \tilde{N}/n'),$$

where we identify a binary string from $\{0, 1\}^{n'}$ with an integer from $[2^{n'}]$. Finally, we define $G : \{0, 1\}^{\tilde{N} + \log \ell} \rightarrow \{0, 1\}^{\tilde{N}}$ as

$$G(d, i) = g_i(d, 1) \parallel \dots \parallel g_i(d, \tilde{N}/n').$$

We claim that G cannot be computed by a circuit of linear size and logarithmic depth (a series-parallel circuit of linear size, respectively). The proofs of the three statements of this theorem follow the same pattern, so we only present the proof of the first one.

Assume, for contradiction, that there is a circuit of size $O(\tilde{N})$ and depth $O(\log \tilde{N})$ that computes G . By Theorem 25, G has an (s, t) -solution in the common bits model, where $s = O(\tilde{N}/\log \log \tilde{N}) = O(2^n n / \log n)$ and $t = \tilde{N}^{\delta/2} < 2^{\delta n}$. Since each output of g is a part of the output of $G(\cdot, i)$ for one of the ℓ values of i , we have that g has an $(s \cdot \ell, t)$ -solution in the common bits model. In particular, g can be computed with preprocessing $s \cdot \ell = O(n'2^{n'}/\log n)$ and $t = 2^{\delta n}$ queries to the input. This, in turn, implies a $(n'2^{n'}/\log n, 2^{\delta n})$ -inverter for f^R .

Finally, we observe that the function G can be computed by \tilde{N}/n' evaluations of g , and g is trivially computable in time $2^{n'} \cdot \text{poly}(n')$. Therefore, $G \in \mathbf{DTIME}[\tilde{N} \cdot 2^{n'}] = \mathbf{DTIME}[2^{O(n)}] = \mathbf{DTIME}[\tilde{N}^{O(1)}] = \mathbf{P}$. \square

Remark 14. We remark that $\varepsilon = 1$ is the strongest form of the theorem, *i.e.*, the premise of the theorem only requires a function f^R which cannot be inverted on *all* inputs. Also, it suffices to have f^R which cannot be inverted by *non-adaptive* algorithms, *i.e.*, algorithms where \mathcal{A}_2 is non-adaptive (see Definition 15).

Acknowledgments

Many thanks to Erik Demaine for sending us a manuscript of his one-query lower bound with Salil Vadhan [DV01]. We also thank Henry Corrigan-Gibbs and Dima Kogan for useful discussions.

The work of AG is supported by a Rabin Postdoctoral Fellowship. The work of TH is supported in part by the National Science Foundation under grants CAREER IIS-1149662, CNS-1237235 and

CCF-1763299, by the Office of Naval Research under grants YIP N00014-14-1-0485 and N00014-17-1-2131, and by a Google Research Award. The work of SP is supported by the MIT Media Lab’s Digital Currency Initiative, and its funders; and an earlier stage of SP’s work was funded by the following grants: NSF MACS (CNS-1413920), DARPA IBM (W911NF-15-C-0236), Simons Investigator award agreement dated June 5th, 2012, and the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370. The work of VV is supported in part by NSF Grants CNS-1350619, CNS-1718161 and CNS-1414119, an MIT-IBM grant, a Microsoft Faculty Fellowship and a DARPA Young Faculty Award.

References

- [AAC⁺17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond Hellman’s time-memory trade-offs with applications to proofs of space. In *ASIACRYPT 2017*, pages 357–379, 2017.
- [AAD⁺12] Oswin Aichholzer, Franz Aurenhammer, Erik D. Demaine, Ferran Hurtado, Pedro Ramos, and Jorge Urrutia. On k -convex polygons. *Comput. Geom.*, 45(3):73–87, 2012.
- [AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [ACH⁺98] Esther M. Arkin, Yi-Jen Chiang, Martin Held, Joseph S. B. Mitchell, Vera Sacristan, Steven S. Skiena, and Tae-Cheon Yang. On minimum-area hulls. *Algorithmica*, 21(1):119–136, 1998.
- [ACLL14] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *ICALP 2014*, pages 114–125. Springer, 2014.
- [AEK05] Daniel Archambault, Willam Evans, and David Kirkpatrick. Computing the set of all the distant horizons of a terrain. *Int. J. Comput. Geom. Appl.*, 15(06):547–563, 2005.
- [AHI⁺01] Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. Smallest color-spanning objects. In *ESA 2001*, pages 278–289. Springer, 2001.
- [AHP08] Boris Aronov and Sarel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [AKL⁺16] Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *ISAAC 2016*, pages 12:1–12:12, 2016.
- [ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In *STOC 2016*, pages 1101–1114, 2016.
- [AV14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS 2014*, pages 434–443. IEEE, 2014.
- [AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP 2014*, pages 39–51. Springer, 2014.

- [BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO 2006*, pages 1–21. Springer, 2006.
- [BCC⁺13] Ayelet Butman, Peter Clifford, Raphaël Clifford, Markus Jalsenius, Noa Lewenstein, Benny Porat, Ely Porat, and Benjamin Sach. Pattern matching under polynomial transformation. *SIAM J. Comput.*, 42(2):611–633, 2013.
- [BFM18] Balthazar Bauer, Pooya Farshim, and Sogol Mazaheri. Combiners for backdoored random oracles. In *CRYPTO 2018*, pages 272–302. Springer, 2018.
- [BHKN13] Itay Berman, Iftach Haitner, Ilan Komargodski, and Moni Naor. Hardness preserving reductions via cuckoo hashing. In *TCC 2013*, volume 7785, pages 40–59. Springer, 2013.
- [BHP01] Gill Barequet and Sarel Har-Peled. Polygon containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. *Int. J. Comput. Geom. Appl.*, 11(04):465–474, 2001.
- [BN16] Elette Boyle and Moni Naor. Is there an oblivious RAM lower bound? In *ITCS 2016*, pages 357–368, 2016.
- [BVKT98] Prosenjit Bose, Marc Van Kreveld, and Godfried Toussaint. Filling polyhedral molds. *Comput.-Aided Des.*, 30(4):245–254, 1998.
- [BW09] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *FOCS 2009*, pages 745–754. IEEE, 2009.
- [Cal08] Chris Calabro. A lower bound on the size of series-parallel graphs dense in long paths. In *ECCC*, volume 15, 2008.
- [CCI⁺19] Sergio Cabello, Jean Cardinal, John Iacono, Stefan Langerman, Pat Morin, and Aurélien Ooms. Encoding 3SUM. *arXiv:1903.02645*, 2019.
- [CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *CRYPTO 2018*, pages 693–721, 2018.
- [CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *EUROCRYPT 2018*, pages 227–258. Springer, 2018.
- [CEHP07] Otfried Cheong, Alon Efrat, and Sarel Har-Peled. Finding a guard that sees most and a shop that sells most. *Discrete Comput. Geom.*, 37(4):545–563, 2007.
- [CHC09] Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao. Approximate matching for run-length encoded strings is 3SUM-hard. In *CPM 2009*, pages 168–179. Springer, 2009.
- [CK19] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *TCC 2019*, pages 393–421. Springer, 2019.
- [CL15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *STOC 2015*, pages 31–40. ACM, 2015.

- [CMG⁺16] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the juniper dual EC incident. In *CCS 2016*, pages 468–479. ACM, 2016.
- [CNE⁺14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In *USENIX 2014*, pages 319–335, 2014.
- [dBdGO97] Mark de Berg, Marko M. de Groot, and Mark H. Overmars. Perfect binary space partitions. *Comput. Geom.*, 7(1-2):81–91, 1997.
- [DGG⁺15] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In *EUROCRYPT 2015*, pages 101–126. Springer, 2015.
- [DGHM13] Grégory Demay, Peter Gazi, Martin Hirt, and Ueli Maurer. Resource-restricted indistinguishability. In *EUROCRYPT 2013*, pages 664–683, 2013.
- [DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT 2017*, pages 473–495. Springer, 2017.
- [DGW19] Zeev Dvir, Alexander Golovnev, and Omri Weinstein. Static data structure lower bounds imply rigidity. In *STOC 2019*, pages 967–978. ACM, 2019.
- [DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *CRYPTO 2010*, pages 649–665. Springer, 2010.
- [DV01] Erik D. Demaine and Salil P. Vadhan. Some notes on 3SUM, December 2001. Unpublished manuscript.
- [EGS75] Paul Erdős, Ronald L. Graham, and Endre Szemerédi. On sparse graphs with dense long paths. *Comp. and Math. with Appl.*, 1:145–161, 1975.
- [EHPM06] Jeff Erickson, Sarel Har-Peled, and David M. Mount. On the least median square problem. *Discrete Comput. Geom.*, 36(4):593–607, 2006.
- [Eri99] Jeff Erickson. Bounds for linear satisfiability problems. *Chicago J. Theor. Comput. Sci.*, 1999.
- [FJM18] Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. In *CSF 2018*, pages 105–118. IEEE, 2018.
- [FN00] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 2000.
- [GGKT05] Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005.

- [GKLP16] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How hard is it to find (honest) witnesses? In *ESA 2016*, pages 45:1–45:16, 2016.
- [GKLP17] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *WADS 2017*, pages 421–436. Springer, 2017.
- [GLP17] Isaac Goldstein, Moshe Lewenstein, and Ely Porat. Orthogonal vectors indexing. In *ISAAC 2017*, pages 40:1–40:12, 2017.
- [GO95] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5(3):165–185, 1995.
- [Gol01] Oded Goldreich. *Foundations of Cryptography*, volume I. Basic tools. Cambridge University Press, 2001.
- [Gre13] Matthew Green. A few more notes on NSA random number generators, 2013.
- [GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *FOCS 2000*, pages 305–313. IEEE, 2000.
- [Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.
- [Imp96] Russell Impagliazzo. Very strong one-way functions and pseudo-random generators exist relative to a random oracle, January 1996. Unpublished manuscript.
- [IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC 1989*, pages 44–61. ACM, 1989.
- [JLN19] Riko Jacob, Kasper Green Larsen, and Jesper Buus Nielsen. Lower bounds for oblivious data structures. In *SODA 2019*, pages 2439–2447, 2019.
- [JV16] Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, Triangles. *Algorithmica*, 74(1):326–343, 2016.
- [KP19] Tsvi Kopelowitz and Ely Porat. The strong 3SUM-INDEXING conjecture is false. *arXiv:1907.11206*, 2019.
- [KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *SODA 2016*, pages 1272–1287. SIAM, 2016.
- [Lar12] Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *FOCS 2012*, pages 293–301. IEEE, 2012.
- [LN93] Richard J. Lipton and Jeffrey F. Naughton. Clocked adversaries for hashing. *Algorithmica*, 9(3):239–252, 1993.
- [LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In *CRYPTO 2018*, pages 523–542, 2018.
- [Mil99] Peter Bro Miltersen. Cell probe complexity - a survey. In *FSTTCS 1999*, 1999.
- [Nao13] Moni Naor. Cryptography and data structures: A match made in heaven. View online at: <https://www.youtube.com/watch?v=hCmbLypK0xE>, 2013. The Sixth Israel CS Theory Day, 13/3/2013.

- [NIS01] NIST (National Institute of Standards and Technology). Advanced encryption standard (aes), November 2001. Federal Information Processing Standards Publication 197. Available online at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.
- [NIS15] NIST (National Institute of Standards and Technology). SHA-3 standard, August 2015. Federal Information Processing Standards Publication 202. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.
- [NSW08] Moni Naor, Gil Segev, and Udi Wieder. History-independent cuckoo hashing. In *ICALP 2008*, pages 631–642, 2008.
- [NY15] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *CRYPTO 2015*, pages 565–584, 2015.
- [Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO 2003*, pages 617–630, 2003.
- [Păt10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *STOC 2010*, pages 603–610. ACM, 2010.
- [Păt11] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011.
- [Pet15] Seth Pettie. Higher lower bounds from the 3SUM conjecture. View online at: <https://www.youtube.com/watch?v=OkagNffn7KQ>, 2015. Simons Institute Program on Fine-Grained Complexity and Algorithm Design, Fall 2015.
- [PTW10] Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *FOCS 2010*, pages 805–814. IEEE, 2010.
- [RR20] Sivaramakrishnan Natarajan Ramamoorthy and Cyrus Rashtchian. Equivalence of systematic linear data structures and matrix rigidity. In *ITCS 2020*, volume 151, page 35. LIPIcs, 2020.
- [RTYZ18] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Correcting subverted random oracles. In *CRYPTO 2018*, pages 241–271. Springer, 2018.
- [SEO03] Michael Soss, Jeff Erickson, and Mark Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Comput. Geom.*, 26(3):235–246, 2003.
- [Sie04] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.
- [Unr07] Dominique Unruh. Random oracles and auxiliary input. In *CRYPTO 2007*, pages 205–223. Springer, 2007.
- [Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS 1977*, pages 162–176, 1977.
- [Vio09] Emanuele Viola. On the power of small-depth computation. *Found. Trends Theor. Comput. Sci.*, 5(1):1–72, 2009.

- [Vio19] Emanuele Viola. Lower bounds for data structures with space close to maximum imply circuit lower bounds. *Theory Comput.*, 15(1):1–9, 2019.
- [VW09] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *STOC 2009*, pages 455–464. ACM, 2009.
- [Yao90] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *STOC 1990*, pages 84–94, 1990.
- [YY96a] Adam L. Young and Moti Yung. Cryptovirology: Extortion-based security threats and countermeasures. In *S&P 1996*, pages 129–140. IEEE, 1996.
- [YY96b] Adam L. Young and Moti Yung. The dark side of “black-box” cryptography, or: Should we trust capstone? In *CRYPTO 1996*, pages 89–103. Springer, 1996.
- [YY97] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *EUROCRYPT 1997*, pages 62–74. Springer, 1997.
- [Zim98] Marius Zimand. Efficient privatization of random bits. In *MFCS 1998, Workshop “Randomized Algorithms”*, 1998.