The Function-Inversion Problem: Barriers and Opportunities

Henry Corrigan-Gibbs and Dmitry Kogan

Stanford University
{henrycg,dkogan}@cs.stanford.edu

Abstract. The task of function inversion is central to cryptanalysis: breaking block ciphers, forging signatures, and cracking password hashes are all special cases of the function-inversion problem. In 1980, Hellman showed that it is possible to invert a random function $f\colon [N] \to [N]$ in time $T = \widetilde{O}(N^{2/3})$ given only $S = \widetilde{O}(N^{2/3})$ bits of precomputed advice about f. Hellman's algorithm is the basis for the popular "Rainbow Tables" technique (Oechslin, 2003), which achieves the same asymptotic cost and is widely used in practical cryptanalysis.

Is Hellman's method the best possible algorithm for inverting functions with preprocessed advice? The best known lower bound, due to Yao (1990), shows that $ST = \widetilde{\Omega}(N)$, which still admits the possibility of an $S = T = \widetilde{O}(N^{1/2})$ attack. There remains a long-standing and vexing gap between Hellman's $N^{2/3}$ upper bound and Yao's $N^{1/2}$ lower bound. Understanding the feasibility of an $S = T = N^{1/2}$ algorithm is cryptanalytically relevant since such an algorithm could perform a key-recovery attack on AES-128 in time 2^{64} using a precomputed table of size 2^{64}.

For the past 29 years, there has been no progress either in improving Hellman's algorithm or in strengthening Yao's lower bound. In this work, we connect function inversion to problems in other areas of theory to (1) explain why progress may be difficult and (2) explore possible ways forward.

Our results are as follows:

- We show that *any* improvement on Yao's lower bound on function-inversion algorithms will imply new lower bounds on depth-two circuits with arbitrary gates. Further, we show that proving strong lower bounds on *non-adaptive* function-inversion algorithms would imply breakthrough circuit lower bounds on linear-size log-depth circuits.
- We take first steps towards the study of the *injective* function-inversion problem, which has manifold cryptographic applications. In particular, we show that improved algorithms for breaking PRGs with preprocessing would give improved algorithms for inverting injective functions with preprocessing.
- Finally, we show that function inversion is closely related to well-studied problems in communication complexity and data structures. Through these connections we immediately obtain the best known algorithms for problems in these domains.

1 Introduction

A central task in cryptanalysis is that of *function inversion.* That is, given a function $f\colon [N] \to [N]$ and a point $y \in [N]$, find a value $x \in [N]$ such that $f(x) = y$, if one exists. The hardness of function inversion underpins the security of almost every cryptographic primitive we use in practice: block ciphers, hash functions, digital signatures, and so on. Understanding the exact complexity of function inversion is thus critical for assessing the security of our most important cryptosystems.

We are particularly interested in function-inversion algorithms that only make *black-box* use of the function f—or formally, that have only oracle access to f—since these algorithms invert *all* functions. A straightforward argument shows that any black-box inversion algorithm that makes at most T queries to its f-oracle succeeds with probability at most $O(T/N)$, over the randomness of the adversary and the random choice of the function. This argument suggests that an attacker running in $o(N)$ time cannot invert a black-box function on domain $[N]$ with good probability.

When the inversion algorithm may use *preprocessing*, this logic breaks down. An algorithm with preprocessing runs in two phases: In the preprocessing phase, the algorithm repeatedly queries f and then outputs an "advice string" about f. In the subsequent online phase, the algorithm takes as input its preprocessed advice string and a challenge point $y \in [N]$. It must then produce a value $x \in [N]$ such that $f(x) = y$. When using these algorithms for cryptanalysis, the attacker typically seeks to jointly minimize the bit-length S of the advice string and the running time T of the online algorithm. The computation required to construct the advice string, though usually expensive, can often be amortized over a large number of online inversions.

A trivial preprocessing algorithm stores a table of f^{-1} in its entirety as its advice string using $S = \widetilde{O}(N)$ bits and can then invert the function on all points using a single lookup into the table. In contrast, constructing algorithms that simultaneously achieve sublinear advice and online time $S = T = o(N)$ is non-trivial.

In a seminal paper, Hellman [48] introduced time-space tradeoffs as a tool for cryptanalysis and gave a black-box preprocessing algorithm that inverts a function $f\colon [N] \to [N]$ using only $S = \widetilde{O}(N^{2/3})$ bits of advice and online time $T = \widetilde{O}(N^{2/3})$, where the algorithm is guaranteed to succeed only on a constant fraction of functions. (More precisely, the algorithm has a constant success probability over the uniformly random choice of the function f.) Fiat and Naor [29, 30] later gave a rigorous analysis of Hellman's algorithm and extended it to invert all possible functions, albeit with a slightly worse trade-off of the form $S^3T = \widetilde{O}(N^3)$ for any choice of $N^{3/4} \leq S \leq N$. Hellman's trade-off is the best known today, and his algorithm is a fundamental tool in real-world cryptanalysis [7, 8, 61, 63].

In this work, we investigate the following question:

Is it possible to improve upon Hellman's time-space trade-off?

Yao first asked this question in 1990 [79] and proved that any preprocessing algorithm for function inversion that uses S bits of advice and T online queries must satisfy $ST = \widetilde{\Omega}(N)$. (Counting only queries—and not online computation—only strengthens lower bounds in this model.) Notably, this lower bound does not rule out an algorithm that achieves $S = T = \widetilde{O}(N^{1/2})$. In contrast, Hellman's algorithm only gives an upper bound of $S = T = \widetilde{O}(N^{2/3})$, even for the slightly easier case of inverting a random function. The question resurfaces in the work of Fiat and Naor [30], Barkan, Biham, and Shamir [5] (who show that Hellman's method is optimal for a certain natural but restricted class of algorithms), De, Trevisan and Tulsiani [23], and Abusalah et al. [1].

In addition to the problem's theoretical appeal, determining the best possible time-space trade-offs for function inversion is relevant to practice, since the difference between an online attack time of $N^{2/3}$ and an $N^{1/2}$ becomes crucial when dealing with 128-bit block ciphers, such as the ubiquitous AES-128. Hellman's algorithm gives the best known preprocessing attack against AES-128, with $S = T \approx 2^{86}$. If we could improve Hellman's algorithm to achieve $S = T = N^{1/2}$, matching Yao's lower bound, we could break AES-128 in time 2^{64} with a data structure of size 2^{64}, albeit after an expensive preprocessing phase. While today's $S = T = 2^{86}$ attack is likely far beyond the power of any realistic adversary, an improved $S = T = 2^{64}$ attack would leave us with an alarmingly narrow security margin.

Recent work proves new lower bounds on preprocessing algorithms for various cryptographic problems, using both incompressibility arguments [1, 25, 34] and the newer presampling method [20, 67]. While this progress might give hope for an improved lower bound for function inversion as well, both techniques mysteriously fail to break the $ST = \widetilde{\Omega}(N)$ barrier.

Non-adaptive algorithms. Another avenue for study is to explore the role of *parallelism* or *adaptivity* in preprocessing algorithms for function inversion. All non-trivial algorithms for function inversion, including Hellman's algorithm and Rainbow-table methods [63], critically use the adaptivity of their queries. It would be very interesting to construct a highly parallelizable preprocessing algorithm for function inversion. Such an algorithm would achieve the same advice and time complexity $S = T = \widetilde{O}(N^{2/3})$ as Hellman's algorithm, but would make all $\widetilde{O}(N^{2/3})$ of its queries to the f-oracle in one non-adaptive batch. Such a non-adaptive inversion algorithm could speed up function inversion on cryptanalytic machines with a very large number of parallel processing cores.

We do not even know if there exists a non-adaptive algorithm with $S = T = o(N)$. Can we find new non-adaptive inversion algorithms, or is adaptivity necessary for good time-space trade-offs? Proving lower bounds in this more restricted model could be a stepping stone to improving the general lower bounds on function inversion.

1.1 Our results

This work establishes new connections between the function-inversion problem and well-studied problems in cryptography, complexity theory, and data structures. These connections are useful in two directions.

First, they shed new light on the function-inversion problem: a connection to circuit complexity suggests that improving on the known lower bounds for function-inversion will be difficult. In particular, we show that new lower bounds for function inversion will imply new circuit lower bounds and could even resolve complexity-theoretic questions that predate Hellman's results [68]. Moreover, a new connection to the problem of breaking PRGs with preprocessing suggests a new avenue for better inversion algorithms for *injective functions*. For many of the cryptanalytic applications, progress on this variant of function inversion would in fact be sufficient.

Second, these connections, together with classic cryptanalytic algorithms, give rise to better algorithms for problems in the other areas of theory. For example, a connection to communication complexity leads to the best known algorithm for the multiparty pointer-jumping problem, improving upon a twenty-year-old upper bound [65]. Similarly, a connection to data structures leads to a new upper bound for the systematic substring-search problem, resolving an open question [31].

We now state our results in detail.

Proving better lower bounds for function-inversion implies new circuit lower bounds. A major question in circuit complexity, open since the 1970s [68, 69], is to give an explicit family of functions $F_n\colon \{0,1\}^n \to \{0,1\}^n$ that cannot be computed by fan-in-two circuits of size $O(n)$ and depth $O(\log n)$. Following ideas of Brody and Larsen [13], we demonstrate a close connection between this classic problem in circuit complexity and non-adaptive preprocessing algorithms for function inversion.

Specifically, we show that proving that every *non-adaptive* black-box function-inversion algorithm that uses $S = N \log N/\log\log N$ bits of advice requires at least $T = \Omega(N^\epsilon)$ oracle queries, for some constant $\epsilon > 0$, would give an explicit family of functions that cannot be computed by linear-size log-depth Boolean circuits. This, in turn, would resolve a long-standing open problem in circuit complexity. Though we cannot prove it, we suspect that the above lower bound holds even for $\epsilon = 1$.

This connection implies that proving lower bounds against non-adaptive function-inversion algorithms that use the relatively large amount of advice $S = N \log N/\log\log N$ should be quite difficult. A much more modest goal would be to rule out any non-adaptive algorithm using $S = T = \widetilde{O}(N^{1/2+\epsilon})$, for some $\epsilon > 0$. This would represent only a slight strengthening of Yao's $ST = \widetilde{\Omega}(N)$ bound for adaptive algorithms. However, we show that achieving even this far-more-modest goal would improve the best known lower bound for circuits in Valiant's common-bits model [68, 69]. This, in turn, would represent substantial progress towards proving lower bounds against linear-size log-depth circuits. In particular, since any lower bound against algorithms without a restriction on adaptivity would only be more general, *improving the $ST = \widetilde{\Omega}(N)$ lower bound for function inversion would imply new circuit lower bounds in Valiant's common-bits model.*

We believe that the difficulty of proving such a circuit lower bound suggests that beating the square-root barrier exhibited by both the compression [35, 79] and presampling [20, 67] techniques might prove more difficult than previously expected.

One-to-one function inversion from PRG distinguishers. Many cryptanalytic applications of Hellman tables (cryptanalysis of block ciphers, password cracking, etc.) only require inverting *injective* functions. Does there exist a better-than-Hellman algorithm for inverting injective functions with preprocessing?

One reason to hope for a better algorithm for injective functions is that for the very special case of *permutations*, there exists an inversion algorithm with preprocessing that achieves the improved trade-off $ST = \widetilde{O}(N)$ (i.e., $S = T = N^{1/2}$) [79]. Can we achieve the same trade-off for injective functions?

While we have not been able to answer this question yet, we do open one possible route to answering it. In particular, we show that the problem of inverting injective functions with preprocessing has a close connection to the problem of breaking pseudorandom generators (PRGs) with preprocessing [2, 20, 23, 25, 26]. Specifically, De, Trevisan, and Tulsiani [23] show that black-box PRG distinguishers with preprocessing can realize the trade-off $S = \widetilde{O}(\epsilon^2 N)$, for $T = \widetilde{O}(1)$ and for any choice of distinguishing advantage ϵ.

We show that achieving a more general trade-off of the form $ST = \widetilde{O}(\epsilon^2 N)$, for any constant ϵ, would imply a better-than-Hellman algorithm for inverting injective functions. Thus, improving the known PRG distinguishers with preprocessing can improve the known injective inversion algorithms.

New protocols for multiparty pointer jumping. We show that algorithms for the black-box function-inversion problem are useful in designing new communication protocols for a well-studied problem in communication complexity. In particular, any black-box preprocessing algorithm for inverting permutations yields a protocol for the permutation variant of the "k-party pointer-jumping" problem ($\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$) [10, 11, 14, 22, 58, 65, 72] in the number-on-the-forehead model of communication complexity [16].

Then, by instantiating the permutation-inversion algorithm with a variant of Hellman's method, we obtain the *best known protocol* for $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$ for $k = \omega(\log N / \log\log N)$ players (this regime is in fact the most consequential for the original motivation for studying this problem), improving the previous best upper bound of $O(N \log\log N / \log N)$, by Pudlák et al. [65], to $\widetilde{O}(N/k + \sqrt{N})$. We thus make progress on understanding the communication complexity of multiparty pointer jumping, a problem with significance to ACC^0 circuit lower bounds [6, 49, 80].

Beyond the quantitative improvement, our protocol is different from all previous approaches to the problem and is an unexpected application of a cryptanalytic algorithm to a communication-complexity problem. While the use of a cryptanalytic algorithm in this context appears new, prior work has found application of results in communication complexity to lower bounds [46] and constructions [12] in the cryptographic setting.

This connection presents a path forward for proving non-adaptive lower bounds for permutation inversion. In particular, we show that for every non-adaptive black-box permutation-inversion algorithm using S bits of advice and T online queries, it must hold that $\max\{S,T\}$ is at least as large as the communication complexity of $\mathsf{MPJ}^{\mathsf{perm}}_{N,3}$. Any improvement on the lower bound for $\mathsf{MPJ}^{\mathsf{perm}}_{N,3}$ would give an improved lower bound for non-adaptive black-box permutation-inversion algorithms. The best lower bound for $\mathsf{MPJ}^{\mathsf{perm}}_{N,3}$ is $\Omega(\sqrt{N})$ [3,75]. Interestingly, this matches the best lower-bound for black-box permutation-inversion algorithms, regardless of their adaptivity.

New time-space trade-off for systematic substring search. Finally, we show that improved algorithms for function inversion will also imply improved data structures for the *systematic substring-search problem* [24, 31, 32, 42, 43]. In particular, we prove that there is a preprocessing algorithm for the function-inversion problem using few bits of advice and few online queries if and only if there is a space- and time-efficient data structure for systematic substring search in the cell-probe model [77]. In the systematic substring-search problem, we are given a bitstring of length N (the "text"), and from it we must construct an S-bit data structure (the "index"). Given a query string, we should be able to determine whether the query string appears as a substring of the text by reading the index and by inspecting at most T bits of the original text.

This connection is fruitful in two directions: First, we show that instantiating this connection with the Fiat-Naor algorithm for function inversion [30] yields an $S^3T = \widetilde{O}(N^3)$ systematic data structure, which is the best known in the parameter regime $S = \widetilde{O}(N^\alpha)$ for $\alpha < 1$. Gál and Miltersen [31] ask whether a very strong $S + T = \widetilde{\Omega}(N)$ lower bound on this problem is possible. By beating this hypothetical lower bound, our algorithm answers their open question in the negative.

Second, Gál and Miltersen prove an $ST = \widetilde{\Omega}(N)$ lower bound for systematic substring search. Our barrier to proving lower bounds against black-box algorithms for function inversion implies that improving this lower bound would also imply new lower bounds in Valiant's circuit model and therefore may be quite challenging.

1.2 Related work

We now recall a few salient related results on function inversion, and we discuss additional related work at relevant points throughout the text.

Fiat and Naor [29,30] proved that Hellman's algorithm [48] achieves a trade-off of the form $S^2T = \widetilde{O}(N^2)$, when the algorithm needs only to invert a random function with constant probability (i.e., in the cryptanalytically interesting case). For the worst-case problem of inverting arbitrary functions, Fiat and Naor give an algorithm that achieves a trade-off of the form $S^3T = O(N^3)$. De, Trevisan, and Tulsiani [23] improve the Fiat-Naor trade-off when the algorithm needs only to invert the function at a sub-constant fraction of points.

For inverting functions, Yao [79] proved that every algorithm that uses S bits of advice and makes T online queries must satisfy $ST = \widetilde{\Omega}(N)$ lower bound. Impagliazzo gives a short alternative proof [50]. Dodis et al. [25], building on prior work [23, 35], extended the lower bound to capture algorithms that invert only a sub-constant fraction of functions f.

Barkan, Biham, and Shamir [5] show that, for a *restricted class of preprocessing algorithms*, a Hellman-style trade-off of the form $S^2T = \widetilde{O}(N^2)$ is the best possible. Their lower bound is powerful enough to capture the known inversion schemes, including Hellman's algorithm and Oechslin's practically efficient "Rainbow tables" technique [63]. At the same time, this restricted lower bound leaves open the possibility that an entirely new type of algorithm could subvert their lower bound.

For inverting *permutations*, Yao [79] observed that a Hellman-style algorithm can achieve the $ST = \widetilde{O}(N)$ upper bound and proved a matching lower bound. Gennaro and Trevisan [35], Wee [73], and De, Trevisan, and Tulsiani [23] extend this lower bound to handle randomized algorithms and those that succeed with small probability.

Two recent works [41, 54] use the function-inversion algorithm of Fiat and Naor to obtain new algorithms for the preprocessing version of the 3-SUM problem.

1.3 Preliminaries

Notation. Through this paper, $\mathbb{Z}^{\geq 0}$ denotes the non-negative integers, and $\mathbb{Z}^{>0}$ denotes the positive integers. For any $N \in \mathbb{Z}^{>0}$ we write $[N] = \{1, 2, \ldots, N\}$. We often identify every element $x \in [N]$ with the binary representation of $x - 1$ in $\{0, 1\}^{\lceil \log N \rceil}$. We use $x \leftarrow 4$ to denote assignment and, for a finite set $\mathcal{X}$, we use $x \overset{\text{R}}{\leftarrow} \mathcal{X}$ to denote a uniform random draw from $\mathcal{X}$. For a function $f : A \to B$, we denote the image of the function as $\text{Im}(f) = \{f(x) \mid x \in A\} \subseteq B$, and $y \in B$, we define the preimage set of y as $f^{-1}(y) := \{x \in A \mid f(x) = y\}$. All logarithms are base-two unless stated otherwise. Parameters S and T are always implicit functions of the parameter N, and to simplify the bounds, we always implicitly take $S = T = \Omega(1)$. The notation $\widetilde{\Omega}(\cdot)$ and $\widetilde{O}(\cdot)$ hides factors polynomial in $\log N$.

Definition 1 (Black-box inversion algorithm with preprocessing). Let $N \in \mathbb{Z}^{>0}$. A black-box inversion algorithm with preprocessing for functions on $[N]$ is a pair $(\mathcal{A}_0, \mathcal{A}_1)$ of oracle algorithms, such that $\mathcal{A}_0$ gets oracle access to a function $f \colon [N] \to [N]$, takes no input, and outputs an *advice string* $\mathsf{st}_f \in \{0, 1\}^*$. Algorithm $\mathcal{A}_1$ gets oracle access to a function $f \colon [N] \to [N]$, takes as input a string $\mathsf{st}_f \in \{0, 1\}^*$ and a point $y \in [N]$, and outputs a point $x \in [N]$. Moreover, for every $x \in [N]$, it holds that $\mathcal{A}_1^f(\mathcal{A}_0^f(), f(x)) \in f^{-1}(f(x))$.

We can define a black-box inversion algorithm *for permutations* analogously by restricting the oracle $f \colon [N] \to [N]$ to implement an injective function. In this case, we will often denote the oracle as π instead of f.

Definition 2 (Adaptivity). We say that an oracle algorithm is *k-round adaptive* if the algorithm's oracle queries consist of k sets, such that each set of queries depends on the advice string, the input, and the replies to the previous rounds of queries. We call a 1-round adaptive algorithm *non-adaptive*. Finally, we say that an algorithm is *strongly non-adaptive* if it issues a single set of queries that only depends on the algorithm's input, but not on the advice string. In all of the above cases, when referring to the number of queries made by the algorithm, we account for the sum over all rounds.

Worst case versus average case. The algorithms in Definition 1 are deterministic and successfully invert all functions on all points. It is also interesting to consider algorithms that invert successfully only with probability $\epsilon < 1$, over the random choice of: the function $f \colon [N] \to [N]$, the point to invert, and/or algorithm's randomness. As most of the results in this paper deal with *barriers for improving lower bounds*, restricting ourselves to deterministic algorithms that always succeed in inverting only makes these results *stronger*. In any case, assume that all algorithms we consider halt with probability 1.

Running time versus query complexity. For the purposes of proving lower bounds, and reductions towards proving lower bounds, it suffices to consider the query complexity of a preprocessing algorithm's online phase. Counting only queries (and not computation time) only strengthens lower bounds proved in this model. The algorithms we construct can be made to use only $\widetilde{O}(N)$ preprocessing time in a suitable RAM model, when they are allowed to fail with small probability. Furthermore, the running time of our algorithms' T-query online phase is $\widetilde{O}(T)$.

Non-uniformity. Our definition allows for "free" non-uniformity in the parameter N. Nevertheless, in a model that only "charges" the online algorithm for queries to the oracle and ignores the actual running time, non-uniformity makes little difference since a uniform algorithm can simply search for the optimal choice of non-uniform advice without increasing its query complexity.

Shared randomness. We allow the preprocessing and online phases to access a common stream of random bits. Allowing the adversary to access correlated randomness in both phases only strengthens the lower bounds. Only one of our upper bounds (Theorem 8) makes use of this correlated randomness.

2 Lower bounds on inversion imply circuit lower bounds

The motivating question of this work is whether Hellman's $S = T = \widetilde{O}(N^{2/3})$ algorithm for inverting random functions is optimal. In this section, we show that resolving this question will require proving significant new lower bounds in Valiant's "common bits" model of circuits [68]. We also show that proving strong lower bounds on *non-adaptive* algorithms for function inversion would imply new lower bounds against linear-sized logarithmic-depth circuits.

We obtain these connections by observing that the function-inversion problem is an example of a class of so called "succinct" static data-structure problems [4, 18, 32, 33, 42, 43, 45, 47, 51, 60, 66]. We show a barrier to proving lower bounds against *systematic* data structures, which are a special case of succinct data structures.

Related work. Brody and Larsen [13] showed that proving certain lower bounds against *linear* data structures for *dynamic* problems would imply strong lower bounds on the wire complexity of linear depth-two circuits. We follow their general blueprint, but we instead focus on *arbitrary* algorithms for solving *static* data-structure problems (e.g., function inversion), and our connection is to Valiant's common-bits model of circuits, rather than to linear depth-two circuits.

In recent independent work, Viola [71, Theorem 3] shows that lower bounds against a large class of static data-structures problems imply circuit lower bounds. In his work, Viola considers an incomparable circuit model that, on the one hand, admits circuits of depth larger than two, but, on the other hand, restricts the number of wires connected to the common bits. As a result, Viola's work does not seem to apply to the function-inversion problem within the relevant parameter regime (namely, in the gap between Hellman's upper bound and Yao's lower bound).

In another recent independent work, Dvir, Golovnev, and Weinstein [28] connect data-structure lower bounds to matrix rigidity and circuit lower bounds. Their focus is on *linear* data structures, whereas the function inversion problem, considered in our work, does not have an apparent linear structure.

Boyle and Naor [9] make a surprising connection between cryptographic algorithms and circuit lower bounds. They show that proving the non-existence of certain "offline" oblivious RAM algorithms (ORAMs) [36, 40, 64] would imply new lower bounds on the size of Boolean circuits for sorting lists of integers. Larsen and Nielsen [56] recently skirted this barrier by proving a lower bound against ORAMs in the "online" setting. Following that, Weiss and Wichs [74] showed that a variant of the Boyle-Naor barrier still holds against "online read-only" ORAMs.

2.1 Systematic data structures and low-depth circuits

A major open question in circuit complexity is whether there exists an explicit family of Boolean functions (from n bits to one bit) that cannot be computed by fan-in-two circuits of size $O(n)$ and depth $O(\log n)$. An easier problem, which is still famously difficult, is to find an explicit family of functions $F_n \colon \{0,1\}^n \to \{0,1\}^n$ with n-bit output—often called *Boolean operators*—that cannot be computed by this same class of circuits. Even this question has been open since the 1970s [53, 68, 69].

More precisely, we say that a family of Boolean operators $\{F_n\}_{n\in\mathbb{Z}^{>0}}$, for $F_n \colon \{0,1\}^n \to \{0,1\}^n$, is an *explicit operator* if the decision problem associated with each bit of the output of F_n is in the complexity class NP.

The main result of this section is that proving a certain type of data-structure lower bound implies the existence of an explicit Boolean operator on n bits that cannot be computed by fan-in-two circuits of size $O(n)$ and depth $O(\log n)$. We then show that a lower bound on function-inversion algorithms can be cast as

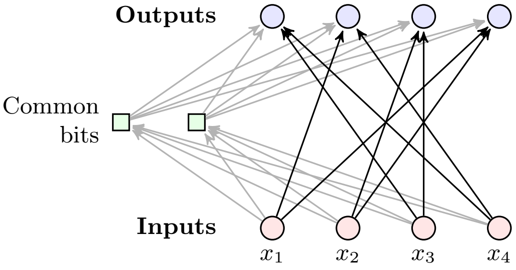

Fig. 1: Common-bits circuit with $n = 4$ inputs, degree $d = 2$, and width $w = 2$.

a data-structure lower bound, and therefore a function-inversion lower bound implies a circuit lower bound.

We now give the necessary background on data-structure problems. A *systematic* data structure of size s and query complexity t for an operator F_n is a pair of algorithms:

- a preprocessing algorithm, which takes as input the data $x \in \{0,1\}^n$ and outputs a string $\mathsf{st} \in \{0,1\}^s$ of length $s = o(n)$, and
- a query algorithm, which takes as input the string st, and an index $i \in [n]$, may probe (read) t bits of the input x, and then outputs the ith bit of $F_n(x)$.

A systematic data structure is *non-adaptive* if the query algorithm probes a set of bits of the input data x whose location depends only on the index i and not on the input data x.

The following theorem is the main result of this section.

Theorem 3. *If an explicit operator $\{F_n\}_{n \in \mathbb{Z}^{>0}}$ has fan-in-two Boolean circuits of size $O(n)$ and depth $O(\log n)$ then, for every $\epsilon > 0$, then this operator admits a non-adaptive systematic data structure of size $O(n/\log\log n)$ and query complexity $O(n^\epsilon)$.*

To prove this, we first recall Valiant's common-bits model of circuits [68, 69].

Valiant's common-bits model. A circuit in the common-bits model of width w and degree d computing a Boolean operator $F_n \colon \{0,1\}^n \to \{0,1\}^n$ contains an input layer, a middle layer, and output layer (Figure 1). The input layer consists of n input bits $x_1, \ldots, x_n \in \{0,1\}$, and the output layer consists of n output gates. There are w gates in the middle layer of the circuit (the "common bits"); each input feeds into each of these w middle gates, and the output of each of the w middle gates feeds into each output gate. Further, each output gate reads from at most d of the inputs. Unlike in a standard circuit, the gates in the middle and output layers of the circuit compute *arbitrary* functions of their inputs. The output of the circuit is the n-bit string formed at the output gates.

It is immediate that any Boolean operator $F_n \colon \{0,1\}^n \to \{0,1\}^n$ has common-bits circuits of width n and degree 0 or, alternatively, of width 0 and degree

n. A non-trivial question is: For a given operator F_n and choice of degree (e.g., $d = n^{1/3}$), what is minimal width of a common-bits circuit that computes F_n?

Lemma 4. *If there exists a circuit in the common-bits model of width w and degree d that computes an operator $F\colon \{0,1\}^n \to \{0,1\}^n$, then there exists a non-adaptive systematic data structure for F of size w and query complexity d.*

Proof. Let $\mathcal{C}$ be a circuit in the common-bits model as in the statement of the lemma. The data structure consists of the outputs of the w middle-layer gates in the circuit $\mathcal{C}$ (i.e., the circuit's common bits). On input $i \in [n]$, the algorithm reads all the input bits connected to the ith output gate of $\mathcal{C}$ and computes the value of the output gate. Since each output gate in the circuit is connected to at most d input bits, the query complexity of the systematic data structure is at most d. □

Theorem 3 then follows from Lemma 4 and the following result of Valiant:

Theorem 5 (Valiant [68, 69]). *If every explicit operator has fan-in-two Boolean circuits of size $O(n)$ and depth $O(\log n)$, then for every constant $\epsilon > 0$, every explicit operator has circuits in the common-bits model of width $O(n/\log\log n)$ and degree n^ϵ.*

Viola [70, Section 3] and Jukna [52, Chapter 13] give detailed proofs of Theorem 5.

2.2 Consequences for function inversion

Observe that every function $f\colon [N] \to [N]$ can be described using $O(N \log N)$ bits, so there is a trivial strongly non-adaptive algorithm that inverts every function using $O(N \log N)$ bits of advice and no queries to the function in the online phase. We know of no non-adaptive function-inversion algorithm that inverts with constant probability using $o(N \log N)$ bits of advice and $o(N)$ queries. The following theorem states that ruling out the existence of such a non-adaptive algorithm is as hard as proving lower bounds against linear-size logarithmic-depth Boolean circuits.

Theorem 6. *If, for some $\epsilon > 0$, every family of strongly non-adaptive black-box algorithms for inverting functions $f\colon [N] \to [N]$ that uses $O(N^\epsilon)$ queries requires $\omega(N \log N/\log\log N)$ bits of advice, then there exists an explicit operator that* cannot *be computed by fan-in-two Boolean circuits of size $O(n)$ and depth $O(\log n)$.*

The theorem considers a restricted class of inversion algorithms that: (i) may only use strongly non-adaptive queries (the most restrictive type of query), (ii) are only allowed, for example, $O(N^{0.0001})$ queries (very few queries), and (iii) must invert arbitrary functions with probability one (the most difficult variant of the inversion problem).

So, even though we may suspect that there are no algorithms for inverting functions $f\colon [N] \to [N]$ using $O(N \log N/\log\log N)$ bits of advice and $O(N^{0.0001})$ non-adaptive queries, proving such an assertion seems very challenging.

Proof of Theorem 6. Let $n = N \log N$, where $N \in \mathbb{Z}^{>0}$ is a power of two. (For all other values of n, define the inversion operator trivially as the identity mapping.) We define the *inversion operator* $F_n^{\mathsf{inv}} \colon \{0,1\}^n \to \{0,1\}^n$ as follows. Let $x \in \{0,1\}^n$ be an input to F_n^{inv}, and view x as the concatenation of N blocks of length $\log N$ bits each: $x = x_1 \| x_2 \| \cdots \| x_N$. For each $i \in [N]$, let $y_i \in [N]$ be the least $j \in [N]$ such that $x_j = i$, if one such j exists. If no such j exists, set $y_i = 0$. We define $F_n^{\mathsf{inv}}(x) = (y_1 \| y_2 \| \cdots \| y_N)$.

Observe that a systematic data structure for F_n^{inv} gives a strongly non-adaptive preprocessing algorithm that inverts every function $f \colon [N] \to [N]$. The preprocessing phase constructs the data structure for operator F_n^{inv} on input $f(1) \| f(2) \| \dots \| f(N)$ and outputs this data structure as the advice string.

In the online phase, on input $i \in [N]$, the algorithm uses the data structure in the advice string and its oracle access to f to compute all $\log N$ bits of the ith output block y_i of F_n^{inv}, which is enough to recover some inverse of i under f, if it exists.

The theorem now follows from Theorem 3, instantiated with F_n^{inv}. For n of the form $n = N \log N$, where $N > 0$ is a power of two, we get that the length of the advice string is $O(N \log N / \log\log(N \log N)) = O(N \log N / \log\log N)$ and the online query complexity is $\log N \cdot O\left((N \log N)^{\epsilon}\right) = O(N^{\epsilon'})$, for any $\epsilon' > \epsilon$. □

Theorem 6 suggests the hardness of proving stronger lower bounds for non-adaptive inversion algorithms, but it applies only to algorithms that use a relatively long advice string, of length $O(N \log N / \log\log N)$. We might still hope to improve upon Yao's $ST = \widetilde{\Omega}(N)$ lower bound for function inversion without breaking the aforementioned barrier.

The following corollary shows that ruling out function-inversion algorithms using advice and time $S = T = \widetilde{O}(N^{1/2+\epsilon})$, for any $\epsilon > 0$, would imply the existence of an explicit operator that cannot be computed by circuits of width $O(n^{1/2+\epsilon'})$ and degree $O(n^{1/2+\epsilon'})$ in the common-bits model, for some $\epsilon' > 0$. As we will discuss, no such lower bound in the common-bits model is known, so proving the optimality of Hellman's $\widetilde{O}(N^{2/3})$ algorithm, or even showing that inverting functions with preprocessing is marginally harder than inverting permutations with preprocessing, would imply an advance in the state of lower bounds on circuits in the common-bits model.

Corollary 7. *If, for some $\epsilon > 0$, there does* not *exist a family of strongly non-adaptive algorithms for inverting functions $f \colon [N] \to [N]$ using $O(N^{1/2+\epsilon})$ bits of advice and $O(N^{1/2+\epsilon})$ queries, then there exists an explicit operator that does* not *have circuits in the common-bits model of width $O(n^{1/2+\epsilon'})$ and degree $O(n^{1/2+\epsilon'})$, for every ϵ' satisfying $0 < \epsilon' < \epsilon$.*

Proof. We prove the contrapositive. Assume that for every $\epsilon' > 0$, every explicit operator has common-bits circuits of width $O(n^{1/2+\epsilon'})$ and depth $O(n^{1/2+\epsilon'})$. Then, as in the proof of Theorem 6, we can apply Lemma 4 to operator F_n^{inv} to show that, for $n = N \log N$, there exists a strongly non-adaptive preprocessing algorithm that inverts functions $f \colon [N] \to [N]$ using $O(n^{1/2+\epsilon'}) =$

$O((N \log N)^{1/2+\epsilon'}) = O(N^{1/2+\epsilon'} \log N)$ bits of advice and $O(n^{1/2+\epsilon'} \log N) = O((N \log N)^{1/2+\epsilon'} \log N)$ online queries. Then, for any $\epsilon > \epsilon'$, the advice usage and number of online queries is $O(N^{1/2+\epsilon})$. □

Notice that while the hypothesis of Corollary 7 considers a lower bound against strongly non-adaptive inversion algorithms, this only *strengthens* the statement. This is true because proving a lower bound against adaptive inversion algorithms implies a lower bound against strongly non-adaptive algorithms as well.

If we instantiate Corollary 7 with $\epsilon = 1/6$, we find that ruling out function-inversion algorithms using $S = T = o(N^{2/3})$, even against the restricted class of strongly non-adaptive algorithms, would give an explicit operator that does not have common-bits circuits of width w and degree d satisfying $w = d = o(n^{2/3-\delta})$, for any $\delta > 0$.

Proving such a lower bound on common-bits circuits is *not* strong enough to yield a lower bound against linear-size log-depth circuits via Valiant's method (Theorem 5). However, this lower bound *would* improve the best known lower bound against circuits in the common-bits model. The best known bound, due to Pudlák, Rödl, and Sgall, gives $d = \Omega(\frac{n}{w} \cdot \log(\frac{n}{w}))$, for a common-bits circuit of width w and degree d [65]. In particular, they construct an explicit operator that does not have common-bits circuits satisfying $w = d = \widetilde{O}(n^{1/2})$. By Corollary 7, ruling out function-inversion algorithms with $S = T = \widetilde{O}(N^{1/2+\epsilon})$, for any $\epsilon > 0$, would thus improve the best lower bounds on common-bits circuits.

2.3 Consequences for other succinct data-structure problems

Theorem 3 and Lemma 4 together imply that proving strong lower bounds for *any* systematic data-structure problem—not only for the function-inversion problem—will be challenging. To explain how this barrier applies to a completely different data-structure problem, we recall the systematic variant of the standard data-structure problem of *polynomial evaluation with preprocessing* [59]. We give an informal description of the problem, and the transformation into a formal systematic data-structure problem (as in Section 2.1) is straightforward.

The problem of polynomial evaluation with preprocessing is parameterized by an integer $N \in \mathbb{Z}^{>0}$ and a finite field $\mathbb{F}$ of size $\Theta(N)$. The input data is a polynomial $p \in \mathbb{F}[X]$ of degree at most $N-1$, represented as its vector of coefficients $\bar{c} = (c_0, c_1, \ldots, c_{N-1}) \in \mathbb{F}^N$. The preprocessing algorithm reads this input (the entire polynomial p) and produces a preprocessed S-bit string st. In a subsequent online phase, the query algorithm takes as input a point $x_0 \in \mathbb{F}$, and must output the evaluation $p(x_0) \in \mathbb{F}$ of the polynomial p at point x_0. To produce its answer, the query algorithm may read the entire preprocessed string st, query at most T coordinates of the coefficient vector $\bar{c}$, and perform an unlimited amount of computation.

For what choices of space usage S and query complexity T does there exist a systematic data structure for polynomial evaluation with preprocessing?

The two naïve approaches to solving this problem are:

1. Have the preprocessing algorithm store in the string st the evaluation of the polynomial p on every point in the field $\mathbb{F}$, using $S = \Omega(N)$ space.
2. Have the online-phase algorithm read the entire coefficient vector $\bar{c}$, using $T = \Omega(N)$ queries, and then evaluate $p(x_0) = \sum_i c_i x_0 \in \mathbb{F}$ directly.

These solutions both have $S + T = \widetilde{\Omega}(N)$.

It seems very difficult to construct an algorithm that simultaneously uses a data structure of size $S = N^\delta$ and query complexity $T = N^\delta$, for some $\delta < 1$. And yet, the best lower bound we have for this problem, implied by a bound of Gál and Miltersen [32], is of the form $ST = \widetilde{\Omega}(N)$. A variant of Corollary 7 implies that proving stronger lower bounds for this problem—or proving any lower bound better than $ST = \widetilde{\Omega}(N)$ for *any* systematic or succinct data-structure problem, for that matter—will also imply new lower bounds in Valiant's common-bits model. Proving even a stronger lower bound could, via Theorem 3, imply a lower bound against linear-size log-depth fan-in-two circuits.

3 Breaking PRGs is as hard as inverting injective functions

Many cryptanalytic applications of Hellman tables only require inverting *injective* functions. That is given a *injective* function $f : [N] \to [M]$ and a point $y \in [M]$, find a value $x \in [N]$ such that $f(x) = y$, if one exists.

For example, consider the classic application of Hellman tables to plaintext attacks on block ciphers: Let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, where k is the key size and n is the block size. If we define $f_E : \{0,1\}^k \to \{0,1\}^n$ such that $f_E(x) = E(x, m_0)$ for some fixed plaintext m_0, then an algorithm with preprocessing for the function f_E essentially gives a known-plaintext attack on the block cipher E. We can (heuristically) expect the resulting function f_E to behave similar to a random function, and therefore be injective only beyond the birthday bound $k \gtrapprox 2n$. However, even for shorter keys, we can reduce a known-plaintext attack to the problem of inverting an injective function by considering the encryption of multiple known plaintexts m_0, m_1, m_2. For example, if $k = n$, then we expect $f_{E^{\times 3}} \colon \{0,1\}^n \to \{0,1\}^{3n}$, defined as $f_{E^{\times 3}}(x) = E(x, m_0) \| E(x, m_1) \| E(x, m_2)$, to have no collisions.

A function-inversion algorithm can invert an injective function $f \colon [N] \to [M]$ without taking any advantage of the fact that it is injective, so Hellman's $S^2 T = \widetilde{O}(N^2)$ upper bound for function inversion [48] applies in this setting as well. However, the fact that for the case of random permutations (i.e., an injective function $f : [N] \to [N]$), Hellman's algorithm gives a significantly better upper bound of $ST = \widetilde{O}(N)$, gives hope that a similar improvement—or at least some improvement—is possible for injective length-increasing functions.

To the best of our knowledge, the injective variant of the function-inversion problem has not been studied directly so far, even though it is a special case with wide cryptanalytic applications. As a first step, we connect the injective inversion problem to the problem of breaking pseudorandom generators (PRGs) with preprocessing [2,20,23,25,26]. In that problem, we model a "black-box" PRG as an oracle $G\colon [N] \to [M]$, with $N < M$. A PRG distinguisher with preprocessing first makes arbitrarily many queries to G and outputs an S-bit advice string. In the online phase, the distinguisher can then use its advice string, along with T queries to G, to distinguish whether a given sample $y \in [M]$ has been drawn from the distribution $\{G(x) \mid x \xleftarrow{\text{R}} [N]\}$ or the distribution $\{y \mid y \xleftarrow{\text{R}} [M]\}$.

In their work, De, Trevisan, and Tulsiani [23] give a distinguisher with $S = O(\epsilon^2 N)$ and $T = \widetilde{O}(1)$ that achieves a distinguishing advantage $\epsilon \leq 1/\sqrt{N}$. They ask whether it is possible to realize the trade-off $ST = \widetilde{O}(\epsilon^2 N)$ for other parameter settings as well. The following theorem shows that a PRG distinguisher that achieves constant distinguishing advantage at points on this trade-off (e.g., $\epsilon = 1/100$, $S = N^{1/4}$, and $T = N^{3/4}$) would imply a better-than-Hellman algorithm for inverting injective functions.

Theorem 8. *Suppose that there is a black-box PRG distinguisher that uses S bits of advice, makes T online queries to a PRG $G\colon [N] \to [M]$, and achieves distinguishing advantage ϵ. Then there exists a black-box algorithm that inverts any injective function $f\colon [N] \to [M]$ using $\widetilde{O}(\epsilon^{-2} S)$ bits of advice and $\widetilde{O}(\epsilon^{-2} T)$ online queries, and that inverts f with probability $1 - 1/\log N$ (over the algorithm's randomness).*

Furthermore, if the preprocessing and online phase algorithms have access to a common random oracle, the online phase also runs in time $\widetilde{O}(T)$.

Remark 9 (Relation to Goldreich-Levin). A classic line of results [38,39,57,78] shows how to use any injective one-way function $f : [N] \to [M]$ to construct an efficient PRG $G_f : [N^2] \to [2N^2]$ which makes black-box use of f. The proof uses the Goldreich-Levin theorem [39] to show that any efficient distinguisher for G_f yields an inversion algorithm for f. (Consult Goldreich's textbook [37, Section 3.5] for the details.) It is not clear to us whether a non-uniform generalization of these classic results directly implies Theorem 8. The problem is that the domain of the PRG G_f has size N^2, whereas the domain of the original function f has size N. Since we are interested in the exact exponent of the advice and time usage of function-inversion algorithms (i.e., $S = N^{3/4}$ versus $S = N^{1/2}$), we are sensitive to this polynomial expansion in the domain size. For example, say that we were able to construct a black-box PRG distinguisher that achieves $S = T = \widetilde{O}(\sqrt{N})$. Applying the classic reduction directly to G_f would only imply the existence of an inverter for the function f that uses the trivial advice and time complexity $S = T = \widetilde{O}(\sqrt{N^2}) = \widetilde{O}(N)$. In contrast, Theorem 8 implies that an $S = T = \widetilde{O}(\sqrt{N})$ distinguisher yields an $S = T = \widetilde{O}(\sqrt{N})$ inverter.

Proof idea for Theorem 8. Given a distinguisher for any length-increasing generator $G\colon [N] \to [M]$, we construct an inversion algorithm for injective functions $f\colon [N] \to [M]$ in two steps. First, for each $i \in [n]$, we construct a bit-recovery algorithm $\mathcal{B}_i$ that, given $f(x)$, achieves a non-trivial advantage in recovering the ith bit of x. We then use the algorithms $(\mathcal{B}_1, \ldots, \mathcal{B}_n)$ to construct an inversion algorithm $\mathcal{I}$ that, given $f(x)$, recovers the full preimage x with good probability.

To give the intuition behind the bit-recovery algorithm $\mathcal{B}_i$: Given a function $f\colon [N] \to [M]$ to invert, we construct a function $G_i\colon [N] \to [M]$ such that a point $y = f(x)$ is in the image of G_i if and only if the ith bit of x is 1. Then, we can apply the PRG distinguisher to G_i and recover the ith bit of y's preimage.

This simple algorithm does not quite work when the PRG distinguisher has small distinguishing advantage ϵ, since the distinguisher may fail on the point y. To fix this, we give $\mathcal{B}_i$ access to two random permutations $\pi\colon [N] \to [N]$ and $\sigma\colon [M] \to [M]$ that allow $\mathcal{B}_i$ to essentially randomize the point it gives as input to the PRG distinguisher.

We then can run $\mathcal{B}_i$ many times with different random permutations and then take the majority vote of the outputs of these runs. This majority vote will yield the ith bit of the x with high probability. To complete the construction, we instantiate the permutations π and σ using correlated randomness between the preprocessing and online algorithms. The full description of the construction appears in Appendix B. □

4 From cryptanalysis to new communication protocols

Communication complexity [55, 76] quantifies the number of bits that a set of players need to communicate amongst themselves in order to compute a function on an input that is split between the players. One of the major open problems in communication complexity is to obtain a non-trivial lower bound for some problem for a super-poly-logarithmic number of players. Such a bound would in turn lead to a breakthrough circuit lower bound for the complexity class ACC^0 [6, 49, 80].

In this section, we develop connections between the function-inversion problem and the multiparty pointer-jumping problem in the number-on-the-forehead (NOF) model of communication complexity [16]. By combining these new connections with the classic cycle-walking algorithm for permutation inversion, we obtain the best known NOF protocols for the permutation variant of the pointer-jumping problem. Since pointer jumping is a candidate hard problem in the k-party NOF setting, understanding the exact communication complexity of pointer jumping for a super-poly-logarithmic number of players is an important step towards the eventual goal of proving circuit lower bounds [10, 11, 14, 22, 58, 65, 72].

4.1 Multiparty pointer-jumping in the NOF model

A classical problem in the NOF model is the *pointer-jumping* problem. We describe the *permutation* variant of the problem, and then discuss the general case. In the pointer-jumping problem $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$, there are k computationally-unbounded players, denoted $P_0, P_1, \dots, P_{k-1}$, and each has an input "written on her forehead." The first player P_0 has a point $x \in [N]$ written on her forehead, the last player P_{k-1} has a Boolean mapping $\beta : [N] \to \{0, 1\}$ written on her forehead, and each remaining player P_i, for $i = 1, \dots, k-2$, has a permutation $\pi_i : [N] \to [N]$ written on her forehead. Each player can see all $k-1$ inputs except the one written on her own forehead. The goal of the players is to compute the value $\beta \circ \pi_{k-2} \circ \cdots \circ \pi_1(x)$, which loosely corresponds to "following a trail of pointers" defined by the permutations, starting from x (Figure 2). The players can communicate by writing messages on a public blackboard. The communication complexity of a protocol is the total number of bits written on the blackboard for a worst-case input.

A *one-way* protocol is a protocol in which each player writes a single message on the blackboard in the fixed order $P_0, \ldots, P_{k-1}$, and the last player's message must be the output. The one-way communication complexity of a function f, denoted $\mathsf{CC}^1(f)$, is the minimum communication complexity of all one-way protocols that successfully compute f. Without the "one-way" restriction, there are protocols for $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$ that require only $O(\log N)$ bits of communication.

Known bounds. The best upper bound for $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$ is due to Pudlák et al. [65], who showed that $\mathsf{CC}^1(\mathsf{MPJ}^{\mathsf{perm}}_{N,k}) = O(N \log\log N / \log N)$. More recently, Brody and Sanchez [14] showed that this upper bound applies to the more general pointer-jumping problem, in which we replace the permutations $\pi_1, \ldots, \pi_{k-2}$ with arbitrary functions. In this general case, Wigderson [75] proved an $\Omega(\sqrt{N})$ lower bound for $k = 3$ players (see also [3]), and Viola and Wigderson [72] proved an $\widetilde{\Omega}(N^{\frac{1}{k-1}})$ lower bound for $k \geq 3$ players.

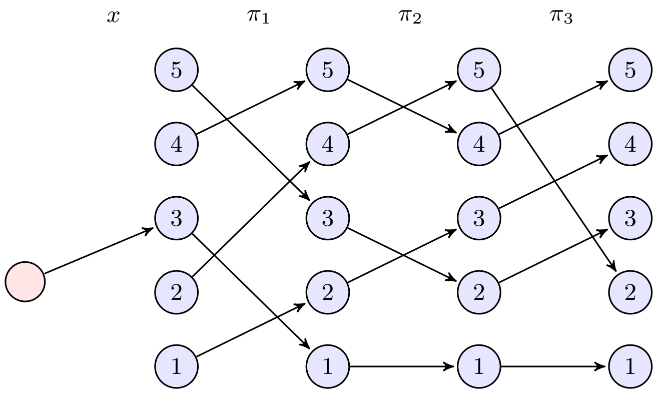

Fig. 2: A pointer-jumping instance for $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{k=4,N=5}$ with $\pi_1 = (1\,2\,4\,5\,3)$, $\pi_2 = (2\,3)(4\,5)$, $\pi_3 = (2\,3\,4\,5)$ and $x = 2$. Lemma 11 reduces this instance to inverting the permutation $\pi_1^{-1}\pi_2^{-1}\pi_3^{-1} = (1\,3\,5\,4)$ on the point $x = 2$.

4.2 A new communication protocol from permutation inversion

We obtain the best known communication protocol for the *permutation* variant of the pointer-jumping game on parameter N for $k = \omega(\log N / \log\log N)$ players. Our result improves the previously best known upper bound of $\widetilde{O}(N)$ to $\widetilde{O}(N/k+$

$\sqrt{N}$). Extending our upper bound to the *general* multiparty pointer-jumping problem remains an open problem, which we discuss in Remark 13.

On the lower-bound side, this connection suggests a path to prove lower bounds against *partially adaptive* permutation-inversion algorithms, as in Definition 2. In contrast, the techniques of Section 2 can only prove lower bounds against strongly non-adaptive algorithms.

In this section, we prove the following new upper bound on $\mathsf{CC}^1(\mathsf{MPJ}^{\mathsf{perm}}_{N,k})$:

Theorem 10. $\mathsf{CC}^1(\mathsf{MPJ}^{\mathsf{perm}}_{N,k}) \leq O\left((N/k + \sqrt{N})\log N\right)$.

To prove Theorem 10, as we do later in this section, we use the integer-valued version the pointer-jumping problem, commonly denoted $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$. In this version, the last player P_{k-1} holds a permutation $\pi_{k-1} : [N] \to [N]$, instead of a boolean mapping, so the output of the problem is a value in $[N]$. The following technical lemma, which we prove in Appendix A, shows that the boolean-valued version of the pointer-jumping problem has communication complexity that is not much larger than that of the integer-valued pointer-jumping problem.

Lemma 11. $\mathsf{CC}^1(\mathsf{MPJ}^{\mathsf{perm}}_{N,k}) \leq \mathsf{CC}^1(\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}) + \lceil \log N \rceil$.

Then, our main lemma technical uses an arbitrary permutation-inversion algorithm with preprocessing to solve the integer-valued problem $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$:

Lemma 12. *If there exists a $(k-2)$-round adaptive algorithm for inverting permutations $\pi : [N] \to [N]$ that uses advice S and time T, then*

$$\mathsf{CC}^1(\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}) \leq S + T\lceil \log N \rceil .$$

Proof. Let $(\mathcal{A}_0, \mathcal{A}_1)$ be a $(k-2)$-round adaptive algorithm for inverting permutations with preprocessing. We give a protocol for $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$.

- Player P_0 runs the preprocessing algorithm $\mathcal{A}_0$ on the permutation $\pi_1^{-1} \circ \cdots \circ \pi_{k-1}^{-1}$ and writes the advice string on the blackboard.
- Player P_1 runs the online inversion algorithm $\mathcal{A}_1$ on the input x (written on player P_0's forehead) using the advice string that has been written on the blackboard, to produce the first round of queries $q_{1,1}, \ldots, q_{1,t_1}$. For each query $q_{1,\ell}$, she computes the partial reply $p_{1,\ell} = \pi_2^{-1}(\ldots(\pi_{k-1}^{-1}(q_{1,\ell}))\ldots)$ and writes it on the blackboard.
- Player P_i, for $i \in \{2, \ldots, k-2\}$, reads the partial replies $p_{i-1,1}, \ldots, p_{i-1,t_{i-1}}$ written by the previous player, computes the (complete) query replies $r_{i-1,1}, \ldots, r_{i-1,t_{i-1}}$ by computing $r_{i-1,\ell} = \pi_1^{-1}(\ldots(\pi_{i-1}^{-1}(p_{i-1,\ell}))\ldots)$. Player P_i then runs (in her head) the first $i-1$ rounds of the online inversion algorithm on input x, using the advice string and the replies to the first $i-1$ rounds of queries, all of which, she can compute using the partial replies written on the blackboard. Player P_i then produces the i^{th} round of queries, on which, similarly to Player P_1, she computes the partial replies and writes them on the blackboard.

- Player P_{k-1} completes the evaluation of round $k-2$ of the queries by evaluating the remaining permutations $\pi_1^{-1} \circ \cdots \circ \pi_{k-2}^{-1}$ on the partial replies written by P_{k-2}. Player P_{k-1} then runs in her head all $k-2$ rounds of the online inversion algorithm and writes the output on the blackboard.

By definition, the output y of the algorithm satisfies $\pi_1^{-1} \circ \cdots \circ \pi_{k-1}^{-1}(y) = x$. Since all π_i are permutations, it must hold $\pi_{k-1} \circ \cdots \circ \pi_1(x) = y$ and so y is the correct output for $\widehat{\mathsf{MPJ}}_{N,k}^{\mathsf{perm}}$.

The communication consists of the advice string written by Player P_0 and a partial reply for each query, giving a total of $S + T\lceil \log N \rceil$. (The last player writes the $\lceil \log N \rceil$-bit output, but does not need to write the response to the T-th query). □

Proof of Theorem 10. To prove Theorem 10, we instantiate Lemma 12 using Hellman's cycle-walking algorithm [48], which we recall in Appendix C. The algorithm inverts permutations using T queries and S bits of advice, for every choice of S and T such that $ST \geq 2N\lceil \log N + 1 \rceil$. Furthermore the algorithm is T-round adaptive. Specifically, for $k \leq \sqrt{N} + 2$, using Hellman's algorithm with $T = k-2$ and $S = \lceil (2N \log N)/T \rceil$ gives a protocol with communication $O((N/k)\log N)$. For $k > \sqrt{N} + 2$, we use Hellman's algorithm with $T = \sqrt{N}$ and $S = 2\sqrt{N}(\lceil \log N \rceil + 1)$ to find that $\mathsf{CC}^1(\widehat{\mathsf{MPJ}}_{N,k}^{\mathsf{perm}}) \leq O(\sqrt{N}\log N)$. Then, applying Lemma 11 lets us conclude that $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\mathsf{perm}}) \leq O(\sqrt{N}\log N)$. □

Remark 13 (The function case). We might hope to show that a good function-inversion algorithm, such as that of Fiat and Naor [30], implies a good protocol for the *general* multiparty pointer-jumping problem, in which each player i has an arbitrary function f_i (which may not be a permutation) written on her forehead. We do not know how to prove such a result. The problem is that the reduction of Lemma 12 requires that the composition $f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_{k-1}^{-1}$ is a function, and this is not true in the general case. (In contrast, when $f_1, \ldots, f_k$ are all permutations it holds that $f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_{k-1}^{-1}$ is a permutation.) Since several upper bounds for the permutation variant of the pointer-jumping problem [11, 22, 65] have led to subsequent upper bounds for the unrestricted case [11, 14], there is still hope to generalize the result.

5 From cryptanalysis to data-structures

In this section, we show how to apply the Fiat-Naor algorithm for function inversion [30] to obtain the best known data structure for the *systematic substring-search* problem [24, 31, 32, 42, 43], in a wide range of parameter regimes. As a consequence of this connection, we show that the open problem of improving the known lower bounds on function inversion is equivalent to the open problem in the data-structure literature of whether it is possible to improve the known lower bounds for systematic substring search.

In the systematic substring-search problem, we are given a bitstring of length N ("the text") and a bitstring of length $P \ll N$ ("the pattern"). If the pattern appears in the text, we must output an index $i \in [N]$ into the text at which the pattern begins. We take the pattern length to be $P = \Theta(\log N)$.

An algorithm for systematic substring search is a two-part algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. The preprocessing algorithm $\mathcal{A}_0$ takes as input only the text, may perform arbitrary computation on it, and then outputs an S-bit "index" into the text. The online algorithm $\mathcal{A}_1$ takes as input the index and the pattern, queries T bits of the text, and then outputs the location of pattern in the text, if one exists.

By applying the Fiat-Naor function inversion algorithm [30], we obtain the *best known algorithm* for systematic substring search on texts of length N when using an index of size $O(N^\epsilon)$ bits, for any $\epsilon < 1$. Gál and Miltersen [32] asked for a strong lower bound against search algorithms using an $O(N/\operatorname{polylog} N)$-bit index, and we answer this question by giving an upper bound that beats their hypothetical lower bound. This connection also gives evidence that finding a faster algorithm for systematic substring search will require a cryptanalytic breakthrough.

Known lower bounds. Demaine and López-Ortiz [24] prove that on texts of length N with pattern length $P = \Theta(\log N)$, any algorithm that uses an S-bit index and makes $T = o(P^2/\log P)$ queries in the online phase must satisfy $ST = \Omega(N \log N)$. Golynski [42,43] gives a stronger version of this bound that applies even for larger $T = o(\sqrt{N}/\log N)$. Gál and Miltersen prove a slightly weaker bound but that holds for all values of T. They show that for certain pattern lengths $P = \Theta(\log N)$, and any choice of T, any algorithm must satisfy $ST = \Omega(N/\log N)$.[1]

The main technical result of this section is the following theorem, which we prove in Appendix D.

Theorem 14. *For any integer $N \in \mathbb{Z}^{>0}$ and integral constant $c > 2$, if there is an algorithm for systematic substring search on texts of length $cN \cdot \lceil \log N \rceil$ with pattern length $c \cdot \lceil \log N \rceil$ that uses an S-bit index and reads T bits of the text in its online phase, then there is a black-box algorithm for inverting functions $f \colon [N] \to [N]$ that uses S bits of advice and makes T online queries.*

For any integer $N \in \mathbb{Z}^{>0}$, if there is a black-box algorithm for inverting functions $f \colon [2N] \to [2N]$ that uses S bits of advice and T queries, then, for any integral constant $c > 1$, there is an algorithm for systematic substring search on texts of length N with pattern length $c \cdot \lceil \log N \rceil$ that uses an $\widetilde{O}(S)$-bit index and reads $\widetilde{O}(T)$ bits of the text in its online phase.

Remark 15. It is possible to make the preprocessing time $\widetilde{O}(N)$ by allowing the algorithm to fail with probability $O(1/N)$ over the randomness of the preprocessing phase. Similarly, the online running time (in addition to the query complexity) is $\widetilde{O}(T)$.

[1] Gál and Miltersen in fact prove their lower bound against algorithms that solve the *decision* version of the problem, rather than the *search* version that we describe here. Using an argument similar to that of Theorem 8, which treats the case of black-box PRG distinguishers, we can show that these problems are equivalent up to log factors when we demand constant success probability.

Proof idea. The full proof appears in Appendix D. In the first part, we must use a substring-search algorithm to invert a function $f\colon [N] \to [N]$. The idea, formalized in Lemma 20, is to construct a text τ of length $\Theta(N \log N)$ by writing out the evaluation of f at all points in its domain, in order, with a few extra bits added as delimiters. To invert a point $y \in [N]$, we use the substring search algorithm to find the location at which y appears in the text τ. This location immediately yields a preimage of y under f. Demaine and López-Ortiz [24] use a similar—but more sophisticated encoding—on the way to proving a data-structure lower bound for systematic substring search. Their encoding maps a function $f\colon [N] \to [N]$ into a string of length $(1 + o(1))N \log N$, while ours maps f into a string of length $3N \log N$.

In the second part, we must use a function-inversion algorithm to solve substring search on a text τ of length N with pattern length $P = c \cdot \lceil \log N \rceil$, for some constant $c > 1$. To do so, we define in Lemma 22 a function $f'\colon [N] \to [N^c]$ such that $f'(i)$ is equal to the length-P substring that starts from the ith bit of the text τ. Given a pattern string $\sigma = \{0,1\}^P$, finding the inverse of y under f' is enough to locate the position of the pattern string σ in the text τ. The only remaining challenge is that f' is length-increasing, rather than length-preserving. In Lemma 21, we use universal hashing to reduce the problem of inverting length-increasing functions to the problem of inverting length-preserving functions, which completes the proof. □

We now apply Theorem 14 to construct a new algorithm for systematic substring search that resolves an open question of Gál and Miltersen. In their 2007 paper, Gál and Miltersen say that "it would be nice to prove a lower bound of, say, the form," $T < N/\mathrm{polylog}\, N \Rightarrow S > N/\mathrm{polylog}\, N$ (using our notation) for systematic substring search [32]. Goyal and Saks [44] use an elegant argument to show that the specific technique of Gál and Miltersen cannot prove this lower bound. As a corollary of Theorem 14, we construct an algorithm for substring search that beats the hypothetical lower bound.

Corollary 16. *For any integral constant $c > 1$ there is an algorithm for systematic substring search on texts of length N with pattern length $c \cdot \lceil \log N \rceil$, that uses an S-bit index, reads T bits of the text in its online phase, and achieves the trade-off $S^3 T = \widetilde{O}(N^3)$.*

Proof. Theorem 14 shows that systematic substring search on strings of length N with pattern length $\Theta(\log N)$ reduces to the problem of inverting arbitrary functions $f\colon [N] \to [N]$. The inversion algorithm of Fiat and Naor [30] inverts such functions f achieving the desired complexity bounds. □

In particular, we get an algorithm that solves systematic substring search using an index size and time satisfying $S = T = \widetilde{O}(N^{3/4})$, for strings of length N and patterns of length $\Theta(\log N)$. Furthermore, this connection, along with the results of Section 2.2, shows that improving on the $ST = \widetilde{\Omega}(N)$ bound of Gál and Miltersen will require advances in techniques for proving lower bounds on the power of depth-two circuits.

6 Discussion and future directions

In this final section, we discuss a few directions for future work.

6.1 Which lower-bound techniques can work?

In Section 2, we showed that improving Yao's lower bound on function-inversion algorithms requires new circuit lower bounds in the common-bits model. What potential approaches do we have to prove such a lower bound?

Function inversion and Yao's "box problem." Yao's "box problem" [62, 79] is a preprocessing problem that is closely related to the function-inversion problem. In the box problem, we are given oracle access to a function $f\colon [N] \to \{0,1\}$. First, we get to look at all of f and write down an S-bit advice string st_f. Later on, we are given our advice string st_f and a point $x \in [N]$. We may then make T queries to f, provided that we do not query $f(x)$, and we must then output a value $y \in \{0,1\}$ such that $y = f(x)$.

The box problem is in some sense the dual of the function-inversion problem: we are given an f-oracle and we must compute f in the *forward* direction, rather than in the *inverse* direction. The same $ST = \widetilde{\Omega}(N)$ lower bound applies to both problems [79]. However, in contrast to the inversion problem, for which we suspect that good parallel (i.e., non-adaptive) algorithms do not exist, the natural algorithm for the box problem is *already* non-adaptive and achieves $ST = O(N \log N)$.[2]

Puzzlingly, the two main techniques for proving time-space lower bounds *do not distinguish* between the function-inversion problem and Yao's box problem. In particular, the known lower bounds use compression [23, 25, 35, 79] or bit-fixing [19, 20, 67]. Both techniques essentially look at the information that the oracle queries and their replies give on the pair $(x, f(x))$ induced by the challenge, regardless of whether the actual challenge is x, and the algorithm has to find $f(x)$ (as in the case of Yao's box problem), or the challenge is $y = f(x)$, and the algorithm has to find $x = f^{-1}(y)$ (as in the case of the inversion problem).

Since there is an $ST = \widetilde{O}(N)$ *upper bound* for Yao's box problem, then any method that proves a lower bound better than $ST = \Omega(N)$ for function inversion must not apply to the box problem. Therefore, a "sanity check" for any improved lower bound for the function-inversion problem is to verify that the same proof technique does not apply to Yao's box problem.

Strong-multiscale-entropy. Drucker [27] shows, at the very least, that improving lower bounds in the common-bits model will require new types of arguments. In particular, Jukna [52, Chapter 13], generalizing earlier arguments of Cherukhin [17] defined the "strong multiscale entropy" (SME) property of

[2] Divide $[N]$ into disjoint blocks of (at most) $T + 1$ points each. For each block, store the sum of the values of the function over all points in the block. In the online phase, query all the other points in the block given by the challenge point, and use the stored sum to recover the value of the function over the given challenge point.

Boolean operators. Jukna proved that an operator on n bits with the SME property cannot be computed by common-bits circuits of width $o(n^{1/2})$ and degree $o(n^{1/2})$. (These results are actually phrased in terms of the wire complexity of depth-two circuits with arbitrary gates, but the implications to the common-bits model are straightforward.)

Strengthening Jukna's lower bound on the circuit complexity of SME operators appeared to be one promising direction for progress on lower bounds. Thwarting this hope, Drucker constructs an explicit operator with the SME property that has circuits in the common-bits model of width $O(n^{1/2})$ and degree $O(n^{1/2})$. Thus, SME-type arguments alone are not strong enough to prove that an operator cannot be computed by circuits of width $O(n^{1/2+\epsilon})$ and degree $O(n^{1/2+\epsilon})$ for $\epsilon > 0$.

6.2 One-to-one functions

Prompted by the fact that many cryptanalytic applications of function inversion only require inverting injective function, we initiated in Section 3 the study of injective function inversion. Though we take the first step by connecting this problem to the problem of distinguishing PRGs, the basic question remains: is it easier to invert a random injective function $f\colon [N] \to [M]$, for $N \ll M$, than it is to invert a random length-preserving function $f\colon [N] \to [N]$? A better-than-Hellman attack against injective functions would be remarkable. Or, can we prove that inverting injective functions is as hard as inverting random functions?

6.3 Barriers for upper bounds

Is there a barrier to getting an $S = T = o(N^{2/3})$ algorithm for function inversion? Barkan, Biham, and Shamir [5] prove a lower bound against a certain *restricted* class of Hellman-like algorithms, which suggests that better algorithms must use new techniques. It would be satisfying to show at least that improving Hellman's upper bound would result in a dramatic algorithmic improvement for a well-studied problem in another domain.

Acknowledgments. We would like to thank Dan Boneh for encouraging us to investigate whether Hellman's method can be improved and for his continued advice as we undertook this project. Iftach Haitner gave us meaningful guidance on our research process early on and, along with Ronen Shaltiel, suggested many possible approaches towards proving new lower bounds. Joshua Brakensiek, Joshua Brody, Clément Canonne, Andrew Drucker, Michael Kim, Peter Bro Miltersen, Ilya Mironov, Omer Reingold, Avishay Tal, Li-Yang Tan, and David Wu made a number of suggestions that improved the presentation of our results. Finally, we would like to thank the anonymous TCC reviewers for their many constructive comments. This work was supported by CISPA, DARPA, NSF, ONR, and the Simons Foundation.

References

1. Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.: Beyond Hellman's time-memory trade-offs with applications to proofs of space. In: ASIACRYPT (2017). https://doi.org/10.1007/978-3-319-70697-9_13
2. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple construction of almost k-wise independent random variables. Random Struct. Algorithms **3**(3), 289–304 (1992). https://doi.org/10.1002/rsa.3240030308
3. Babai, L., Hayes, T.P., Kimmel, P.G.: The cost of the missing bit: Communication complexity with help. Combinatorica **21**(4), 455–488 (2001). https://doi.org/10.1007/s004930100009
4. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms **7**(4), 52:1–52:27 (2011). https://doi.org/10.1145/2000807.2000820
5. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: CRYPTO (2006). https://doi.org/10.1007/11818175_1
6. Beigel, R., Tarui, J.: On ACC. Computational Complexity **4**, 350–366 (1994). https://doi.org/10.1007/BF01263423
7. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: ASIACRYPT (2000). https://doi.org/10.1007/3-540-44448-3_1
8. Biryukov, A., Shamir, A., Wagner, D.A.: Real time cryptanalysis of A5/1 on a PC. In: FSE (2000). https://doi.org/10.1007/3-540-44706-7_1
9. Boyle, E., Naor, M.: Is there an oblivious RAM lower bound? In: ITCS (2016). https://doi.org/10.1145/2840728.2840761
10. Brody, J.: The maximum communication complexity of multi-party pointer jumping. In: CCC (2009). https://doi.org/10.1109/CCC.2009.30
11. Brody, J., Chakrabarti, A.: Sublinear communication protocols for multi-party pointer jumping and a related lower bound. In: STACS (2008). https://doi.org/10.4230/LIPIcs.STACS.2008.1341
12. Brody, J., Dziembowski, S., Faust, S., Pietrzak, K.: Position-based cryptography and multiparty communication complexity. In: TCC (2017). https://doi.org/10.1007/978-3-319-70500-2_3
13. Brody, J., Larsen, K.G.: Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. Theory of Computing **11**(19), 471–489 (2015). https://doi.org/10.4086/toc.2015.v011a019
14. Brody, J., Sanchez, M.: Dependent random graphs and multi-party pointer jumping. In: APPROX/RANDOM (2015). https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.606
15. Carter, L., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences **18**(2), 143–154 (1979). https://doi.org/10.1016/0022-0000(79)90044-8
16. Chandra, A.K., Furst, M.L., Lipton, R.J.: Multi-party protocols. In: STOC (1983). https://doi.org/10.1145/800061.808737
17. Cherukhin, D.Y.: Lower bounds for the complexity of boolean circuits of finite depth with arbitrary elements. Discrete Mathematics and Applications **23**(4), 39–47 (2011). https://doi.org/10.1515/dma.2011.031
18. Clark, D.R., Munro, J.I.: Efficient suffix trees on secondary storage. In: SODA (1996)
19. Coretti, S., Dodis, Y., Guo, S.: Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In: CRYPTO (2018). https://doi.org/10.1007/978-3-319-96884-1_23

20. Coretti, S., Dodis, Y., Guo, S., Steinberger, J.P.: Random oracles and non-uniformity. In: EUROCRYPT (2018). https://doi.org/10.1007/978-3-319-78381-9_9
21. Coron, J., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: CRYPTO (2008). https://doi.org/10.1007/978-3-540-85174-5_1
22. Damm, C., Jukna, S., Sgall, J.: Some bounds on multiparty communication complexity of pointer jumping. Computational Complexity **7**(2), 109–127 (1998). https://doi.org/10.1007/PL00001595
23. De, A., Trevisan, L., Tulsiani, M.: Time space tradeoffs for attacks against one-way functions and prgs. In: CRYPTO (2010). https://doi.org/10.1007/978-3-642-14623-7_35
24. Demaine, E.D., López-Ortiz, A.: A linear lower bound on index size for text retrieval. Journal of Algorithms **48**(1), 2–15 (2003). https://doi.org/10.1016/S0196-6774(03)00043-9
25. Dodis, Y., Guo, S., Katz, J.: Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In: EUROCRYPT (2017). https://doi.org/10.1007/978-3-319-56614-6_16
26. Dodis, Y., Steinberger, J.P.: Message authentication codes from unpredictable block ciphers. In: CRYPTO (2009). https://doi.org/10.1007/978-3-642-03356-8_16
27. Drucker, A.: Limitations of lower-bound methods for the wire complexity of boolean operators. In: CCC (2012). https://doi.org/10.1109/CCC.2012.39
28. Dvir, Z., Golovnev, A., Weinstein, O.: Static data structure lower bounds imply rigidity. In: STOC (2019). https://doi.org/10.1145/3313276.3316348
29. Fiat, A., Naor, M.: Rigorous time/space tradeoffs for inverting functions. In: STOC (1991). https://doi.org/10.1145/103418.103473
30. Fiat, A., Naor, M.: Rigorous time/space trade-offs for inverting functions. SIAM Journal on Computing **29**(3), 790–803 (1999). https://doi.org/10.1137/S0097539795280512
31. Gál, A., Miltersen, P.B.: The cell probe complexity of succinct data structures. In: ICALP (2003). https://doi.org/10.1007/3-540-45061-0_28
32. Gál, A., Miltersen, P.B.: The cell probe complexity of succinct data structures. Theoretical Computer Science **379**(3), 405–417 (2007). https://doi.org/10.1016/j.tcs.2007.02.047
33. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms **2**(4), 510–534 (2006). https://doi.org/10.1145/1198513.1198516
34. Gennaro, R., Gertner, Y., Katz, J., Trevisan, L.: Bounds on the efficiency of generic cryptographic constructions. SIAM Journal on Computing **35**(1), 217–246 (2005). https://doi.org/10.1137/S0097539704443276
35. Gennaro, R., Trevisan, L.: Lower bounds on the efficiency of generic cryptographic constructions. In: FOCS (2000). https://doi.org/10.1109/SFCS.2000.892119
36. Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs. In: STOC (1987). https://doi.org/10.1145/28395.28416
37. Goldreich, O.: Foundations of Cryptography, vol. 1. Cambridge University Press (2006)
38. Goldreich, O., Krawczyk, H., Luby, M.: On the existence of pseudorandom generators. SIAM J. Comput. **22**(6), 1163–1175 (1993). https://doi.org/10.1137/0222069
39. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: STOC (1989). https://doi.org/10.1145/73007.73010
40. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. Journal of the ACM **43**(3), 431–473 (1996). https://doi.org/10.1145/233551.233553

41. Golovnev, A., Guo, S., Horel, T., Park, S., Vaikuntanathan, V.: 3SUM with preprocessing: Algorithms, lower bounds and cryptographic applications. arXiv:1907.08355 [cs.DS] (2019), http://arxiv.org/abs/1907.08355
42. Golynski, A.: Stronger lower bounds for text searching and polynomial evaluation (2007), https://cs.uwaterloo.ca/research/tr/2007/CS-2007-25.pdf
43. Golynski, A.: Cell probe lower bounds for succinct data structures. In: SODA (2009). https://doi.org/10.1137/1.9781611973068.69
44. Goyal, N., Saks, M.: A parallel search game. Random Structures & Algorithms **27**(2), 227–234 (2005). https://doi.org/10.1002/rsa.20068
45. Grossi, R., Orlandi, A., Raman, R.: Optimal trade-offs for succinct string indexes. In: ICALP (2010). https://doi.org/10.1007/978-3-642-14165-2_57
46. Haitner, I., Mazor, N., Oshman, R., Reingold, O., Yehudayoff, A.: On the communication complexity of key-agreement protocols. In: ITCS (2019). https://doi.org/10.4230/LIPIcs.ITCS.2019.40
47. He, M., Munro, J.I., Satti, S.R.: Succinct ordinal trees based on tree covering. ACM Transactions on Algorithms **8**(4), 42:1–42:32 (2012). https://doi.org/10.1145/2344422.2344432
48. Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory **26**(4), 401–406 (1980). https://doi.org/10.1109/TIT.1980.1056220
49. Håstad, J., Goldmann, M.: On the power of small-depth threshold circuits. Computational Complexity **1**, 113–129 (1991). https://doi.org/10.1007/BF01272517
50. Impagliazzo, R.: Relativized separations of worst-case and average-case complexities for NP. In: CCC (2011). https://doi.org/10.1109/CCC.2011.34
51. Jacobson, G.: Space-efficient static trees and graphs. In: FOCS (1989). https://doi.org/10.1109/SFCS.1989.63533
52. Jukna, S.: Boolean Function Complexity. No. 27 in Algorithms and Combinatorics (2012). https://doi.org/10.1007/978-3-642-24508-4
53. Jukna, S., Schnitger, G.: Min-rank conjecture for log-depth circuits. Journal of Computer and System Sciences **77**(6), 1023–1038 (2011). https://doi.org/10.1016/j.jcss.2009.09.003
54. Kopelowitz, T., Porat, E.: The strong 3SUM-INDEXING conjecture is false. arXiv:1907.11206 [cs.DS] (2019), http://arxiv.org/abs/1907.11206
55. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press (1997)
56. Larsen, K.G., Nielsen, J.B.: Yes, there is an oblivious RAM lower bound! In: CRYPTO (2018). https://doi.org/10.1007/978-3-319-96881-0_18
57. Levin, L.A.: One-way functions and pseudorandom generators. Combinatorica **7**(4), 357–363 (1987). https://doi.org/10.1007/BF02579323
58. Liang, H.: Optimal collapsing protocol for multiparty pointer jumping. Theory Comput. Syst. **54**(1), 13–23 (2014). https://doi.org/10.1007/s00224-013-9476-x
59. Miltersen, P.B.: On the cell probe complexity of polynomial evaluation. Theoretical Computer Science **143**(1), 167–174 (1995). https://doi.org/10.1016/0304-3975(95)80032-5
60. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations and functions. Theoretical Computer Science **438**, 74–88 (2012). https://doi.org/10.1016/j.tcs.2012.03.005
61. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: CCS (2005). https://doi.org/10.1145/1102120.1102168
62. Nayebi, A., Aaronson, S., Belovs, A., Trevisan, L.: Quantum lower bound for inverting a permutation with advice. Quantum Information & Computation **15**(11-12), 901–913 (2015)

63. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: CRYPTO (2003). https://doi.org/10.1007/978-3-540-45146-4_36
64. Ostrovsky, R.: Efficient computation on oblivious RAMs. In: STOC (1990). https://doi.org/10.1145/100216.100289
65. Pudlák, P., Rödl, V., Sgall, J.: Boolean circuits, tensor ranks, and communication complexity. SIAM Journal on Computing **26**(3), 605–633 (1997). https://doi.org/10.1137/S0097539794264809
66. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: SODA (2006). https://doi.org/10.1145/1109557.1109693
67. Unruh, D.: Random oracles and auxiliary input. In: CRYPTO (2007). https://doi.org/10.1007/978-3-540-74143-5_12
68. Valiant, L.G.: Graph-theoretic arguments in low-level complexity. In: MFCS (1977). https://doi.org/10.1007/3-540-08353-7_135
69. Valiant, L.G.: Why is Boolean Complexity Theory Difficult, pp. 84–94. No. 169 in London Mathematical Society Lecture Note Series (1992). https://doi.org/10.1017/cbo9780511526633.008
70. Viola, E.: On the power of small-depth computation. Foundations and Trends in Theoretical Computer Science **5**(1), 1–72 (2009). https://doi.org/10.1561/0400000033
71. Viola, E.: Lower bounds for data structures with space close to maximum imply circuit lower bounds. Electronic Colloquium on Computational Complexity (ECCC), Report 2018/186 (2018)
72. Viola, E., Wigderson, A.: One-way multiparty communication lower bound for pointer jumping with applications. Combinatorica **29**(6), 719–743 (2009). https://doi.org/10.1007/s00493-009-2667-z
73. Wee, H.: On obfuscating point functions. In: STOC (2005). https://doi.org/10.1145/1060590.1060669
74. Weiss, M., Wichs, D.: Is there an oblivious RAM lower bound for online reads? Cryptology ePrint Archive, Report 2018/619 (2018)
75. Wigderson, A.: (1996), unpublished
76. Yao, A.C.: Some complexity questions related to distributive computing (preliminary report). In: STOC (1979). https://doi.org/10.1145/800135.804414
77. Yao, A.C.: Should tables be sorted? Journal of the ACM **28**(3), 615–628 (1981). https://doi.org/10.1145/322261.322274
78. Yao, A.C.: Theory and applications of trapdoor functions. In: FOCS (1982). https://doi.org/10.1109/SFCS.1982.45
79. Yao, A.C.: Coherent functions and program checkers (extended abstract). In: STOC (1990). https://doi.org/10.1145/100216.100226
80. Yao, A.C.: On ACC and threshold circuits. In: FOCS (1990). https://doi.org/10.1109/FSCS.1990.89583

A Proof of Lemma 11

The first step is to define a permutation $\pi_\beta\colon [N] \to [N]$, for every function $\beta\colon [N] \to \{0,1\}$. We then use this permutation π_β to convert a Boolean-valued pointer-jumping instance $(x, \pi_1, \ldots, \pi_{k-1}, \beta)$ to an integer-valued pointer-jumping instance $(x, \pi_1, \ldots, \pi_{k-1}, \pi_\beta)$. Solving the integer-valued instance using a protocol for $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$ is then enough—with a few extra bits of communication—to solve the Boolean-valued instance of $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$.

Table 3: Example of the encoding procedure of Lemma 11. $N = 2^3$, and $\beta\colon [N] \to \{0,1\}$. Note that the last column is a permutation over the elements of $[N]$. Also note how β can be recovered from $\pi_\beta(x)$ for all $x \neq 0$.

x	$\beta(x)$	$(x\|_3, x\|_2, x\|_1)$	$\beta(x\|_3)\beta(x\|_2)\beta(x\|_1)$	$y = \pi_\beta(x)$
000	1	$(100, 010, 001)$	001	100
001	**1**	$(101, 011, 001)$	011	10**1**
010	**0**	$(110, 010, 001)$	101	0**0**0
011	**1**	$(111, 011, 001)$	111	0**1**1
100	**0**	$(100, 010, 001)$	001	**0**10
101	**0**	$(101, 011, 001)$	011	**0**01
110	**1**	$(110, 010, 001)$	101	**1**00
111	**1**	$(111, 011, 001)$	111	**1**11

Towards constructing π_β, consider first the case when N is a power of two. For $N = 2^n$, consider the following mapping from $\{0,1\}^N$ to permutations on $\{0, 1, \ldots, N-1\} = \{0,1\}^n$. On $\beta\colon \{0,1\}^n \to \{0,1\}$ we construct a permutation π_β on $\{0,1\}^n$ as follows: let $x \in \{0,1\}^n$ and let $x = x_n x_{n-1} \ldots x_1$ be the binary representation of x. Set $\pi_\beta(x) = y = y_n y_{n-1} \ldots y_1$ defined by $y_i = \beta(x|_i) \oplus x_i \oplus 1$ where $x|_i = 0 \ldots 01 x_{i-1} x_{i-2} \ldots x_1$. The following two properties hold:

- The mapping π_β defined above is a permutation. To see this let $x \neq x'$ be two distinct elements in $\{0,1\}^n$, and let $y = \pi_\beta(x)$ and $y' = \pi_\beta(x')$. Let $i \in [n]$ be the rightmost bit position on which x and x' differ. Then $x_i \neq x'_i$ but $x|_i = x'|_i$. Therefore $y_i = \beta(x|_i) \oplus x_i \oplus 1 \neq \beta(x'|_i) \oplus x'_i \oplus 1 = y'_i$, so $y \neq y'$.
- For any $x \in \{0,1\}^n$ such that $x = x_n \ldots x_1 \neq 0$, let i be the leftmost bit position such that $x_i = 1$. It then holds that $\beta(x)$ is equal to the ith bit of $\pi_\beta(x)$.

Note that the latter property guarantees that the value of $\beta(x)$ for every $x \neq 0$ can be recovered from a single bit of $\pi_\beta(x)$.

For N which is not a power of 2, we can view N as a sum $\sum_{j=1}^{\ell} 2^{n_j}$ of at most $\lceil \log N \rceil$ powers of 2, and construct a permutation π_β on $\{0, \ldots, N-1\} = \{0,1\}^{n_1} \cup \cdots \cup \{0,1\}^{n_\ell}$ as a union of permutations on $\{0,1\}^{n_j}$. By the properties above, for all but $\ell = \lceil \log N \rceil$ *bad* points, the value of β can be recovered from the corresponding value of π_β. Note that the set of bad points depends only on N and not on β. We give an example of this encoding procedure in Table 3.

Therefore, given a communication protocol for $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$, we construct a protocol for $\mathsf{MPJ}^{\mathsf{perm}}_{N,k}$ as follows. Let $\beta \in \{0,1\}^N$ be the input (on the forehead) of the last player. Each of the first $k-1$ players computes the permutation π_β from β according to the mapping above. The first player also writes on the blackboard the value of β evaluated on all of the bad points of π_β. The players then run the protocol for $\widehat{\mathsf{MPJ}}^{\mathsf{perm}}_{N,k}$ on the instance $(x, \pi_1, \ldots, \pi_{k-2}, \pi_\beta)$.

The last player computes the output of the original protocol $\pi_\beta \circ \pi_{k-2} \circ \cdots \circ \pi_1(x) = \pi_\beta(\hat{x}) \in \{0, 1, \ldots, N-1\}$ where $\hat{x} = \pi_{k-2} \circ \cdots \circ \pi_1(x)$. If $\hat{x}$ is not a bad point she can recover and output $\beta \circ \pi_{k-2} \circ \cdots \circ \pi_1(x) = \beta(\hat{x}) \in \{0,1\}$ from $\pi_\beta(\hat{x})$. Otherwise, if $\hat{x}$ is a bad point, she outputs the value $\beta(\hat{x})$, which the first player wrote on the blackboard.

The new protocol increases the communication complexity of the original protocol by $\lceil \log N \rceil$.

B Proof of Theorem 8

Let $(\mathcal{A}_0, \mathcal{A}_1)$ be a distinguisher for any length-increasing generator $G\colon [N] \to [M]$ with an advantage $\mathsf{PRGadv}\left[(\mathcal{A}_0, \mathcal{A}_1), G\right] \geq \epsilon$. We assume that N is power of two, and let $n = \log N$. (To handle the general case, one can, for instance, extend the function domain to the next power of two.) We construct an inversion algorithm for injective functions $f\colon [N] \to [M]$ in two steps.

First, for each $i \in [n]$, we construct a bit-recovery algorithm $\mathcal{B}_i$ that, given $f(x)$, achieves a non-trivial advantage in recovering the ith bit of x. We then use the algorithms $(\mathcal{B}_1, \ldots, \mathcal{B}_n)$ to construct an inversion algorithm $\mathcal{I}$ that, given $f(x)$, recovers the full preimage x with good probability.

For every $i \in [n]$, and for every $z \in \{0,1\}^n$ let $[z]_{i\to 1}$ denote z with its ith bit z_i set to 1. The bit-recovery algorithm $\mathcal{B}_i = (\mathcal{B}_{i0}, \mathcal{B}_{i1})$, for every $i \in [n]$, is given access to random permutations π and σ and operates as follows:

- **Preprocessing.**
  - Define a function $G_i\colon [N] \to [M]$ such that $G_i(x) = \pi\left(f\left([\sigma(x)]_{i\to 1}\right)\right)$.
  - Run the preprocessing phase for the PRG distinguisher on function G_i to get an advice string st_{G_i}: $\mathsf{st}_{G_i} \leftarrow \mathcal{A}_0^{G_i}()$.
- **Online.**
  - On input $y \in [M]$, run the online phase for the PRG distinguisher $\mathcal{A}_1(\mathsf{st}_{G_i}, \pi(y))$.
  - Answer each of $\mathcal{A}_1$'s oracle queries to G_i using oracle access to f, σ, and π.
  - Output the bit b_i that $\mathcal{A}_1$ outputs.

The inversion algorithm $\mathcal{I}$ uses the bit-recovery algorithms $(\mathcal{B}_1, \ldots, \mathcal{B}_n)$, a random oracle $\mathcal{O}$ (implemented using correlated randomness between the preprocessing and online algorithms), and a parameter k, which we choose later, and operates as follows:

- **Preprocessing.**
  For $j \in \{1, \ldots, k\}$:
  - Derive from the random oracle $\mathcal{O}$ two random permutations π_j and σ_j using standard techniques [21].
  - For each $i \in \{1, \ldots, n\}$, generate an advice string: $\mathsf{st}_{ij} \leftarrow \mathcal{B}_{i0}^{f,\pi_j,\sigma_j}()$.

  Finally, output the nk advice strings $\{\mathsf{st}_{ij}\}_{i\in[n],j\in[k]}$.
- **Online.**
  For $j \in \{1, \ldots, k\}$:

- Derive from the random oracle $\mathcal{O}$ the same two random permutations π_j and σ_j as in the offline phase.
- For each $i \in \{1, \ldots, n\}$, compute a guess of the ith bit of the preimage of y under f: $b_{ij} \leftarrow \mathcal{B}_{i1}^{f,\pi_j,\sigma_j}(\mathsf{st}_{ij}, y)$. Since f is injective, there is exactly one such preimage.

For each $i \in \{1, \ldots, n\}$, let $\hat{b}_i \in \{0, 1\}$ be the majority vote of the bits $\{b_{i1}, \ldots, b_{ik}\}$. Finally, output $\hat{x} = \hat{b}_1 \hat{b}_2 \ldots \hat{b}_n \in \{0, 1\}^n$.

We now analyze both algorithms and prove that the resulting inverter succeeds with probability $1 - 1/\log N$, uses an advice string of length $\tilde{O}(\epsilon^{-2} S)$, and makes $\tilde{(}\epsilon^{-2}T)$ online queries to f and $\tilde{(}\epsilon^{-2}T)$ queries to the random oracle.

Proposition 17. *For every injective function $f \colon [N] \to [M]$, every $x \in [N]$, and every $i \in [n]$,*

$$\Pr_{\pi,\sigma}\left[\mathcal{B}_{i1}^{f,\pi,\sigma}\left(\mathcal{B}_{i0}^{f,\pi,\sigma}(), f(x)\right) = x_i\right] \geq 1/2 + \Omega(\epsilon)\,,$$

where x_i denotes the ith bit of x.

Proof. Algorithm $\mathcal{B}_{i1}$, on input y, runs $\mathcal{A}$ on the point $\pi(y)$, thus

$$\Pr_{\pi,\sigma}\left[\mathcal{B}_{i1}^{f,\pi,\sigma}\left(\mathcal{B}_{i0}^{f,\pi,\sigma}(), f(x)\right) = 1\right] = \Pr_{\pi,\sigma}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), \pi(f(x))\right) = 1\right]\,.$$

Since $\mathcal{A}$ distinguishes the output of any length-increasing PRG from random with advantage ϵ, we may assume without the loss of generality that

$$\Pr_{x \overset{\mathrm{R}}{\leftarrow} [N]}\left[\mathcal{A}_1^{G_i}(\mathcal{A}_0^{G_i}, G_i(x)) = 1\right] \geq 1/2 + \Omega(\epsilon)\,, \tag{1}$$

and

$$\Pr_{y \overset{\mathrm{R}}{\leftarrow} [M]}\left[\mathcal{A}_1^{G_i}(\mathcal{A}_0^{G_i}, y) = 0\right] \geq 1/2 + \Omega(\epsilon)\,. \tag{2}$$

Consider now each of the two possible values of the bit x_i.

If $x_i = 1$, let $z = \sigma^{-1}(x)$ and note that

$$\pi\left(f(x)\right) = \pi\left(f([x]_{i\to 1})\right) = \pi\left(f\left([\sigma(z)]_{i\to 1}\right)\right) = G_i(z)\,.$$

Since σ is a random permutation, then $z = \sigma^{-1}(x)$ is a random point in $[N]$, and, since f is injective and π is a random permutation, the PRG $G_i = \pi \circ f \circ [\,]_{i\to 1} \circ \sigma$ is a random two-to-one function. Moreover, since π is independent of σ, then, even when we condition on $z = \sigma^{-1}(x)$, G_i is still a uniformly random two-to-one function. Hence z and G_i are *independent*, and

$$\Pr_{\pi,\sigma}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), \pi(f(x))\right) = 1\right] = \Pr_{\substack{G_i \\ z \overset{\mathrm{R}}{\leftarrow} [N]}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), G_i(z)\right) = 1\right] \geq 1/2 + \Omega(\epsilon)\,, \tag{3}$$

where the inequality follows from (1).

If $x_i = 0$, then x is not in the image of $[\,]_{i \to 1} \circ \sigma$, and thus $\pi(f(x))$ is a random point in $[M] \setminus \mathrm{Im}(G_i)$. Similarly to the argument above, we can show that this point is also independent of G_i, and thus

$$\Pr_{\pi,\sigma}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), \pi(f(x))\right) = 1\right] = \Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} [M]\setminus \mathrm{Im}(G_i)}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] .$$

The final step is to show that, in the $x_i = 0$ case, even though we run the PRG distinguisher $\mathcal{A}$ on samples from the uniform distribution over $[M] \setminus \mathrm{Im}(G_i)$ instead of over $[M]$, $\mathcal{A}$ still achieves good distinguishing advantage.

We can think of the uniform distribution over $[M]$ as a weighted sum of the distributions over $\mathrm{Im}(G_i)$ and $[M] \setminus \mathrm{Im}(G_i)$. Moreover, since G_i is a two-to-one function. Then the weight on $\mathrm{Im}(G_i)$ is $N/2M$ and the weight on $[M] \setminus \mathrm{Im}(G_i)$ is $(M - N/2)/M$. Therefore

$$\begin{aligned}\Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} [M]\setminus \mathrm{Im}(G_i)}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] =& \tfrac{M}{M-N/2} \cdot \Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} [M]}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] \\ &- \tfrac{N/2}{M-N/2} \cdot \Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} \mathrm{Im}(G_i)}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] .\end{aligned} \tag{4}$$

Since G_i is a two-to-one function, the distributions $\{w \xleftarrow{\mathrm{R}} \mathrm{Im}(G_i)\}$ and $\{w = G_i(x) : x \xleftarrow{\mathrm{R}} [N]\}$ are identical. Substituting the latter for the former in Eq. (4), we get

$$\begin{aligned}\Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} [M]\setminus \mathrm{Im}(G_i)}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] =& \tfrac{M}{M-N/2} \cdot \Pr_{\substack{G_i \\ y \xleftarrow{\mathrm{R}} [M]}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), y\right) = 0\right] \\ &- \tfrac{N/2}{M-N/2} \cdot \Pr_{\substack{G_i \\ x \xleftarrow{\mathrm{R}} [N]}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), G_i(x)\right) = 0\right] .\end{aligned} \tag{5}$$

Plugging in Inequalities (1) and (2) into Eq. (5), we obtain

$$\begin{aligned}\Pr_{\substack{G_i \\ w \xleftarrow{\mathrm{R}} [M]\setminus \mathrm{Im}(G_i)}}\left[\mathcal{A}_1^{G_i}\left(\mathcal{A}_0^{G_i}(), w\right) = 0\right] &\geq \tfrac{M}{M-N/2} \cdot (1/2 + \Omega(\epsilon)) - \tfrac{N/2}{M-N/2} \cdot (1/2 - \Omega(\epsilon)) \\ &\geq 1/2 + \Omega(\epsilon) .\end{aligned} \tag{6}$$

The claim follows from (3) and (6). □

Proposition 18. *For every injective function $f\colon [N] \to [M]$ and every $x \in [N]$,*

$$\Pr_{\mathcal{O}}\left[\mathcal{I}_1^{f,\mathcal{O}}(\mathcal{I}_0^{f,\mathcal{O}}(), f(x)) = x\right] \geq 1 - 1/\log N.$$

Proof. Since algorithm $\mathcal{B}$ correctly guesses the ith bit of x with probability $1/2 + \Omega(\epsilon)$, we have

$$\Pr_{\pi_{ij},\sigma_{ij}}[b_{ij} = x_i] \geq 1/2 + \Omega(\epsilon)\,.$$

Since the permutations $\{\pi_j, \sigma_j\}_{j\in[k]}$ are sampled independently from the random oracle, using a standard Chernoff bound we get

$$\Pr_{\mathcal{O}}[\hat{b}_i \neq x_i] \leq e^{-\Omega(\epsilon^2 k)}\,.$$

Therefore setting $k = O(\epsilon^{-2}\log n) = O(\epsilon^{-2}\log\log N)$ we get

$$\Pr_{\mathcal{O}}[\hat{b}_i \neq x_i] \leq 1/n^2\,.$$

Taking the union bound over all $i \in [n]$ we get

$$\Pr_{\mathcal{O}}[\hat{x} \neq x] \leq 1/n = 1/\lceil \log N \rceil\,.$$

□

Time and space analysis. If algorithm $\mathcal{A}$ uses an advice string of length S and makes T online queries, then the inverter uses an advice string of length $kn \cdot S = O(\epsilon^{-2} S \log N \log\log N)$ and makes $kn \cdot T = O(\epsilon^{-2} S \log N \log\log N)$ online queries to f.

Moreover, constructing a random permutation from a random oracle requires only a constant number of queries to the random oracle for each evaluation of the random permutation [21], therefore the number of online queries to the random oracle is also $O(\epsilon^{-2} S \log N \log\log N)$.

C Background: Cycle walking and Hellman tables

We briefly recall the cycle-walking algorithm for inverting permutations with preprocessing.

This algorithm, which Yao [79] makes explicit, is implicit in Hellman's more general algorithm for inverting random functions with preprocessing [48]. This cycle-walking algorithm also serves as a building-block for other inversion algorithms [30, 63].

Theorem 19 (Hellman [48]). *There exists a black-box algorithm for inverting permutations $\pi\colon [N] \to [N]$ that, for any $S, T \in \mathbb{Z}^{>0}$ satisfying $ST \geq 2N\lceil \log N + 1\rceil$, uses T queries and S bits of advice and is T-round adaptive.*

Proof. In the preprocessing phase, consider the cycle structure of the permutation $\pi\colon [N] \to [N]$ given by the oracle. There are at most $N/(T+1)$ cycles of length greater than T. For every such cycle, store a sequence of "checkpoints" in the order they appear in the cycle, such that every point on the cycle is at a distance of at most T points on the cycle from the previous checkpoint.

If the ith cycle has length ℓ_i, that cycle requires $\lfloor \ell_i/T \rfloor$ checkpoints to cover its first $\lfloor \ell_i/T \rfloor \cdot T$ points and one checkpoint to cover the remaining $\ell_i \bmod T$ points. In the worst case, all cycles have length $\ell_i = T+1$, in which case we have $N/(T+1)$ cycles with two checkpoints each. This amounts to a total of $2N/(T+1) \leq 2N/T$ checkpoints.

We can therefore store the checkpoints as a list of lists using $2N\lceil \log N \rceil/T$ bits. We add one additional bit to each checkpoint (i.e., at most $2N/T$ bits total) to indicate whether the checkpoint belongs to the same cycle as the prior one.

In the online phase, given a point $y \in [N]$ as input and the list of checkpoints as an advice string, take $y_0 = y$ and $y_{i+1} = \pi(y_i)$ iteratively, until either (i) $y_i = y$, at which point output y_{i-1} as the preimage of y, or (ii) y_i is one of the stored checkpoints, at which point set y_{i+1} to be the previous checkpoint in the list on the same cycle and continue iterating. As each point on the permutation is either on a cycle of length at most T, or at a distance of at most T from a checkpoint, the total number of oracle queries in the online phase is at most T. □

We can think of this cycle-walking algorithm as dividing each cycle in the permutation π to at most $2N/T$ "chains" of length at most T. The online algorithm then traverses one of those chains by computing iterates of the permutation π.

To see why the cycle-walking algorithm does not apply to general functions: If we view a general function $f\colon [N] \to [N]$ as a graph G_f on vertices $[N]$ with edges $\{(i, f(i))\}_{i \in [N]}$, then, because of collisions in the function f, it will almost never be possible to cover the entire graph G_f with $O(N/T)$ chains each of length T.

Hellman's algorithm [48] gets around this difficulty by creating many rerandomized versions $g_1, g_2, \ldots$ of f, and covering part of each graph $G_{g_1}, G_{g_2}, \ldots$ with chains. By balancing the number of functions g_i, the number of chains per graph, and the length of each chain, Hellman's algorithm can invert a constant fraction of points in the image of f. See De, Trevisan, and Tulsiani [23, Section 2.1] for details.

D Proof of Theorem 14

The following lemma proves the first part of Theorem 14.

Lemma 20. *For any integral constant $c > 2$, If there is an algorithm for systematic substring search on texts of length $cN \cdot \lceil \log N \rceil$ with pattern length $c \cdot \lceil \log N \rceil$ that uses an S-bit index and reads T bits of the text in its online phase, then there is a black-box algorithm for inverting functions $f\colon [N] \to [N]$ that uses S bits of advice and makes T online queries.*

Proof. We prove the lemma for the case when $c = 3$, but the generalization is immediate. Given an algorithm $(\mathcal{A}_0, \mathcal{A}_1)$ for substring search, we describe the preprocessing algorithm for function inversion.

- **Preprocessing.** Let $\ell = \lceil \log N \rceil$. View f as a function that outputs ℓ-bit strings. Construct a text $\tau \in \{0,1\}^{3N\ell}$ by writing out the function table of f delimited by strings of zeros and ones.

  $$\tau \ = \ 0^\ell \parallel f(1) \parallel 1^\ell \parallel 0^\ell \parallel f(2) \parallel 1^\ell \parallel \cdots \parallel 0^\ell \parallel f(N) \parallel 1^\ell \quad \in \{0,1\}^{3N\ell}.$$

  Then, run the preprocessing algorithm $\mathcal{A}_0$ on τ and return the S-bit index st_τ it produces.
- **Online.** We are given a challenge $y \in [N]$ and we must find a value $x \in [N]$ such that $f(x) = y$. Write the challenge as a pattern $p_y = 0^\ell \| y \| 1^\ell \in \{0,1\}^{3\ell}$. Then, run the substring-search algorithm $\mathcal{A}_1(\mathsf{st}_\tau, p_y)$.

  As the algorithm runs, it makes at most T queries for the bits of τ. If the query is to the constant part of τ, we can respond with a "0" or "1" without making any f queries. Otherwise, we can respond to $\mathcal{A}_1$'s query by making a single query to f.

  When the algorithm outputs the position i^* of the substring, we can uniquely identify the location of the inverse as $i^*/3\ell \in [N]$.

The claimed efficiency properties follow by construction. We must only show that the constructed algorithm solves the function-inversion problem.

We say that an position i of a symbol in the text τ is on a "block boundary" if $i = 0 \pmod{3\ell}$, where we count the bits of τ starting from zero. We now claim that every substring of the form $q_y = 0^\ell \| y \| 1^\ell$ in τ must begin on block boundary.

To prove the claim: Towards a contradiction, assume that there exists a pattern $p_y \in \{0,1\}^{3\ell}$ that is a substring of τ but that does not appear on a block boundary. There are three cases:

- $0 < i \bmod 3\ell < \ell$: In this case, i points to a substring that ends with a "0", while q_y ends with a "1," so the strings cannot match.
- $\ell \leq i \bmod 3\ell < 2\ell$: If this case, i points to a substring whose 2ℓ-th symbol is a "0," while q_y's 2ℓ-th character is a "1", so the strings cannot match.
- $2\ell \leq i \bmod 3\ell < 3\ell$: In this case, i points to a substring that begins with a "1", while q_y begins with a "0," so the strings cannot match.

In all cases, we derive a contradiction, which proves the claim.

If the substring q_y appears at position i^* in τ, then by the claim just proved, it must be that $i^* \pmod{3\ell} = 0$. This implies that the middle ℓ bits of the substring of τ beginning at position i^* must be the value $f(i^*/3\ell)$. Thus, $f(i^*/3\ell) = y$ and the algorithm successfully inverts f. □

The following two lemmata together prove the second part of Theorem 14. We show (roughly) in Lemma 21 that inverting a length-preserving function $f\colon [2N] \to [2N]$ is enough to invert a length-increasing function $f\colon [N] \to [N^c]$. Then, we show in Lemma 22 that inverting a length-increasing function $f\colon [N] \to [N^c]$, for constant $c > 1$ is enough to solve substring search.

Lemma 21. *If there is a black-box inversion algorithm for functions $f\colon [2N] \to [2N]$ that uses S bits of advice and makes T online queries, then for any integral* *constant $c > 1$, there is a a black-box inversion algorithm for functions $f\colon [N] \to [N^c]$, that uses $O(S \log N + \log^2 N)$ bits of advice and makes $O(T \log N)$ online queries.*

The idea of the proof of Lemma 21 is that we choose a hash function $h\colon [N^c] \to [2N]$ from a universal family. Then we use our preprocessing algorithm to invert the function $h \circ f$.

To sketch why this works: Let $y = f(x)$ be the point that we aim to invert. As long as $z = h(y)$ has a *unique* preimage under h, then any inverse of $h(f(x))$ will also be a preimage of y under f. If the point $h(y)$ has multiple preimages under h, then this is not so.

However, we can just choose many hash functions $h_1, \dots, h_k$ and run the preprocessing algorithm on each. Then, as long as there exists an $h_i \in \{h_1, \dots, h_k\}$ such that $h_i(y)$ has a unique preimage under h_i, we will be able to invert. By choosing the number of hash functions k appropriately, we will invert all points with good probability.

The idea of using "reduction functions" that map a large range into a small domain in this setting goes back to Hellman [48] and features in the work of Fiat and Naor [30] and Oechslin's Rainbow tables [63]. As far as we know, the analysis and the application to the setting of inverting length-increasing functions are new.

Proof. Given an algorithm $(\mathcal{A}_0, \mathcal{A}_1)$ for inverting a function from $[2N]$ to $[2N]$, we construct an algorithm for inverting a function $f\colon [N] \to [N^c]$, for any integral constant $c > 1$.

Preliminaries. The algorithm makes use of a family $\mathcal{H}$ of universal hash functions mapping $[N^c]$ into $[2N]$. Using the Carter-Wegman construction [15], we can evaluate any function $h \in \mathcal{H}$ in $O(\log N)$ time and we can represent an element $h \in \mathcal{H}$ using $O(\log N)$ bits. We define a "domain-extended" version of the function $f\colon [N] \to [N^c]$ that we wish to invert. The domain-extended version $\hat{f}$ maps $[2N]$ to $[N^c]$. We define $\hat{f}(x) = f(x)$ for all $x \in [N]$, and $\hat{f}(x) = 1$ otherwise.

We first describe the algorithm as using a randomized preprocessing phase. We then explain how to derandomize it. The algorithm proceeds as follows:

- **Preprocessing.** Sample $k = 2\lceil \log N \rceil$ functions $h_1, \dots, h_k$ independently at random from the universal hash function family $\mathcal{H}$. For every $i \in \{1, \dots, k\}$:
  - Define a function $g_i\colon [2N] \to [2N]$ as $g_i = h_i \circ \hat{f}$.
  - Run the preprocessing algorithm $\mathcal{A}_0$ on g_i to get advice string st_{g_i}.

  As the algorithm's advice string, output:
  - the strings $\mathsf{st}_{g_1}, \dots, \mathsf{st}_{g_k}$,
  - the short descriptions of the hash functions $h_1, \dots, h_k$, and
  - a preimage of 1 under f, if one exists.
- **Online.** We are given as input a point $y \in \mathrm{Im}(f) \subset [N^c]$. If $y = 1$, output the preimage of 1 hardcoded into the advice string.

Otherwise, for each $i \in \{1, \ldots, k\}$:

- Run the online algorithm $x_i \leftarrow \mathcal{A}_1(\mathsf{st}_{g_i}, h_i(y)) \in [N]$. As $\mathcal{A}_1$ runs, it makes queries to g_i. We can reply to each query by making at most a single query to f and applying h_i to the output.
- Use a single query to f to test whether $f(x_i) = y$. If so, return x_i as the preimage of y under f.

If the algorithm has not yet found a preimage of x, output the failure symbol $\perp$.

By construction, the algorithm satisfies the claimed bounds on advice and time complexity. By construction, it also always outputs a preimage of y under f. To complete the proof, we need only show that the failure probability is as claimed.

Towards this goal, say that in the online phase our task is to invert a point $y \in \mathrm{Im}(f)$. First, observe that the algorithm trivially inverts the point $y = 1$, so assume that the input point $y \neq 1$. Next, notice that if there exists some function $h_i \in \{h_1, \ldots, h_k\}$ such that $h_i(y)$ has a unique preimage under $\hat{f}$, then the online algorithm will succeed.

In this case, the algorithm $\mathcal{A}_1(\mathsf{st}_{g_i}, h_i(y))$ must output a value x such that $g_i(x) = h_i(y)$. But, by definition of g_i, this implies that $h_i(\hat{f}(x)) = h_i(y)$. Since $h_i(y)$ only has a single preimage in the image of $\hat{f}$, we know that $\hat{f}(x) = y$. By construction of $\hat{f}$, whenever $y \neq 1$, all preimages of y under $\hat{f}$ are in the range $\{1, \ldots, N\}$. Furthermore, for every such preimage x_0, $f(x_0) = \hat{f}(x_0)$. Therefore, the point x that the algorithm outputs is indeed a preimage of y under f.

Thus, our task now is to prove that, with high probability over the random choice of the hash functions $h_1, \ldots, h_k$ from the universal family $\mathcal{H}$, for every point $y \in \mathrm{Im}(f)$, there exists a hash function h_i such that $h_i(y)$ has a single inverse in the image of $\hat{f}$.

Consider one such function $h_i \colon [N^c] \to [2N]$. By definition of universal hashing, for distinct values $y, y' \in [N^c]$, $\Pr_{h_i}[h_i(y) = h_i(y')] \leq 1/(2N)$. Therefore, by the Union Bound, for every $y \in \mathrm{Im}(f)$, the probability (over the choice of $h_i \in \mathcal{H}$) that $h_i(y)$ has more than one preimage in $\mathrm{Im}(f)$ under h_i is at most $(|\mathrm{Im}(f)| - 1)/(2N) \leq 1/2$.

If we sample k hash functions independently from the family $\mathcal{H}$, then for every $y \in \mathrm{Im}(f)$, the probability that for every $i \in [k]$ the point $h_i(y)$ has more than one preimage under h_i is at most $1/2^k$. Then the probability that there exists a point $y \in \mathrm{Im}(f)$ that does not have a unique preimage under any of the k functions is, by the Union Bound, at most $|\mathrm{Im}(f)| \cdot (1/2)^k \leq N \cdot (1/2)^k$. For $k = 2 \log N$, this failure probability is at most $1/N$.

To get an algorithm that never fails, we can repeat the preprocessing phase with fresh randomness until we find an advice string that inverts all points. □

Lemma 22. *Let $c > 1$ be an integral constant. If there is a black-box algorithm for inverting length-increasing functions $f \colon [N] \to [N^c]$ that uses S bits of advice and makes T online queries, then there is an algorithm for systematic substring* *search on texts of length N with pattern length $c \cdot \lceil \log N \rceil$ that uses an S-bit index and reads $cT\lceil \log N \rceil$ bits of the text in its online phase.*

Proof. Given an algorithm $(\mathcal{A}_0, \mathcal{A}_1)$ for inverting length-increasing functions, we describe the preprocessing algorithm for systematic substring search. Let $\ell = \lceil \log N \rceil$.

- **Preprocessing.** We are given a text $\tau \in \{0,1\}^N$ to preprocess. Define a function $f \colon [N] \to [N^c]$ such that the value $f(i)$ is equal to the $c\ell$ bits of the text τ beginning at position i. For the extremal values $i \geq N^c - c\ell$, set $f(i)$ to be some special $(c\ell)$-bit string that appears nowhere in τ. (Such a string is guaranteed to exist, since $c > 1$.)

  Run the preprocessing algorithm $\mathcal{A}_0$ on f and output the advice string st_f it outputs as the index.
- **Online.** We are given a pattern $p \in \{0,1\}^{c\ell}$ and we must find a value $i \in [N]$ such that the $(c\ell)$-bit substring of τ beginning at position i is equal to s. As $\mathcal{A}_1$ runs, it makes queries to f. We can respond to each query to f using at most $c\ell$ queries to the text τ.

  To do so, run $i \leftarrow \mathcal{A}_1(\mathsf{st}_f, q)$. If $0 \leq i < (N - c\ell)$ output i. Otherwise output "$\perp$."

The efficiency and correctness properties follow immediately by construction. □

We can now assemble the results of this section to prove Theorem 14:

Proof of Theorem 14. Lemma 20 proves the first part of the theorem.

To prove the second part: Lemma 21 then shows that, for any $N \in \mathbb{Z}^{\geq 0}$, if there is an algorithm for inverting length-preserving functions $f \colon [2N] \to [2N]$ that uses S bits of advice and makes T online queries, then for any integral constant $c > 1$, there is algorithm for inverting length-increasing functions $f' \colon [N] \to [N^c]$, that uses $S' = \widetilde{O}(S)$ bits of advice and makes $T' = \widetilde{O}(T)$ online queries.

Then, Lemma 22 shows that if there is an algorithm that inverts such functions f' that uses S' bits of advice and that makes T' online queries, then there an algorithm for systematic substring search on texts of length N with pattern length $c \cdot \lceil \log N \rceil$ that uses an index of size $S' = \widetilde{O}(S)$ bits and that makes $\widetilde{O}(T') = \widetilde{O}(T)$ online queries. □