Electronic Colloquium on Computational Complexity, Report No. 182 (2018)

The Function-Inversion Problem: Barriers and Opportunities

Henry Corrigan-Gibbs
henrycg@cs.stanford.edu

Dmitry Kogan
dkogan@cs.stanford.edu

Stanford University

Abstract

We study preprocessing algorithms for the function-inversion problem. In this problem, an algorithm gets oracle access to a function $f\colon [N] \to [N]$ and takes as input S bits of auxiliary information about f, along with a point $y \in [N]$. After running for time T, the algorithm must output an $x \in [N]$ such that $f(x) = y$, if one exists. This problem, first studied by Hellman (1980), has manifold applications to cryptanalysis.

Hellman's algorithm for this problem achieves the upper bound $S = T = \widetilde{O}(N^{2/3})$ when f is a random function, while the best known lower bound, due to Yao (1990) shows that $ST = \widetilde{\Omega}(N)$, which admits the possibility of an $S = T = \widetilde{O}(N^{1/2})$ algorithm. There remains a long-standing and vexing gap between these upper and lower bounds.

By uncovering connections between function inversion and other areas of theoretical computer science, we explain why making progress on either the lower-bound or upper-bound side of this problem will be difficult. Along the way, we use these connections—in concert with Hellman-style algorithms—to improve the best upper bounds for well-studied problems in communication complexity and data structures.

In particular, we obtain the following results:

- We show that *any* improvement on Yao's lower bound for function inversion will imply new lower bounds on depth-two circuits with arbitrary gates.
- We show that proving strong lower bounds for function inversion would imply breakthrough lower bounds against linear-size log-depth circuits.
- We use a cryptanalytic algorithm to obtain an $O\big((N/k + \sqrt{N}) \log N\big)$-bit protocol for the permutation variant of the k-party pointer jumping problem in the number-on-the-forehead model of communication complexity. For any $k = \omega\big(\frac{\log N}{\log \log N}\big)$, we improve the previous best bound of $O\big(N \cdot \frac{\log \log N}{\log N}\big)$, due to Pudlák, Rödl, and Sgall (1997).
- We give the first data structure for the systematic substring-search problem achieving index size and query time $\widetilde{O}(N^{\delta})$, for some $\delta < 1$. In fact, we achieve $\delta = 3/4$. In doing so, we answer an open question of Gál and Miltersen (2003).

ISSN 1433-8092

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.

One particularly interesting class of function-inversion algorithms are those that only have *black-box* access to the function f—or formally, that have only oracle access to f—since these algorithms invert *all* functions. A straightforward argument shows that any such inversion algorithm that makes at most T queries to its f-oracle succeeds with probability at most $O(T/N)$. 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 $f(x) = y$. We seek 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 [Hel80] 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 random function. (More precisely, the algorithm has a constant success probability where the probability is taken over a uniform choice of function f.) Fiat and Naor [FN91, FN99] 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 $N^{3/4} \le S \le N$. This trade-off is the best known today, and is a fundamental tool in real-world cryptanalysis [BS00, BSW00, Oec03, NS05].

In this work, we investigate the hardness of the following question:

Is Hellman's time-space trade-off optimal?

Yao first asked this question in 1990 [Yao90a] 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.) This lower bound does not rule out an algorithm that achieves $S = T = \widetilde{O}(N^{1/2})$. In contrast, Hellman's algorithm 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 [FN99], Barkan, Biham, and Shamir [BBS06] (who show that Hellman's method is optimal for a certain natural class of algorithms), De, Trevisan and Tulsiani [DTT10], and Abusalah et al. [AACKPR17]. 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 $N^{2/3}$ and an $N^{1/2}$ online time attack becomes crucial when dealing with 128-bit block ciphers, such as the ubiquitous AES-128.

Recent work proves new lower bounds on preprocessing algorithms for various cryptographic problems, using both incompressibility arguments [GGKT05, DGK17, AACKPR17] and the newer presampling method [Unr07, CDGS18]. 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 [Oec03], 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 makes two primary contributions. First, we show that improving on the known lower bounds for the function-inversion problem will be very difficult. In particular, new lower bounds for inversion will imply new circuit lower bounds and could even resolve complexity-theoretic questions that predate Hellman's results [Val77]. Second, we show that improving upper bounds for function inversion will immediately yield new upper bounds for standard problems in communication complexity and data structures. Even further, we obtain the *best known upper bounds* for these problems by instantiating our new connection with Hellman-style algorithms. In sum, we demonstrate that making essentially any progress on the function-inversion problem would have important consequences—not only for cryptography, but also for other areas of theoretical computer science.

Proving the optimality of Hellman's algorithm requires new circuit lower bounds. A major question in circuit complexity, open since the 1970s [Val77, Val92], 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 [BL15], 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 $S = N \log\log N / \log N$ 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 this 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 for non-adaptive function-inversion algorithms that use the relatively large amount of advice $S = N \log\log N / \log N$ 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 [Val77, Val92]. 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, *proving that Hellman's method is optimal 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 [Yao90a, GT00] and presampling [Unr07, CDGS18] techniques might prove more difficult than previously expected.

New protocols for multiparty pointer jumping. We show that improved algorithms for the black-box function-inversion problem imply improved upper bounds 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 ($\mathsf{MPJ}^{\mathrm{perm}}_{N,k}$) [PRS97, DJS98, BC08, Bro09, VW09, Lia14, BS15] in the number-on-the-forehead model of communication complexity [CFL83].

Then, by instantiating the permutation-inversion algorithm with a variant of Hellman's method, we obtain the *best known protocol* for $\mathsf{MPJ}^{\mathrm{perm}}_{N,k}$ for $k = \omega(\log N/\log\log N)$ players, improving the previous best upper bound of $O(N \log\log N/\log N)$, by Pudlák et al. [PRS97], 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 [Yao90b, HG91, BT94].

Beyond the quantitative improvement, our protocol is different from all previous approaches to the problem and, as far as we know, is the first application of a cryptanalytic algorithm to a communication-complexity problem.

Taking an optimistic view: this connection also 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 $\mathsf{MPJ}^{\mathrm{perm}}_{N,3}$. Any improvement on the lower bound for $\mathsf{MPJ}^{\mathrm{perm}}_{N,3}$ would give an improved lower bound for non-adaptive black-box permutation-inversion algorithms. The best lower bound for $\mathsf{MPJ}^{\mathrm{perm}}_{N,3}$ is $\Omega(\sqrt{N})$ [Wig96, BHK01]. Interestingly, this matches the best lower-bound for black-box permutation-inversion algorithms, regardless of their adaptivity.

Or, taking a pessimistic view: this connection suggests that constructing non-adaptive or partially adaptive function-inversion algorithms will be difficult—as difficult as devising better communication protocols for pointer jumping.

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* [DLO03, GM03, GM07, Gol07, Gol09]. 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 [Yao81]. 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 [FN99] yields an $S^3T = \widetilde{O}(N^3)$ systematic data structure, which is the best known in the parameter regime $S = \widetilde{O}(N^\epsilon)$ for $\epsilon < 1$. Gál and Miltersen [GM03] 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 [FN91, FN99] proved that Hellman's algorithm [Hel80] 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 [DTT10] 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 [Yao90a] proved that every algorithm that uses S bits of advice and makes T online queries must satisfy $ST = \widetilde{\Omega}(N)$ lower bound and Dodis et al. [DGK17], building on prior work [GT00, DTT10], gave a similar lower bound even for algorithms that invert on a sub-constant fraction of functions f.

Barkan, Biham, and Shamir [BBS06] show that, for a *restricted class of preprocessing algorithms*, a Hellman-style trade-off of the form $S^2T = \widetilde{O}(N^2)$ is 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 [Oec03]. 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 [Yao90a] observed that a Hellman-style algorithm can achieve the $ST = \widetilde{O}(N)$ upper bound and he proved the matching lower bound. Gennaro and Trevisan [GT00], Wee [Wee05], and De, Trevisan, and Tulsiani [DTT10] extend this this lower bound to handle randomized algorithms and those that succeed with small probability.

1.3 Notation and definitions

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 $x \xleftarrow{\text{R}} S$ to denote a uniform random draw from $\mathcal{X}$. For a function $f : A \to B$, we denote the image of the function as $\mathrm{Im}(f) = \{f(x) \mid x \in A\} \subseteq B$, and for every $y \in B$, we define the preimage set of y as $f^{-1}(y) = \{x \in A | f(x) = y\}$. All logarithms are base-two unless stated otherwise. Parameters S and T are always implicit functions of the parameter N, and to to simplify the bounds, we always implicitly take $S = T = \Omega(1)$. The notation $\widetilde{\Omega}(\cdot)$ and $\widetilde{O}(\cdot)$ hide 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 $\mathcal{A}_1^f(\mathcal{A}_0^f(), f(x)) \in f^{-1}(x)$.

We can define a black-box inversion algorithm *for permutations* analogously by restricting the oracle $f\colon [N] \to [N]$ to implement a one-to-one 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,
- *non-adaptive* if given its input and its advice string, the algorithm first outputs a single set of queries, then receives the oracle's responses to this queries, and finally outputs its output, and
- *strongly non-adaptive* if the algorithm is non-adaptive, and furthermore the set of queries 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.

We discuss additional subtleties with our computational model in Appendix A.

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 [Val77]. Towards this goal, we first show that proving strong lower bounds on *non-adaptive* algorithms for function inversion would imply new lower bounds against linear-sized logarithmic-depth circuits.

Related work. Brody and Larsen [BL15] 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 a *static* data-structure problem (i.e., function inversion), and our connection is to Valiant's common-bits model of circuits, rather than to linear depth-two circuits.

Boyle and Naor [BN16] 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) [Gol87, Ost90, GO96] would imply new lower bounds on the size of Boolean circuits for sorting lists of integers. Larsen and Nielsen [LN18] recently skirted this barrier by proving a lower bound against ORAMs in the "online" setting. Following that, Weiss and Wichs [WW18] showed that a variant of the Boyle-Naor barrier still holds against "online read-only" ORAMs.

2.1 A strong lower bound on non-adaptive function inversion implies a lower bound against linear-size log-depth circuits

A major open question in circuit complexity is whether there exists an explicit family of Boolean functions on n bits that cannot be computed by fan-in-two circuits of size $O(n)$ and depth $O(\log n)$. An easier question, which is still famously difficult, is to ask instead for 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 latter question has been open since the 1970s [Val77, Val92, JS11].

More precisely, we say that a family of Boolean operators $\{F_n\}_{n=1}^{\infty}$, 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 following theorem is the main result of this section.

Theorem 3. *If every explicit operator has fan-in-two Boolean circuits of size $O(n)$ and depth $O(\log n)$ then, for every $\epsilon > 0$, there exists a family of strongly non-adaptive black-box algorithms that inverts all functions $f\colon [N] \to [N]$ using $O(N \log N/ \log\log N)$ bits of advice and $O(N^\epsilon)$ online queries.*

We give the proof idea in Section 2.2 and the full proof in Appendix B.

Observe that a function $f\colon [N] \to [N]$ requires $O(N \log N)$ bits to describe, so there is a trivial algorithm that inverts f using $O(N \log N)$ bits of advice and no queries to f. The algorithm implied by the consequence of Theorem 3 is non-trivial because it uses $o(N \log N)$ bits of advice—just slightly fewer bits than it takes to describe the function. We know of no non-adaptive algorithm that inverts with constant probability using $o(N \log N)$ bits of advice and $o(N)$ queries.

The contrapositive of Theorem 3 immediately yields the following corollary:

Corollary 4. *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 corollary considers a restricted class of inversion algorithms that:

- may only use strongly non-adaptive queries (the most restrictive type of query),
- are only allowed, for example, $O(N^{0.0001})$ queries (very few queries), and
- must invert arbitrary functions with probability one (the most difficult variant of the inversion problem).

The corollary states that proving that such restricted algorithms must use a large amount of space is as hard as proving lower bounds against linear-size logarithmic-depth circuits. 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.

2.2 Proof idea for Theorem 3

To prove Theorem 3, we first recall Valiant's common-bits model of circuits [Val77, Val92].

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. (See Figure 1 in Appendix B.) The n input variables $x_1, \dots, x_n \in \{0,1\}$ are presented as wires at the input layer of the circuit and there are n output gates at the output layer of the circuit. There are w gates in the middle layer of the circuit (the "common bits"); each input wire 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 input wires. 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, and also 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?

Proof idea for Theorem 3. The full proof appears in Appendix B.

1. First, we define an *inversion operator* F_n^{inv} (Definition 12.) This operatator takes as input the description of a function $f\colon [N] \to [N]$, for $n = N \log N$, represented as its function table $(f(1), f(2), \ldots, f(N)) \in [N]^N$. The inversion operator F_n^{inv} outputs an inverse of *every* point $(1, 2, \ldots, N) \in [N]^N$ under the function f, whenever these inverses exist.

2. Next, we prove a result about circuits in the common-bits model. We show in Lemma 13 that if the inversion operator F_n^{inv} has a width-w degree-d circuit in Valiant's common-bits model of circuits [Val77, Val92], then there is a strongly non-adaptive inversion algorithm for f using w bits of advice and $d \log N$ online queries.

3. Finally, we apply a result of Valiant (Theorem 14), who proved that if all explicit operators have linear-size logarithmic-depth circuits, then all explicit operators have circuits in the common-bits model of width $w = O(n/\log\log n)$ and degree $d = n^{\epsilon}$, for any $\epsilon > 0$.

Putting these pieces together, we conclude that if all explicit operators have linear-size log-depth circuits, then there is a strongly non-adaptive inversion algorithm using $O(N \log N/\log\log N)$ bits of advice and $O(N^{\epsilon})$ online queries, for any $\epsilon > 0$. $\square$

2.3 Why proving the optimality of Hellman's algorithm is difficult

Corollary 4 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 5. *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 3, we can apply Lemma 13 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 $O(N^{1/2+\epsilon})$. $\square$

Notice that while the hypothesis of Corollary 5 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 5 with $\epsilon = \frac{1}{6}$, 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 14). 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 [PRS97]. In particular, they construct an explicit operator that does not have common-bits circuits satisfying $w = d = \widetilde{O}(n^{1/2})$. By Corollary 5, 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.

Should we hope for new lower bounds in the common-bits model? Drucker [Dru12] discusses the limitations of current techniques, and we recall these and other limitations in more detail in Appendix C. A generalization of Corollary 5 actually applies more generally to proving strong lower bounds for *any* so-called "succinct" data-structure problem [Jac89, CM96, GRR06, SG06, GM07, Gol07, Gol09, GOR10, BHMS11, MRRR12, HMS12] of which function inversion is a special case. We discuss this extension in Appendix C.3.

3 From cryptanalysis to new communication protocols

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 [CFL83]. 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.

3.1 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 $\mathsf{MPJ}^{\text{perm}}_{N,k}$, 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. (See Figure 3 in Appendix D.) 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 $P_0, \dots, P_{k-1}$, 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 $\mathsf{MPJ}^{\text{perm}}_{N,k}$ that require only $O(\log N)$ bits of communication.

Pudlák et al. [PRS97] develop a connection between NOF communication complexity and common-bits circuits, which we elaborate on in Appendix D.1. There, we also discuss applications of NOF complexity to data-structure lower bounds [Pǎt10, CEEP16].

One of the major open problems in NOF 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 [Yao90b, HG91, BT94]. 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 [PRS97, DJS98, BC08, Bro09, VW09, Lia14, BS15].

Known bounds. The best upper bound for $\mathsf{MPJ}_{N,k}^{\text{perm}}$ is due to Pudlák et al. [PRS97], who showed that $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\text{perm}}) = O(N \log\log N/\log N)$. More recently, Brody and Sanchez [BS15] showed that this upper bound applies to the more general pointer-jumping problem, in which we replace the permutations $\pi_1, \dots, \pi_{k-2}$ with arbitrary functions. In this general case, Wigderson [Wig96] proved an $\Omega(\sqrt{N})$ lower bound for $k = 3$ players (see also [BHK01]), and Viola and Wigderson [VW09] proved an $\widetilde{\Omega}(N^{\frac{1}{k-1}})$ lower bound for $k \geq 3$ players.

3.2 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 $\widetilde{O}\big(N/k+\sqrt{N}\big)$. Extending our upper bound to the *general* multiparty pointer-jumping problem remains an open problem, which we discuss in Appendix D.2.

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 $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\text{perm}})$:

Theorem 6. $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\text{perm}}) \leq O\left((N/k + \sqrt{N})\log N\right)$.

To prove Theorem 6, we use the integer-valued version the pointer-jumping problem, commonly denoted $\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}$. 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 $[N]$. We then use the following lemma, which we prove in Appendix D.3:

Lemma 7. $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\text{perm}}) \leq \mathsf{CC}^1(\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}) + \lceil \log N \rceil$.

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

Lemma 8. *If there exists a $(k-2)$-round adaptive algorithm for inverting permutations $\pi : [N] \to [N]$ that uses advice S and time T, then* $\mathsf{CC}^1(\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}) \leq S + 2T\lceil \log N \rceil$.

Proof idea. Consider the composition of the players' permutations in *reverse* order. Namely, define a permutation $\pi = \pi_1^{-1} \circ \cdots \circ \pi_{k-1}^{-1}$. We can then reinterpret the players' task of computing $\pi_{k-1} \circ \cdots \circ \pi_1(x)$ as the task of computing the *inverse* of x under π.

Now, we observe that the k players can simulate a black-box permutation-inversion algorithm with preprocessing in the one-way NOF model at the cost of paying a single round of communication for each round of queries. The players then can use an inversion algorithm to compute $\pi^{-1}(x)$, which yields the desired output. The players run the simulation as follows:

- First, Player P_0, who can see all the permutations, runs the preprocessing step of the inversion algorithm on the composed permutation π and writes the S-bit advice string on the blackboard.

- Next, Player P_1, who can see the input $x \in [N]$ and the permutations $\pi_2, \dots, \pi_{k-1}$, runs the online phase of the inversion algorithm on input x. As the inversion algorithm runs, it makes queries to π. To evaluate π at the first queried point $q \in [N]$, Player P_1 follows the pointers *in reverse order* from point q in the last layer until she reaches her own layer: $p \leftarrow \pi_2^{-1} \circ \cdots \circ \pi_{k-1}^{-1}(q)$. Since Player P_1 cannot see π_1—it is written on her forehead—she cannot compute $\pi(q) = \pi_1^{-1}(p)$. So, Player P_1 writes this partial response $p \in [N]$ on the blackboard.
- Next, Player P_2, who can see π_1, computes the answer $\pi(q) = \pi_1^{-1}(p)$ to the first query and writes it on the blackboard. Player P_2 then re-runs the online inversion algorithm on input x and replies to the inversion algorithm's first query q with the value $\pi(q)$.
- Players $P_2, \dots, P_{k-1}$ reply to the remaining rounds of queries this way, until Player P_{k-2} is able to answer all queries and compute the output $\pi^{-1}(x) = \pi_{k-1} \circ \cdots \circ \pi_1(x)$.

Finally, since the algorithm is $(k-2)$-round adaptive, it makes its queries in $k-2$ sets. The k players can respond to each set of queries in parallel.

We give the formal protocol and analyze its communication complexity in Appendix D.3. $\square$

To complete the proof of Theorem 6 we instantiate Lemma 8 using Hellman's cycle-walking algorithm [Hel80], which we recall in Appendix E. 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 $S = \sqrt{2}N(\lceil \log N \rceil + 1)$ to get a protocol with communication $O(\sqrt{N} \log N)$.

4 Function inversion is equivalent to substring search

In this section, we demonstrate an equivalence between the function-inversion problem and the *systematic substring-search problem*, a static data-structure problem studied by Demaine and López-Ortiz [DLO03], Gál and Miltersen [GM03, GM07], and Golynski [Gol07, Gol09]. We use this connection to demonstrate 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 [GM07] 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.

In the systematic substring-search problem, we are given a bitstring of length N ("the text") and a string 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.

Known lower bounds. Demaine and López-Ortiz [DLO03] 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 [Gol07, Gol09] 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 F.

Theorem 9. *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.

Proof idea for Theorem 9. 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 16, 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 [DLO03] 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 18 a function $f'\colon [N] \to [N^c]$ such that $f'(i)$ is equal to the length-P substring that starts from the i^{th} 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 17, 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. $\square$

We now apply Theorem 9 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," $T < N/\operatorname{polylog} N \Rightarrow S > N/\operatorname{polylog} N$ (using our notation) for systematic substring search [GM07]. Goyal and Saks [GS05] use an elegant argument to show that the specific technique of Gál and Miltersen cannot prove this lower bound. As a corollary of Theorem 9, we construct an algorithm for substring search that beats the hypothetical lower bound.

Corollary 10. *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 9 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

[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 11, 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.

algorithm of Fiat and Naor [FN99] inverts such functions f achieving the desired complexity bounds. $\square$

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 2.3, 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.

5 Future directions

We close with a few directions for future work on the function-inversion problem.

One-to-one functions. Almost all cryptanalytic applications of Hellman tables only require inverting *one-to-one* functions. Is it easier to invert a random one-to-one 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 one-to-one functions would be remarkable.

Pseudorandom generators. De, Trevisan, and Tulsiani [DTT10] introduced the problem of breaking pseudorandom generators (PRGs) with preprocessing. 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 give a distinguisher with $S = O(N/\epsilon^2)$ 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. While we still cannot answer that question, 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 one-to-one functions.

Theorem 11 (Informal)**.** *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 constant distinguishing advantage. Then there exists a black-box inversion algorithm for any one-to-one function $f\colon [N] \to [M]$ that uses $\widetilde{O}(S)$ bits advice, makes $\widetilde{O}(T)$ online queries, and inverts f with constant probability.*

We prove the theorem in Appendix G.

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 [BBS06] 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. Peter Bro Miltersen, Joshua Brody, Andrew Drucker, Michael Kim, Ilya Mironov, Omer Reingold, Avishay Tal, Li-Yang Tan, and David Wu made a number of suggestions that improved the presentation of our results.

A Model of computation

We discuss some choices in the model of computation for black-box function inversion with preprocessing as appears Definition 1.

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 probabilistic inversion algorithms, which invert successfully only with probability $\epsilon < 1$. Depending on the application, this probability may be taken over any of the random choice of a function $f\colon [N] \to [N]$, the point to invert, and algorithm's randomness. However, as most of the results in this paper deal with lower bounds, restricting ourselves to deterministic algorithms that always succeed in inverting only makes our results stronger. In any case, even if we bound only the number of queries, we 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.

When running an algorithm with preprocessing, the computation time (e.g., in the RAM model)—of both the preprocessing and online phases—is practically important. The standard algorithms with preprocessing for function inversion, including those of Hellman [Hel80], Fiat and Naor [FN99], and De, Trevisan, and Tulsiani [DTT10] all can use $\widetilde{O}(N)$ preprocessing time in a suitable RAM model, when they are allowed to fail with small probability. The running time of these algorithms' T-query online phase is $\widetilde{O}(T)$.

In the case of our new preprocessing algorithm for systematic substring search (Section 4), it is similarly 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 running time of the T-query algorithm 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. To see this, consider a non-uniform family of algorithms that make fewer than $T(N)$ oracle queries and use an $|\mathsf{st}_f| = S(N)$ bits of advice about the oracle function f. We can then construct a uniform algorithm that given N, $S(N)$, and $T(N)$ enumerates all possible non-uniform algorithms that use advice $S(N)$ and make $T(N)$ queries, on all possible functions $f\colon [N] \to [N]$ and all possible inputs $x \in [N]$ and chooses one that successfully inverts all functions on all points. Note that that for a given N, all of the enumerated sets are finite. Also note that this simulation does not require making any actual oracle queries.

Shared randomness. We allow the preprocessing and online phases to share an unlimited number of random bits. In other words, our reductions hold also with respect to random oracles.

B Proof of Theorem 3

Theorem 3 (restated). *If every explicit operator has fan-in-two Boolean circuits of size $O(n)$ and depth $O(\log n)$ then, for every $\epsilon > 0$, there exists a family of strongly non-adaptive black-box algorithms that inverts all functions $f\colon [N] \to [N]$ using $O(N \log N/ \log\log N)$ bits of advice and $O(N^\epsilon)$ online queries.*

The proof of Theorem 3 uses the following operator:

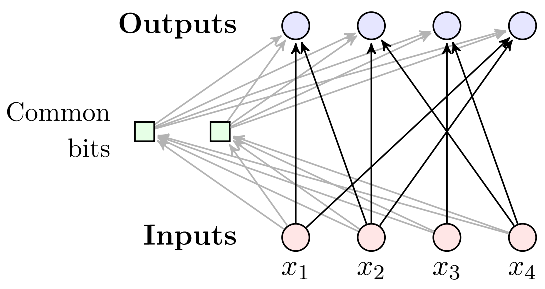

Figure 1: A circuit in the common-bits model with $n = 4$ inputs, degree $d = 2$, and width $w = 2$.

Definition 12 (Inversion operator)**.** 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 strings 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 $y_i = 0 \in [N]$. We define $F_n^{\mathsf{inv}}(x) = \big(y_1 \| y_2 \| \cdots \| y_N\big)$.

Lemma 13. *Let $n = N \log N \in \mathbb{Z}^{>0}$, where $N \in \mathbb{Z}^{>0}$ is a power of two. If there exists a circuit in the common-bits model of width w and degree d that computes the inversion operator F_n^{inv}, then there exists a strongly non-adaptive preprocessing algorithm that inverts a function $f \colon [N] \to [N]$ using an advice string of length w and $d \log N$ oracle queries.*

Proof. Let $\mathcal{C}$ be a circuit in the common-bits model as in the statement of the lemma. The advice string consists of the outputs of the middle-layer gates in the circuit $\mathcal{C}$ (i.e., the circuit's common bits). Divide the output gates into N blocks of size $\log N$ each. On input $y \in [N]$, the algorithm queries f on all the points containing input bits connected to the y-th block of output gates of $\mathcal{C}$. Using the advice string and the oracle replies, it then computes and outputs the value of all of the values in the y-th block of output gates of the circuit $\mathcal{C}$. These output bits are enough to recover some inverse of y under f, if it exists. $\square$

Finally, we need the following result of Valiant:

Theorem 14 (Valiant [Val77, Val92])**.** *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 [Vio09, Section 3] and Jukna [Juk12, Chapter 13] give detailed proofs of Theorem 14.

Now we can prove Theorem 3:

Proof of Theorem 3. First, observe that the inversion operator F_n^{inv} is explicit, since it is possible to compute any bit of its output with a linear scan over the input.

Under the hypothesis of Theorem 3, the operator F_n^{inv} has fan-in two Boolean circuits of size $O(n)$ and depth $O(\log n)$. Therefore, by Valiant's result (Theorem 14), F_n^{inv} has circuits in the common-bits model of width $w = O(n/\log\log n)$ and degree $d = n^{\epsilon}$, for any $\epsilon > 0$. For n of the form $n = N \log N$, where $N > 0$ is a power of two, we get $w = O(N \log N / \log\log(N \log N)) = O(N \log N / \log\log N)$ and $d = (N \log N)^{\epsilon} = O(N^{\epsilon'})$, for any $\epsilon' > \epsilon$. The theorem then follows from Lemma 13. $\square$

C Limits of current lower-bound approaches

C.1 Limits of the strong-multiscale-entropy method

Drucker [Dru12] shows, at the very least, that improving lower bounds in the common-bits model will require new types of arguments. In particular, Jukna [Juk12, Chapter 13], generalizing earlier arguments of Cherukhin [Che11] defined the "strong multiscale entropy" (SME) property of 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 $\epsilon > 0$.

C.2 Yao's box problem and limits of compression and presampling

Yao's "box problem" [Yao90a, NABT15] 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 [Yao90a]. 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 [Yao90a, GT00, DTT10, DGK17] or bit-fixing [Unr07, CDG18, CDGS18]. 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.

C.3 Barriers for other succinct data-structure problems

Brody and Larsen [BL15] showed that lower bounds on a certain class of dynamic data-structures (*linear data structures*) would imply lower bounds on the wire complexity of depth-two circuits with arbitrary gates.

[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.

Along similar lines, Corollary 5 gives a barrier to proving strong lower bounds for *any* so-called "succinct" data-structure problem [GM07], a class of static data-structure problems that includes function inversion. In fact the barrier applies to proving lower bounds against *systematic* data structures, which are a special case of succinct data structures. The barrier to proving lower bounds against weaker (i.e., systematic) data structures implies that the same barrier applies to proving lower bounds against stronger (i.e., succinct) data structures.

To explain how this barrier applies to a data-structure problem, consider the systematic variant of the standard problem of *polynomial evaluation with preprocessing* [Mil95]: On parameter $N \in \mathbb{Z}^{>0}$, let $\mathbb{F}$ be a finite field of size $\Theta(N)$. We are given a polynomial $p \in \mathbb{F}[X]$ of degree at most $N-1$, represented as its vector of coefficients $\bar{c} = (c_0, c_1, \dots, c_{N-1}) \in \mathbb{F}^N$. In a preprocessing phase, we may read these coefficients and produce an preprocessed S-bit data structure about p. In a subsequent online phase, we are given a point $x_0 \in \mathbb{F}$, and we must output the value $p(x_0) \in \mathbb{F}$ after reading the entire S-bit data structure, querying at most T coordinates of the coefficient vector $\bar{c}$, and performing an unlimited amount of computation.

It seems very difficult to construct an algorithm that simultaneously uses a data structure of size $S = o(N)$ and only $T = o(N)$ online queries. And yet, the best lower bound we have for this problem, implied by a bound of Gál and Miltersen [GM07], is of the form $ST = \widetilde{\Omega}(N)$. A variant of Corollary 5 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.

D Deferred material from Section 3

D.1 Relevance of NOF complexity to data structures

There is a close connection between communication complexity in the NOF model and circuit complexity in Valiant's common-bits model, which we introduced in Section 2.2. Pudlák et al. [PRS97] showed that an operator $F_n \colon \{0,1\}^n \to \{0,1\}^n$ can be computed by a small common-bits circuit if and only if there exists a low-communication three-party protocol that computes the i^{th} bit of the operator F_n, for some partition of the value i and the input to F_n. Applying the result of Pudlák et al. to Corollary 5 yields a NOF communication game that captures the hardness of the function-inversion problem. Proving a communication-complexity lower bound for this game would give a time-space lower bound for strongly non-adaptive algorithms with preprocessing for function inversion. Unfortunately, the resulting communication game is rather ad hoc, and we can say little about its communication complexity.

Therefore, rather than going through Valiant's common-bits model, Section 3 takes a more direct path, in the form of a reduction from the permutation variant of the pointer-jumping problem in communication complexity to the permutation-inversion problem.

Another relevant connection is an asymmetric variant of the three-party NOF model, introduced by Pătraşcu [Păt10], and later denoted by Chattopadhyay et al. [CEEP16] by $A \xrightarrow{B} (B \leftrightarrow C)$. In this model, the inputs are written on the foreheads of the players as in the standard NOF model. However, the first player is allowed to send a single "advice" message to the second player, after which only the second and the third player can communicate bidirectionally. Pătraşcu conjectured a lower bound for a variant of the set-disjointness problem in this model, and showed that such a lower bound implies new lower bounds on a plethora of data-structure problems. Chattopadhyay et al. [CEEP16] later refuted this conjecture, but used the model to prove the non-adaptive data-structure lower bounds obtained by Brody and Larsen [BL15] via different means.

D.2 The function case

One might have hoped for reducing the general (non-permutation) case of multiparty pointer jumping to the function-inversion problem, and then use a function-inversion algorithm with preprocessing [Hel80, FN99] to obtain a communication protocol. However, we could not construct the analogue of Theorem 8 for the function case. The problem is that for the reduction to succeed, the composition $f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_{k-1}^{-1}$, rather than $f_{k-1} \circ f_{k-2} \circ \cdots \circ f_1$, needs to be a function, and this is not true in the general case (unlike when $f_1, \ldots, f_k$ are all permutations, in which case both $f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_{k-1}^{-1}$ and $f_{k-1} \circ f_{k-2} \circ \cdots \circ f_1$ are permutations).

That said, even the permutation variant is often considered to be the hard case of the pointer-jumping problem [DJS98, BC08]. In addition, several upper bounds for the permutation case [PRS97, DJS98, BC08] led to subsequent upper bounds for the unrestricted case [BC08, BS15].

D.3 Omitted proofs from Section 3

Lemma 7 (restated)**.** $\mathsf{CC}^1(\mathsf{MPJ}_{N,k}^{\text{perm}}) \le \mathsf{CC}^1(\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}) + \lceil \log N \rceil$.

Proof. 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 $\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}$ is then enough—with a few extra bits of communication—to solve the Boolean-valued instance of $\mathsf{MPJ}_{N,k}^{\text{perm}}$.

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 $x|_i = 0 \ldots 0 1 x_{i-1} x_{i-2} \ldots x_1$. 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 i^{th} 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 2.

Therefore, given a communication protocol for $\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}$, we construct a protocol for $\mathsf{MPJ}_{N,k}^{\text{perm}}$ 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 $\widehat{\mathsf{MPJ}}_{N,k}^{\text{perm}}$ 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

Table 2: Example of the encoding procedure of Lemma 7. $N = 2^3$, 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

$\beta \circ \pi_{k-2} \circ \cdots \circ \pi_1(x) = \beta(\hat{x}) \in \{0,1\}$ 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 $\lceil \log N \rceil$. $\square$

Lemma 8 (restated). *If there exists a $(k-2)$-round adaptive algorithm for inverting permutations $\pi : [N] \to [N]$ that uses advice S and time T, then $\mathsf{CC}^1(\widehat{\mathsf{MPJ}}^{\text{perm}}_{N,k}) \leq S + 2T\lceil \log N \rceil$.*

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 $\widehat{\mathsf{MPJ}}^{\text{perm}}_{N,k}$.

- 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 $r_{i-1,\ell} = \pi_1^{-1}(\ldots(\pi_{i-1}^{-1}(p_{i-1,\ell}))\ldots)$, and writes them down on the blackboard. 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, by this time, have already been 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 answers 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 $\widehat{\mathsf{MPJ}}^{\text{perm}}_{N,k}$.

The communication consists of the advice string written by Player P_0 as well as a partial reply and a complete reply for each query, giving a total of $S + 2T\lceil \log N \rceil$. (The last player writes the $\lceil \log N \rceil$-bit output, but does not need to write the response to the T-th query). $\square$

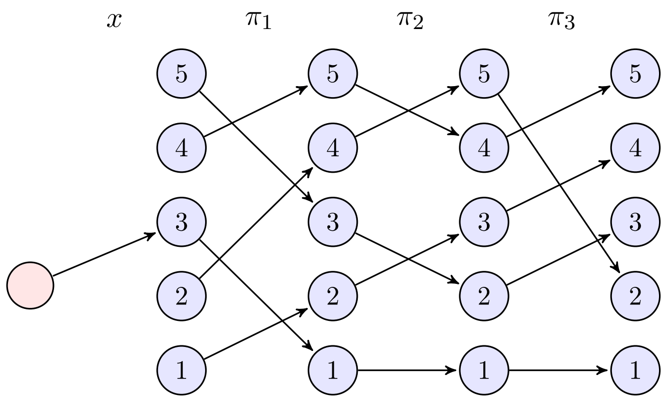

Figure 3: A pointer-jumping instance for $\widehat{\mathsf{MPJ}}^{\text{perm}}_{k=4,N=5}$ with $\pi_1 = (1\,2\,4\,5\,3)$, $\pi_2 = (2\,3)(4\,5)$, $\pi_3 = (2\,3\,4\,5)$ and $x = 2$. Lemma 7 reduces this instance to inverting the permutation $\pi_1^{-1}\pi_2^{-1}\pi_3^{-1} = (1\,3\,5\,4)$ on the point $x = 2$.

E Background: Cycle walking and Hellman tables

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

This algorithm, which Yao [Yao90a] makes explicit, is implicit in Hellman's more general algorithm for inverting random functions with preprocessing [Hel80]. This cycle-walking algorithm also serves as a building-block for other inversion algorithms [FN99, Oec03].

Theorem 15 (Hellman [Hel80])**.** *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 i^{th} 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 [Hel80] 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 [DTT10, Section 2.1] for details.

F Proof of Theorem 9

Theorem 9 (restated)**.** *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.

The following lemma proves the first part of the theorem.

Lemma 16. *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 9. We show (roughly) in Lemma 17 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 18 that inverting a length-increasing function $f\colon [N] \to [N^c]$, for constant $c > 1$ is enough to solve substring search.

Lemma 17. *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 17 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 $h_1, \ldots, h_k$ and run the preprocessing algorithm on each. Then, as long as there exists an $h_i \in \{h_1, \ldots, h_k\}$ 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 [Hel80] and features in the work of Fiat and Naor [FN99] and Oechslin's Rainbow tables [Oec03]. 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 [CW79], 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 $i \in \{1, \dots, k\}$:
  - 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 $h_i \in \{h_1, \dots, h_k\}$ 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 $\{1, \dots, N\}$. 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 $h_1, \dots, h_k$ 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 18. *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 9:

Proof of Theorem 9. Lemma 16 proves the first part of the theorem.

To prove the second part: Lemma 17 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 18 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. □

G Breaking PRGs with preprocessing

De, Trevisan, and Tulsiani [DTT10] introduced the problem of breaking pseudorandom generators (PRGs) with preprocessing. In that problem, we are given oracle access to a function $G\colon [N] \to [M]$, for $N < M$, that we think of as a "black-box" PRG.

We can then define a black-box PRG distinguisher with preprocessing $(\mathcal{A}_0, \mathcal{A}_1)$ as follows: In the preprocessing phase, the algorithm $\mathcal{A}_0$ may make arbitrarily many queries to G and then must output an S-bit advice string st_G. In the online phase, the algorithm $\mathcal{A}_1$ takes as input (1) the advice string st_G and (2) a point $y \in [M]$, and it must distinguish whether y has been drawn from the distribution $\{G(x) \mid x \xleftarrow{\text{R}} [N]\}$ or the distribution $\{y \mid y \xleftarrow{\text{R}} [M]\}$. More formally, we define the *distinguishing advantage* of a PRG distinguisher with preprocessing $(\mathcal{A}_0, \mathcal{A}_1)$ with respect to G as follows:

$$\mathsf{PRGadv}\big[(\mathcal{A}_0, \mathcal{A}_1), G\big] := \left| \Pr_{x \xleftarrow{\text{R}} [N]} \left[\mathcal{A}_1^G\left(\mathcal{A}_0^G(), G(x)\right) = 1\right] - \Pr_{y \xleftarrow{\text{R}} [M]} \left[\mathcal{A}_1^G\left(\mathcal{A}_0^G(), y\right) = 1\right] \right| .$$

De, Trevisan, and Tulsiani [DTT10] and Dodis, Guo, and Katz [DGK17] prove that algorithms that achieve PRG distinguishing advantage ϵ, must satisfy $ST = \widetilde{\Omega}(\epsilon^2 N)$, while the lower bound for inverting functions with probability ϵ is $ST = \widetilde{\Omega}(\epsilon N)$.

For what range of these parameters can we achieve the PRG lower bound? For sub-constant advantage $\epsilon \le 1/\sqrt{N}$, De, Trevisan, and Tulsiani give a black-box PRG distinguisher with preprocessing that satisfies $S = \widetilde{O}(\epsilon^2 N)$ and $T = \widetilde{O}(1)$. They ask whether there are other distinguishers that match the lower bound. The following theorem shows that devising, for example, an $S = T = \widetilde{O}(\sqrt{N})$ black-box PRG distinguisher with constant distinguishing advantage ϵ would imply a very powerful (i.e., better than Hellman) algorithm for inverting one-to-one functions:

Theorem 11 (restated formally)**.** *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 one-to-one function $f\colon [N] \to [M]$ using $\widetilde{O}(\epsilon^{-2} S)$ bits advice, $\widetilde{O}(\epsilon^{-2} T)$ online queries, and a random oracle, and that inverts f with probability $1 - 1/\log N$ (over the choice of the random oracle). Furthermore, the online phase of the inversion algorithm makes $\widetilde{O}(\epsilon^{-2} T)$ queries to the random oracle.*

Proof. In the following discussion, 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.) For every $i \in [n]$, and for every $z \in \{0,1\}^n$ let $[z]_{i\to 1}$ denote z with its i^{th} bit z_i set to 1.

Let $(\mathcal{A}_0, \mathcal{A}_1)$ be a distinguisher for any length-increasing generator $G\colon [N] \to [M]$ such that $\mathsf{PRGadv}\left[(\mathcal{A}_0, \mathcal{A}_1), G\right] \ge \epsilon$. We construct an inversion algorithm for one-to-one 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 i^{th} bit of x. We then use the algorithms $(\mathcal{B}_1, \dots, \mathcal{B}_n)$ 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 i^{th} bit of x is 1. Then, we can apply the PRG distinguisher to G_i recover the i^{th} 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 i^{th} bit of the x with high probability. To complete the construction, we instantiate the permutations π and σ using a random oracle.

We now give the construction of the bit-recovery algorithm $\mathcal{B}_i$, for every $i \in [n]$. Given access to random permutations π and σ, the algorithm $\mathcal{B}_i = (\mathcal{B}_{i0}, \mathcal{B}_{i1})$ 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.

Claim 19. *For every one-to-one 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 i^{th} 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

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

and

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

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 one-to-one 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

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

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

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

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

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

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

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

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

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

The claim follows from (3) and (6). □

We can now use the algorithms $(\mathcal{B}_1, \dots, \mathcal{B}_n)$ to construct an inversion algorithm $\mathcal{I}$, that makes use of a random oracle $\mathcal{O}$. The inverter uses a parameter k, which we choose later, and operates as follows:

- **Preprocessing.**
  For $j \in \{1, \dots, k\}$:
  - Derive from the random oracle $\mathcal{O}$ two random permutations π_j and σ_j using standard techniques [CPS08].
  - For each $i \in \{1, \dots, n\}$, 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 $j \in \{1, \dots, k\}$:
  - Derive from the random oracle $\mathcal{O}$ two random permutations π_j and σ_j.
  - For each $i \in \{1, \dots, n\}$, compute a guess of the i^{th} 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 one-to-one, there is exactly one such preimage.
  - Let $\hat{b}_i \in \{0, 1\}$ be the majority vote of the bits $\{b_{1j}, \dots, b_{nj}\}$.

  Finally, output $\hat{x} = \hat{b}_1\hat{b}_2 \dots \hat{b}_n \in \{0, 1\}^n$.

Claim 20. *For every one-to-one 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 i^{th} 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

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

The resulting inverter succeeds with probability $1 - 1/\log N$. Furthermore 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 [CPS08], therefore the number of online queries to the random oracle is also $O(\epsilon^{-2} S \log N \log\log N)$. $\square$

The last proof can also be adapted to the setting in which the given distinguisher only inverts a random PRG. Moreover, as a random function $f\colon [N] \to [M]$ with $M > \Omega(N^2)$ is one-to-one with high probability, a similar argument shows that a distinguisher for random PRGs can also be used to invert random functions whose co-domain is sufficiently larger than their domain.

References

[AACKPR17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. "Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space". In: *ASIACRYPT* 2017. DOI: 10.1007/978-3-319-70697-9_13. Available at Cryptology ePrint Archive, Report 2017/893.

[BHK01] László Babai, Thomas P. Hayes, and Peter G. Kimmel. "The Cost of the Missing Bit: Communication Complexity with Help". In: *Combinatorica* 21.4 (2001), pp. 455–488. DOI: 10.1007/s004930100009.

[BHMS11] Jérémy Barbay, Meng He, J. Ian Munro, and Srinivasa Rao Satti. "Succinct Indexes for Strings, Binary Relations and Multilabeled Trees". In: *ACM Transactions on Algorithms* 7.4 (2011), 52:1–52:27. DOI: 10.1145/2000807.2000820.

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. "Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs". In: *CRYPTO* 2006. DOI: 10.1007/11818175_1.

[BT94] Richard Beigel and Jun Tarui. "On ACC". In: *Computational Complexity* 4 (1994), pp. 350–366. DOI: 10.1007/BF01263423.

[BS00] Alex Biryukov and Adi Shamir. "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers". In: *ASIACRYPT* 2000. DOI: 10.1007/3-540-44448-3_1.

[BSW00] Alex Biryukov, Adi Shamir, and David A. Wagner. "Real Time Cryptanalysis of A5/1 on a PC". In: *FSE* 2000. DOI: 10.1007/3-540-44706-7_1.

[BN16] Elette Boyle and Moni Naor. "Is There an Oblivious RAM Lower Bound?" In: *ITCS* 2016. DOI: 10.1145/2840728.2840761. Available at Electronic Colloquium on Computational Complexity, Report 2015/146.

[Bro09] Joshua Brody. "The Maximum Communication Complexity of Multi-party Pointer Jumping". In: *CCC* 2009. DOI: 10.1109/CCC.2009.30. Available at Electronic Colloquium on Computational Complexity, Report 2009/017.

[BC08] Joshua Brody and Amit Chakrabarti. "Sublinear Communication Protocols for Multi-party Pointer Jumping and a Related Lower Bound". In: *STACS* 2008. DOI: 10.4230/LIPIcs.STACS.2008.1341. arXiv: 0802.2843.

[BL15] Joshua Brody and Kasper Green Larsen. "Adapt or Die: Polynomial Lower Bounds for Non-Adaptive Dynamic Data Structures". In: *Theory of Computing* 11.19 (2015), pp. 471–489. DOI: 10.4086/toc.2015.v011a019.

[BS15] Joshua Brody and Mario Sanchez. "Dependent Random Graphs and Multi-Party Pointer Jumping". In: *APPROX/RANDOM* 2015. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2015.606.

[CW79] Larry Carter and Mark N. Wegman. "Universal Classes of Hash Functions". In: *Journal of Computer and System Sciences* 18.2 (1979), pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8.

[CFL83] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. "Multi-Party Protocols". In: *STOC* 1983. DOI: 10.1145/800061.808737.

[CEEP16] Arkadev Chattopadhyay, Jeff Edmonds, Faith Ellen, and Toniann Pitassi. "Upper and Lower Bounds on the Power of Advice". In: *SIAM Journal on Computing* 45.4 (2016), pp. 1412–1432. DOI: 10.1137/15M1031862.

[Che11] Dmitriy Yu. Cherukhin. "Lower bounds for the complexity of Boolean circuits of finite depth with arbitrary elements". In: *Discrete Mathematics and Applications* 23.4 (2011), pp. 39–47. DOI: 10.1515/dma.2011.031. Available at Electronic Colloquium on Computational Complexity, Report 2008/032.

[CM96] David R. Clark and J. Ian Munro. "Efficient Suffix Trees on Secondary Storage". In: *SODA* 1996.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. "Non-Uniform Bounds in the Random-Permutation, Ideal-Cipher, and Generic-Group Models". In: *CRYPTO* 2018. DOI: 10.1007/978-3-319-96884-1_23. Available at Cryptology ePrint Archive, Report 2018/226.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. "Random Oracles and Non-uniformity". In: *EUROCRYPT* 2018. DOI: 10.1007/978-3-319-78381-9_9. Available at Cryptology ePrint Archive, Report 2017/937.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. "The Random Oracle Model and the Ideal Cipher Model Are Equivalent". In: *CRYPTO* 2008. DOI: 10.1007/978-3-540-85174-5_1. Available at Cryptology ePrint Archive, Report 2008/246.

[DJS98] Carsten Damm, Stasys Jukna, and Jiří Sgall. "Some Bounds on Multiparty Communication Complexity of Pointer Jumping". In: *Computational Complexity* 7.2 (1998), pp. 109–127. DOI: 10.1007/PL00001595.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. "Time Space Tradeoffs for Attacks against One-Way Functions and PRGs". In: *CRYPTO* 2010. DOI: 10.1007/978-3-642-14623-7_35. Available at Electronic Colloquium on Computational Complexity, Report 2009/113.

[DLO03] Erik D. Demaine and Alejandro López-Ortiz. "A linear lower bound on index size for text retrieval". In: *Journal of Algorithms* 48.1 (2003), pp. 2–15. DOI: 10.1016/S0196-6774(03)00043-9.

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. "Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited". In: *EUROCRYPT* 2017. DOI: 10.1007/978-3-319-56614-6_16.

[Dru12] Andrew Drucker. "Limitations of Lower-Bound Methods for the Wire Complexity of Boolean Operators". In: *CCC* 2012. DOI: 10.1109/CCC.2012.39. Available at Electronic Colloquium on Computational Complexity, Report 2011/125.

[FN91] Amos Fiat and Moni Naor. "Rigorous Time/Space Tradeoffs for Inverting Functions". In: *STOC* 1991. DOI: 10.1145/103418.103473.

[FN99] Amos Fiat and Moni Naor. "Rigorous Time/Space Trade-Offs for Inverting Functions". In: *SIAM Journal on Computing* 29.3 (1999), pp. 790–803. DOI: 10.1137/S0097539795280512.

[GM03] Anna Gál and Peter Bro Miltersen. "The cell probe complexity of succinct data structures". In: *ICALP* 2003. DOI: 10.1007/3-540-45061-0_28.

[GM07] Anna Gál and Peter Bro Miltersen. "The cell probe complexity of succinct data structures". In: *Theoretical Computer Science* 379.3 (2007), pp. 405–417. DOI: 10.1016/j.tcs.2007.02.047.

[GRR06] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. "Succinct Ordinal Trees with Level-Ancestor Queries". In: *ACM Transactions on Algorithms* 2.4 (2006), pp. 510–534. DOI: 10.1145/1198513.1198516.

[GGKT05] Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. "Bounds on the Efficiency of Generic Cryptographic Constructions". In: *SIAM Journal on Computing* 35.1 (2005), pp. 217–246. DOI: 10.1137/S0097539704443276.

[GT00] Rosario Gennaro and Luca Trevisan. "Lower Bounds on the Efficiency of Generic Cryptographic Constructions". In: *FOCS* 2000. DOI: 10.1109/SFCS.2000.892119. Available at Cryptology ePrint Archive, Report 2000/017.

[Gol87] Oded Goldreich. "Towards a Theory of Software Protection and Simulation by Oblivious RAMs". In: *STOC* 1987. DOI: 10.1145/28395.28416.

[GO96] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *Journal of the ACM* 43.3 (1996), pp. 431–473. DOI: 10.1145/233551.233553.

[Gol07] Alexander Golynski. *Stronger lower bounds for text searching and polynomial evaluation.* Tech. rep. CS-2007-25. University of Waterloo, Cheriton School of Computer Science, 2007. URL: https://cs.uwaterloo.ca/research/tr/2007/CS-2007-25.pdf.

[Gol09] Alexander Golynski. "Cell Probe Lower Bounds For Succinct Data Structures". In: *SODA* 2009. DOI: 10.1137/1.9781611973068.69.

[GS05] Navin Goyal and Michael Saks. "A Parallel Search Game". In: *Random Structures & Algorithms* 27.2 (2005), pp. 227–234. DOI: 10.1002/rsa.20068.

[GOR10] Roberto Grossi, Alessio Orlandi, and Rajeev Raman. "Optimal Trade-Offs for Succinct String Indexes". In: *ICALP* 2010. DOI: 10.1007/978-3-642-14165-2_57. arXiv: 1006.5354.

[HMS12] Meng He, J. Ian Munro, and Srinivasa Rao Satti. "Succinct ordinal trees based on tree covering". In: *ACM Transactions on Algorithms* 8.4 (2012), 42:1–42:32. DOI: 10.1145/2344422.2344432.

[Hel80] Martin Hellman. "A Cryptanalytic Time-Memory Trade-Off". In: *IEEE Transactions on Information Theory* 26.4 (1980), pp. 401–406. DOI: 10.1109/TIT.1980.1056220.

[HG91] Johan Håstad and Mikael Goldmann. "On the Power of Small-Depth Threshold Circuits". In: *Computational Complexity* 1 (1991), pp. 113–129. DOI: 10.1007/BF01272517.

[Jac89] Guy Jacobson. "Space-efficient Static Trees and Graphs". In: *FOCS* 1989. DOI: 10.1109/SFCS.1989.63533.

[Juk12] Stasys Jukna. In: *Boolean Function Complexity. Advances and Frontiers.* Algorithms and Combinatorics 27. 2012. DOI: 10.1007/978-3-642-24508-4.

[JS11] Stasys Jukna and Georg Schnitger. "Min-rank conjecture for log-depth circuits". In: *Journal of Computer and System Sciences* 77.6 (2011), pp. 1023–1038. DOI: 10.1016/j.jcss.2009.09.003. arXiv: 1005.1009.

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. "Yes, There is an Oblivious RAM Lower Bound!" In: *CRYPTO* 2018. DOI: 10.1007/978-3-319-96881-0_18. Available at Cryptology ePrint Archive, Report 2018/423.

[Lia14] Hongyu Liang. "Optimal Collapsing Protocol for Multiparty Pointer Jumping". In: *Theory of Computing Systems* 54.1 (2014), pp. 13–23. DOI: 10.1007/s00224-013-9476-x.

[Mil95] Peter Bro Miltersen. "On the cell probe complexity of polynomial evaluation". In: *Theoretical Computer Science* 143.1 (1995), pp. 167–174. DOI: 10.1016/0304-3975(95)80032-5.

[MRRR12] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. "Succinct representations of permutations and functions". In: *Theoretical Computer Science* 438 (2012), pp. 74–88. DOI: 10.1016/j.tcs.2012.03.005.

[NS05] Arvind Narayanan and Vitaly Shmatikov. "Fast dictionary attacks on passwords using time-space tradeoff". In: *CCS* 2005. DOI: 10.1145/1102120.1102168.

[NABT15] Aran Nayebi, Scott Aaronson, Aleksandrs Belovs, and Luca Trevisan. "Quantum Lower Bound for Inverting a Permutation with Advice". In: *Quantum Information & Computation* 15.11-12 (2015), pp. 901–913. arXiv: 1408.3193.

[Oec03] Philippe Oechslin. "Making a Faster Cryptanalytic Time-Memory Trade-Off". In: *CRYPTO* 2003. DOI: 10.1007/978-3-540-45146-4_36.

[Ost90] Rafail Ostrovsky. "Efficient Computation on Oblivious RAMs". In: *STOC* 1990. DOI: 10.1145/100216.100289.

[Păt10] Mihai Pătraşcu. "Towards polynomial lower bounds for dynamic problems". In: *STOC* 2010. DOI: 10.1145/1806689.1806772.

[PRS97] Pavel Pudlák, Vojtech Rödl, and Jiří Sgall. "Boolean circuits, tensor ranks, and communication complexity". In: *SIAM Journal on Computing* 26.3 (1997), pp. 605–633. DOI: 10.1137/S0097539794264809.

[SG06] Kunihiko Sadakane and Roberto Grossi. "Squeezing succinct data structures into entropy bounds". In: *SODA* 2006. DOI: 10.1145/1109557.1109693.

[Unr07] Dominique Unruh. "Random Oracles and Auxiliary Input". In: *CRYPTO* 2007. DOI: 10.1007/978-3-540-74143-5_12. Available at Cryptology ePrint Archive, Report 2007/168.

[Val77] Leslie G. Valiant. "Graph-theoretic arguments in low-level complexity". In: *MFCS* 1977. DOI: 10.1007/3-540-08353-7_135.

[Val92] Leslie G. Valiant. “Why is Boolean Complexity Theory Difficult”. In: *Boolean Function Complexity.* London Mathematical Society Lecture Note Series 169. 1992, pp. 84–94. DOI: 10.1017/cbo9780511526633.008.

[Vio09] Emanuele Viola. “On the Power of Small-Depth Computation”. In: *Foundations and Trends in Theoretical Computer Science* 5.1 (2009), pp. 1–72. DOI: 10.1561/0400000033.

[VW09] Emanuele Viola and Avi Wigderson. “One-way multiparty communication lower bound for pointer jumping with applications”. In: *Combinatorica* 29.6 (2009), pp. 719–743. DOI: 10.1007/s00493-009-2667-z.

[Wee05] Hoeteck Wee. “On Obfuscating Point Functions”. In: *STOC* 2005. DOI: 10.1145/1060590.1060669. Available at Cryptology ePrint Archive, Report 2005/001.

[WW18] Mor Weiss and Daniel Wichs. *Is there an Oblivious RAM Lower Bound for Online Reads?* Cryptology ePrint Archive, Report 2018/619. 2018.

[Wig96] Avi Wigderson. unpublished. 1996.

[Yao81] Andrew Chi-Chih Yao. “Should Tables Be Sorted?” In: *Journal of the ACM* 28.3 (1981), pp. 615–628. DOI: 10.1145/322261.322274.

[Yao90a] Andrew Chi-Chih Yao. “Coherent Functions and Program Checkers (Extended Abstract)”. In: *STOC* 1990. DOI: 10.1145/100216.100226.

[Yao90b] Andrew Chi-Chih Yao. “On ACC and Threshold Circuits”. In: *FOCS* 1990. DOI: 10.1109/FSCS.1990.89583.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il