Electronic Colloquium on Computational Complexity, Report No. 145 (2022)

Revisiting Time-Space Tradeoffs for Function Inversion

Alexander Golovnev* Siyao Guo† Spencer Peters‡ Noah Stephens-Davidowitz§

Abstract

We study the black-box function inversion problem, which is the problem of finding $x \in [N]$ such that $f(x) = y$, given as input some challenge point y in the image of a function $f : [N] \to [N]$, using T oracle queries to f *and* preprocessed advice $\sigma \in \{0,1\}^S$ depending on f. We prove a number of new results about this problem, as follows.

1. We show an algorithm that works for any T and S satisfying
$$TS^2 \cdot \max\{S, T\} = \widetilde{\Theta}(N^3) \ .$$
In the important setting when $S < T$, this improves on the celebrated algorithm of Fiat and Naor [STOC, 1991], which requires $TS^3 \gtrsim N^3$. E.g., Fiat and Naor's algorithm is only non-trivial for $S \gg N^{2/3}$, while our algorithm gives a non-trivial tradeoff for any $S \gg N^{1/2}$. (Our algorithm and analysis are quite simple. As a consequence of this, we also give a self-contained and simple proof of Fiat and Naor's original result, with certain optimizations left out for simplicity.)
2. We show a *non-adaptive* algorithm (i.e., an algorithm whose ith query x_i is chosen based entirely on σ and y, and not on the $f(x_1), \ldots, f(x_{i-1})$) that works for any T and S satisfying
$$S = \Theta(N \log(N/T)) \ ,$$
giving the first non-trivial non-adaptive algorithm for this problem. E.g., setting $T = N/\mathrm{poly}\log(N)$ gives $S = \Theta(N \log\log N)$. This answers a question due to Corrigan-Gibbs and Kogan [TCC, 2019], who asked whether it was possible for a non-adaptive algorithm to work with parameters T and S satisfying $T + S/\log N < o(N)$. We also observe that our non-adaptive algorithm is what we call a *guess-and-check* algorithm, that is, it is non-adaptive *and* its final output is always one of the oracle queries $x_1, \ldots, x_T$. For guess-and-check algorithms, we prove a matching lower bound, therefore completely characterizing the achievable parameters (S, T) for this natural class of algorithms. (Corrigan-Gibbs and Kogan showed that any such lower bound for *arbitrary* non-adaptive algorithms would imply new circuit lower bounds.)
3. We show equivalence between function inversion and a natural decision version of the problem in both the worst case and the average case, and similarly for functions $f : [N] \to [M]$ with different ranges.

All of the above results are most naturally described in a model with *shared randomness* (i.e., random coins shared between the preprocessing algorithm and the online algorithm). However, as an additional contribution, we show (using a technique from communication complexity due to Newman [IPL, 1991]) how to generically convert any algorithm that uses shared randomness into one that does not.

*Georgetown University. Email: alexgolovnev@gmail.com.

†NYU Shanghai. Email: sg191@nyu.edu. Supported by National Natural Science Foundation of China Grant No. 62102260, Shanghai Municipal Education Commission (SMEC) Grant No. 0920000169, NYTP Grant No. 20121201 and NYU Shanghai Boost Fund.

‡Cornell University. Email: sp2473@cornell.edu. Supported in part by the NSF under Grant No. CCF-2122230.

§Cornell University. Email: noahsd@gmail.com. Supported in part by the NSF under Grant No. CCF-2122230.

ISSN 1433-8092

Contents

1 Introduction

We revisit the fundamental problem of black-box *function inversion*. That is, we study the problem of finding $x \in [N]$ such that $f(x) = y$, given as input some challenge point y in the image of $f : [N] \to [N]$ and oracle access to f.

Of course, given *only* oracle access to f, inverting general functions f will clearly require roughly N queries, which is not very interesting. However, if we allow our inversion algorithm access to some additional information about f, then inversion might be possible with *much* fewer queries. So, we consider the following model. First, using unlimited computational power, a *preprocessing* algorithm $\mathcal{P}$ analyzes f and outputs S bits of *advice* $\sigma \in \{0,1\}^S$. Then, an *online* algorithm $\mathcal{A}$ is given a point y in the image of f, the advice σ, and oracle access to f and, using at most T oracle queries, must output some x such that $f(x) = y$. We wish to design such algorithms that minimize the complexity measures S and T, which are often referred to informally as "space" and "time." For example, notice that it is trivial to invert f if $S/\log N + T \geq N$, by simply including the first $S/\log N$ values of f as advice and querying the remaining $N - S/\log N \leq T$ values.

This model is very well studied, since it arises naturally in a number of contexts, from cryptography [Hel80, Yao90, FN91, GT00, Wee05, Unr07, DTT10, DGK17, CDGS18, CDG18, CK19] (where an appropriate version of this problem corresponds to the problem of breaking a black-box one-way function in the non-uniform model) to data structures and complexity theory [Yao90, CK19, GGH+20, DKKS21]. Indeed, many variants of the problem have been studied. For example, we might ask for algorithms that invert arbitrary functions f [FN91], random functions f [Hel80] (in which case the algorithm should work with reasonable probability over the function f), or special classes of functions f, like permutations [Yao90]; or one might place restrictions on the algorithm by, e.g., requiring the oracle queries to be non-adaptive [CK19, CHM20] or requiring that the algorithm otherwise has some special structure [BBS06]. Other work has considered stronger models of computation, such as quantum algorithms [NABT15, CLQ20, CGLQ20].

In his celebrated 1980 work, Hellman [Hel80] published the first non-trivial function inversion algorithm. Hellman's algorithm inverts random functions for any S and T satisfying $TS^2 \gtrsim N^2$, under certain heuristic assumptions. (Here and elsewhere in the introduction, we use $\gtrsim$ to represent an inequality that holds up to factors polylogarithmic in N.) In their seminal 1991 paper, Fiat and Naor [FN91] presented an algorithm that (1) provably achieves Hellman's tradeoff for random functions f; and (2) achieves a different non-trivial tradeoff for *any* function f. Specifically, their algorithm can invert any function f provided that S and T satisfy

$$TS^3 \gtrsim N^3 \,. \tag{1}$$

For example, when $T = S$, this works for any $S = T \gtrsim N^{3/4}$, while the result becomes trivial for $S \lesssim N^{2/3}$ (since in that case they require $T \geq N$, which can be matched by the trivial algorithm).

Despite thirty years of effort, no improvements have been made to Eq. (1). This has naturally led to a search for matching *lower bounds*. Indeed, Barkan, Biham, and Shamir showed that Hellman's algorithm (or Fiat and Naor's variant with proven correctness) gives essentially the optimal tradeoff between S and T for inverting random functions if we restrict our attention to a certain rather specific class of algorithms [BBS06]. However, the best known lower bound [Yao90, GT00, DTT10] against arbitrary algorithms (which applies for random functions and even random permutations) only says that S and T must satisfy

$$ST \gtrsim N \,, \tag{2}$$

which is much weaker than Eq. (1). (While the lower bound in Eq. (2) is quite far from the best upper bounds known for arbitrary functions or even for random functions, it is actually known to be tight in the special case when the function f is a permutation [Hel80].)

Corrigan-Gibbs and Kogan explained the lack of progress on lower bounds by showing that any significant improvement to the lower bound in Eq. (2) would yield a breakthrough in circuit lower bounds [CK19]. (See also [DKKS21], which showed that lower bounds on function inversion are closely related to many other major open problems, such as the hardness of sorting and the Network Coding Conjecture.) In fact, [CK19] showed that even a lower bound against *non-adaptive* algorithms that improves upon Eq. (2) would imply new circuit lower bounds. An online algorithm $\mathcal{A}$ is *non-adaptive* if the queries $x_1, \ldots, x_T$ that it makes to its oracle are functions only of its input y, the preprocessed advice σ, and shared randomness r—i.e., if x_{i+1} is chosen independently of the answers $f(x_1), \ldots, f(x_i)$ to the previous queries. This result is quite tantalizing because (1) all of the non-trivial algorithms described above rely crucially on adaptive queries; (2) very strong lower bounds are in fact known for slightly weaker models [CHM20]; and (3) it seems intuitively clear that non-adaptive algorithms should not be able to do much better than the trivial algorithm, which requires $S/\log N + T \geq N$. (Notice that in the context of non-adaptive algorithms, we do not leave out logarithmic factors, as even small improvements are interesting here.) Indeed, Corrigan-Gibbs and Kogan naturally speculated that no non-adaptive algorithm can do significantly better than the trivial algorithm—specifically, that no non-adaptive algorithm can solve function inversion with $S < o(N \log N)$ and $T < o(N)$.

1.1 Our results

Improving on the Fiat-Naor algorithm for $T > S$. Our first main result is an algorithm that inverts any function $f : [N] \to [N]$ on any challenge y for any T and S satisfying

$$T^2 S^2 \gtrsim N^3 \,. \tag{3}$$

Recall that the original Fiat-Naor algorithm requires $TS^3 \gtrsim N^3$ (as in Eq. (1)). So, our algorithm is better than Fiat and Naor's algorithm if (and only if) $T > S$. This is arguably the most interesting setting, since non-uniform advice is arguably a more expensive resource than queries (as Hellman pointed out in [Hel80]).[1] In particular, our algorithm remains non-trivial (i.e., outperforms the trivial algorithm that requires $S + T \gtrsim N$) as long as $S \gtrsim N^{1/2}$, whereas the original Fiat-Naor algorithm is trivial for $S \lesssim N^{2/3}$.

In fact, our algorithm is a surprisingly simple variant of Fiat and Naor's original. Our presentation of the algorithm and its analysis is also notably simpler. So, as an additional benefit, we also give a significantly simpler presentation of the original result in [FN91].[2] Indeed, we present the two algorithms together, as a single algorithm (that behaves differently in one step depending on whether $S > T$) that solves function inversion for any S and T satisfying

$$TS^2 \cdot \max\{S, T\} \gtrsim N^3 \,. \tag{4}$$

[1] However, a big part of the reason that advice is considered to be expensive is because memory is often considered to be more expensive than computing time. Unfortunately, though our algorithm can use much less than T bits of advice, our online algorithm still must use roughly T bits of space. So, though we do show an algorithm that uses less advice, we do not show an algorithm that uses less space.

[2] Admittedly, this simplicity is partially (though not entirely) due to the fact that we chose not to optimize for parameters other than S and T, while Fiat and Naor were quite careful to optimize, e.g., the actual running time and space of both the query algorithm and the preprocessing algorithm. See Section 1.4 for more discussion.

In other words, we give a unified presentation that achieves the best of both worlds, matching the original tradeoff achieved by Fiat and Naor in Eq. (1) and our new tradeoff in Eq. (3).

A better-than-trivial non-adaptive algorithm, and a matching lower bound. Our second main result addresses Corrigan-Gibbs and Kogan's question about whether non-trivial non-adaptive algorithms are possible. Though [CK19] naturally speculated that such algorithms were not possible, we show an algorithm that (slightly) outperforms the trivial algorithm.

Specifically, we show a *non-adaptive* algorithm that inverts *any* function f using T queries for any $T \leq N$ and preprocessed advice with bit length

$$S = O(N \log(N/T)) \,. \tag{5}$$

For example, for $T = N/\log^C N$ for any constant $C > 0$, our algorithm uses $S = O(N \log\log N)$ bits of advice. Recall that the trivial algorithm (which is also non-adaptive) requires $S/\log N + T \geq N$. So, our algorithm beats the trivial algorithm by a polylogarithmic factor. As another example, we can take, e.g., $S = O(N \log\log\log N)$ and $T = O(N/\log\log N)$.

Our algorithm is relatively simple. (See Section 1.2 for a high-level description and Section 4 for the details.) In particular, it has the natural property that it always outputs one of its query points x_i. In other words, the online algorithm takes as input a challenge point $y \in [N]$ in the image of f and preprocessed advice $\sigma \in \{0,1\}^S$ (and any shared randomness), uses these to (non-adaptively!) choose a list of queries $x_1, \ldots, x_T \in [N]$, and outputs an x_i such that $f(x_i) = y$ (if one exists). We call algorithms with this property *guess-and-check (non-adaptive) algorithms*, since such an algorithm can be viewed as using its queries to *check* whether any of its *guesses* $x_1, \ldots, x_T$ are in fact a preimage of y.[3]

To our knowledge, we are the first to consider this class of algorithms, though we find them to be quite natural. For example, we note in passing that guess-and-check algorithms can be thought of as "highly parallel algorithms" in the sense that they capture the model in which T processors independently compute and check one potential preimage of y (i.e., one "guess") each $x_1, ..., x_T$, and the algorithm succeeds if and only if any of these processors discovers that x_i is in fact a preimage of y. Indeed, [CK19] introduced non-adaptive algorithms in part because of their relationship with parallelism. (Other special classes of non-adaptive algorithms were studied in [CK19] and [CHM20], but none of the previously defined classes captures guess-and-check algorithms, as we explain in Section 1.3.)

Our third main contribution is a lower bound showing that no *guess-and-check* algorithm can do significantly better than Eq. (5) (even for inverting permutations). Specifically, we show that Eq. (5) is tight up to a constant factor in S and T. We therefore characterize the query-preprocessing tradeoff for guess-and-check non-adaptive function inversion up to a constant factor. If our lower bound could be extended to general non-adaptive algorithms, it would imply new strong circuit lower bounds, using the result of Corrigan-Gibbs and Kogan [CK19].[4]

Search-to-decision reductions. Next, we consider a natural variant of function inversion, which we call *decision function inversion* (DFI). In DFI, the goal is simply to determine whether

[3] In general, a non-adaptive algorithm might output x^* that depends in a more complicated way on $f(x_1), \ldots, f(x_T)$, and in particular might succeed even when $f(x_i) \neq y$ for all x_i.

[4] In fact, two of the authors of the present work originally proved the lower bound without realizing the distinction between general non-adaptive algorithms and guess-and-check algorithms. We were quite excited to have proven a new circuit lower bound, before we realized this subtle error.

the input point $y \in [M]$ is in the image of a function $f : [N] \to [M]$, given oracle access to f, shared randomness r, and S bits of preprocessed advice σ that may depend on r and f. (Notice that in the context of DFI, it is natural to consider functions with a range $[M]$ for $M \gg N$. In Appendix A, we show that many versions of function inversion are equivalent to their respective variants when the range is changed.) Given the very slow progress on the *search* function inversion (SFI) problem that we discussed above, it is natural to ask whether the decision variant is any easier.

Unfortunately, we show that this cannot be the case—for either random functions or worst-case functions. Specifically, we show a reduction from average-case SFI to average-case DFI (in which both the function and the target are uniformly random, as in definitions Definitions 2.4 and 2.5), and a reduction from worst-case SFI to worst-case DFI. These reductions incur very little overhead—only increasing S and T by a factor that is polylogarithmic in N—and both reductions are non-adaptive, in the sense that they convert non-adaptive DFI algorithms into non-adaptive SFI algorithms. (See Remarks 5.3 and 5.5.)

These reductions can be viewed as variants of a reduction in [CK19] (as we discuss in Sections 1.2 and 1.3). In Appendix B, we show another search-to-decision reduction for injective functions, which is a more direct adaptation of the reduction in [CK19].

Removing shared randomness. Our final contribution is a generic way to convert a function inversion algorithm with shared randomness into an algorithm without shared randomness, at the expense of a small (additive) increase in S. Indeed, prior work used slightly different models for function inversion—depending on whether the preprocessing and query algorithms are allowed access to a shared random string, which does not count as part of the preprocessed advice. Often, this shared random string is represented by shared access to a random oracle.

E.g., Corrigan-Gibbs and Kogan [CK19] allowed their query and preprocessing algorithms access to the same random oracle. In contrast, Fiat and Naor [FN91] did not allow for this. Even in this more conservative setting, however, it is often far more convenient to first describe algorithms that *do* have access to shared randomness, typically in the form of a random oracle, and then to describe how to remove this shared randomness by, e.g., replacing the random oracle with a suitable carefully chosen hash function (with a suitably short key that can be included as part of the preprocessed advice) and arguing that this has little to no effect on the correctness of the algorithm.

We show a generic way to convert *any* function inversion algorithm with shared randomness into a function inversion algorithm without shared randomness. Our conversion is quite simple (and actually applies to a more general class of problems; see Section 6), as it simply replaces the shared randomness r with a string r_i chosen by the preprocessing algorithm from a relatively small number of fixed strings $r_1, \ldots, r_k$. (In fact, a random list of strings will work with high probability.[5]) Because the number of such strings is relatively low (e.g., $k \leq N \cdot \operatorname{poly}\log(N)$ in all of our settings), the index i can be appended to the preprocessed advice essentially for free (costing only an additional $\log k \approx \log N$ bits of advice).

In particular, nearly all of the results listed are most naturally presented using shared randomness, but this procedure shows that this shared randomness can be removed without changing any of our stated results (up to a lower-order additive term in S)! And, this shows that the carefully

[5] At first, this statement might sound trivial, since we started with an algorithm that works with shared randomness r, and we seem to have converted into an algorithm with *more* shared randomness. The difference, however, is in the order of quantifiers. In the shared randomness model, we ask that for any function f with high probability over the randomness r, the algorithm inverts f. Here, we show that with high probability over the random strings $r_1, \ldots, r_k$, for every function f there exists i such that the algorithm inverts f with randomness r_i.

chosen hash functions in much prior work were *in some sense* not necessary. (In particular, our result implies that it is not necessary to use these hash functions to remove shared randomness. However, these hash functions are still useful for optimizing additional complexity measures that we ignore in this work, like the running time of the query and preprocessing algorithms. See Section 1.4.)

Our proof of this result is an adaptation to our setting of a celebrated result in communication complexity. Specifically, we adapt Newman's beautiful technique for converting public-coin protocols to private-coin protocols [New91].

This does not come completely for free, however. Our proof shows that a random list of strings $r_1, \ldots, r_k$ will work with high probability. But, these strings still need to be stored somehow. So, while our conversion process does not increase the number of queries T and only (additively) increases the size S of the advice by a very small amount, it *does* require both the preprocessing algorithm $\mathcal{P}$ and the online algorithm $\mathcal{A}$ to be *non-uniform*.

Since non-uniformity is often assumed in this setting, this does not bother us much. But, there do exist practical applications of function inversion algorithms, e.g., in cryptanalysis, for which truly non-uniform algorithms are an unreasonable model. We note, however, that in practical applications it is typically sufficient to simply use a cryptographic hash function as a replacement for a random oracle.

1.2 Our techniques

Improving Fiat-Naor. Our improvement to Fiat and Naor's algorithm starts by recalling the following. In the original Fiat-Naor procedure, the preprocessing algorithm first generates a list of nearly S "heavy hitters"—that is, elements in the image of f such that $|f^{-1}(y)|$ is large—and it includes this list together with a preimage for each heavy hitter in its advice to the online algorithm.

The online algorithm then operates in two phases. It first checks this list to see if its input y is a heavy hitter, in which case it immediately outputs the corresponding preimage contained in the advice. Otherwise, (ignoring important technical details for simplicity) the algorithm effectively runs a function inversion algorithm on the function f *restricted to elements whose images are not heavy hitters*. With the heavy hitters removed, the new restricted function is much better behaved than the original, allowing for the final tradeoff. (In particular, the restricted function will have relatively low collision probability, which Fiat and Naor show is sufficient for a Hellman-like algorithm to invert it with the desired tradeoff. See Section 3 for the details.)

In fact, as Fiat and Naor observe, it is sufficient to simply include a list of nearly S pairs $(x_i, f(x_i))_{1 \le i \lesssim S}$ for uniformly random $x_i \sim [N]$ as part of the advice, rather than explicitly looking for heavy hitters. (Notice that any elements $y \in [N]$ with very many preimages will still be contained in such a list with high probability, which is why this works.)

At this high (and slightly misleading) level of detail, our modification to Fiat and Naor's algorithm is straightforward: rather than having the preprocessing algorithm include many random queries $(x_i, f(x_i))_i$ as part of the preprocessing, we have the online algorithm generate this list itself. This allows us to replace a list of length S with a list of length T, which gives us our advantage over the original algorithm when $T > S$.

Of course, many details must be worked out to make this actually work. Most significantly, it is crucial that the same list $(x_i, f(x_i))_i$ is known to both the preprocessing algorithm and the online algorithm, so that they both work with the same restricted function f'. For this, we rely on shared randomness (which can then be removed quite painlessly using the result from Section 6), allowing

the online algorithm and the preprocessing algorithm to share the same list $(x_i)_i$ of random query points.

Our reliance on shared randomness also greatly simplifies the description and analysis of both our algorithm *and* Fiat and Naor's original. Indeed, as we mentioned above, we give a simple presentation of a single unified algorithm that works whenever

$$S^2T \cdot \max\{S, T\} \gtrsim N^3 \ .$$

This simplified presentation might itself be of independent interest.

A non-trivial non-adaptive (guess-and-check) algorithm. To describe our non-adaptive algorithm, we first consider the (rather silly) toy problem in which $T = 0$ and $S = \log N$, i.e., the case in which S is large enough to write down precisely one element $\sigma \in [N]$. In particular, suppose that we would like to find some preprocessing $\sigma \in [N]$ such that the algorithm $\mathcal{A}$ can use σ to find a preimage of some image $y := f(x)$ for uniformly random $x \sim [N]$ and any function $f : [N] \to [N]$ without making any queries at all to f. (Equivalently, we can think of this as a one-query *guess-and-check* algorithm.) Of course, such an algorithm cannot hope to succeed with high probability. But, what if we simply wish to maximize the probability that our algorithm succeeds—even if it's still a tiny probability?

A naive approach would achieve success probability $2/N$ by, e.g., setting $\sigma := f(0)$ and having $\mathcal{A}$ output 0 if $y = \sigma$ and 1 otherwise. But, it is actually possible to achieve a success probability of roughly $(\log N/\log\log N)/N$. To see this, let $h : [N] \to [N]$ be a random function, and suppose that our online algorithm simply outputs $g(\sigma, y) := h(y) + \sigma \bmod N$ on input $y \in [N]$. For each possible $\sigma \in [N]$, consider the set $X_\sigma := \{x \in [N] \ : \ f(g(\sigma, f(x))) = f(x)\}$, i.e., the set of all x such that $g(\sigma, f(x))$ is a valid inverse to $f(x)$. Then, a relatively straightforward balls-and-bins argument shows that with high probability there exists a $\sigma \in [N]$ such that $|X_\sigma| \geq \Omega(\log N/\log\log N)$. The preprocessing algorithm can output such a σ, and the result follows.

In our actual algorithm (in Section 4), we (carefully!) repeat this idea many times, with many independent random functions h_i (constructed using shared randomness) and shifts σ_i, and use our oracle queries to check which of the $h_i(y) + \sigma_i \bmod N$ yields a preimage to y. This more-or-less already works in the case when $T = S/\log N \approx N \log\log N/\log N$, but to achieve the full optimal tradeoff for all $S \approx N\log(N/T)$, we must apply additional tricks. In particular, we effectively partition the domain of the function f, so that a single "piece of advice" σ as above can represent many queries. *And*, we similarly partition the range of f, so that the online algorithm "only uses advice corresponding to the part of the partition containing the challenge point y."

A tight bound against guess-and-check algorithms. The proof of our lower bound against guess-and-check algorithms follows the high-level framework used by [DTT10] and [DGK17]. The idea here is to show that a function inversion algorithm with certain properties would imply an unreasonably succinct way to *encode* a function $f : [N] \to [N]$—i.e., a succinct bit string that can be used to recover f. (In this high-level description, we ignore for simplicity the fact that our algorithms $(\mathcal{P}, \mathcal{A})$ may be randomized and the related fact that they might fail some fraction of the time. To fix this, we must work with randomized encodings that themselves have some chance of failure.) In fact, we restrict our attention to permutations f, so that in order to encode f, it suffices to encode the unique inverse of each element $y \in [N]$. (This only makes our lower bound stronger.)

Our encoding will consist of the S bits of preprocessed advice $\sigma \in \{0,1\}^S$ together with some additional information. Recall that a non-adaptive algorithm has the property that the queries $x_1^{(y)}, \ldots, x_T^{(y)}$ made by $\mathcal{A}$ on input y are fixed for fixed σ (where here we are ignoring any randomness for simplicity). Furthermore, if a guess-and-check (non-adaptive) algorithm succeeds, then one of the x_i must be a preimage of y. Our encoding will therefore simply record for each $y \in [N]$ the index $i_y \in [T]$ such that $x_{i_y}^{(y)}$ is the unique preimage of y. Notice that this information, together with σ, is actually enough to completely reconstruct the function f. (Notice also that this argument relies quite heavily on *guess-and-check* non-adaptivity. For a general non-adaptive algorithm, it might be necessary to include the responses to all queries $x_1^{(y)}, \ldots, x_T^{(y)}$.)

This gives an encoding of f that uses only $N \log T + S$ bits. Since there are $N!$ permutations over $[N]$, this is a contradiction unless $N \log T + S \geq \log(N!) \geq \Omega(N \log N)$. Rearranging gives our lower bound of $S \geq \Omega(N \log(N/T))$.

Search-to-decision reductions. Corrigan-Gibbs and Kogan [CK19] observed that there is a reduction from SFI on *injective* functions $f : [N] \to [M]$ to (a different version of) DFI on worst-case functions, where the reduction works by essentially "asking the DFI oracle for the ith bit of the unique preimage." Specifically, at a high level their reduction works by essentially running the DFI algorithm separately on the functions $f_i : [N/2] \to [M]$ corresponding to f restricted to inputs whose ith bit is, say, zero. By solving DFI on the functions f_i and target point y_i, they can recover the unique preimage to y "one bit at a time."[6] Notice in particular that this reduction is careful to only work with a small number of functions f_i that are defined independently of the target point, which allows the SFI algorithm to work with preprocessed advice from the DFI algorithm for a small number of functions.

Both of our search-to-decision reductions start with the simple (and, on its own, not particularly interesting) observation that the above idea can be generalized to invert *any* function $f : [N] \to [M]$, *provided that the target point y that we are inverting has a unique preimage.*

At a high level, our reduction from worst-case SFI to worst-case DFI then works by directly reducing from worst-case SFI with a general target point y to the variant in which y is promised to have a unique preimage. For this, we use an idea inspired by Valiant and Vazirani's celebrated Isolation Lemma [VV85]. Specifically, we find a small number of subsets $U_j \subseteq [N]$ of the domain of f (which are chosen independently of y!) such that with high probability y has exactly one preimage when f is *restricted to U_j*. Then, (ignoring many technical details) we can use the ideas described above to solve this search problem using only a DFI algorithm.

For our reduction from average-case SFI to average-case DFI, we can more-or-less assume that the target point y has a unique preimage, since a large fraction of the elements in the image of a random function f have a unique preimage. However, here we run into a different problem: an average-case DFI oracle is only guaranteed to work with some reasonable probability when the function $f : [N] \to [M]$ is uniformly random (see Section 5.2 for the details). While the restrictions f_i (as described above) of a uniformly random function f are themselves uniformly random, they are certainly not independent. This means that a DFI oracle could potentially have very high success probability but still could, e.g., *always* fail on one (or even many) of the functions f_i (out of

[6] We are oversimplifying quite a bit here and leaving out many important details. Perhaps most importantly, we are assuming here for simplicity that the DFI oracle always outputs the correct answer, while Corrigan-Gibbs and Kogan worked with a much weaker DFI oracle. They were also careful to keep the domain of the functions f_i the same as the domain of the function f, while we are not concerned with this.

$\log N$ total functions $f_1, \ldots, f_{\log N}$), which would cause our search-to-decision reduction to always fail to find the ith bit of the preimage (and therefore to fail).

We solve the above problem by using good error-correcting codes. That is, instead of working with the functions f_i corresponding to the bits of elements in $[N]$ written in binary, we work with a larger number of functions f'_i corresponding to the bits of *encodings* of elements in $[N]$ using a good error-correcting code. That is, f'_i is the function f restricted to the set of elements in $[N]$ whose corresponding codeword has ith bit equal to zero. By using a good enough code, we can recover a preimage of the target by solving just $O(\log N)$ decision problems, even if a $1/4 - \varepsilon$ fraction of the answers are wrong. (Indeed, we can even decode efficiently, though we mostly do not worry about this.)

1.3 Related work

Here, we describe some of the related work that has not already been discussed, as it relates to the present work.

De, Trevisan, and Tulsiani [DTT10] showed improvements to Fiat and Naor's algorithm along a different axis. Specifically, they showed how to achieve surprisingly small values of S and T in the setting in which the algorithm is only required to invert $y := f(x)$ for uniformly random $x \sim [N]$ with some very small probability ε. (In contrast, all of our algorithms invert such a y with high probability.) They show a slight variant of Fiat and Naor's algorithm that works for any S, T, and ε satisfying $ST \gtrsim \varepsilon N$ for $\varepsilon < N^{-1/3}$ (which they show is optimal) and $TS^3 \gtrsim \varepsilon^5 N^3$ otherwise.

Like us, Chawin, Haitner, and Mazor [CHM20] showed lower bounds on special cases of non-adaptive algorithms. In particular, they considered the function $g_{\sigma,y} : [N]^T \to [N]$ that maps the responses $f(x_1), \ldots, f(x_T)$ to the queries made by $\mathcal{A}$ to the final output of $\mathcal{A}$ (i.e., the guess that $\mathcal{A}$ makes for the preimage of y). For example, they showed that $S \geq \Omega(N)$ (regardless of T) if $g_{\sigma,y}$ is an affine function. They also showed that $dS \log N + T \geq \Omega(N)$ if $g_{\sigma,y}$ can be implemented by a depth-d affine decision tree. We note that neither of these models captures guess-and-check algorithms, for which $g_{\sigma,y}(y_1, \ldots, y_T) = x_i$, where i is such that $y_i = y$. (Such a $g_{\sigma,y}$ is certainly not affine, and it seems that it requires depth $d \approx T$ to implement such a function as an affine decision tree, as one must sequentially check whether $y_i = y$ for all i.)

Corrigan-Gibbs and Kogan also defined a special case of non-adaptive algorithms, which they call *strongly* non-adaptive [CK19]. For a strongly non-adaptive algorithm, the function $g_{\sigma,y}$ may be arbitrary, but the queries $x_1, \ldots, x_T$ must be computed independently of the preprocessing (and non-adaptively), so that they are effectively completely independent of the function f. [CK19] showed that lower bounds against even such weak models would imply new circuit lower bounds. However, strongly non-adaptive algorithms are incomparable to our model of guess-and-check algorithms, so that our lower bound on guess-and-check algorithms unfortunately does not directly apply.

For general non-adaptive algorithms, Dvořák, Koucký, Král, and Slívová [DKKS21] showed a conditional lower bound of $T \geq \Omega(\log N / \log \log N)$ for any $S \leq \varepsilon N \log N$ for some small constant $\varepsilon > 0$, assuming the Network Coding Conjecture. Notice that this lower bound holds in a more general setting than our lower bound or those of [CHM20] but it requires an unproven conjecture and is quantitatively weaker than ours and those in of [CHM20]. (E.g., for guess-and-check algorithms with $S \leq \varepsilon N \log N$, our lower bound implies that $T \geq N^{1-O(\varepsilon)}$.)

There is also a long line of work [DTT10, DGK17, CDGS18, CDG18, CK19, GGKL21] studying a different version of DFI than the one that we study, which is sometimes simply called the PRG

problem. Here, the goal is to distinguish (perhaps with relatively small distinguishing advantage) a uniformly random element $y \sim [M]$ from $f(x)$ for uniformly random $x \sim [N]$, where $f : [N] \to [M]$. In particular, Corrigan-Gibbs and Kogan show a search-to-to-decision reduction from SFI over injective functions to the worst-case PRG problem. Our search-to-decision reductions are essentially generalizations of the reduction from [CK19] to the setting of non-injective functions. We pay for this non-injectivity by requiring our DFI algorithm to solve problems that are harder than the PRG problem, and by requiring significantly more complicated reductions.

1.4 A note on the many facets of function inversion

There are *many* variants of the function inversion problem and *many* different complexity measures that one can use to assess algorithms in this context. The landscape is therefore quite complicated. Indeed, our search-to-decision reductions and our proof that shared randomness can be removed (as well as the reductions between versions of SFI with different range sizes in Appendix A) can be viewed as small steps towards simplifying the picture a bit.

But, there are still certainly many variants and complexity measures that we simply do not address in this work. E.g., while we mostly focus on the number of queries T and the length S of the preprocessed advice, much prior work was also interested in the time and space complexity of the algorithms $\mathcal{P}$ and $\mathcal{A}$, which we largely ignore. E.g., prior work [FN91, DTT10] used specialized hash functions to replace shared randomness because removing shared randomness is itself a worthy goal, but *also* to optimize the running time of their algorithms (which is not the same as the query complexity T). For the sake of simplicity, we have chosen to largely ignore these additional complexity measures in our algorithms, and we have therefore not optimized our algorithms for these complexity measures at all. (We do note that our algorithms run in essentially optimal time when they are implemented with shared randomness in the form of shared access to a random oracle. In particular, the preprocessing algorithms can be implemented in time $\widetilde{O}(N)$, and the online algorithms can be implemented in time $T \cdot \operatorname{poly}\log(N)$.)

As another example, as we discussed above, De, Trevisan, and Tulsiani [DTT10] studied the dependence of S and T in terms of the fraction ε of inputs $x \in [N]$ for which the algorithm successfully inverts $f(x)$. They showed that for small ε one can do *much* better than Eq. (1), using essentially the same algorithm. It is natural to ask whether their techniques can be applied to our new version of the Fiat-Naor algorithm, but we leave this to future work.

2 Preliminaries

We define $\mathbb{1}_\mu$ as $\mathbb{1}_\mu = 1$ if μ is true, and 0 otherwise. All logarithms are base 2, i.e., $\log 2^n = n$.

2.1 Definitions of function inversion problems

In the following definitions, M and N are positive integers, and $(\mathcal{P}, \mathcal{A})$ is a pair of randomized algorithms. The first few definitions are core to our study of function inversion.

Definition 2.1. *We say that*

1. $(\mathcal{P}, \mathcal{A})$ *uses S bits of preprocessing if for all inputs, the output of $\mathcal{P}$ has bitlength at most S.*
2. $(\mathcal{P}, \mathcal{A})$ *uses T queries if for all inputs, $\mathcal{A}^f$ makes at most T queries to f.*

Definition 2.2. *We say that* $(\mathcal{P}, \mathcal{A})$ solves (N, M)-search function-inversion ((N, M)-SFI) with success probability $\delta \in (0, 1]$ *if for all* $f : [N] \to [M]$ *and* $y \in f([N])$*,*

$$\Pr_{r \sim \{0,1\}^l}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) \in f^{-1}(y)] \geq \delta.$$

Here r is the shared randomness between the algorithms $\mathcal{A}$ and $\mathcal{P}$. It has some (typically unspecified) finite bitlength l.

Definition 2.3. *We say that* $(\mathcal{P}, \mathcal{A})$ solves (N, M)-decision function-inversion ((N, M)-DFI) with advantage $\varepsilon \in (0, 1/2]$ *if for all* $f : [N] \to [M]$ *and* $y \in [M]$*,*

$$\Pr_{r \sim \{0,1\}^l}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) = \mathbb{1}_{y \in f([N])}] \geq 1/2 + \varepsilon.$$

In words, $\mathcal{A}$ *is likely to output* 1 *when* y *is in the image of* f*, but is unlikely to output* 1 *when* y *is in* $[M] \setminus f([N])$*.*

We will abuse terminology slightly and simply refer to $(\mathcal{P}, \mathcal{A})$ as an algorithm when the meaning is clear from context. When $N = M$, we will drop the parameters and just write SFI or DFI. We will also sometimes write "worst-case SFI" or "worst-case DFI" to distinguish from the average-case variants that we define next.

Definition 2.4. *We say that* $(\mathcal{P}, \mathcal{A})$ solves average-case (N, M)-SFI with success probability δ *if*

$$\Pr_{\substack{r \sim \{0,1\}^l \\ f \sim \{g:[N] \to [M]\} \\ x \sim [N]; y \leftarrow f(x)}}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) \in f^{-1}(y)] \geq \delta.$$

Definition 2.5. *We say that* $(\mathcal{P}, \mathcal{A})$ solves average-case (N, M)-DFI with advantage ε *if*

$$\Pr_{\substack{r \sim \{0,1\}^l \\ f \sim \{g:[N] \to [M]\} \\ x \sim [N]; y \leftarrow f(x)}}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) = 1] \geq 1/2 + \varepsilon,$$

and

$$\Pr_{\substack{r \sim \{0,1\}^l \\ f \sim \{g:[N] \to [M]\} \\ y \sim [M] \setminus f([N])}}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) = 0] \geq 1/2 + \varepsilon.$$

In order to state our results removing shared randomness, we need the following definition of function-inversion algorithms without shared randomness.

Definition 2.6. *We say that* $(\mathcal{P}, \mathcal{A})$ solves (N, M)-SFI with success probability δ without shared randomness *if for all* $f : [N] \to [M]$ *and all* $y \in f([N])$*,*

$$\Pr_{r_1, r_2 \sim \{0,1\}^l}[\mathcal{A}^f(\mathcal{P}(f, r_1), y, r_2) \in f^{-1}(y)] \geq \delta.$$

We make analogous definitions for the 3 other problems ((N, M)-DFI, average-case (N, M)-SFI, average-case (N, M)-DFI).

Note that we will say, for example, "(N, M)-SFI for injective functions", when we mean Definition 2.2, but with the function f ranging over all injective functions from $[N] \to [M]$. Finally, we define some special classes of algorithms that will be studied in Section 4.

Definition 2.7. *An algorithm $\mathcal{A}$ is* non-adaptive *if $\mathcal{A}^f(\sigma, y, r)$ only queries f on points $x_1(\sigma, y, r), \ldots, x_T(\sigma, y, r)$ depending only on the inputs σ, y, and r (i.e., not depending on the results of previous queries).*

Definition 2.8. *An algorithm $\mathcal{A}$ is a* guess-and-check *algorithm if it is non-adaptive and whenever $x \leftarrow \mathcal{A}^f(\sigma, y, r)$, then x is one of the points queried by $\mathcal{A}^f$.*

We will say that $(\mathcal{P}, \mathcal{A})$ is non-adaptive (resp. a guess-and-check algorithm) if $\mathcal{A}$ is non-adaptive (resp. a guess-and-check algorithm).

2.2 Some basic probability results

We will use the following version of Chernoff's bound (see, e.g., [MU17]).

Lemma 2.9. *Suppose $X_1, \ldots, X_n$ are independent random variables taking values in $\{0, 1\}$. Let X denote their sum, and $\mu := \mathbb{E}[X]$. Then for any $\delta \geq 0$,*

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(\frac{-\delta^2\mu}{2+\delta}\right).$$

Moreover, for $0 \leq \varepsilon \leq 1$,

$$\Pr[X \leq (1-\varepsilon)\mu] \leq \exp\left(\frac{-\varepsilon^2\mu}{2}\right).$$

We will also need the following simple bound.

Claim 2.10. *For any integers $N \geq 1$ and $M \geq 2$,*

$$\Pr_{f \sim \{g:[N]\to[M]\}, x\sim[N]}[|\{x' \in [N] \ : \ f(x) = f(x')\}| = 1] \geq e^{-N/M - N/M^2}.$$

Proof. This is exactly equal to

$$\Pr_{y_1,\ldots,y_{N-1}\sim[M]}[\forall i, \ y_i \neq 0] = (1-1/M)^{N-1} \geq e^{-N/M-N/M^2}. \qquad \square$$

2.3 Binary linear codes

Recall that a binary linear code $\mathcal{C}$ with rank n is an n-dimensional subspace $\mathcal{C} \subseteq \mathbb{F}_2^m$, and $\mathbf{C} \in \mathbb{F}_2^{m\times n}$ is a generator matrix for $\mathcal{C}$ if $\mathcal{C} = \mathbf{C}\mathbb{F}_2^n$. For $\mathbf{x} \in \mathbb{F}_2^m$, we write $\|\mathbf{x}\|_H$ for the Hamming weight of $\mathbf{x}$ (i.e., the number of non-zero coordinates).

Theorem 2.11 ([Jus72, MS77, ABN+92, Spi95])**.** *For every constant $\varepsilon > 0$, there exists a family $\mathcal{C}_{n,\varepsilon} \subseteq \mathbb{F}_2^m$ with rank n and $m = m_{n,\varepsilon} \leq O_\varepsilon(n)$, an efficiently computable generator matrices $\mathbf{C}_{n,\varepsilon} \in \mathbb{F}_2^{m\times n}$, and an efficient decoding algorithm Dec such that for every $\mathbf{x} \in \mathbb{F}_2^n$ and every $\mathbf{e} \in \mathbb{F}_2^m$ with $\|\mathbf{e}\|_H \leq (1/4 - \varepsilon)\cdot m$, $\mathsf{Dec}(\mathbf{C}_{n,\varepsilon}\mathbf{x} \oplus \mathbf{e}) = \mathbf{x}$.*

For any $\mathcal{C} \subseteq \mathbb{F}_2^m$ and $1 \leq i \leq m$, we can easily define the subcode $\mathcal{C}_i := \{\mathbf{c} = (c_1, \ldots, c_m) \in \mathcal{C} \ : \ c_i = 0\}$. Notice that we have either $|\mathcal{C}_i| = |\mathcal{C}|$ or $|\mathcal{C}_i| = |\mathcal{C}|/2$ (where the first case only occurs if all $\mathbf{c} \in \mathcal{C}$ have zero ith coordinate), and that given a generator matrix $\mathbf{C} \in \mathbb{F}_2^{m\times n}$ for a code $\mathcal{C}$, it is trivial to compute a generator matrix for $\mathcal{C}_i$. Notice also that we may assume without loss of generality that the codes $\mathcal{C} := \mathcal{C}_{n,\varepsilon}$ in Theorem 2.11 satisfy $|\mathcal{C}_i| = |\mathcal{C}|/2$ for all i (since we may simply remove any coordinates that are always zero).

3 An improvement to Fiat and Naor's algorithm

From our perspective, there are two core techniques used in Fiat and Naor's algorithm [FN91]. First, Fiat and Naor's algorithm generates a list L of pairs $(x, f(x))$ for random domain elements $x \in [N]$, which effectively serves as a list of preimages of "heavy hitters"—i.e., elements x such that $f(x)$ has many preimages. In the original algorithm, L is included as part of the preprocessed advice. Second, (following Hellman [Hel80]) Fiat and Naor describe a randomized subroutine $(\mathcal{P}', \mathcal{A}')$ that takes L as auxiliary input and for all $y \in f([N])$, inverts y with some small probability. This subroutine is then run many times to boost its success probability (with a fixed list L but independent randomness for $\mathcal{P}'$ and $\mathcal{A}'$). Our improvement differs from the original only in the first part, and the difference can be described in one sentence: if $T > S$, instead of including the list L in the preprocessed advice, we reconstruct it using queries to f. This can be done because the random domain elements x can be derived from *shared* randomness (which we also show in Corollary 6.3 is available in the non-uniform model for essentially no cost). This allows us to construct a larger list L in the case when $T > S$, with $|L| \approx T$ instead of $|L| \approx S$.

Our formal theorem is the following.

Theorem 3.1. *For all S, T satisfying $S^2 T \max\{S, T\} \geq N^3$, there exists an algorithm that solves SFI with success probability 1 using $O(S \log^2 N)$ bits of preprocessing and $O(T \log^2 N)$ queries.*

As mentioned above, this improves on Fiat and Naor's tradeoff in the important setting where $S < T$. On the other hand, when $S \geq T$ our algorithm is essentially just Fiat and Naor's algorithm. However, even in this case, we believe that our presentation and analysis is significantly simpler, which we view as an additional contribution. Some (though certainly not all) of this simplicity is because of our choice to optimize only for T and S and not for additional complexity measures like the running time of the online algorithm (see Section 1.4) or the use of shared randomness (which we show is essentially without loss of generality in Section 6). Fiat and Naor made careful use of k-wise independent hash functions in order to optimize these parameters.

Below, we present an algorithm which succeeds with probability $1 - O(1/N)$. By Corollary 6.3, this implies the result.

3.1 The algorithm

Let $K := \max\{S, T\}$, and let $\alpha := 2K\lceil \log N \rceil$. Let $z_1, \ldots, z_\alpha \sim [N]$ be uniformly random and independent elements generated using the shared randomness. Let $L := \{(z_i, f(z_i)) : i \in [\alpha]\}$. Intuitively, we think of L as a list of inverses for "heavy hitters," that is, elements y in the image of f that have many preimages. Let $\widehat{L} := \{y : (x, y) \in L\}$, and let $D := \{x \in [N] : f(x) \notin \widehat{L}\}$ be the domain elements whose images are not trivially inverted by lookup in L. Finally, let $N' := |D|$.

We will show a subroutine $(\mathcal{P}', \mathcal{A}')$ that takes L as input and, provided that $\widehat{L}$ contains all points with at least N/K preimages, inverts any challenge $y \in f(D)$ with small but decent probability. It uses parameters $m := \lfloor N/3T \rfloor$ and $t := \lfloor N'/3S \rfloor$. The subroutine works by constructing m chains of length t as in Figure 1.

At a high level, the full algorithm $(\mathcal{P}, \mathcal{A})$ then works by constructing L, and running $(\mathcal{P}', \mathcal{A}')$ many times to boost the success probability. More precisely, let $\ell := \lceil 100ST \log(N)/N \rceil$, and let $r_1, \ldots, r_\ell$ be independent random strings derived from shared randomness. On input a function f, the preprocessing algorithm $\mathcal{P}$ first constructs L as described above, then for $i \in [\ell]$, it runs $\mathrm{st}_i \leftarrow \mathcal{P}'(L, f, r_i)$. If $S \geq T$, $\mathcal{P}$ outputs $\sigma := (L, \mathrm{st}_1, \ldots, \mathrm{st}_\ell)$. Otherwise, it just outputs $\sigma := (\mathrm{st}_1, \ldots, \mathrm{st}_\ell)$.

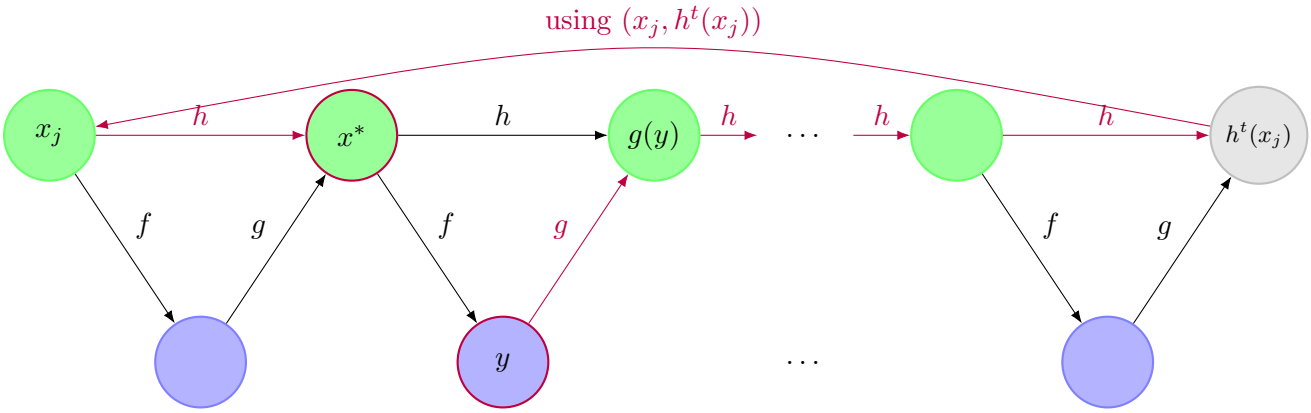

Figure 1: The picture captures the basic workings of chain-based algorithms, including Hellman's algorithm, Fiat and Naor's algorithm, and our improvement. Here $h = g \circ f$, where g is randomly sampled from some appropriate distribution. Preprocessing constructs the green chain $C(x_j)$ by sampling a random point x_j and iterating h. It stores the pair $(x_j, h^t(x_j))$. On challenge y, online assumes y is a blue point, and follows the red arrows. That is, it proceeds by computing $g(y)$, then iterating h until it reaches the stored endpoint $h^t(x_j)$. Once there, it jumps back to x_j and iterates h until $x^* \in f^{-1}(y)$ is found.

On input a challenge y and preprocessed advice σ, the online algorithm $\mathcal{A}$ first recovers L as follows. If $S \geq T$, $\mathcal{A}$ just reads L from σ. Otherwise, it queries f on the points $z_1, \ldots, z_\alpha$ to recover L. Then $\mathcal{A}$ checks if $y \in \widehat{L}$; if so, it returns the corresponding inverse. If not, for $i \in [\ell]$, it runs $o_i \leftarrow \mathcal{A}'^f(L, \mathrm{st}_i, y, r_i)$. If any run i returns $o_i \neq \perp$, $\mathcal{A}$ outputs o_i. Otherwise, it outputs $\perp$.

3.1.1 The subroutine

It remains to describe the subroutine $(\mathcal{A}', \mathcal{P}')$. The subroutine receives L as input, but we will view it as receiving g as input instead, where $g : [N] \to [D]$ is a uniformly random function, constructed using L as follows. Let $J := \lceil N/N' \cdot 2 \log N \rceil$, and let $g' : [N] \times [J] \to [N]$ be a random function sampled independently using the shared randomness of $\mathcal{P}'$ and $\mathcal{A}'$. We say that g' is *bad* if there exists an $i \in [N]$ such that $g'(i, j) \notin D$ for all $j \in [J]$. If g' is bad, our subroutine will simply fail. But, it is easy to see that for our choice of J this happens with probability at most $2/N$. We will therefore assume below that g' is not bad, which will cost us at most an additive factor of $2/N$ in the success probability of our subroutine. Now, define $g(y) := g'(y, k)$, where $k \in [J]$ is minimal such that $g'(y, k) \in D$. Notice that g is a uniformly random function $g : [N] \to D$, and that, given L, $g(y)$ can be computed using at most J queries to f by finding the minimal i such that $f(g'(y, i)) \notin \widehat{L}$. [7]

Finally, let $h := g \circ f$, and for each $x \in [N]$ and $s \geq 1$, define the *chain* $C^s(x) := \{x, h(x), \ldots, h^s(x)\}$. (Here and below, we use the notation h^q to represent h composed with itself q times.) See Figure 1.

[7] Indeed, this is the whole purpose of this rather subtle construction of g (which is only a slight variant of the construction in Fiat and Naor [FN91])—to provide $\mathcal{P}'$ and $\mathcal{A}'$ with access to a shared random function from $[N]$ to D without requiring $\mathcal{A}'$ to make too many queries. Notice that this is non-trivial because the set D is not known to $\mathcal{A}'$ and might not have a succinct description. ($\mathcal{A}'$ instead only knows the image $\widehat{L}$ of $[N] - D$ under f.)

Preprocessing: Stores $(x_i, h^t(x_i))$ for independent $x_1, \ldots, x_m \sim D$. The $h^t(x_i)$ will be called *endpoints*.

Online: On challenge $y \in f(D)$ (recall that $f(D) = f([N]) - \widehat{L}$), online computes $C_y := C^{t-1}(g(y))$ and checks if there is a unique $i \in [m]$ such that $h^t(x_i) \in C_y$.[8] If not, it gives up. Then it computes $C^{t-1}(x_i)$ and checks whether any $x^* \in C^{t-1}(x_i)$ satisfies $f(x^*) = y$. If so, it returns x^*; else it returns $\perp$.

3.2 Analysis

First we analyze the resource costs. It is clear that the sub-algorithm stores at most $2m\lceil \log N \rceil$ bits of advice, and makes at most $2t \cdot J$ queries to f. Hence the data structures $\mathrm{st}_1, \ldots, \mathrm{st}_\ell$ have total bitlength at most

$$\ell \cdot 2m\lceil \log N \rceil = \left\lceil \frac{100ST \log N}{N} \right\rceil \cdot (2m\lceil \log N \rceil) \leq \frac{300ST \log^2 N}{N} \frac{N}{3T} = 100S \log^2 N \, .$$

If $S > T$, storing the list L, which consists of $\alpha = 2S\lceil \log N \rceil$ pairs of elements of $[N]$, requires at most an additional $10S \log^2 N$ bits. So $(\mathcal{P}, \mathcal{A})$ uses at most $110S\lceil \log N \rceil^2$bits of preprocessing. And the total number of queries to f made by $\mathcal{A}$ is at most

$$\ell \cdot (2tJ) \leq \ell \cdot 5t(N/N') \log N = \left\lceil \frac{100ST \log N}{N} \right\rceil \cdot 5 \left\lfloor \frac{N'}{3S} \right\rfloor \cdot \frac{N \log N}{N'} \leq 200T \log^2 N \, .$$

To analyze the success probability, we first observe that

Lemma 3.2. *Except with probability $2/N$, all $x \in D$ satisfy $|f^{-1}(f(x))| \leq N/K$.*

Proof. The condition above is equivalent to the list $\widehat{L}$ containing all $u \in [N]$ with $|f^{-1}(u)| \geq N/K$. But since $\alpha := 2K\lceil \log N \rceil$, we have $N/K \geq 2 \log N \cdot N/\alpha$, and so for any u with $|f^{-1}(u)| \geq N/K$, there exists $i \in [\alpha]$ with $f(z_i) = u$ (which implies $u \in \widehat{L}$) except with probability $2/N^2$. The lemma then follows by union bound. □

We claim that the subalgorithm satisfies the following guarantee:

Theorem 3.3. *Let $f : [N] \to [N]$ for some $N \geq 1$. Let $U \geq 1$, and suppose that for all $x \in D$, $|f^{-1}(f(x))| \leq U$. Let $y \in f(D)$. Then the sub-algorithm with parameters $0 \leq m, t \leq N$ finds an inverse of y with probability at least*

$$(1 - 6mt^2U/N') \cdot (1 - t^2U/N') \cdot |f^{-1}(y)| \cdot mt/N' - 2/N \, .$$

In particular, if N is sufficiently large, the bound $U = N/K$ from Lemma 3.2 holds, and the parameter settings are $m = \lfloor N/3T \rfloor$, $t = \lfloor N'/3S \rfloor$ as above, the probability is at least

$$mt/(2N') \geq N/(100ST).$$

Using Theorem 3.3, it is straightforward to show Theorem 3.1.

[8] The requirement of uniqueness substantially simplifies the analysis. However, it is possible to use a weaker condition.

Proof of Theorem 3.1 assuming Theorem 3.3.
Lemma 3.2 states that all $x \in D$ satisfy $|f^{-1}(f(x))| \le U$ except with probability $2/N$ over the random choices of $z_1, \ldots, z_\alpha$. Assuming this holds, Theorem 3.3 says that for all $y \in f(D)$ (i.e., all $y \notin \widehat{L}$), the subalgorithm $(\mathcal{P}', \mathcal{A}')$ inverts y with probability at least $N/(100ST)$. Thus, for all $y \notin \widehat{L}$, except with probability $O(1/N)$, at least one of the $\ell = \lceil 100 \log N \cdot (ST/N) \rceil$ iterations of $(\mathcal{P}', \mathcal{A}')$ inverts y. Of course, the points $y \in \widehat{L}$ are trivially inverted by lookup in L. Hence for all $y \in f([N])$, $(\mathcal{P}, \mathcal{A})$ inverts y except with probability $O(1/N)$. By Corollary 6.3, this implies the result. □

It remains to prove Theorem 3.3.

Proof of Theorem 3.3. The particular statement easily follows from the general statement. Indeed,

$$\begin{aligned} mt^2U/N' &\le (N/3T) \cdot (N'/3S)^2 \cdot (N/K)/N' \\ &\le N^2N'/(27S^2TK) \le N^3/(27S^2TK) \le 1/27. \end{aligned}$$

And for sufficiently large N, it follows that

$$\begin{aligned} (1 - 6mt^2U/N') \cdot (1 - t^2U/N') \cdot |f^{-1}(y)| \cdot mt/N' - 2/N \\ &\ge (1 - 7mt^2U/N') \cdot mt/N' - 2/N \\ &\ge (1 - 7/27) \cdot mt/N' - 2/N \\ &\ge 1/2 \cdot (mt/N') \\ &\ge (N/3T) \cdot (N'/3S)/(2N') \\ &\ge N/(18ST). \end{aligned}$$

We now prove the general statement of Theorem 3.3. Fix f, U, and y as in the theorem statement. In what follows, we will assume that g' is not bad (so that g is a random function from $[N]$ to $[D]$), at the cost of an additive $2/N$ in the success probability. By inspection, the subalgorithm inverts y if and only if the following event E_i occurs for some $i \in [m]$: (1) y is contained in $f(C^{t-1}(x_i))$ (which implies $h^t(x_i) \in C_y$), and (2), for all $j \ne i$, $h^t(x_j) \notin C_y$. Moreover, these events E_i are disjoint and symmetric. So the probability that the subalgorithm inverts y is exactly $m \Pr[E_1]$.

Let E_1^1 be the event that $h^t(x_j) \notin C_y$ for all $j \ne 1$, and let E_1^2 be the event that $y \in f(C^{t-1}(x_1))$; then $E_1 = E_1^1 \cap E_1^2$. To lower bound $\Pr[E_1]$, we will first lower bound $\Pr[E_1^2]$, then lower bound $\Pr[E_1^1 \mid E_1^2]$.

We claim that

Claim 3.4.
$$\Pr[E_1^2] := \Pr[y \in f(C^{t-1}(x_1))] \ge (1 - t^2U/N') \cdot |f^{-1}(y)| \cdot t/N'.$$

For convenience, define

$$(Z_1, \ldots, Z_t) := (x_1, h(x_1), \ldots, h^{t-1}(x_1)) = C^{t-1}(x_1)\ .$$

Let A_0 be the universal event (i.e., $\Pr[A_0] = 1$) and for $1 \le i \le t-1$, let A_i be the event that (1) A_{i-1} holds, (2) $Z_i \notin f^{-1}(y)$, and (3) $f(Z_i) \notin f(\{Z_1, \ldots, Z_{i-1}\})$. More explicitly, for $1 \le i \le t-1$, A_i is the event that (1) $Z_1, Z_2 \ldots, Z_i \notin f^{-1}(y)$, and (2) the values $f(Z_1), f(Z_2), \ldots, f(Z_i)$ are all distinct.

It is not hard to see that for all $1 \le i \le t$, conditioned on A_{i-1}, Z_i is uniformly random and independent of $(Z_1, \ldots, Z_{i-1})$. (Here the probability is over $x_1, \ldots, x_m$ and the random function g.) Indeed, the claim is trivial for $i = 1$. For $i > 1$, observe that conditioned on A_{i-1}, it holds that $f(Z_{i-1}) \notin f(\{Z_1, \ldots, Z_{i-2}\})$, so $Z_i = g(f(Z_{i-1}))$ is a fresh uniform sample from D, independent of $(Z_1, \ldots, Z_{i-1})$.

For $1 \le i \le t$, let B_i be the event that (1) A_{i-1} holds, and (2) $Z_i \in f^{-1}(y)$. That is, B_i is the event that (1) $Z_i \in f^{-1}(y)$, (2) $Z_j \notin f^{-1}(y)$ for all $j < i$, and (3), the values $f(Z_1), f(Z_2), \ldots, f(Z_{i-1})$ are all distinct. By construction, the events B_i are mutually exclusive. So,

$$\Pr[y \in f(C^{t-1}(x_1))] \ge \Pr\Big[\bigcup_{i=1}^{t} B_i\Big] = \sum_{i=1}^{t} \Pr[B_i] \ge \sum_{i=0}^{t-1} \Pr[A_i] \Pr[B_{i+1} \mid A_i] \,.$$

First we obtain a lower bound on $\Pr[A_i]$.

$$\begin{aligned}
\Pr[A_{i+1} \mid A_i] &= \Pr[Z_{i+1} \notin f^{-1}(y) \text{ and } f(Z_{i+1}) \notin f(\{Z_1, \ldots, Z_i\}) \mid A_i] \\
&= \Pr[Z_{i+1} \notin \big(f^{-1}(y) \cup f^{-1}(f(Z_1)) \cup \cdots \cup f^{-1}(f(Z_i))\big) \mid A_i] \\
&= 1 - |f^{-1}(y) \cup f^{-1}(f(Z_1)) \cup \cdots \cup f^{-1}(f(Z_i))|/N' \\
&\ge 1 - ((i+1)U)/N' \\
&\ge 1 - tU/N'.
\end{aligned}$$

It follows that for all $0 \le i \le t - 1$,

$$\Pr[A_i] \ge (1 - tU/N')^t \ge 1 - t^2U/N'.$$

By a similar calculation, for all $0 \le i \le t - 1$,

$$\Pr[B_{i+1} \mid A_i] = \Pr[Z_{i+1} \in f^{-1}(y) \mid A_i] = |f^{-1}(y)|/N' \,.$$

Putting everything together, we have the desired lower bound:

$$\Pr[E_1^2] := \Pr[y \in f(C^{t-1}(x_1))] \ge (1 - t^2U/N') \cdot |f^{-1}(y)| \cdot t/N' \,.$$

Next we turn to lower bounding $\Pr[E_1^1 \mid E_1^2]$. We claim that

Claim 3.5.

$$\Pr[E_1^1 \mid E_1^2] \ge 1 - 6mt^2U/N' \,.$$

It suffices to prove this claim. Indeed, combining it with Claim 3.4 gives

$$\Pr[I] \ge m \Pr[E_1] \ge m \Pr[E_1^2] \cdot \Pr[E_1^1 \mid E_1^2] \ge m(1 - 6mt^2U/N') \cdot (1 - t^2U/N') \cdot |f^{-1}(y)| \cdot t/N' \,.$$

Next we prove Claim 3.5. By union bound and symmetry,

$$\Pr[E_1^1 \mid E_1^2] := \Pr[\forall j \ne 1, h^t(x_j) \notin C_y \mid E_1^2] \ge 1 - m \cdot \Pr[h^t(x_2) \in C_y \mid y \in f(C^{t-1}(x_1))] \,. \tag{6}$$

Thus, our goal is to upper bound $\Pr[h^t(x_2) \in C_y \mid y \in f(C^{t-1}(x_1))]$. We reason similarly to the proof of Claim 3.4.

Notice that, if $y \in f(C^{t-1}(x_1))$, then $g(y) \in C^t(x_1)$, and so $C_y := C^{t-1}(g(y)) \subseteq C^{2t}(x_1)$. It follows that

$$\Pr[h^t(x_2) \in C_y \mid y \in f(C^{t-1}(x_1))] \leq \Pr[h^t(x_2) \in C^{2t}(x_1) \mid y \in f(C^{t-1}(x_1))] \,.$$

This is convenient, since we have combined two events that would otherwise need to be considered separately; namely, the event that the chain $C^t(x_2)$ starting at x_2 intersects C_y, and the event that $C^t(x_2)$ intersects $C^{t-1}(x_1)$. Next, we reason as follows.

$$\begin{aligned}
&\Pr[h^t(x_2) \in C^{2t}(x_1) \mid y \in f(C^{t-1}(x_1))] \\
&\leq \Pr[f(h^t(x_2)) \in f(C^{2t}(x_1)) \mid y \in f(C^{t-1}(x_1))] \\
&\leq \Pr[\bigvee_{j=0}^{t} f(h^j(x_2)) \in f(C^{2t}(x_1)) \mid y \in f(C^{t-1}(x_1))] \\
&\leq \sum_{j=0}^{t} \Pr[f(h^j(x_2)) \in f(C^{2t}(x_1)) \mid \forall k < j, f(h^k(x_2)) \notin f(C^{2t}(x_1)), y \in f(C^{t-1}(x_1))] \,.
\end{aligned}$$

Intuitively, the j-th term in the sum corresponds to the chain starting at x_2 intersecting the chain starting at x_1 after j steps, but not before. We write

$$\mathsf{COND}_{1,j} := \forall k < j, f(h^k(x_2)) \notin f(C^{2t}(x_1)) \text{ and } \mathsf{COND}_2 := y \in f(C^{t-1}(x_1))$$

We claim that for all $0 \leq j \leq t$, the j-th term satisfies the following bound:

$$\Pr[f(h^j(x_2)) \in f(C^{2t}(x_1)) \mid \mathsf{COND}_{1,j}, \mathsf{COND}_2] \leq |f^{-1}(f(C^{2t}(x_1)))|/N' \,.$$

Notice that if $j = 0$, then $\mathsf{COND}_{1,j}$ is vacuous, $h^j(x_2) = x_2$ is a fresh independent uniform sample from D, and the claimed bound holds with equality.

For $j \geq 1$, consider the event $\mathsf{COND}_{3,j}$ that, for some $k < j-1$, $f(h^{j-1}(x_2)) = f(h^k(x_2))$. It is not hard to see that

$$\Pr[f(h^j(x_2)) \in f(C^{2t}(x_1)) \mid \mathsf{COND}_{1,j}, \mathsf{COND}_2, \mathsf{COND}_{3,j}] = 0 \,.$$

Indeed, applying $f \circ g$ to both sides of $\mathsf{COND}_{3,j}$ gives $f(h^j(x_2)) = f(h^{k+1}(x_2))$, but $\mathsf{COND}_{1,j}$ implies $f(h^{k+1}(x_2)) \notin f(C^{2t}(x_1))$.

On the other hand, if we condition on $\neg\mathsf{COND}_{3,j}$ (and $\mathsf{COND}_{1,j}$ and COND_2), we know that $v_j := f(h^{j-1}(x_2))$ is distinct from the values $f(h^k(x_2))$ for $0 \leq k < j-1$. By $\mathsf{COND}_{1,j}$ and COND_2, v_j is also distinct from the values $f(h^i(x_1))$ for $0 \leq i \leq 2t$. In other words, v_j is not in the set V_j defined by

$$V_j := \{f(h^k(x_2)) \mid 0 \leq k < j-1\} \cup \{f(h^i(x_1)) \mid 0 \leq i \leq 2t\}.$$

But it is not difficult to verify that the events $\mathsf{COND}_{1,j}, \mathsf{COND}_2$, and $\mathsf{COND}_{3,j}$ can be expressed solely in terms of x_1, x_2, and the random variables $g(x)$ for $x \in V_j$. (As a sanity check, it is helpful to note that $h^{j-1}(x_2) = g(f(h^{j-2}(x_2)))$ only depends on the random variables $g(f(h^k(x_2)))$ for $k < j-1$.) In particular, these events are independent of $g(v_j)$. It follows that, even conditional on $\mathsf{COND}_{1,j}, \mathsf{COND}_2$, and $\neg\mathsf{COND}_{3,j}$, $h^j(x_2) = g(v_j)$ is a fresh uniform sample from D, independent of the random variables in the conditional. So we have

$$\Pr[f(h^j(x_2)) \in f(C^{2t}(x_1)) \mid \mathsf{COND}_{1,j}, \mathsf{COND}_2, \neg\mathsf{COND}_{3,j}] = |f^{-1}(f(C^{2t}(x_1)))|/N' \,,$$

and we have established the claimed bound on the terms of the sum. Plugging the bound in, we see

$$\begin{aligned}
&\Pr[h^t(x_2) \in C^{2t}(x_1) \mid y \in f(C^{t-1}(x_1))] \\
&\leq \sum_{j=0}^{t} |f^{-1}(f(C^{2t}(x_1)))|/N' \\
&\leq \sum_{i=0}^{t} U \cdot (2t+1)/N' \\
&\leq U \cdot (t+1) \cdot (2t+1)/N' \leq 6t^2 U/N' \,.
\end{aligned}$$

(The last line only holds if $t > 0$, but otherwise Claim 3.5 is trivial.) Combining this with Eq. (6) concludes the proof of Claim 3.5 and hence the proof of Theorem 3.3. □

4 Non-adaptive algorithms

All known non-trivial algorithms for function inversion make queries adaptively (as far as the authors are aware). That is, the answers of previous queries are used to decide which domain element to query next. If instead all query points must be specified at once, the only known scheme is the *trivial inverter* that stores inverses for (say) the first k elements and, for worst-case functions, queries all the non-stored elements. Thus in the worst case, it uses advice of bitlength $S = k \log N$ and makes $T = N - k$ queries, so that $S/\log N + T = N$.

We make progress on non-adaptive function inversion, by giving a non-trivial non-adaptive algorithm. E.g., our algorithm achieves $S = \Theta(N \log\log N)$ for any $T = N/\mathrm{poly}\log(N)$. This resolves a question of Corrigan-Gibbs and Kogan [CK19], who asked whether it was possible to give a non-adaptive algorithm with $S/\log N + T < o(N)$. Our algorithm is even a *guess-and-check* algorithm in the sense that the preimage that it outputs is always one of its query points. (See Definition 2.8.) Specifically, we show the following.

Theorem 4.1. *For any $1 \leq T \leq N/2$, there exists a guess-and-check algorithm that solves SFI with success probability $1 - \exp(-\Omega(\min\{T, 2^{N/T}\}))$ using $S \leq O(N \log(N/T))$ bits of preprocessing and T queries.*

We remark that for T not too extreme—say $10 \log N < T < N/(10 \log\log N)$—the failure probability can be made 1 using Corollary 6.3. We also show a lower bound, which shows that our algorithm is optimal among guess-and-check algorithms, up to a constant factor in S or T. Specifically, we prove the following lower bound.

Theorem 4.2. *Any guess-and-check algorithm that solves SFI for permutations with success probability at least $3/4$ using S bits of preprocessing and T queries must have $S \geq (N/2) \log(N/6T) - 4$.*

4.1 A non-trivial non-adaptive (guess-and-check) algorithm

In this section, we prove Theorem 4.1. We will use the following fact about balls and bins. The lemma is just a special case of [Mit96, Lemma 2.13], except that it includes explicit constants. We include a proof in Appendix C for completeness.

Lemma 4.3. *Suppose $M \le N/\log N$ balls are thrown uniformly and independently into $N \ge 300$ bins. Let $t(M,N)$ be the largest integer t such that with probability at least 0.9 at least one bin has at least t balls in it. Then $t(M,N) \ge \lfloor \log N/(3\log(N/M))\rfloor$.*

Our main theorem will be a corollary of the following lemma, which gives a parameterized family of guess-and-check algorithms.

Lemma 4.4. *Let T', N', N be positive integers such that $T', N' \le N$. Then, there exists a guess-and-check algorithm $(\mathcal{P},\mathcal{A})$ that solves (N,N)-SFI with success probability $1-\exp(-\Omega(T'))$ using $S \le (16N\log N')/t^*$ bits of preprocessing and $T \le T'\lceil N/N'\rceil$ queries, where $t^* := t(\lceil T'/2\rceil, N')$.*

Before proving the lemma, we show how to use it to prove our main theorem. This amounts to carefully setting parameters and some simple case analysis.

Proof of Theorem 4.1. If $1 \le T < 2N/\log N$, we set $N' = N$, $T' = T$, and we may assume without loss of generality that $N' \ge 300$. Then by Lemma 4.3, the t^* in Lemma 4.4 satisfies $t^* \ge \lfloor \log N/(3\log(2N/T))\rfloor$, so that

$$S \le (16N\log N')/t^* \le 100N\log(2N/T)\ ,$$

and the error probability is $\exp(-\Omega(T')) = \exp(-\Omega(T))$, as needed.

For $2N/\log N \le T \le N$, we set $N' := \lceil 2^{2N/T}\rceil \le N$ and $T' = \lfloor N'/\log N'\rfloor$. We may again assume without loss of generality that $N' \ge 300$. And, it is easy to verify that $T'\lceil N/N'\rceil \le T$. Again, by Lemma 4.3, t^* in Lemma 4.4 satisfies $t^* \ge \lfloor \log N'/3\log(2N'/T')\rfloor$ and

$$S \le (16N\log N')/t^* \le 100N\log(2N'/T') \le 100N\log(4\log N') \le 100N\log(16N/T)\ ,$$

where the third inequality is due to $N'/T' \ge \log(N')/2$, and the fourth inequality is due to $\log N' \le 4N/T$ by our choice of N' and T'. Moreover, the error probability is $\exp(-\Omega(N'/\log N')) \le \exp(-\Omega(2^{N/T}))$, as claimed. □

Proof of Lemma 4.4. For positive integers $A \le N$, we define the map $\lfloor\cdot\rfloor_A : [N] \to [A]$ as $\lfloor B\rfloor_A := \lfloor AB/N\rfloor$. We think of $\lfloor\cdot\rfloor_A$ as partitioning $[N]$ into A sets, each with size roughly N/A (and we could equivalently use any function with this property). Let $g_1,\dots,g_{T'} : [N] \to [N']$ be uniformly random functions derived from the shared randomness of $\mathcal{P}$ and $\mathcal{A}$.

Before describing the algorithm in detail, we provide a high-level overview. The algorithm divides the elements $y \in [N]$ in the range of f into K sets of roughly equal size, according to $\lfloor y\rfloor_K$, where $K \approx N/(T't^*)$. Our algorithm will treat these sets separately.

In particular, the online algorithm will attempt to invert $y \in [N]$ with $\lfloor y\rfloor_K = i$ using the advice $b_{i,1},\dots,b_{i,T'} \in [N']$. Intuitively, we think of each $b_{i,j}$ as suggesting that the preimage x of y might satisfy $\lfloor x\rfloor_{N'} = b_{i,j} + g_j(y) \bmod N'$. I.e., each $b_{i,j}$ suggests roughly N/N' queries that $\mathcal{A}$ should make, corresponding to the roughly N/N' elements $x' \in [N]$ with $\lfloor x'\rfloor_{N'} = b_{i,j} + g_j(y)$. The total number of queries made by our algorithm will therefore be essentially $N/N' \cdot T'$, as claimed.

To choose the $b_{i,j}$, the preprocessing algorithm essentially just greedily picks for each $j = 1,\dots,T'$ the *bin* $b_{i,j}$ that inverts the maximum number of elements y that have not yet been inverted. This corresponds to throwing one ball for each y with $\lfloor y\rfloor_K = i$ that has not yet been inverted into the bin $\lfloor x\rfloor_{N'} - g_j(y) \bmod N'$. Notice that these bins are uniformly random and independent in $[N']$ (because g_j is a random function). Therefore, if the number of uninverted y

with $\lfloor y \rfloor_K = i$ in the jth step is not much smaller than T', then we expect $b_{i,j}$ to successfully invert at least t new choices of these y. After T' steps, we thus expect to invert roughly $T't^\star \approx N/K$ of these elements y. Since we do this for each of the K sets in the partition of the range $[N]$, we expect to invert the entire range. (Here, for simplicity, we have ignored the possibility that the image of f is smaller than the range. Of course, below we handle this carefully.)

We now describe the algorithm in careful detail.

Preprocessing: The preprocessing algorithm $\mathcal{P}$ first fixes an arbitrary preimage $x_y \in [N]$ for each image $y \in f([N])$, and partitions $f([N])$ into K sets $R_{1,0}, \ldots, R_{K,0}$ where $K := \lceil 4N/(T't^\star) \rceil$ and $R_{i,0} := \{y \in f([N]) : \lfloor y \rfloor_K = i\}$. Let $h_j := g_j \circ f$. The preprocessing algorithm then performs the following operations for each $i = 1, 2, \ldots, K$.

1. For each $j = 1, 2, \ldots, T'$,
   (a) Define the *bin* $b_y^{(i,j)} \in [N']$ corresponding to *ball* $y \in R_{i,j-1}$ as $b_y^{(i,j)} := \lfloor x_y \rfloor_{N'} - h_j(x_y) \bmod N'$. Notice that $(h_j(x_y))_{y \in R_{i,j-1}}$ is a uniformly random list of $|R_{i,j-1}|$ elements in $[N']$, and the bins $(b_y^{(i,j)})_{y \in R_{i,j-1}}$ are therefore independent and uniformly distributed in $[N']$.
   (b) Given a bin $b \in [N']$, define $\mathrm{L}_{i,j}(b) := \{y \in R_{i,j-1} : b_y^{(i,j)} = b\}$. Set $b_{i,j} \in [N']$ to maximize $|\mathrm{L}_{i,j}(b_{i,j})|$.
   (c) Let $\mathrm{L}_{i,j} := \mathrm{L}_i(b_{i,j})$. Set $R_{i,j} := R_{i,j-1} \setminus \mathrm{L}_{i,j}$.

Finally, the preprocessing outputs as its advice $\sigma := (b_{i,j})_{i,j}$. Notice that the final advice string consists of KT' elements in $[N']$. Therefore $S \le KT' \lceil \log N' \rceil \le (16N \log N')/t^\star$.

Online: In the online phase, given the advice $\sigma := (b_{i,j})_{i,j}$ and the challenge $y \in [N]$, the algorithm computes $i := \lfloor y \rfloor_K$, and uses the advice $b_{i,j}$ to compute the set of non-adaptive queries $Q_y := \{x : \lfloor x \rfloor_{N'} = b_{i,j} + g_j(y) \bmod N' \text{ for any } j = 1, \ldots, T'\}$. Finally, it simply queries all elements in Q_y and outputs any preimage that it finds. (If no preimages are found, it simply outputs $\perp$.) Notice that this is a guess-and-check algorithm and Q_y has size at most $T' \lceil N/N' \rceil$. Therefore $T \le T' \lceil N/N' \rceil$.

Analysis: Notice that $y \in f([N])$ lies in $R_{i,0}$ for $i := \lfloor y \rfloor_K$, and that y is correctly inverted if $y \in \mathrm{L}_{i,j}$ for some $j \in [T']$. Notice further that $R_{i,0}$ is equal to the disjoint union

$$R_{i,0} = R_{i,T'} \cup \left(\bigcup_j \mathrm{L}_{i,j} \right).$$

Therefore, if $R_{i,T'}$ is empty, it immediately follows that our algorithm successfully inverts y (and indeed successfully inverts all elements in $R_{i,0}$). This is captured by the following claim.

Claim 4.5. *Consider the sequence of random variables $B_0, B_1, \ldots$, where $B_0 \le \lceil T't/4 \rceil$ and $t^\star := t(\lceil T'/2 \rceil, N')$, and B_{i+1} is obtained by throwing B_i balls uniformly and independently into N' bins, and taking $B_{i+1} = B_i - X_i$, where X_i is the maximum number of balls in any bin. With probability at least $1 - \exp(-\Omega(T'))$, it holds that $B_{T'} = 0$.*

Proof. Since $B_{j+1} \le \max\{0, B_j - 1\}$ (i.e., as long as the number of balls thrown is non-zero, there is always a bin with at least one ball in it), it suffices to show that $B_M < M$ where $M := \lceil T'/2 \rceil$ except with probability at most $\exp(-\Omega(M))$.

To that end, notice that for each $j \in [M]$, either $B_{j-1} < M$ (which immediately implies that $B_M < M$) or with probability at least $9/10$, $B_j \le B_{j-1} - t^\star$. Thus, with probability at least $9/10$, $B_j \le \max\{B_{j-1} - t^\star, M - 1\}$. But if this happens at least $\lceil M/2 \rceil$ times, then

$$B_M \le \max\{B_0 - \lceil M/2 \rceil t^\star, M - 1\} \le \max\{\lceil T' t^\star/4 \rceil - \lceil T'/4 \rceil t^\star, M - 1\} < M \; .$$

The probability that $B_M \ge M$ is thus at most the probability that the sum of M independent Bernoulli random variables, each with probability $9/10$ of equaling 1, is no more than $M/2$. But by the Chernoff bound (Lemma 2.9), this probability is at most $\exp(-\Theta(M))$. □

□

4.2 A tight lower bound against guess-and-check algorithms.

In this section, we prove Theorem 4.2. Following De et al. [DTT10] and Dodis et al. [DGK17], we will consider randomized encoding and decoding procedures for a set of functions, and rely on the following lemma which lower bounds the encoding length.

Lemma 4.6. *([DTT10, DGK17]) Suppose there exist randomized encoding and decoding procedures* $(\mathrm{Enc}, \mathrm{Dec})$ *for a set* $\mathcal{F}$*. We say such an encoding has recovery probability* δ *if for all* $f \in \mathcal{F}$*,*

$$\Pr_{r \sim \{0,1\}^\ell}[\mathrm{Dec}(\mathrm{Enc}(f, r), r) = f] \ge \delta \; .$$

The encoding length of $(\mathrm{Enc}, \mathrm{Dec})$*, defined to be* $\max_{f,r}\{|\mathrm{Enc}(f, r)|\}$*, is at least* $\log |\mathcal{F}| - \log 1/\delta$*.*

Our main lemma gives a randomized encoding for the family of permutations given a guess-and-check inversion algorithm.

Lemma 4.7. *Suppose that there exists a guess-and-check algorithm* $(\mathcal{P}, \mathcal{A})$ *that solves SFI for permutations with success probability* $3/4$ *using* S *bits of preprocessing and* T *queries. Then there exists a randomized encoding for the set of all permutations from* $[N]$ *to* $[N]$*, with recovery probability at least* $1/2$ *and encoding length at most*

$$S + \lceil N/2 \rceil \cdot \log T + \log \frac{N!}{\lceil N/2 \rceil!} + 3 \; .$$

We first observe that Theorem 4.2 follows immediately from the above lemmas. Indeed, combining the two lemmas and recalling that there are $N!$ permutations from $[N]$ to $[N]$, we have

$$S + \lceil N/2 \rceil \cdot \log T + \log \frac{N!}{\lceil N/2 \rceil!} + 3 \ge \log N! - \log 2 \; .$$

Hence,

$$S \ge \log \lceil N/2 \rceil! - \lceil N/2 \rceil \cdot \log T - 4 \ge \frac{N}{2} \log \frac{N}{6T} - 4 \; ,$$

where the second inequality is due to the fact $m! \ge (m/e)^m \ge (m/3)^m$ (by Stirling's approximation) and $\lceil N/2 \rceil \ge N/2$.

Proof of Lemma 4.7. Fix an arbitrary permutation $f\colon [N] \to [N]$. We encode f as follows. Given f and randomness r, the encoder simulates $(\mathcal{P}, \mathcal{A})$ on every $y \in [N]$. Let st be the output of $\mathcal{P}(f, r)$ and G be the set of y such that $\mathcal{A}^f(\mathrm{st}, y, r) = f^{-1}(y)$. By an averaging argument,

$$\Pr_{r\sim\{0,1\}^\ell}[\Pr_{y\sim[N]}[\mathcal{A}^f(\mathrm{st}, y, r) = f^{-1}(y)] \geq \frac{1}{2}] \geq \frac{1}{2}\,.$$

In other words, with probability at least $1/2$ the size of G is at least $N' := \lceil N/2 \rceil$. Assuming $|G| \geq N'$, we pick a set $G' \subseteq G$ with size exactly N' and encode f as follows,

1. Include st, and a description of G'. This requires $S + \lceil \log \binom{N}{N'} \rceil$ bits.
2. For each $y \in G'$ (in lexicographic order), run $\mathcal{A}^f(\mathrm{st}, y, r)$ and include the index i such that the answer to the ith oracle query is y. This requires $\lceil N' \cdot \log T \rceil$ bits in total.
3. Store the mapping from $[N] \setminus f^{-1}(G')$ to $[N] \setminus G'$ corresponding to f restricted to $[N] \setminus f^{-1}(G')$ using $\lceil \log(N - N')! \rceil$ bits.

Given the shared randomness r, the decoder does the following:

1. Recover st and G'.
2. For each $y \in G'$, run $\mathcal{A}(\mathrm{st}, y, r)$ to generate T non-adaptive queries $x_1, \ldots, x_T$, recover the index i and set $f(x_i) = y$. We remark that this step heavily relies on the guess-and-check property of $\mathcal{A}$.
3. After the above two steps, the decoder reconstructs $f^{-1}(G')$ and G' (hence $[N] \setminus f^{-1}(G')$ and $[N] \setminus G'$). Then the decoder recovers the values of $[N] \setminus f^{-1}(G')$ using the remainder of the encoding.

Assuming $|G| \geq N/2$, the decoding procedure recovers f. The encoding length is

$$S + \lceil \log \binom{N}{N'} \rceil + \lceil N' \cdot \log T \rceil + \lceil \log(N - N')! \rceil \leq S + N' \cdot \log T + \log \frac{N!}{N'!} + 3\,,$$

as claimed. □

5 Comparing variants of function inversion

In this section, we prove that different formulations of the function inversion problem are equivalent (up to polylogarithmic factors in S and T). First, we prove that the decision version of the Function Inversion problem, that merely asks to check whether a query y is in the image of the preprocessed function f, is as hard as the search version of the problem where the goal is to find a preimage of y. We prove this equivalence for three different settings: for arbitrary (i.e., worst-case) functions in Section 5.1, for random functions in Section 5.2, and for injective functions in Appendix B.[9] Also, [CK19, Lemma 21] proves that for worst-case functions and $M > N$, inverting $f\colon [N] \to [M]$ is as

[9] We remark that the result for injective functions is very similar to [CK19, Theorem 8]. We simply include it for completeness.

hard as inverting $f'\colon [N] \to [N]$. In Appendix A, we show that this result can be extended to the setting of random functions.

These equivalences suggest that the hardness of function inversion is specified by the domain size and the class of functions (worst-case/injective/random), but not by the search/decision type of the problem or the range size.

5.1 Search-to-decision reduction for arbitrary functions

In this section, we prove an essentially tight search-to-decision reduction for worst-case function inversion. Namely, given an algorithm that solves DFI (for all functions; see Definition 2.3) in query time T and preprocessing S, we design an algorithm that solves SFI (for all functions) in query time $T \cdot \mathrm{poly}(\log N)$ and preprocessing $S \cdot \mathrm{poly}(\log N)$ (or even query time $O(T \cdot \log N)$ and preprocessing $O(S \cdot \log N)$, see Remark 5.3).

First, in Lemma 5.1 we observe that, given an algorithm for DFI, one can solve SFI on all inputs y that have unique preimages. Then, in Theorem 5.2 we use the Isolation Lemma [VV85, MVV87, Ta-15] to reduce the general case of SFI to the case where y has a unique preimage.

Lemma 5.1. *Let $N = 2^n$ and $\varepsilon := \varepsilon(N) \in (0, 1/2]$. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves (N, M)-DFI with advantage ε using S bits of preprocessing and T queries. Then there exists an algorithm $(\mathcal{P}', \mathcal{A}')$ that uses $S' \le O(Sn(\log n)/\varepsilon^2)$ bits of preprocessing and $T' \le O(Tn(\log n)/\varepsilon^2)$ queries with the following guarantees. For every $f\colon [N] \to [M]$ and* every $y \in [M]$ satisfying $|\{f^{-1}(y)\}| = 1$,

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f, r), y, r) \ : \ f(x') = y] \ge 1 - 1/(10n^2) \, .$$

Furthermore, for every $f\colon [N] \to [M]$ and every $y \in [M]$,

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f, r), y, r) \ : \ x' \neq \perp \text{ and } f(x') \neq y] \le 1/(10n^2) \, .^{10}$$

Proof. By running $(\mathcal{P}, \mathcal{A})$ a total of $k = O((\log n)/\varepsilon^2)$ times with fresh randomness and taking the majority of the query answers, we may assume that $(\mathcal{P}, \mathcal{A})$ has failure probability at most $1/(20n^3)$ (by the Chernoff bound in Lemma 2.9). Therefore, it suffices to show an algorithm $(\mathcal{P}', \mathcal{A}')$ that uses $O(Tn)$ queries and $O(Sn)$ bits of preprocessed advice for the special case when $\varepsilon = 1/2 - 1/(20n^3)$. For an $x \in [N]$, by x_i we denote the ith bit in the binary representation of x. For inverting a function $f\colon [N] \to [M]$, we first define the following $2n = 2 \log N$ functions from $[N]$ to $[M]$. For $i \in [n]$, let

$$g_i^0(x) = \begin{cases} f(x) & \text{if } x_i = 0 \, , \\ M & \text{otherwise} \, . \end{cases} \qquad g_i^1(x) = \begin{cases} f(x) & \text{if } x_i = 1 \, , \\ M & \text{otherwise} \, . \end{cases}$$

Notice that an oracle query to g_i^b can be trivially simulated by making at most one oracle query to f.

We now present the algorithms $\mathcal{P}'$ and $\mathcal{A}'$. For each $i \in [n]$ and $b \in \{0,1\}$, $r_i^b \sim \{0,1\}^\ell$, $\mathcal{P}'$ runs $\sigma_i^b \leftarrow \mathcal{P}(g_i^b, r_i^b)$, and outputs $\sigma' = (\sigma_1^0, \sigma_1^1 \ldots, \sigma_n^0, \sigma_n^1)$. Finally, $\mathcal{P}'$ outputs a preimage of M if it exists: $z \in f^{-1}(M)$. This implies that $S' \le O(Sn + n) = O(Sn)$.

[10] One could reduce the latter probability of failure to 0 with an adaptive reduction, but we prefer to keep the reduction non-adaptive with a small probability of error.

Given a query $y \in [M]$, the algorithm $\mathcal{A}'$ (with oracle access to f) proceeds as follows. If $y = M$, the algorithm just outputs the stored preimage z of M. For $y \in [M-1]$, the algorithm $\mathcal{A}'$ for each $i \in [n]$ and $b \in \{0,1\}$, computes $d_i^b = \mathcal{A}^{g_i^b}(\sigma_i^b, y, r_i^b)$.

Intuitively, we expect $d_i^0 = 1$ if there exists a preimage of y with the ith bit 0, and $d_i^1 = 1$ if there exists a preimage of y with the ith bit 1. Now we check that y has a unique preimage as follows. If for some $i \in [n]$, $d_i^0 = d_i^1$, then $\mathcal{A}'$ outputs $\perp$ and terminates. Otherwise, for each i, $\mathcal{A}'$ sets $x'_i = b$ where $b \in \{0,1\}$ is such that $d_i^b = 1$ and $d_i^{1-b} = 0$. Finally, the algorithm $\mathcal{A}'$ outputs $x' = (x'_1, \ldots, x'_n) \in [N]$.

It is clear that the number of queries made by $\mathcal{A}'$ is at most $O(Tn)$, as needed. If $y \in [M]$ has a unique preimage x under f, then all $2n$ queries to $\mathcal{A}$ are simultaneously answered correctly with probability at least $1 - 2n/(20n^3) \geq 1 - 1/(10n^2)$, which immediately implies

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f,r), y, r) \ : \ x' = x] \geq 1 - 1/(10n^2) \ .$$

If $y \in [M]$ doesn't have a preimage under f, then

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f,r), y, r) \ : \ x' = \perp] \geq \Pr_{r \sim \{0,1\}^{\ell'}}[d_1^0 = 0 \text{ and } d_1^1 = 0] \geq 1 - 1/(10n^3) \ .$$

Finally, if $y \in [M]$ has at least two preimages that differ in ith bit, then

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f,r), y, r) \ : \ x' = \perp] \geq \Pr_{r \sim \{0,1\}^{\ell'}}[d_i^0 = 1 \text{ and } d_i^1 = 1] \geq 1 - 1/(10n^3) \ .$$

The result follows. □

We are now ready to prove the main result of this section. The main difference in the statements of Lemma 5.1 and Theorem 5.2 is that the SFI algorithm in Lemma 5.1 is only guaranteed to succeed on queries that have a unique preimage, while the SFI algorithm in Theorem 5.2 works for all queries.

Theorem 5.2. *Let $N = 2^n$, and let $\varepsilon := \varepsilon(N) \in (0, 1/2]$. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves (N, M)-DFI with advantage ε using S bits of preprocessing and T queries. Then there exists an algorithm $(\mathcal{P}'', \mathcal{A}'')$ that solves (N, M)-SFI with success probability 0.9, $S'' \leq O(Sn^2(\log n)/\varepsilon^2)$ bits of preprocessing, and $T'' \leq O(Tn^2(\log n)/\varepsilon^2)$ queries.*

Proof. Let $n = \log N$ and $k = 50$. Using shared randomness, the algorithms $\mathcal{P}''$ and $\mathcal{A}''$ define kn sets $U_{i,j} \subseteq [N]$ for $i \in [n]$ and $j \in [k]$, where each each element $x \in [N]$ is placed in $U_{i,j}$ independently with probability 2^{-i-1}.

In order to invert a function $f\colon [N] \to [M]$, we first define the following kn functions from $[N]$ to $[M]$. For $i \in [n]$ and $j \in [k]$, let

$$g_{i,j}(x) = \begin{cases} f(x) & \text{if } x \in U_{i,j} \ , \\ M & \text{otherwise} \ . \end{cases}$$

Notice that an oracle query to $g_{i,j}$ can be trivially simulated by making at most one oracle query to f.

The algorithm $\mathcal{P}''$ runs the algorithm $\mathcal{P}'$ guaranteed in Lemma 5.1 for the kn functions $g_{i,j}$. That is, let $r_{1,1}, \ldots, r_{n,k}$ be independent random binary strings of length ℓ. For $i \in [n]$ and $j \in [k]$,

$\mathcal{P}''$ runs $\sigma_{i,j} \leftarrow \mathcal{P}'(g_{i,j}, r_{i,j})$, and outputs $\sigma' = (\sigma_{1,1}, \ldots, \sigma_{n,k})$. Finally, $\mathcal{P}''$ outputs a preimage of M if it exists: $z = f^{-1}(M)$. In particular, $S'' \le O(S'kn + n) \le O(Sn^2(\log n)/\varepsilon^2)$.

Given a query $y \in [M]$, the algorithm $\mathcal{A}''$ (with oracle access to f) proceeds as follows. If $y = M$, then $\mathcal{A}''$ just outputs the stored preimage of M. If $y < M$, then for each $i \in [n]$ and $j \in [k]$, $\mathcal{A}''$ computes $x_{i,j} = (\mathcal{A}')^{g_{i,j}}(\sigma_{i,j}, y, r_{i,j})$, where $\mathcal{A}'$ is the algorithm guaranteed in Lemma 5.1. If for all $i \in [n]$ and $j \in [k]$, $x_{i,j} = \perp$, then $\mathcal{A}''$ outputs $\perp$, otherwise $\mathcal{A}''$ outputs any $x_{i,j}$ with $x_{i,j} \neq \perp$.

It is immediate that $\mathcal{A}''$ makes at most $T'' \le O(T'nk) = O(Tn^2(\log n)/\varepsilon^2)$ queries to f. By Lemma 5.1, with probability at least $1 - kn/(10n^2) \ge 0.99$, all kn queries to the algorithms $\mathcal{P}'$ and $\mathcal{A}'$ are answered correctly. In particular, the algorithm $(\mathcal{A}')^{g_{i,j}}$ returns $x_{i,j}$ satisfying $f(x_{i,j}) = y$ for $y \in [M-1]$ having a unique preimage under $g_{i,j}$, and returns $\perp$ (or a correct preimage of y) otherwise. Therefore, it suffices to prove that for every $y \in f([N])$,

$$\Pr_{r \sim \{0,1\}^{\ell'}}[\exists i \in [n], j \in [k] \colon |g_{i,j}^{-1}(y)| = 1] \ge 0.99 \,.$$

For a $y \in f([N])$, let $S_y = \{x \colon f(x) = y\}$. Let $i \in [n]$ be such that $2^{i-1} \le |S_x| \le 2^i$. For $x \in S_y$, let $p = \Pr_r[x \in U_{i,j}] = 2^{-i-1}$. We have that $1/4 \le p|S_y| \le 1/2$. Then for each $j \in [k]$,

$$\Pr_{r \sim \{0,1\}^{\ell'}}[|g_{i,j}^{-1}(y)| = 1] = |S_y| \cdot p \cdot (1-p)^{|S_y|-1} \ge 1/4 \cdot e^{-2p|S_y|} \ge 1/(4e),$$

where we used $1 - z \ge e^{-2z}$ for $0 \le z \le 1/2$. From independence of $U_{i,j}$ and $k = 50$,

$$\Pr_{r \sim \{0,1\}^{\ell'}}[\exists j \in [k] \colon |g_{i,j}^{-1}(y)| = 1] \ge 1 - (1 - 1/(4e))^k \ge 0.99 \,,$$

which finishes the proof. □

Remark 5.3. *A few extensions of Theorem 5.2 are in order.*

1. *To simplify the proof of Theorem 5.2, we used shared randomness to gain access to* the same *random sets $U_{i,j}$ in the preprocessing and query algorithms. As always, shared randomness can be removed using Corollary 6.3. Alternatively, one can avoid using shared randomness in the proof of Theorem 5.2 by sampling $O(n)$ functions from a family of pairwise independent hash functions (see, e.g., [VV85]), and storing their descriptions using additional $O(n^2)$ bits of advice.*

2. *We remark that the presented search-to-decision reduction is non-adaptive, so a non-adaptive algorithm for DFI implies a non-adaptive algorithm for SFI (and an adaptive algorithm for DFI implies an adaptive algorithm for SFI).*

3. *In the proof of Lemma 5.1, the* $\log n$ *factor in the advice length and the number of queries comes from the amplification of the success probability of the assumed DFI algorithm from $1/2 + \varepsilon$ to $1 - O(1/n^3)$. We remark that one can get rid of this* $\log n$ *factor by recovering the bits of $C(x)$ rather than the bits of x for a good linear code C (similarly to how it is done in the proof of Theorem 5.4). This modification will also improve the parameters S'' and T'' in Theorem 5.2 by a* $\log n$ *factor (though unfortunately it does* not *preserve non-adaptivity).*

4. *Furthermore, in the proof of Theorem 5.2, one can define the functions $g_{i,j} \colon [N/2^i] \to [M]$ with domain of size $N/2^i$. If the assumed DFI algorithm works for all (large enough) values*

of N and has advice length and number of queries $S(N)$ and $T(N)$, then this modification would give us $S'' \le O((S(N) + S(N/2) + \cdots + S(N/2^n))n)$ and $T'' \le O((T(N) + T(N/2) + \cdots + T(N/2^n))n)$. For the most interesting regime of $S(N), T(N) = N^{\Theta(1)}$, this gives us $S'' \le O(Sn)$ and $T'' \le O(Tn)$.

5.2 Search-to-decision reduction for average-case functions

In this section, we show a different search-to-decision reduction for average-case function inversion. (See Definition 2.4 for the formal definition of average-case SFI and Definition 2.5 for the formal definition of average-case DFI.) The proof of Theorem 5.2 does not work for the case of average-case functions as Lemma 5.1 heavily relies on the fact that the assumed DFI algorithm works for all functions. Nevertheless, we can extend the techniques of the previous section to recover bits of a certain encoding of x rather than the individual bits of x and prove an essentially tight search-to-decision reduction for average-case function inversion in Theorem 5.4.

Theorem 5.4. *Let $N = 2^n$. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves average-case $(2N, M)$-DFI with advantage $\varepsilon \ge 1/2 - \exp(-2N/M - 2N/M^2)/4$, using S bits of preprocessing and T queries. Then for any constant $\delta \in (0, 1/4)$, there exists an algorithm $(\mathcal{P}', \mathcal{A}')$ that solves average-case (N, M)-SFI with success probability $\exp(-2N/M - 2N/M^2) - (1/2 - \varepsilon)/(1/4 - \delta)$ using $S' \le O_\delta(nS)$ bits of preprocessing and $T' \le O_\delta(nT)$ queries.*

Proof. Let $\mathcal{C} := \mathcal{C}_{n+1,\delta}$ be a code as guaranteed by Theorem 2.11, with generator matrix $\mathbf{C} := \mathbf{C}_{n+1,\delta} \in \mathbb{F}_2^{m \times (n+1)}$ with $m \le O_\delta(n)$ and efficient decoding algorithm Dec. Recall that for $1 \le i \le m$, we write $\mathcal{C}_i := \{\mathbf{c} \in \mathcal{C} \ : \ c_i = 0\}$ and that we may assume without loss of generality that $|\mathcal{C}_i| = |\mathcal{C}|/2 = 2^n = N$. Notice that $\mathcal{X}_i := \{\mathbf{z} \ : \ \mathbf{Cz} \in \mathcal{C}_i\}$ is an n-dimensional subspace of $\mathbb{F}_2^{n+1}$, and let $\mathbf{B}_i \in \mathbb{F}_2^{(n+1)\times n}$ be a basis for $\mathcal{X}_i$. ($\mathbf{B}_i$ can be computed efficiently, given $\mathbf{C}$ and i.) Finally, we define $f_i : [N] \to [M]$ by $f_i(x) := f(\mathbf{B}_i \mathbf{x})$, where here we interpret $x \in [N]$ as an n-bit vector $\mathbf{x} \in \mathbb{F}_2^n$ by writing x in binary.

Given the setup above, our algorithms $\mathcal{P}'$ and $\mathcal{A}'$ are relatively simple. On input $f \sim \{g : [2N] \to [M]\}$ and $r \sim \{0,1\}^\ell$, $\mathcal{P}'$ runs $\sigma_i \leftarrow \mathcal{P}(f_i, r)$ for $i = 1, \ldots, m$ and outputs $\sigma' := (\sigma_1, \ldots, \sigma_m)$.[11] In particular $S' \le mS \le O_\delta(nS)$, as claimed.

The algorithm $\mathcal{A}'$ (with oracle access to f) behaves as follows on input $\sigma' = (\sigma_1, \ldots, \sigma_m)$, $y := f(x)$, and r. For $i = 1, \ldots, m$, it computes $d_i := 1 - \mathcal{A}^{f_i}(\sigma_i, y, r)$, which can be done using at most T queries to f. (Intuitively, "$d_i = 0$ means that there is likely to be a preimage of y in $\mathcal{X}_i$.") Finally, the algorithm outputs $x' := \mathsf{Dec}(\mathbf{d}) \in [2N]$ (where we interpret the $(n+1)$-bit string returned by Dec as the binary representation of an element of $[2N]$). In particular, the total number of oracle queries made by $\mathcal{A}'$ is $T' \le mT \le O_\delta(nT)$, as claimed.

Let $\mathbf{c} := \mathbf{Cx} \in \mathbb{F}_2^m$, where $\mathbf{x} \in \mathbb{F}_2^{n+1}$ is the bit string corresponding to the binary representation of $x \in [2N]$. Let $\mathbf{e} := \mathbf{c} \oplus \mathbf{d} \in \mathbb{F}_2^m$. Notice that $x' = x$ if $\|\mathbf{e}\|_H \le (1/4 - \delta)m$. So, it suffices to argue that $\|\mathbf{e}\|_H \le (1/4 - \delta)m$ with the claimed probability.

Let E_{u} be the event that $|\{x^* \ : \ f(x^*) = y\}| = 1$, i.e., the event that x is the unique preimage of y under f. Let $e'_i = 1$ if and only if "$\mathcal{A}^{f_i}(\sigma_i, y, r)$ fails," i.e., $e'_i = 1$ if *either* $d_i = 0$ but there is a preimage of y in $\mathcal{X}_i$ *or* $d_i = 1$ but there is a preimage of y in $\mathcal{X}_i$. Notice that $\Pr[e'_i = 1] \le (1/2 - \varepsilon)$ by assumption, so that $\mathbb{E}[\|\mathbf{e}'\|_H] \le (1/2 - \varepsilon)m$. (Here, we are using the fact that (1) f_i is a uniformly

[11] We really can run $\mathcal{P}$ with the same random bit string r in each of these calls, as our proof will not require independence of the randomness used by the m evaluations of $\mathcal{P}$.

random function; (2) conditioned on y being in the image of f_i, y is distributed identically to $f_i(x^*)$ for uniformly random $x^* \sim [N]$; and (3) conditioned on y not being in the image of f_i, y is uniformly random in $[M] \setminus f_i([N])$.) Notice that, if E_{u} holds, then $\mathbf{e} = \mathbf{e}'$. Therefore,

$$\begin{aligned}\Pr[f(x') = y] &\geq \Pr\left[\|\mathbf{e}\|_H \leq (1/4 - \delta)m \text{ and } E_{\mathsf{u}}\right] \\ &= \Pr\left[\|\mathbf{e}'\|_H \leq (1/4 - \delta)m \text{ and } E_{\mathsf{u}}\right] \\ &\geq \Pr[E_{\mathsf{u}}] - \Pr\left[\|\mathbf{e}'\|_H > (1/4 - \delta)m\right] .\end{aligned}$$

Finally, by Claim 2.10, we have $\Pr[E_{\mathsf{u}}] \geq e^{-2N/M - 2N/M^2}$, and by Markov's inequality, we have

$$\Pr\left[\|\mathbf{e}'\|_H > (1/4 - \delta)m\right] \leq \frac{\mathbb{E}[\|\mathbf{e}'\|_H]}{(1/4 - \delta)m} \leq \frac{1/2 - \varepsilon}{1/4 - \delta} .$$

The result follows. □

Remark 5.5.

1. *Similarly to the reduction in Theorem 5.2, the search-to-decision reduction of Theorem 5.4 is non-adaptive.*

2. *For simplicity of presentation, in the proof of Theorem 5.4 we only invert queries that have a unique preimage. For the standard setting of $M \geq \Omega(N)$ this already gives relatively high constant probability of success. One can amplify this probability to an arbitrarily large constant by using the Isolation Lemma (see the proof of Theorem 5.2) to invert points with more than one preimage, too.*

3. *A drawback of Theorem 5.4 is that it requires the DFI algorithm to have very large advantage ε. This is because we actually need the DFI algorithm to have non-negligible advantage in distinguishing between (1) uniformly random y that is not in the image of f; and (2) uniformly random y with $|f^{-1}(y)| = 1$ (i.e., a random image that has a unique preimage). (This is true even if we use the Isolation Lemma idea described above.) We could have worked directly with this assumption on the DFI algorithm, but we prefer the simpler (but strictly stronger) assumption in Theorem 5.4.*

6 Removing shared randomness

In this section, we adapt to our setting Newman's technique for converting public-coin protocols to private-coin protocols [New91] in the context of communication complexity. We first define a general notion of a computational problem with preprocessing to which our technique will apply.

Definition 6.1. *Let $\mathcal{F}$ be a set of functions $f : D \to R$, and let $\mathcal{Y}$, $\mathcal{X}$ be sets. A preprocessing-queries tradeoff problem is a function $g : \mathcal{F} \times \mathcal{Y} \to 2^{\mathcal{X}}$, where $2^{\mathcal{X}}$ denotes the powerset of $\mathcal{X}$. Let $(\mathcal{P}, \mathcal{A})$ be a pair of randomized algorithms. We say that*

1. *$(\mathcal{P}, \mathcal{A})$ solves g with success probability $\delta \in (0, 1]$ if for all $f \in \mathcal{F}$ and $y \in \mathcal{Y}$,*

$$\Pr_{r \sim \{0,1\}^l}[\mathcal{A}^f(\mathcal{P}(f, r), y, r) \in g(f, y)] \geq \delta.$$

2. $(\mathcal{P}, \mathcal{A})$ *solves* g *without shared randomness with success probability* $\delta \in (0,1]$ *if for all* $f \in \mathcal{F}$ *and* $y \in \mathcal{Y}$,

$$\Pr_{r_1, r_2 \sim \{0,1\}^{l'}}[\mathcal{A}^f(\mathcal{P}(f, r_1), y, r_2) \in g(f, y)] \geq \delta.$$

Our generic lemma for removing shared randomness is as follows.

Lemma 6.2. *Suppose there exists an algorithm that solves a preprocessing-queries tradeoff problem* $g : \mathcal{F} \times \mathcal{Y} \to 2^{\mathcal{X}}$ *with success probability* $1-\varepsilon$ *using preprocessing* S *and* T*. Then there exists another algorithm that solves* g *without shared randomness, with success probability* $1 - 2\varepsilon$*, preprocessing* $S + \log(K/\varepsilon^2) + O(1)$*, and* T *queries, where* $K = \log|\mathcal{F} \times \mathcal{Y}|$*. If the first algorithm is non-adaptive (resp. guess-and-check) then so is the second. Moreover, the success probability can be increased to* 1 *at the cost of an additional* $4\varepsilon|\mathcal{Y}|\lceil\log|\mathcal{Y}|\rceil$ *bits of preprocessing.*

Proof. The proof is adapted from the proof of Newman's technique given in [RY20]. Sample $k = O(2K/\varepsilon^2)$ independent random strings $r_1, \ldots, r_k \in \{0,1\}^l$.

We claim that with probability at least $1 - 2^{-K}$, these random strings satisfy the following property: For all functions $f \in \mathcal{F}$ and inputs $y \in \mathcal{Y}$, we have

$$\Pr_{i \sim [k]}[\mathcal{A}^f(\mathcal{P}(f, r_i), y, r_i) \in g(f, y)] \geq 1 - 2\varepsilon. \tag{7}$$

From the claim, it follows that k fixed strings $r_1^*, \ldots, r_k^*$ with this property must exist. Then the algorithms $(\mathcal{A}', \mathcal{P}')$ are simple. On input f, $\mathcal{P}'$ first samples $i \sim [k]$, then simulates $\mathcal{P}$ to compute $st := \mathcal{P}(f, r_i^*)$. It outputs advice (st, i). On input y, $\mathcal{A}'^f$ simply returns $\mathcal{A}^f(st, y, r_i^*)$. Clearly $\mathcal{A}'$ is non-adaptive (resp. guess-and-check) if $\mathcal{A}$ is.

It remains to prove the claim. Fix a function f and an input y. For each independent random string r_i we have

$$\Pr_{r_i}[\mathcal{A}^f(\mathcal{P}(f, r_i), y, r_i) \in g(f, y)] \geq 1 - \varepsilon.$$

Hence by the Chernoff bound (Lemma 2.9), the probability that $2\varepsilon k$ strings r_i satisfy $\mathcal{A}^f(\mathcal{P}(f, r_i), y, r_i) \notin g(f, y)$ is at most $2^{\Omega(\varepsilon^2 k)} \leq 2^{-2K}$. Since there are at most 2^K possible pairs (f, y), by union bound, the probability that this occurs for any f, y is at most 2^{-K}, as claimed.

For the "Moreover", fix a function $f \in \mathcal{F}$. Notice that by an averaging argument, Eq. (7) implies that for some $i^* \in [k]$, $r_{i^*}^*$ satisfies

$$\Pr_{y \in \mathcal{Y}}[\mathcal{A}^f(\mathcal{P}(f, r_i^*), y, r_i^*) \in g(f, y)] \geq 1 - 2\varepsilon\ .$$

Thus there are only $b = 2\varepsilon|\mathcal{Y}|$ inputs $y_1, \ldots, y_b$ for which $\mathcal{A}^f(\mathcal{P}(f, r_{i^*}^*), y_j, r_{i^*}^*) \notin g(f, y)$. $\mathcal{P}'$ outputs (st, i^*, E), where $E := \{(y_j, x_j)\}_{j \in [b]}$, and for each $j \in [b]$, $x_j \in g(f, y)$. (Such an x_j is guaranteed to exist because the original algorithm $(\mathcal{A}, \mathcal{P})$ is assumed to have positive success probability on all input-challenge pairs (f, y).) Given challenge y, $\mathcal{A}'^f$ first checks if $(y, x) \in E$ for some $x \in \mathcal{X}$. If so, it returns x. Otherwise, it returns $\mathcal{A}^f(st, y, r_{i^*}^*)$ as before. It is easy to see that $(\mathcal{P}', \mathcal{A}')$ always succeeds, uses at most $S + \log(K/\varepsilon^2) + 4\varepsilon|\mathcal{Y}|\lceil\log|\mathcal{Y}|\rceil + O(1)$ bits of preprocessing, and uses at most T queries. And again, $\mathcal{A}'$ is clearly non-adaptive (resp. guess-and-check) if $\mathcal{A}$ is. □

(Of course, choosing the strings at random will not allow us to obtain success probability 1.) The following is an immediate corollary in our setting.

Corollary 6.3. *Suppose that for some class $\mathcal{F}$ of functions $f : [N] \to [M]$ there exists a function-inversion algorithm that solves (N, M)-SFI (resp. solves (N, M)-DFI) for $\mathcal{F}$ with success probability $1 - \varepsilon$, using preprocessing S, and queries T. Then there exists a function-inversion algorithm that solves (N, M)-SFI (resp. solves (N, M)-DFI) for $\mathcal{F}$ with success probability $1 - 2\varepsilon$ without shared randomness, using $S + \log(N/\varepsilon^2) + \log\log M + O(1)$ bits of preprocessing and T queries. If the first algorithm is non-adaptive (resp. guess-and-check) then so is the second. Moreover, the success probability can be made 1 at the cost of an additional $4\varepsilon N \log N$ bits of preprocessing.*

Proof. It is easy to check that each of these function-inversion problems is a preprocessing-queries tradeoff problem, with $\mathcal{Y} = [M]$. Thus Lemma 6.2 applies. So it suffices to observe that

$$\begin{aligned}
\log(K/\varepsilon^2) &= \log K - \log \varepsilon^2 \\
&= \log\log|\mathcal{F} \times \mathcal{Y}| - \log \varepsilon^2 \\
&= \log\log M^{N+1} - \log \varepsilon^2 \\
&= \log((N+1)\log M) - \log \varepsilon^2 \\
&= \log(N+1) + \log\log M - \log \varepsilon^2 \\
&= O(1) + \log N - \log \varepsilon^2 + \log\log M \\
&= O(1) + \log(N/\varepsilon^2) + \log\log M.
\end{aligned}$$

□

References

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992. 11

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO*, 2006. 1

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *CRYPTO*, 2018. 1, 8

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John Steinberger. Random oracles and non-uniformity. In *Eurocrypt*, 2018. 1, 8

[CGLQ20] Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *FOCS*, 2020. 1

[CHM20] Dror Chawin, Iftach Haitner, and Noam Mazor. Lower bounds on the time/memory tradeoff of function inversion. In *TCC*, 2020. 1, 2, 3, 8

[CK19] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *TCC*, 2019. 1, 2, 3, 4, 7, 8, 9, 18, 22, 32, 34

[CLQ20] Kai-Min Chung, Tai-Ning Liao, and Luowen Qian. Lower bounds for function inversion with quantum advice. In *ITC*, 2020. 1

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT*, 2017. 1, 6, 8, 21

[DKKS21] Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová. Data structures lower bounds and popular conjectures. In *ESA*, 2021. 1, 2, 8

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *CRYPTO*, 2010. 1, 6, 8, 9, 21

[FN91] Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *STOC*, 1991. 1, 2, 4, 9, 12, 13

[GGH+20] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. Data structures meet cryptography: 3SUM with preprocessing. In *STOC*, 2020. 1

[GGKL21] Nick Gravin, Siyao Guo, Tsz Chiu Kwok, and Pinyan Lu. Concentration bounds for almost k-wise independence with applications to non-uniform security. In *SODA*, 2021. 8

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *FOCS*, 2000. 1

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. 1, 2, 12

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972. 11

[Mit96] Michael David Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, 1996. 18, 34, 35

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977. 11

[MU17] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017. 11

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *STOC*, 1987. 23

[NABT15] Aran Nayebi, Scott Aaronson, Aleksandrs Belovs, and Luca Trevisan. Quantum lower bound for inverting a permutation with advice. *Quantum Information & Computation*, 15(11-12):901–913, 2015. 1

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Information processing letters*, 39(2):67–71, 1991. 5, 27

[RY20] Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, 2020. 28

[Spi95] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In *STOC*, 1995. 11

[Ta-15] Noam Ta-Shma. A simple proof of the isolation lemma, 2015. https://eccc.weizmann.ac.il//report/2015/080/. 23

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *CRYPTO*, 2007. 1

[VV85] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. In *STOC*, 1985. 7, 23, 25

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, 2005. 1

[Yao90] Andrew Chi-Chih Yao. Coherent functions and program checkers. In *STOC*, 1990. 1, 33

A Reducing the range size

In this section, we prove that for worst-case and average-case function inversion, the size of the range does not matter. The result of Lemma A.1 was already proven in [CK19, Lemma 21], and the presented proof is similar to that of [CK19, Lemma 21]. We include it for completeness and to simplify the presentation in Lemma A.2. Recall that we denote (N, N)-SFI by SFI.

Lemma A.1. *Let $0 < \varepsilon, \delta < 1$ be constants, and let $M \geq N$. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves worst-case SFI with success probability ε using S bits of preprocessing and T queries. Then there exists an algorithm $(\mathcal{P}', \mathcal{A}')$ that solves worst-case (N, M)-SFI with success probability δ, $S' = O(S)$ bits of preprocessing, and $T' = O(T)$ queries.*

Proof. Let $k := \lceil \log(1-\delta)/\log(1-\varepsilon/e^2) \rceil$. Using shared randomness, the algorithms $\mathcal{P}'$ and $\mathcal{A}'$ define k independent random functions $h_1, \ldots, h_k \colon [M] \to [N]$.

In order to invert a function $f \colon [N] \to [M]$, we define the functions $g_1, \ldots, g_k \colon [N] \to [N]$ as $g_i(x) = h_i(f(x))$ for $x \in [N]$ and $i \in [k]$. Let $r_1, \ldots, r_k \in \{0,1\}^\ell$ be independent random binary strings of length ℓ. The algorithm $\mathcal{P}'$ runs $\sigma_i \leftarrow \mathcal{P}(g_i, r_i)$ for each $i \in [k]$, and outputs $\sigma' = (\sigma_1, \ldots, \sigma_k)$. In particular, $S' \leq Sk = O(S)$.

The algorithm $\mathcal{A}'$, given a query $y \in [M]$ proceeds as follows. For $i \in [k]$, it computes $x^{(i)} \leftarrow \mathcal{A}^{g_i}(\sigma_i, h_i(y), r_i)$. Then $\mathcal{A}'$ makes k additional queries to f, and outputs $x^{(i)}$ if $f(x^{(i)}) = y$ for an $i \in [k]$. If for all $i \in [k]$, $f(x^{(i)}) \neq y$, then $\mathcal{A}'$ outputs $\perp$. We have that the number of queries made by $\mathcal{A}'$ is $T' \leq Tk + k = O(T)$.

Clearly, if $\mathcal{A}'$ outputs $x' \neq \perp$, then $f(x') = y$. It remains to show that for each $y \in f([N])$, the algorithm $\mathcal{A}'$ outputs $x \in f^{-1}(y)$ with probability at least δ. For every $y \in f([N])$, let E_y^i be the event that $|\{y' \in [M] \colon h_i(y') = h_i(y)\}| = 1$, i.e., the event that y is the unique preimage of $h_i(y)$ under h_i. Notice that if E_y^i holds, then every preimage of $h_i(y)$ under g_i is also a preimage of y under f. By Claim 2.10 and $M \geq N$,

$$\Pr_{h_i \sim \{h:[M]\to[N]\}}[E_y^i] \geq e^{-N/M - N/M^2} \geq e^{-2} \,.$$

For every function f and every $y \in f([N])$, if E_y^i holds, then $\mathcal{A}$ returns a correct preimage of y under f with probability at least ε. By repeating this $k = \lceil \log(1-\delta)/\log(1-\varepsilon/e^2) \rceil$ times, we have that for every f and every $y \in [M]$, the algorithm $\mathcal{A}'$ finds a correct preimage of y with probability at least δ. □

Lemma A.2. *Let $\delta := \delta(N) \in (0, 1]$, $n = \log N$, and let M be a multiple of N. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves average-case SFI with success probability δ using S bits of preprocessing and T queries. Then there exists an algorithm $(\mathcal{P}', \mathcal{A}')$ that solves average-case (N, M)-SFI with success probability $\delta - 1/(10n^2)$, $S' = O(S \log n)$ bits of preprocessing, and $T' = O(T \log n)$ queries.*

Proof. Let $k = \lceil 4 \log n + 6 \rceil$. Using shared randomness, the algorithms $\mathcal{P}'$ and $\mathcal{A}'$ define k independent random bijective functions $h_1, \ldots, h_k \colon [M] \to [M]$.

In order to invert a function $f \colon [N] \to [M]$, we define the functions $g_1, \ldots, g_k \colon [N] \to [N]$ as $g_i(x) = h_i(f(x)) \bmod N$ for $x \in [N]$ and $i \in [k]$. Let $r_1, \ldots, r_k \in \{0,1\}^\ell$ be independent random binary strings of length ℓ. The algorithm $\mathcal{P}'$ runs $\sigma_i \leftarrow \mathcal{P}(g_i, r_i)$ for each $i \in [k]$, and outputs $\sigma' = (\sigma_1, \ldots, \sigma_k)$. In particular, $S' \leq Sk = O(S \log n)$.

The algorithm $\mathcal{A}'$, given a query $y \in [M]$ proceeds as follows. For $i \in [k]$, it computes $x^{(i)} \leftarrow \mathcal{A}^{g_i}(\sigma_i, h_i(y) \bmod N, r_i)$. Then $\mathcal{A}'$ makes k additional queries to f, and outputs $x^{(i)}$ if $f(x^{(i)}) = y$ for an $i \in [k]$. If for all $i \in [k]$, $f(x^{(i)}) \neq y$, then $\mathcal{A}'$ outputs $\perp$. We have that the number of queries made by $\mathcal{A}'$ is $T' \leq Tk + k = O(T \log n)$.

For every a function $f\colon [N] \to [M]$ and $x \in [N], y = f(x)$, let E^i_x be the event that $|\{y' \in f([N])\colon h_i(y') \equiv h_i(y) \bmod N\}| = 1$. Observe that if E^i_x holds, then every preimage of $h_i(y) \bmod N$ under g_i is also a preimage of y under f. Indeed, if E^i_x holds, then every x satisfying $h_i(y) \equiv g_i(x) \equiv h_i(f(x)) \bmod N$ must also satisfy $y = f(x)$. Since $h\colon [M] \to [M]$ is bijective, for every f and x,

$$\Pr_{h_1}[E^1_x] \geq \binom{M - M/N}{N-1} \Big/ \binom{M-1}{N-1} \geq \left(\frac{M - M/N - (N-2)}{M-(N-1)}\right)^{N-1} \geq \left(1 - \frac{1}{N}\right)^{N-1} \geq 1/e\,.$$

From independence of $h_1, \ldots, h_k$, for every $x \in [N]$ and every $f\colon [N] \to [M]$ we have that

$$\Pr_{h_1,\ldots,h_k}[\forall i \in [k]\colon \neg E^i_x] = \Pr_{h_1}[\neg E^1_x]^k \leq (1 - 1/e)^k \leq 1/(10n^2)\,.$$

The success probability of the algorithm $(\mathcal{P}', \mathcal{A}')$ is at least

$$\begin{aligned}
&\Pr_{\substack{r\sim\{0,1\}^l \\ f:[N]\to[M] \\ x\sim[N];\, y=f(x)}} [(\mathcal{A}')^f(\mathcal{P}(f,r), f(x), r) \in f^{-1}(y)] \\
&\geq \Pr_{\substack{r\sim\{0,1\}^l \\ f:[N]\to[M] \\ x\sim[N]}} [\exists i \in [k]\colon E^i_x \text{ and } (\mathcal{A})^{g_i}(\mathcal{P}(g_i,r), h_i(x) \bmod N, r) \in g_i^{-1}(h_i(x) \bmod N)] \\
&\geq 1 - 1/(10n^2) - \Pr_{\substack{r\sim\{0,1\}^l \\ f:[N]\to[M] \\ h:[M]\to[M] \text{ bij.} \\ g=(\bmod N)\circ h\circ f \\ x\sim[N];\, y=g(x)}} [\mathcal{A}^g(\mathcal{P}(g,r), g(x), r) \notin g^{-1}(y)] \\
&\geq 1 - 1/(10n^2) - \Pr_{\substack{r\sim\{0,1\}^l \\ g:[N]\to[N] \\ x\sim[N];\, y=g(x)}} [\mathcal{A}^g(\mathcal{P}(g,r), g(x), r) \notin g^{-1}(y)] \\
&\geq 1 - 1/(10n^2) - (1 - \delta) = \delta - 1/(10n^2)\,,
\end{aligned}$$

where the last inequality follows from the success guarantee of the algorithm $(\mathcal{P}, \mathcal{A})$. □

B Search-to-decision reduction for injective functions

We observe that the reduction of Theorem 5.2 that preserves the domain and the range of the functions cannot be extended to the case of injective function inversion. Indeed, if $M = N$, then the injective DFI problem is trivial (the output to every query is one), while the injective SFI problem is the permutation inversion problem that is known to require $ST \gtrsim N$ [Yao90]. In the following lemma we provide a search-to-decision reduction for injective function inversion that increases the range by a factor of two. The main difference between the proofs of Lemma 5.1 and Lemma B.1 is in the definitions of functions g^b_i. We define injective SFI and DFI problems similarly to Definition 2.2 and Definition 2.3 with the only difference being that the algorithms are guaranteed to work only for injective functions f.

We include the proof of Lemma B.1 for completeness. This proof closely follows the proof of [CK19, Theorem 8] with the following minor differences. [CK19] reduces injective (N, M)-SFI to (N, M)-DFI over an arbitrary function but average-case input y, while we reduce injective (N, M)-SFI to injective $(N, M+N)$-DFI. It is easy to see that the functions that [CK19] constructs in the proof are 2-to-1, so a straightforward modification of their reduction results in injective functions (but with a larger range of size $M+N$).

Lemma B.1. *Let $\varepsilon := \varepsilon(N) \in (0, 1/2]$. Suppose there exists an algorithm $(\mathcal{P}, \mathcal{A})$ that solves injective $(N, M+N)$-DFI with advantage ε using S bits of preprocessing and T queries. Then there exists an algorithm $(\mathcal{P}', \mathcal{A}')$ that solves injective (N, M)-SFI with success probability $1 - 1/(10n^2)$ using $S' \le O(Sn(\log n)/\varepsilon^2)$ bits of preprocessing and $T' \le O(Tn(\log n)/\varepsilon^2)$ queries.*

Proof. By running $\mathcal{A}$ and $\mathcal{P}$ $k = O((\log n)/\varepsilon^2)$ times with fresh randomness and taking the majority of the query answers, we may assume that $\mathcal{A}$ and $\mathcal{P}$ have failure probability at most $1/(20n^3)$ (by the Chernoff bound in Lemma 2.9). Therefore, it suffices to show an algorithm $(\mathcal{P}', \mathcal{A}')$ that uses $O(Tn)$ queries and $O(Sn)$ bits of preprocessed advice for the special case when $\varepsilon = 1/2 - 1/(20n^3)$.

For inverting an injective function $f\colon [N] \to [M]$, we define the following $2n$ injective functions from $[N]$ to $[M+N]$. For $i \in [n]$, let

$$g_i^0(x) = \begin{cases} f(x) & \text{if } x_i = 0\ , \\ x + M & \text{otherwise}\ . \end{cases} \qquad g_i^1(x) = \begin{cases} f(x) & \text{if } x_i = 1\ , \\ x + M & \text{otherwise}\ . \end{cases}$$

Notice that an oracle query to g_i^b can be trivially simulated by making at most one oracle query to f.

For each $i \in [n]$ and $b \in \{0,1\}$, $r_i^b \sim \{0,1\}^\ell$, $\mathcal{P}'$ runs $\sigma_i^b \leftarrow \mathcal{P}(g_i^b, r_i^b)$, and outputs $\sigma' = (\sigma_1^0, \sigma_1^1 \ldots, \sigma_n^0, \sigma_n^1)$. In particular, $S' \le O(Sn)$.

Given a query $y \in [M]$, for each $i \in [n]$ and $b \in [k]$, $\mathcal{A}'$ computes $d_i^b = \mathcal{A}^{g_i^b}(\sigma_i^b, y, r_i^b)$, which is done using at most $O(Tn)$ queries to f. Informally, $d_i^0 = 1$ if there exists a preimage of y with the ith bit 0, and $d_i^1 = 1$ if there exists a preimage of y with the ith bit 1. If for some $i \in [n]$, $d_i^0 = d_i^1$, then $\mathcal{A}'$ outputs $\perp$. Otherwise, for each i, $\mathcal{A}'$ sets $x_i' = b$ where $b \in \{0,1\}$ is such that $d_i^b = 1$ and $d_i^{1-b} = 0$. Finally, the algorithm $\mathcal{A}'$ outputs $x' = (x_1', \ldots, x_n') \in [N]$. In particular, the number of queries is $T' \le O(Tn)$.

All $2n$ queries to $\mathcal{A}$ are answered correctly with probability at least $1 - 2n/(20n^3) \ge 1 - 1/(10n^2)$. If $y \in [M]$ has a (unique) preimage x under f, we have that $\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f, r), y, r) : x' = x] \ge 1 - 1/(10n^2)$. If $y \in [M]$ doesn't have a preimage under f, then $\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f, r), y, r) : x' = \perp] \ge 1 - \Pr_{r \sim \{0,1\}^{\ell'}}[d_1^0 = 1 \text{ or } d_1^1 = 1] \ge 1 - 1/(10n^3)$. The result follows. □

C Balls and bins: Proof of Lemma 4.3

We will need the following lemma, which is a special case of [Mit96, Corollary 2.12]. It relates the case when a fixed number M of balls are thrown into N bins to the case when the number of balls in each bin is independently chosen from a Poisson distribution with mean M/N.

Lemma C.1. *For any positive integers M, N, k with $1 \le k \le M$, if M balls are thrown independently and uniformly at random into N bins, then the probability that there is no bin that has*

exactly k balls in it is at most $4(1-p)^N \leq 4e^{-pN}$, where

$$p := e^{-M/N} \cdot \frac{(M/N)^k}{k!}$$

is the probability that a Poisson distribution with mean M/N takes the value k.

Our proof of Lemma 4.3 is a minor adaptation of the proof of [Mit96, Lemma 2.13]. The only difference between our Lemma 4.3 and [Mit96, Lemma 2.13] is that our lemma makes the constant explicit.

Proof of Lemma 4.3. Let $k := \lfloor \log N/(3\log(N/M)) \rfloor$. Notice that the statement is trivial if $k \leq 1$ (since there is always a bin with at least one ball in it), so we may assume that $k \geq 2$. By Lemma C.1, it suffices to prove that $4e^{-pN} \leq 1/10$, where

$$p := e^{-M/N} \cdot \frac{(M/N)^k}{k!} \geq (M/(kN))^k \,,$$

where we have used the assumption that $M \leq N/\log N$ and that $N \geq 300$, which in particular implies that $e^{-M/N} > 1/2 \geq k!/k^k$ for $k \geq 2$. Furthermore, by the same assumption, we have that

$$M/(kN) > M/(N\log N) \geq (M/N)^2 \,.$$

Therefore, we have

$$pN > (M/N)^{2k} \cdot N \geq (M/N)^{(2/3)\cdot \log N/\log(N/M)} \cdot N = N^{1/3} \,.$$

It follows that $4e^{-pN} \leq 4e^{-N^{1/3}} \leq 1/10$, since $N \geq 300$. □

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il