# Revisiting Time-Space Tradeoffs for Function Inversion

Alexander Golovnev[1], Siyao Guo[2], Spencer Peters[3], and Noah Stephens-Davidowitz[3]

[1] Georgetown University, Georgetown, USA
alexgolovnev@gmail.com
[2] NYU Shanghai, Shanghai, China
sg191@nyu.edu
[3] Cornell University, Ithaca, USA
sp2473@cornell.edu, noahsd@gmail.com

**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 observe that there is a very simple *non-adaptive* algorithm (i.e., an algorithm whose $i$th query $x_i$ is chosen based entirely on $\sigma$ and $y$, and not on the $f(x_1), \ldots, f(x_{i-1})$) that improves slightly on the trivial algorithm. It works for any $T$ and $S$ satisfying $S = \Theta(N \log(N/T))$, for example, $T = N/\operatorname{poly}\log(N)$, $S = \Theta(N/\log\log N)$. This answers a question due to Corrigan-Gibbs and Kogan [TCC, 2019], who asked whether non-trivial non-adaptive algorithms exist; namely, algorithms that 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. Some of these equivalence results are deferred to the full version [ECCC, 2022].

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.

# Table of 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 $\sigma$, 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 [6–10, 12, 13, 17, 27, 29, 30] (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 [8, 11, 14, 30]. Indeed, many variants of the problem have been studied. For example, we might ask for algorithms that invert arbitrary functions $f$ [12], random functions $f$ [17] (in which case the algorithm should work with reasonable probability over the function $f$), or special classes of functions $f$, like permutations [30]; or one might place restrictions on the algorithm by, e.g., requiring the oracle queries to be non-adaptive [3, 8] or requiring that the algorithm otherwise has some special structure [2]. Other work has considered stronger models of computation, such as quantum algorithms [4, 5, 22].

In his celebrated 1980 work, Hellman [17] 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 [12] 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 [2]. However, the best known lower bound [9, 13, 30] 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, Hellman proved it is tight in the special case when the function $f$ is a permutation [17].)

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 [8]. (See also [11], 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, Corrigan-Gibbs and Kogan [8] 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 $\sigma$, 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 [3]; 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 $y$ in its image 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 [17]).[4] 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 [12].[5] 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

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

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 lower bound against guess-and-check (non-adaptive) algorithms.** We next address Corrigan-Gibbs and Kogan's question about whether non-trivial non-adaptive algorithms are possible. Corrigan-Gibbs and Kogan naturally guessed the answer was negative. But, surprisingly, we observe that there is a very simple algorithm that (slightly) outperforms the trivial algorithm.[6] Recall that the trivial algorithm simply stores inverses for as many range elements as it can, and achieves parameters $S/\log N + T = N$.

The simple algorithm, by contrast, stores only *part of* an inverse for each range element. Specifically, for each $y \in [N]$ having at least one inverse, the preprocessing algorithm stores the first $\log(N) - \log(T) = \log(N/T)$ bits of an inverse $x_y$. On challenge $y$, the online algorithm queries all $T = 2^{\log T}$ elements whose first $\log(N/T)$ bits match the stored prefix of $x_y$. One of these queries will discover that $f(x_y) = y$. This simple algorithm evidently achieves the tradeoff

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

For example, setting $T = N/\log^C(N)$ for any constant $C > 0$, the simple algorithm uses $S = O(N \log\log N)$ bits of advice, beating the trivial algorithm by a polylogarithmic factor in both time and space.

[4] 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.

[5] 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.

[6] In fact, we also missed this algorithm. An earlier version of this paper described a much more complicated algorithm that achieves the same parameters. We are very grateful to the anonymous CRYPTO reviewer who reviewed that version and discovered the simple algorithm.

The simple algorithm is very straightforward by any standard, and in particular, it always outputs one of the points $x_i$ that it queries. We call non-adaptive algorithms with this property *guess-and-check* algorithms, since such an algorithm can be viewed as making $T$ guesses $x_1, \ldots, x_T$ up front, and then using its queries to check whether any of its guesses is in fact a inverse of $y$.

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 $x_i$ of $y$ (i.e., one "guess"), and the algorithm succeeds if and only if any of these processors discovers that $x_i$ is in fact a preimage of $y$. Indeed, Corrigan-Gibbs and Kogan [8] introduced non-adaptive algorithms in part because of their relationship with parallelism. (Other special classes of non-adaptive algorithms were studied in [8] and [3], but none of the previously defined classes captures guess-and-check algorithms, as we explain in Section 1.3.)

Our second 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 [8].

**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 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 $\sigma$ 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 the full version [15], 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 4 and 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 1 and 2.)

These reductions can be viewed as variants of a reduction presented by Corrigan-Gibbs and Kogan in [8] (as we discuss in Sections 1.2 and 1.3). In

the full version [15], we show another search-to-decision reduction for injective functions, which is a more direct adaptation of the reduction in [8].

**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 [8] allowed their query and preprocessing algorithms access to the same random oracle. In contrast, Fiat and Naor [12] 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.[7]) 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 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

[7] 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 $r_i$.

complexity measures that we ignore in this work, like the size of the description of the (nonuniform) preprocessing algorithm. 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 [23].

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. If this is done, our algorithm becomes uniform, while retaining the desirable property from Fiat and Naor's algorithm that preprocessing only requires $\tilde{O}(N)$ time. Thus our improvement over Fiat and Naor's algorithm in the low-space regime $S < T$ also applies in this setting.

### 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 $f$ having many inverses—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 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 [9] and [10]. 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 $\sigma$ (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 $\sigma$, 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 [8] 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 $i$th 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 $i$th 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."[8] 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 [28]. 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 $\log N$ total functions $f_1, \ldots, f_{\log N}$), which would cause our search-to-decision reduction to always fail to find the $i$th 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 $f'_i$ correspond-

[8] 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.

ing 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 $i$th 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 [9] 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 $\varepsilon$. (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 $\varepsilon$ 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 [3] 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 [8]. 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 $f$. [8] 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 [11] 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 [3] but it requires an unproven conjecture and is quantitatively weaker than ours and those in of [3]. (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 [6–10, 16] 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 their reduction from [8] 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 the full version [15] ) 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 of Fiat and Naor, and of De, Trevisan, and Tulsiani [9, 12] 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 [9] studied the dependence of $S$ and $T$ in terms of the fraction $\varepsilon$ of inputs $x \in [N]$ for which the algorithm successfully inverts $f(x)$. They showed that for small $\varepsilon$ 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; we believe that they can be, but we leave this to future work.

## 2 Preliminaries

We define $\mathbb{1}_\mu$ as $\mathbb{1}_\mu = 1$ if $\mu$ 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. For a set $X \subseteq [N]$, $f(X)$ denotes the image of $X$ under $f$, and for $y \in [N]$, $f^{-1}(y)$ denotes the preimage of $y$ under $f$. The first few definitions are core to our study of function inversion.

**Definition 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.** *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 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 4.** *We say that* $(\mathcal{P}, \mathcal{A})$ solves average-case $(N, M)$-SFI with success probability $\delta$ *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 5.** *We say that* $(\mathcal{P}, \mathcal{A})$ solves average-case $(N, M)$-DFI with advantage $\varepsilon$ *if*

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

*and*

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

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

**Definition 6.** *We say that* $(\mathcal{P}, \mathcal{A})$ solves $(N, M)$-SFI with success probability $\delta$ without shared randomness *if for all* $f : [N] \to [M]$ *and all* $y \in f([N])$,

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

*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, 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 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* $\sigma, y,$ *and* $r$ *(i.e., not depending on the results of previous queries).*

**Definition 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., [20]).

**Lemma 1.** *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.

**Lemma 2.** *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

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

For the last inequality, it suffices to show that $1 - x \geq e^{-x-x^2}$ for $0 \leq x \leq 1/2$. Indeed, plugging in $x = 1/M$ gives $1 - 1/M \geq e^{-1/M-1/M^2}$, which implies $(1-1/M)^{N-1} > (1-1/M)^N \geq e^{-N/M-N/M^2}$. To prove this, let $f(x) = 1 - x, g(x) = e^{-x-x^2}$, and $h(x) = f(x)/g(x)$. Computing $\frac{d}{dx}\log(h(x)) = \frac{d}{dx}(\log(1-x) - (-x-x^2)) = -1/(1-x) + 1 + 2x = x(1-2x)/(1-x)$, we see that it is nonnegative on $[0, 1/2]$. Since the logarithm is increasing, it follows that $\frac{d}{dx}h(x)$ is also nonnegative on $[0, 1/2]$, and so $h(x) \geq h(0) = 1$ on $[0, 1/2]$. But this implies $f(x) \geq g(x)$ for all $0 \leq x \leq 1/2$, which is what we wanted to prove. □

### 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). . The notation $m_{n,\varepsilon} \leq O_\varepsilon(n)$ means that there exists a function $f(\varepsilon)$ such that $m_{n,\varepsilon} \leq f(\varepsilon)O(n)$.

**Theorem 1 ([1, 18, 19, 25]).** *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 $i$th 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 1 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 [12]. 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 [17]) 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 1 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 2.** *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 1, 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.

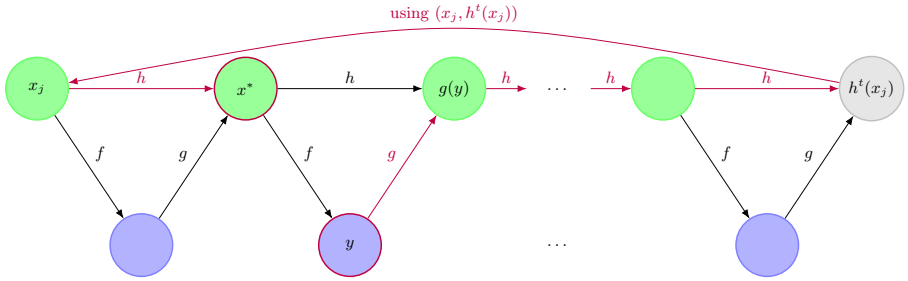


**Fig. 1.** The picture captures the basic workings of chain-based algorithms, including Hellman's algorithm, Fiat and Naor's algorithm, and our improvement. Here $h = g \circ f$, where $g$ is randomly sampled from some appropriate distribution. Preprocessing constructs the green chain $C(x_j)$ by sampling 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.

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 $\sigma := (\mathrm{st}_1, \ldots, \mathrm{st}_\ell)$.

On input a challenge $y$ and preprocessed advice $\sigma$, the online algorithm $\mathcal{A}$ first recovers $L$ as follows. If $S \geq T$, $\mathcal{A}$ just reads $L$ from $\sigma$. 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 $\perp$.

**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 $f(g'(y,i)) \notin \widehat{L}$. [9]

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.

**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 $h^t(x_i) \in C_y$.[10] 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.** *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. □

[9] 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 [12])—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 $f$.)

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

We claim that the subalgorithm satisfies the following guarantee:

**Theorem 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 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, it is straightforward to show Theorem 2.

*Proof of Theorem 2 assuming Theorem 3.*
Lemma 3 states that all $x \in D$ satisfy $|f^{-1}(f(x))| \leq U$ except with probability $2/N$ over the random choices of $z_1, \ldots, z_\alpha$. Assuming this holds, Theorem 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 1, this implies the result. □

It remains to prove Theorem 3.

*Proof of Theorem 3.* The particular statement easily follows from the general statement. Indeed,

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

And for sufficiently large $N$, it follows that

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

We now prove the general statement of Theorem 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 $j \neq i$, $h^t(x_j) \notin C_y$. 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 $j \neq 1$, 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

**Lemma 4.**

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

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 $1 \leq i \leq t-1$, 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 $1 \leq i \leq t-1$, $A_i$ 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 $1 \leq i \leq t$, 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 $1 \leq i \leq t$, 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,

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

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

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

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

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

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

**Lemma 5.**

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

It suffices to prove this claim. Indeed, combining it with Lemma 4 gives

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

Next we prove Lemma 5. By union bound and symmetry,

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

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 Lemma 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

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

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.

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

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 $0 \le j \le t$, the $j$-th term satisfies the following bound:

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

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 $j \ge 1$, 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 $\mathsf{COND}_2$), we know that $v_j := f(h^{j-1}(x_2))$ is distinct from the values $f(h^k(x_2))$ for $0 \le k < j-1$. By $\mathsf{COND}_{1,j}$ and $\mathsf{COND}_2$, $v_j$ is also distinct from the values $f(h^i(x_1))$ for $0 \le i \le 2t$. In other words, $v_j$ is not in the set $V_j$ defined by

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

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

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

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

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

(The last line only holds if $t > 0$, but otherwise Lemma 5 is trivial.) Combining this with Eq. (6) concludes the proof of Lemma 5 and hence the proof of Theorem 3. □

## 4 A lower bound against guess-and-check non-adaptive algorithms

In this section, we prove our lower bound against guess-and-check non-adaptive algorithms. The precise statement is as follows.

**Theorem 4.** *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* $S \geq (N/2)\log(N/6T) - 4$.

Following De et al. [9] and Dodis et al. [10], 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 6.** *([9, 10]) 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* $\delta$ *if for all* $f \in \mathcal{F}$*,*

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

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

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

Hence,

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

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

*Proof.* 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 $i$th oracle query is $y$. This requires $\lceil N' \cdot \log T \rceil$ bits in total.
3. Store the mapping from $[N]\backslash f^{-1}(G')$ to $[N]\backslash G'$ 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 the full version [15].[11] Also, [8, Lemma 21] proves that

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

for worst-case functions and $M > N$, inverting $f\colon [N] \to [M]$ is as hard as inverting $f'\colon [N] \to [N]$. (Of course, for $M < N$, inverting worst-case functions $f\colon [N] \to [M]$ trivially reduces to inverting worst-case functions $f'\colon [N] \to [N]$.) In the full version [15], 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 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 1).

First, in Lemma 8 we observe that, given an algorithm for DFI, one can solve SFI on all inputs $y$ that have unique preimages. Then, in Theorem 5 we use the Isolation Lemma [21, 26, 28] to reduce the general case of SFI to the case where $y$ has a unique preimage.

**Lemma 8.** *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 $\varepsilon$ 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 $y \in [M]$,*

$$\Pr_{r \sim \{0,1\}^{\ell'}}[x' \leftarrow (\mathcal{A}')^f(\mathcal{P}'(f, r), y, r) \ : \ x' \neq \perp \text{ and } f(x') \neq y] \le 1/(10n^2) \ .$$[12]

For space reasons, we defer the proofs of Lemma 8 and the other results in this section to the full version [15]. We can now state the main result of this section. The main difference in the statements of Lemma 8 and Theorem 5 is that the SFI algorithm in Lemma 8 is only guaranteed to succeed on queries that have a unique preimage, while the SFI algorithm in Theorem 5 works for all queries.

[12] 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.

**Theorem 5.** *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 $\varepsilon$ 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, $S'' \le O(Sn^2(\log n)/\varepsilon^2)$ bits of preprocessing, and $T'' \le O(Tn^2(\log n)/\varepsilon^2)$ queries.*

*Remark 1.* A few extensions of Theorem 5 are in order.

1. This 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). See the full version [15].
2. In the proof of Lemma 8 in the full version [15], 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 6). This modification will also improve the parameters $S''$ and $T''$ in Theorem 5 by a $\log n$ factor (though unfortunately it does *not* preserve non-adaptivity).

### 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 4 for the formal definition of average-case SFI and Definition 5 for the formal definition of average-case DFI.) The proof of Theorem 5 does not work for the case of average-case functions as Lemma 8 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 6.

**Theorem 6.** *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.*

*Remark 2.*

1. Similarly to the reduction in Theorem 5, the search-to-decision reduction of Theorem 6 is non-adaptive.
2. A drawback of Theorem 6 is that it requires the DFI algorithm to have very large advantage $\varepsilon$. 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). We could have worked directly with this assumption on the DFI algorithm, but we prefer the simpler (but strictly stronger) assumption in Theorem 6.

## 6 Removing shared randomness

In this section, we adapt to our setting Newman's technique for converting public-coin protocols to private-coin protocols [23] 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 9.** *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 9.** *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 [24]. 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 1), 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. □

It's worth noting that while the proof uses the probabilistic method (and so is nonconstructive), it is essentially constructive in the sense that choosing the required strings at random works with very high probability. (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 1.** *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 9 applies.

So it suffices to observe that

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

□

**Acknowledgements** Siyao Guo was 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. Spencer Peters and Noah Stephens-Davidowitz were supported in part by the NSF under Grant No. CCF-2122230. We are indebted to all reviewers of this paper, but we would like to acknowledge specifically the anonymous CRYPTO reviewer who pointed out the existence of the very simple non-adaptive algorithm.

# Bibliography


[1] Alon, N., Bruck, J., Naor, J., Naor, M., Roth, R.M.: Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. IEEE Transactions on Information Theory **38**(2), 509–516 (1992) 13

[2] Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: CRYPTO (2006) 1, 2

[3] Chawin, D., Haitner, I., Mazor, N.: Lower bounds on the time/memory tradeoff of function inversion. In: TCC (2020) 1, 2, 4, 9

[4] Chung, K.M., Guo, S., Liu, Q., Qian, L.: Tight quantum time-space tradeoffs for function inversion. In: FOCS (2020) 1

[5] Chung, K.M., Liao, T.N., Qian, L.: Lower bounds for function inversion with quantum advice. In: ITC (2020) 1

[6] Coretti, S., Dodis, Y., Guo, S.: Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In: CRYPTO (2018) 1, 10

[7] Coretti, S., Dodis, Y., Guo, S., Steinberger, J.: Random oracles and non-uniformity. In: Eurocrypt (2018)

[8] Corrigan-Gibbs, H., Kogan, D.: The function-inversion problem: Barriers and opportunities. In: TCC (2019) 1, 2, 4, 5, 8, 9, 10, 22

[9] De, A., Trevisan, L., Tulsiani, M.: Time space tradeoffs for attacks against one-way functions and PRGs. In: CRYPTO (2010) 2, 7, 9, 10, 21

[10] Dodis, Y., Guo, S., Katz, J.: Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In: EUROCRYPT (2017) 1, 7, 10, 21

[11] Dvořák, P., Koucký, M., Král, K., Slívová, V.: Data structures lower bounds and popular conjectures. In: ESA (2021) 1, 2, 9

[12] Fiat, A., Naor, M.: Rigorous time/space tradeoffs for inverting functions. In: STOC (1991) 1, 3, 5, 10, 13, 16

[13] Gennaro, R., Trevisan, L.: Lower bounds on the efficiency of generic cryptographic constructions. In: FOCS (2000) 1, 2

[14] Golovnev, A., Guo, S., Horel, T., Park, S., Vaikuntanathan, V.: Data structures meet cryptography: 3SUM with preprocessing. In: STOC (2020) 1

[15] Golovnev, A., Guo, S., Peters, S., Stephens-Davidowitz, N.: Revisiting time-space tradeoffs for function inversion (2022), https://eccc.weizmann.ac.il/report/2022/145/ 4, 5, 10, 22, 23, 24

[16] Gravin, N., Guo, S., Kwok, T.C., Lu, P.: Concentration bounds for almost $k$-wise independence with applications to non-uniform security. In: SODA (2021) 10

[17] Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory **26**(4), 401–406 (1980) 1, 2, 3, 14

[18] Justesen, J.: Class of constructive asymptotically good algebraic codes. IEEE Transactions on Information Theory **18**(5), 652–656 (1972) 13

[19] MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. Elsevier (1977) 13
[20] Mitzenmacher, M., Upfal, E.: Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge University Press (2017) 12
[21] Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. In: STOC (1987) 23
[22] Nayebi, A., Aaronson, S., Belovs, A., Trevisan, L.: Quantum lower bound for inverting a permutation with advice. Quantum Information & Computation **15**(11-12), 901–913 (2015) 1
[23] Newman, I.: Private vs. common random bits in communication complexity. Information processing letters **39**(2), 67–71 (1991) 6, 25
[24] Rao, A., Yehudayoff, A.: Communication Complexity and Applications. Cambridge University Press (2020) 25
[25] Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. In: STOC (1995) 13
[26] Ta-Shma, N.: A simple proof of the isolation lemma (2015), https://eccc.weizmann.ac.il//report/2015/080/ 23
[27] Unruh, D.: Random oracles and auxiliary input. In: CRYPTO (2007) 1
[28] Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solutions. In: STOC (1985) 8, 23
[29] Wee, H.: On obfuscating point functions. In: STOC (2005) 1
[30] Yao, A.C.C.: Coherent functions and program checkers. In: STOC (1990) 1, 2