Time Space Tradeoffs for Attacks against One-Way Functions and PRGs

Anindya De[1,⋆], Luca Trevisan[2,⋆⋆], and Madhur Tulsiani[3,⋆⋆⋆]

[1] University of California at Berkeley
anindya@cs.berkeley.edu
[2] University of California at Berkeley and Stanford University
luca@cs.berkeley.edu
[3] Institute for Advanced Study, Princeton
madhurt@math.ias.edu

Abstract. We study time space tradeoffs in the complexity of attacks against one-way functions and pseudorandom generators.

Fiat and Naor [7] show that for every function $f : [N] \to [N]$, there is an algorithm that inverts f everywhere using (ignoring lower order factors) time, space and advice at most $N^{3/4}$.

We show that an algorithm using time, space and advice at most

$$\max\{\epsilon^{\frac{5}{4}} N^{\frac{3}{4}} \ , \ \sqrt{\epsilon N}\}$$

exists that inverts f on at least an ϵ fraction of inputs. A lower bound of $\tilde{\Omega}(\sqrt{\epsilon N})$ also holds, making our result tight in the "low end" of $\epsilon \leq \sqrt[3]{\frac{1}{N}}$.

(Both the results of Fiat and Naor and ours are formulated as more general trade-offs between the time and the space and advice length of the algorithm. The results quoted above correspond to the interesting special case in which time equals space and advice length.)

We also show that for every length-increasing generator $G : [N] \to [2N]$ there is a algorithm that achieves distinguishing probability ϵ between the output of G and the uniform distribution and that can be implemented in polynomial (in $\log N$) time and with advice and space $O(\epsilon^2 \cdot N \log N)$. We prove a lower bound of $S \cdot T \geq \Omega(\epsilon^2 N)$ where T is the time used by the algorithm and S is the amount of advice. This lower bound applies even when the distinguisher has oracle access to G.

We prove stronger lower bounds in the *common random string* model, for families of one-way permutations and of pseudorandom generators.

Keywords: One-way functions, pseudorandom generators, random permutations, time-space tradeoffs.

⋆ Supported by the "Berkeley fellowship for Graduate Study" and by the BSF under grant 2006060.

⋆⋆ This material is based upon work supported by the National Science Foundation under grant No. CCF-0729137 and by the BSF under grant 2006060.

⋆⋆⋆ This material is based upon work supported by the National Science Foundation under grant No. CCF-0832797 and IAS Sub-contract no. 00001583. Work done partly when the author was a graduate student at UC Berkeley.

T. Rabin (Ed.): CRYPTO 2010, LNCS 6223, pp. 649–665, 2010.

1 Introduction

In the applied cryptography literature, a cryptographic primitive with a key of length k is typically considered "broken" if the key can be recovered in time less than 2^k, that is, faster than via an exhaustive brute force search. Implicit in this attitude is the belief in the existence of primitives for which a brute force attack is optimal. A time t brute force attack against a one-way function $f : \{0,1\}^n \to \{0,1\}^n$, consisting in trying about t random guesses for the inverse, only succeeds with probability about $t/2^n$, and a brute force attack that attempts to distinguish a length increasing generator $G : \{0,1\}^{n-1} \to \{0,1\}^n$ from the uniform distribution by attempting to guess the seed achieves distinguishing probability about $t/2^n$. Is it plausible that such trade-offs are optimal? Would it be plausible to assume that AES with 128 key bit cannot be distinguished from a random permutation with distinguishing probability more than 2^{-40} by adversaries running in time 2^{60}?[1]

If we apply a non-uniform measure of complexity, that is, if we restrict ourselves to a fixed finite one-way function or pseudorandom generator, and allow our adversary to use precomputed information as advice, then it turns out that the above "brute force" bounds can always be improved upon.

In 1980, Hellman [12] proved that for every one-way *permutation* $f : [N] \to [N]$ (for this discussion, it will be convenient to set $N = 2^n$ and identify $\{0,1\}^n$ with $[N]$) and for every parameters S, T satisfying $S \cdot T \geq N$, there is a data structure of size $\tilde{O}(S)$ and an algorithm that, with the help of the data structure, given $f(x)$ is always able to find x in time $\tilde{O}(T)$. The notation $\tilde{O}(\cdot)$ hides lower order factors that are polynomial in $\log N$; we will ignore such factors from now on in the interest of readability. We shall refer to S, the size of the pre-computed data structure used by the algorithm, as the *space* used by the algorithm.

In particular, every one-way permutation can be inverted in time $\sqrt{N}$ using $\sqrt{N}$ bits of advice.[2]

Hellman's algorithm only requires oracle access to the permutation. Yao [18] proves that, in this oracle setting, Hellman's trade-off is tight for random permutations. (See also [10].)

[1] The answer to the last question is no. It follows from our results that there is a distinguisher that makes two queries, then performs a computation realizable as a circuit of size 2^{56}, assuming a complete basis of fan-in two gates, and achieves distinguishing probability $\geq 2^{-40}$ between AES_{128} and a random permutation $\{0,1\}^{128} \to \{0,1\}^{128}$. Otherwise, after the two oracle queries, the distinguisher can be implemented in a 64-bit architecture with two table look-ups and three unit-cost RAM operations, given access to a precomputed table of 2^{49} entries.

[2] This doesn't mean that there is a circuit of size $\tilde{O}(\sqrt{N})$; the running time of $\tilde{O}(\sqrt{N})$ is in the RAM model. The relationship between non-uniform time/space complexity measures and circuit complexity is the following: a circuit of size C can be simulated using time at most $\tilde{O}(C)$ given a pre-computed data structure of size $\tilde{O}(C)$; and an algorithm that uses time T and a pre-computed data structure of size S can be simulated by a circuit of size $\tilde{O}((S+T)^2)$.

Hellman also considers the problem of inverting a random function $f : [N] \to [N]$ given oracle access to f. He provides a heuristic argument suggesting that for every S, T satisfying $TS^2 \geq N^2$, and with high probability over the choice or a random function $f : [N] \to [N]$, there is a data structure of size S and an algorithm of complexity T that inverts f everywhere using the data structure and given oracle access f. This trade-off yields the interesting special case $S = T = N^{2/3}$.

Fiat and Naor [7] prove Hellman's result rigorously, and are able to handle arbitrary functions, not just random functions. If the given function $f : [N] \to [N]$ has collision probability[3] λ, then the algorithm of Fiat and Naor requires the trade-off $TS^2 \geq \lambda \cdot N^3$. Note that with high probability a random function has collision probability about $1/N$ (recall that we ignore $(\log N)^{O(1)}$ terms), and so one recovers Hellman's tradeoff. For general functions, Fiat and Naor are able to prove the trade-off $TS^3 \geq N^3$, which has the special case $S = T = N^{3/4}$.

Barkan, Biham, and Shamir [4] prove that the $TS^2 = N^2$ trade-off of Fiat and Naor for random functions is optimal under certain assumptions on what is stored in the data structure and on the behavior of the algorithm.

The result of Fiat and Naor can also be applied to the task of distinguishing a given pseudorandom generator from the uniform distribution (and hence a given pseudorandom permutation from a random permutation or a given pseudorandom function from a random function) by recovering the seed. We are not aware of previous work that focused specifically on the complexity of distinguishers for pseudorandom generators. Two related results, however, should be mentioned. It has been known for a long time (going back to, as far we know, [2]) that every distribution that has constant statistical distance from the uniform distribution, and, in particular, the output of any length increasing generator, can be distinguished from the uniform distribution over n bits using a parity function (of linear circuit complexity), and with distinguishing probability $\Omega(2^{-\frac{n}{2}})$. The other result is due to Andreev, Clementi and Rolim [3], who prove that for every boolean predicate $P : \{0,1\}^n \to \{0,1\}$ and every ϵ there is a circuit of size $O(\epsilon^2 2^n)$ that computes P on at least a $1/2 + \epsilon$ fraction of inputs. This implies that for every pseudorandom generator of the form $x \to f(x)P(x)$, where f is a permutation and P is a hard-core predicate for f, and every $\epsilon > 0$, there is a circuit of size $O(\epsilon^2 2^n)$ that achieves distinguishing probability ϵ.

Our Results

Upper Bounds for Inverting One-Way Functions. We introduce a new way to analyze the Fiat-Naor construction. Instead of being limited by the collision probability, it is limited by the "irregularity" of the function. In particular, if f is a regular function, then our bound is as good as that for a random function. While this approach yields no improvement for the worst-case complexity of inverting a function everywhere, it improves the complexity if we only seek

[3] Here by the collision probability of a function we mean the probability that after sampling two independent random inputs x, y we have $f(x) = f(y)$.

to invert on an ϵ fraction of the inputs. In particular, we show that there is an algorithm such that for every $f : [N] \to [N]$ and every ϵ, the algorithm inverts f on an ϵ fraction of inputs and its time complexity, space complexity and advice length are bounded by

$$\tilde{O}\left(\max\left\{\sqrt{\epsilon N}\ , \epsilon^{\frac{5}{4}} N^{\frac{3}{4}}\right\}\right)$$

Here the $\tilde{O}$ hides factors of $2^{\text{poly}\log\log}$. It follows from known results, and we present a proof in the full version, that, in an oracle setting, it is not possible to do better than $\Omega(\sqrt{\epsilon N})$, so our result is best possible when $\epsilon < N^{-1/3}$.

Indeed, we establish the following more general trade-off: for every $T < 1/\epsilon$ and S satisfying the trade-off $ST = \epsilon N$ we can construct an algorithm that has time T and uses a data structure of size S (up to lower order factors); for every $T > 1/\epsilon$, we can use time T and space S provided $TS^3 = \epsilon^5 N^3$. As we discuss below, a straight-forward application of the analysis of Fiat and Naor would have given a trade-off $TS^3 = \epsilon^3 N^3$, or a time and space complexity $\tilde{O}(\epsilon^{\frac{3}{4}} N^{\frac{3}{4}})$ in the $T = S$ case. For comparison, when $\epsilon = N^{-1/3}$, we can achieve (optimal) time and space $N^{1/3}$; the straight-forward use of the Fiat-Naor analysis would have given time and space $\sqrt{N}$. Given an upper bound λ on the collision probability, we can achieve the optimal trade-off $TS = \epsilon N$ if $S \geq \epsilon^2 N^2 \lambda$, and the trade-off $TS = \epsilon^2 N^2 \lambda$ otherwise. For example, if we have a function with collision probability close to $1/N$, and we want to achieve inversion probability $\epsilon = N^{-1/4}$, then we can do so, using the latter construction, employing time, space and advice at most $N^{5/12} = N^{.416\cdots}$; using our generic construction (which applies to functions of arbitrary collision probability) would have given a complexity of $N^{7/16} = N^{.4375}$. We note however that all our tradeoffs (as well as the previous ones that we state) apply only for $S = \tilde{\Omega}(\sqrt{\epsilon N})$. The difference between our analysis and the one in [7] is explained in Section 2.

Upper Bounds for Breaking pseudorandom Generators. We also give non-uniform attacks to distinguish between distributions with significant statistical difference. For the sake of simplicity, in this version, we only consider the case of distinguishing the output of a pseudorandom generator from the uniform distribution. Given an arbitrary length-increasing generator $G : \{0,1\}^n \to \{0,1\}^m$, $m > n$, we show that, for every ϵ, there is a distinguisher that runs in polynomial time, uses a data structure of size $O(\epsilon^2 2^n)$, and achieves distinguishing probability ϵ. The distinguisher can also be implemented as a circuit of size $O(\epsilon^2 2^n)$. Notably, the distinguisher need not have oracle access to G, and so our result applies to generators constructed for applications in derandomization, in which the generator may have complexity $2^{O(n)}$, or even higher. In this setting, in which the complexity of the generator is not bounded, it is easy to see that advice $\Omega(\epsilon^2 2^n)$ is necessary. We also present a simpler construction that achieves the slightly worse circuit size $O(\epsilon^2 n 2^n)$.

Lower Bounds. We prove lower bounds for non-uniform attacks on one-way permutations and pseudorandom generators. Our lower bound for permutations

is proven in the following model : Given any permutation f, the algorithm (call it A) is allowed to store a data structure of size S which can be arbitrarily dependent on f. Further, on any input x, A is allowed to make T queries to f along with any other computation it may perform. We say that there is a lower bound on time space tradeoff of time T and space S for inverting permutations on ϵ fraction of the inputs, if for any such algorithm A, there exists a permutation f such that to invert f on ϵ fraction of the inputs, if A stores a data structure of size S, then it must make T queries on some of its inputs. In this model, we prove that $S \cdot T = \tilde{\Omega}(\epsilon N)$. This in particular implies that the technique to invert a permutation described previously is optimal. While such lower bounds had previously been proven by Yao [18], Gennaro and Trevisan [8] and Wee [17], they were only applicable till $T = O(\sqrt{\epsilon N})$ while our proof shows the lower bound for the full range of T. Also, arguably our proof is simpler than the previous proofs.

Another problem we consider is that of showing lower bounds on time space tradeoffs for attacks on pseudorandom generators. The model is the same as that of permutations except that the algorithm is given access to the stretching function G and it is required to distinguish between the output of the pseudorandom generator from the uniform by at least ϵ. In this model, we get a lower bound of $S \cdot T = \tilde{\Omega}(\epsilon^2 N)$. From the previous discussion, this is tight even when restricted to distinguishers with no oracle access. To the best of our knowledge, this question has not been considered previously. Interestingly, the family G that we use to prove the lower bound is a random permutation $f : [N] \to [N]$ followed by a random predicate $P : [N] \to \{0, 1\}$ i.e. $G(x) = f(x) \circ P(x)$.

Common Random String Model. Finally, we prove time space lower bounds for the problem of inverting a function (or breaking a pseudorandom generator) sampled from a family of functions (or a family of pseudorandom generators). This is the case when a common source of randomness is available to all the parties and this randomness is used to sample the one-way permutation or the pseudorandom generator as the case may be. We prove stronger lower bounds in this model. In particular, we show that if there is an algorithm which inverts any family of permutation $f : [N] \times [K] \to [N]$ (where K denotes the common randomness), then for large K, the brute force attack in the best possible. Similarly, if there is an algorithm which for any family of pseudorandom generators, $G : [N] \times [K] \to [N] \times \{0, 1\}$ distinguishes the output of G from uniform by more than ϵ, then $S \cdot T = \tilde{\Omega}(\epsilon^2 KN)$ provided K is large enough. Here S and T have their usual meanings. We specify the exact trade-offs with K in the next section.

Open Questions

It remains open to either improve the Fiat-Naor construction or to prove a stronger lower bound for the problem of inverting a random function or an arbitrary function everywhere. It is plausible that the optimal trade-off $ST = N$, while achievable for permutations, is impossible to achieve for general functions,

maybe even impossible for random functions. Such a separation between the complexity of dealing with general or random functions versus permutations would be extremely interesting.

If one wants to invert a random permutation or function uniformly (that is, given no advice), then the lower bound $T \geq N$ (ignoring lower-order factors) holds. A quantum computer, however, can achieve $T = \sqrt{N}$ [11], which is optimal [5]. What is the complexity of inverting a random permutation, a random function, or an arbitrary function with a quantum computation that takes advice? It was pointed out to us by Scott Aaronson that extension of the techniques in [1] can prove that any pointer-jumping arguments (as the one in Hellman's scheme) cannot beat the $\sqrt{N}$ bound even with access to quantum advice. Hence, if at all quantum computation can beat the classical $\sqrt{N}$ bound, it will have to use significantly new techniques.

We do not have matching upper and lower bounds for the problem of constructing distinguishers for pseudorandom generators, except in the extremal case $T = O(1)$, $S = \epsilon^2 N$. Is $T = \epsilon^2 N$, $S = O(1)$ achievable? More generally, for what range of parameters is it possible to achieve distinguishability even though inversion of one-way permutations or functions is impossible?

2 Inverting One-Way Functions

How can one invert one-way functions, in general, faster than by brute force?

2.1 An Overview of the Ideas of Hellman and of Fiat and Naor

If we are given a one-way *permutation* $f : [N] \to [N]$, then it is easy to construct an inverter for $f()$ that uses time and space $\tilde{O}(\sqrt{N})$. Suppose for simplicity that $f()$ is a cyclic permutation and that $N = s^2$ is a perfect square: then pick $\sqrt{N}$ "equally spaced" points $x_1, \ldots, x_s$, such that $x_{i+1} = f^{(s)}(x_i)$, and create a data structure to store the pairs (x_i, x_{i+1}). Then given y, we compute $f(y)$, $f(f(y))$, and so on, until, for some j, we reach a point $f^{(j)}(x)$ which is one of the special points in the data structure. Then we can read from the data structure the value $f^{(j-s)}(y)$, and then by repeatedly computing f again we will eventually reach $f^{(-1)}(y)$. Note that this takes $O(s)$ evaluations of f and table look-ups, so both the time and space complexity are approximately $s = \sqrt{N}$. If $f()$ is not cyclic, we do a similar construction for each cycle of length less than s, and if N is not a perfect square we can round s to $\lceil \sqrt{N} \rceil$.

Abstractly, this construction works for the following reason. Consider the graph $G_f = ([N], E)$ that has $[N]$ as set of vertices and that for every x has the directed edge $(x, f(x))$. Then, if f is a permutation, it is possible to cover G_f using $\sqrt{N}$ edge-disjoint paths, each of length $\sqrt{N}$ or, more generally, S edge-disjoint paths of length T, provided $ST \geq N$. Furthermore, if f is a function such that G_f can be covered using S edge-disjoint paths, each of length at most T, then we have an algorithm to invert f using space S and time T.

The problem is that, in general, no good collection of paths may exist. Suppose, for example, that G_f looks like the graph on the left in Figure 1: a directed

path of length $\frac{1}{3}N$ with a length-2 path joining in at each point. Then we see that there is a set S (the vertices of indegree zero in the picture) of size $N/3$ such that no path can contain more than one vertex of S, and so no collection of $o(N)$ paths can cover the entire graph.

Hellman [12] considers the case in which $f()$ is a *random* function. Then, even though it's not clear how many edge-disjoint paths of what length can cover G_f it is not hard to see that one can find $N^{\frac{1}{3}}$ paths of length $N^{\frac{1}{3}}$ having very few "collisions." This gives a construction that uses time and space $N^{\frac{1}{3}}$ and that inverts $f()$ at $N^{\frac{2}{3}}$ points. Hellman then suggests to modify $f()$ by composing it with a fixed permutation of the input bits, and to reason heuristically as if the new function behaved as an independently chosen new random function. Then one can repeat the construction, and have a new algorithm of time and space complexity $N^{\frac{1}{3}}$ that inverts $f()$ at $N^{\frac{2}{3}}$ points, which are assumed to be an independent random subset of size $N^{\frac{2}{3}}$. After iterating this process $N^{\frac{1}{3}}$ times one has $N^{\frac{1}{3}}$ candidate algorithms, each of time and space complexity $N^{\frac{1}{3}}$, such that, for every x, $f(x)$ is inverted by at least one of the algorithms. Overall, one gets an algorithm of complexity $N^{\frac{2}{3}}$ that inverts f everywhere.

Fiat and Naor [7] make Hellman's argument rigorous. The idea of Fiat and Naor is to pick a good random hash function g, and then work with the new function $h(x) := g(f(x))$. (See Figure 1 for an example of the effect of this randomization.) If g were a truly random function, and f where a function such that every output has few pre-images, then one can repeat Hellman's calculation that $N^{\frac{1}{3}}$ nearly disjoint paths of length $N^{\frac{1}{3}}$ exist. Picking $N^{\frac{1}{3}}$ random functions g_i then would give a rigorous version of the full argument, except for the dependency on several random oracles. For a more general trade-off, it is possible to pick m nearly disjoint paths of length t provided that $m \cdot t^2 < N$, and then iterate the construction r times, where $r = N/mt$. Thus one gets a data structure of size $r \cdot m$, plus the space needed to store the descriptions of the hash functions, and an inversion procedure whose complexity is dominated by the complexity of evaluating the r random hash functions at t points each. Fiat and Naor then show that each g_i only needs to be k-wise independent where k is approximately t, the length of the paths. While one evaluation of a t-wise independent hash function would take time t, Fiat and Naor show that the overall time for the rt evaluations can be made $t^2 + rt$ via a careful evaluation process and amortized analysis. The different g_i, in turn, only need to be pair-wise independent with respect to each other. Overall, the r hash functions can be represented using only about t bits, so that the space complexity is of the order of $r \cdot m + t$. Choosing the parameters r, m, t optimally shows that the time-space tradeoff $TS^2 = N^2$ is achievable.

For general functions, the above ideas continue to work if the collision probability λ of the distribution $f(U_{[N]})$ is small. In particular, one can have an algorithm of space $S = m \cdot r + t$ and time $T = t^2 + t \cdot r$ provided that $m \cdot t^2 \leq 1/\lambda$ and $m \cdot t \cdot r \geq N$. This optimizes to the time-space tradeoff $TS^2 = \lambda \cdot N^3$.

For functions having large collision probability, the idea is to create an additional look-up table L (we also refer to it as a list), containing, for each of the ℓ

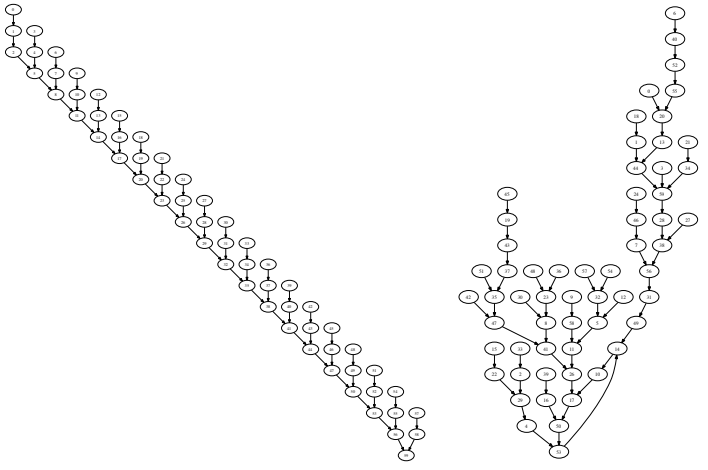

Fig. 1. A graph G_f that cannot be partitioned into few edge-disjoint paths and the graph $G_{f \circ g}$ where g is a random permutation

elements y such that $f^{(-1)}(y)$ is largest, the pair (x, y) where x is an arbitrary pre-image of y. Then, given $f(x) \in L$ we can immediately find an inverse by searching L, and the problem of inverting f reduces to the problem of inverting the restriction of f to $\{0,1\}^n - f^{(-1)}(L)$, which, intuitively, is the problem of inverting a function of low collision probability. More precisely, if we define the "effective" collision probability of f relative to L as the probability that, picking x, x' uniformly at random we have $f(x) = f(x')$ conditioned on $f(x) \notin L$, then the effective collision probability is at most $1/\ell$. The $TS^2 = \lambda N^3$ trade-off can be extended to the case in which λ is the effective collision probability, although at the additional cost of ℓ in the space. The optimal choice ends up being $\ell = S$, and so the trade-off becomes $TS^3 = N^3$. One additional difficulty that comes up in the analysis is that we need hash functions g_i with the property that $g_i(f(x)) \notin f^{(-1)}(L)$ if $f(x) \notin L$. This is achieved by realizing g_i by starting from a sequence of functions $g_i^1, \ldots, g_i^k$, and then defining $g_i(y)$ to be $h_i^j(y)$ for the first j such that $h_i^j(y) \notin f^{(-1)}(L)$.

2.2 Scaling Down the Fiat-Naor Construction

Consider now the issue of scaling down this construction in order to invert only ϵN points.

If we fix parameters r, m, t, ℓ such that $r \cdot m \cdot t = \epsilon N$ and $m \cdot t^2 \leq \ell$, then we have an algorithm that inverts the function at ϵN points and whose time complexity is $t^2 + rt$ and whose space complexity is $\ell + rm + t$. Some calculations show that this gives a time-space trade-off of $TS^3 = (\epsilon N)^3$.

Parameters

ℓ := Size of list L
t := Length of each walk
r := Number of independent functions $g \in \mathcal{F}$ used
m := Number of walks according to each function g_i
$\mathcal{F}$:= Family of $k = 2t \cdot (\log N)^2$-wise independent functions.

Construction of Data Structure

1. Consider the ℓ elements in the range of f with highest value of $I(y)$. For each such element y, store an entry (y, x) in the list for some $x \in f^{-1}(y)$.
2. Choose functions $g_1, \ldots, g_r \in \mathcal{F}$ pairwise independently at random. For each function g_i, define the partial function $g_i^* : [N] \to [N]$ as

$$g_i^*(x) = \begin{cases} g_i(x,u) & \text{if } u \text{ is the least index such that } f(g_i(x,u)) \notin L \\ \text{undefined if } \forall u \in [(\log N)^2].\ f(g_i(x,u)) \in L \end{cases}$$

   For each i, define the partial function $h_i = g_i^* \circ f$.
3. For each $i \in [r]$ and $j \in [m]$, construct a walk W_{ij} of length t; by starting at a random point x_{ij} and computing the sequence $x_{ij}, h_i(x_{ij}), \ldots, h_i^t(x_{ij})$. Discard the walk if
   - for some $t_1 \leq t$, $h_i^{t_1}(x_{ij})$ is undefined.
   - the walk cycles i.e. for $t_1, t_2 \leq t$, $h_i^{t_1}(x_{ij}) = h_i^{t_2}(x_{ij})$.

   For walks W_{ij} that are not discarded, store the pairs $(x_{ij}, h_i^t(x_{ij}))$.

Fig. 2. Description of data structure for inverting f

A first improvement comes by considering that if $|f^{(-1)}(L)| \geq \epsilon N$, then just by constructing L we are done. This means that we may assume that the elements not in L have each at most $\epsilon N/\ell$ pre-images, and there are $(1-\epsilon)N > N/2$ elements not in $f^{(-1)}(L)$, meaning that the collision probability of f restricted to $\{0,1\}^n - f^{(-1)}(L)$ is at most ϵ/ℓ. This is a stronger bound than the "effective collision probability" bound $1/\ell$ in the Fiat-Naor analysis. This means that we can set the parameters so that $rmt = \epsilon N$, $mt^2 \leq \ell/\epsilon$, and have $S = \ell + rm + t$ and $T = t^2 + rt$. This leads to the improved trade-off $TS^3 = \epsilon^4 N^3$, provided $\epsilon N > T > \epsilon^{-2}$.

A second improvement comes by using new constructions of k-wise independent hash functions (with $k = \tilde{O}(t)$) that can be evaluated in time negligible in t. We present such a construction in the full version of the paper. Using such a construction, the running time of the algorithm becomes just rt, rather than $t^2 + rt$. In the original Fiat-Naor construction, the two bounds are of the same order, because optimizing the parameters always leads to $r > t$. In the scaled-down construction we described above, however, $r > t$ is optimal only as long as $T > \epsilon^{-2}$, which is why we added such a constraint above. Hence, we require a family of hash functions with two properties:

Invert(y)

1. If $(x, y) \in L$ for some L, return x.
2. For each $i \in [r]$
   (a) Construct the sequence $(g_i^*(y), h_i(g_i^*(y)), \ldots, h_i^{t-1}(g_i^*(y)))$.
   (b) If there are indices $j_0 \in [m]$ and $t_0 \leq t-1$ such that $h_i^{t_0}(g_i^*(y)) = h_i^t(x_{ij_0})$, then compute $h_i^{t-t_0-1}(x_{ij_0})$. In case there are multiple choices for j_0, pick the smallest one.
   (c) If $f(h_i^{t-t_0-1}(x_{ij_0})) = y$, output $h_i^{t-t_0-1}(x_{ij_0})$ else output fail.

Fig. 3. Procedure for inverting a given element y

- *Small size:* it is sufficient for our purposes that each function be representable with $\Theta(t) + N^{o(1)}$ bits;
- *Efficient evaluation:* given the description of a function in the family and a point in the domain, we would like the evaluation of the function at that point to take time $t^{o(1)} \cdot N^{o(1)}$

We note that most known constructions with small size do not satisfy the efficient computation requirement. The construction that we use in this paper is based on an observation by Siegel [15] coupled with the lossless expander construction by Capalbo *et al.* [6]. The only other construction to us, which satisfies both the properties is the construction by Ostlin and Pagh [14] but their construction can differ from being uniform on a set of size t by an inverse polynomial in t which is too large an error for us. Using these hash functions would lead to the same trade-off $TS^3 = \epsilon^4 N^3$, but for the wider range of parameters $\epsilon N > T > \epsilon^{-1}$.

2.3 The Main New Idea

Our main improvement over the techniques of Fiat and Naor comes from the use of a more precise counting of the number of inputs x such that $f(x)$ can be inverted using a given data structure.

We note that if we have the endpoints of a path of length t in our data structure, then we are able to invert f not just at t inputs, but rather at as many inputs as the sum of the indegrees (in G_f) of the vertices of the path.[4] If, for example, the function f is k-regular (meaning that, for every x, $f(x)$ has exactly k pre-images), then a special case of the analysis that we provide shows that we can invert everywhere with trade-off $TS^2 = N^2/k^2$, while the Fiat-Naor analysis would give a trade-off $TS^2 = N^2 \cdot k$. They are the same when $k = \tilde{O}(1)$, but for larger k the analysis of Fiat and Naor provides worse bounds, because the collision probability increases, while our analysis provides better bounds.

[4] Said differently, Fiat and Naor count the number of y which are inverted, while one should count the number of x such that $f(x)$ is inverted.

For functions that are not regular, providing a good bound on the number of elements that are inverted by the data structure is more challenging.

If the function has collision probability λ (or "effective" collision probability λ after discounting the elements in the high-indegree table), and we construct r data structures, each having m paths of length t, then the average sum of the indegrees of the vertices in the data structure is $m \cdot t \cdot r \cdot \lambda \cdot N$, which is potentially much more than mtr if the collision probability is large. It seems, then, that we could fix parameters m, t, r such that

$$\begin{aligned} m \cdot t^2 &\leq \lambda^{-1} \\ m \cdot t \cdot r \cdot \lambda N &\geq \epsilon N \end{aligned} \tag{1}$$

and be able to invert ϵN elements using time rt and space $rm+t$. This would optimize, in the interesting case in which space and time are equal, to having space and time $\max\{\sqrt{\epsilon N}, \epsilon N^{2/3}\}$, which would be great. In particular, it would improve the Fiat-Naor construction even when $\epsilon = 1$. Unfortunately, while $mrt\lambda N$ is the expectation of the sum of the indegrees of the vertices in all the paths of the data structure, it is not the expectation of the number of x such that $f(x)$ is inverted: the problem is that, if the collision probability is very high, there might be elements y with many pre-images that occur in multiple data structures, and which would then be counted multiple times.

We then proceed by considering three cases. If the collision probability is small, that is, less than ϵ^2/S, where S is the amount of space we plan to use, then we find parameters m, t, r such that

$$\begin{aligned} mt^2 &\leq S/\epsilon^2 \\ mtr &\geq \epsilon N \end{aligned}$$

that is, we take advantage of the bound on collision probability but we do not attempt to improve the Fiat-Naor count on the number of inverted elements. This allows us to invert an ϵ fraction of elements using time and space at most $\max\{\sqrt{\epsilon N}, \epsilon^{5/4} N^{3/4}\}$.

If the collision probability is more than ϵ^2/S, then we consider how much $mrt\lambda N$ is overcounting the real number of inverted elements. The overcounting is dominated by the elements x such that, for a given choice of r, m, t, $f(x)$ has probability $\Omega(1)$, say, probability $\geq 1/100$, of belonging to one of the data structures. Call such a $y = f(x)$ a *heavy* image to invert.

If the number of pre-images of heavy elements is at least $100\epsilon N$, then we are done, because we expect to be able to invert at least a $1/100$ fraction of heavy elements.

The remaining case, then, is when the collision probability is more than ϵ^2/S, but the total number of pre-images of heavy elements is less than $100\epsilon N$. This information, together with the fact that (thanks to the size-S high-indegree table) we are only trying to invert elements with at most $\epsilon N/S$ pre-images, allows us to bound the total number of occurrences of heavy elements in the data structure, and to conclude that the total number of pre-images of non-heavy elements (which are inverted) is at least $\Omega(mrt\lambda N)$. This means that a

choice of m, r, t satisfying (1) leads us to invert an ϵ fraction of inputs, and to do so with time and space at most $\max\{\sqrt{\epsilon N}, \epsilon N^{2/3}\}$.

Applying these ideas to get full time-space trade-offs give us that, if $\epsilon < 1/N^{1/3}$ we can have the optimal trade-off $TS = \epsilon N$; otherwise we achieve the trade-off $TS^3 = \epsilon^5 N^3$. We now formally state the main theorem.

Theorem 1. *There is an oracle algorithm* Invert *such that given any* $f : \{0,1\}^n \to \{0,1\}^n$*, there is a data structure* DS *with parameters* ℓ*,* m*,* t *and* r *such that*

$$\Pr_{x \in [N]} \left[\mathsf{Invert}^{f, \mathsf{DS}}(f(x)) \in f^{-1}(f(x)) \right] \geq \epsilon$$

where the total space required for DS *is* $\tilde{O}(\ell + mr + t)$*, space required by* Invert *is* $\tilde{O}(t)$ *and the total time required by* Invert *is* $\tilde{O}(tr)$*. Hence, assuming that* $(\ell + mr + t) = O(S)$ *and* $tr = T$*, there is an algorithm (in the RAM model) which uses space* $\tilde{O}(S)$ *and time* $\tilde{O}(T)$ *and inverts* f *on an* ϵ *fraction of inputs. Here* $\tilde{O}$ *hides factors of* $2^{\mathrm{poly}\log\log N}$*.*

Remark 1. For most of the allowed range of the parameters we have $r < t$, and that in the "low-end" range $\epsilon < N^{-1/3}$ for which our result is optimal we have $r = 1$. For this reason there is a notable improvement in using our efficient hash functions instead of the amortized hash function evaluation of Fiat and Naor.

As described above, the algorithm by Hellman, Fiat and Naor, as well as ours involve a significant amount of search and hence it is interesting to ask if this search can be parallelized. There have been results in this direction by van Oorschot and Wiener [16] and more recently by Joux and Lucks [13]. Some of these results can be helpful in parallelizing even in the regime when ϵ is small (as in our case).

3 Attacks on Pseudorandom Generators

The starting point of our result for pseudorandom generators is the fact [2] that if two random variables ranging over $\{0,1\}^m$ have constant statistical distance, then there is a linear function (of $O(m)$ circuit complexity) that distinguishes the two random variables with advantage at least $2^{-m/2}$.

Suppose that we are given a length-increasing pseudorandom generator $G : \{0,1\}^{n-1} \to \{0,1\}^n$ and that we want to construct a distinguisher achieving distinguishing probability ϵ.

Our idea is to partition $\{0,1\}^n$ into $\epsilon^2 2^n$ sets each of size ϵ^{-2}, for example based on the value of the first $n - 2\log 1/\epsilon$ bits, and then apply within each block the linear function that provides, within that block, the best distinguishing probability. Overall, this defines a function of circuit complexity $O(\epsilon^2 \cdot n \cdot 2^n)$. Then, intuitively, within each block we achieve distinguishing probability at least ϵ, because each block is a set of size ϵ^{-2}, and the distinguishing probability is at least the square root of the inverse of the block size.

The straightforward implementation of this intuition would be to use, in each block, the linear function that best distinguishes the uniform distribution within

the block from the conditional distribution of the output of the generator conditioned on landing in the block. Unfortunately this approach would not work because the overall distinguishing probability is not a convex combination of the conditional distinguishing probabilities.[5]

Instead, in each block, we choose the linear function that most contributes to the overall distinguishing probability. In order to quantify this contribution we need a slight generalization of the result of [2].

We then present a more efficient distinguisher of circuit complexity $O(\epsilon^2 \cdot 2^n)$ which employs a hash function sampled from a 4-wise independent family, and whose analysis employs a more involved fourth-moment argument, inspired by [3]. As noted before, all our ideas apply to the case when we want to distinguish between two arbitrary distributions D_1 and D_2. In particular, given two distributions D_1 and D_2 with statistical distance δ, we can construct a circuit of size $O(\epsilon^2 2^n)$ which distinguishes between D_1 and D_2 with probability $\epsilon\delta$. More, formally, we have the following result.

Theorem 2. *Given any two distributions D_1 and D_2 over $\{0,1\}^n$ such that their statistical distance is δ and $\epsilon \leq 2^{n/2}$, there is a circuit C of size $O(\epsilon^2 \cdot 2^n)$ such that*

$$\mathbf{P}[C(D_1) = 1] - \mathbf{P}[C(D_2) = 1] \geq 2\epsilon\delta$$

4 Lower Bounds

Using techniques of Yao [18], Gennaro and Trevisan [8], and Wee [17], it is possible to show that, in the generic oracle setting that we consider in this paper, there are permutations for which the amount of advice S and the oracle query complexity T must satisfy

$$S \cdot T \geq \tilde{\Omega}(\epsilon N)$$

for any algorithm that inverts an ϵ fraction of inputs. More precisely, we prove the following theorem.

Theorem 3. *If A is an oracle algorithm that runs in time at most T and such that for every permutation $f : [N] \to [N]$ there is a data structure adv of size $\leq S$ such that*

$$\mathbf{P}_x[A^f_{adv}(f(x)) = x] \geq \epsilon$$

Then

$$S \cdot T = \tilde{\Omega}(\epsilon N)$$

[5] This is a subtle issue related to the fact that the condition of landing in a given block might have different probabilities in the uniform distribution versus the output of the generator. If so, then the respective conditional probabilities are normalized differently, and the use of a distinguisher for the conditional distributions in a block does not necessarily contribute to the task of distinguishing the original distributions.

Such lower bound proofs are based on the idea that an algorithm with better performance could be used to encode every permutation $f : [N] \to [N]$ using strictly less than $\log N!$ bits, which is impossible. Here, we simplify such proofs by using randomized encodings. (Even a randomized encodings cannot represent every permutation using less than $\log N!$, and showing that such an encoding would be possible if the lower bound were wrong is easier by using randomization.) In fact, while previous proofs gave a lower bound on the trade-off only when $T = \tilde{O}(\sqrt{\epsilon N})$, our lower bound works for the full range of parameters.

We then consider the question of the security of pseudorandom generators in the oracle setting. By using the aforementioned results for permutations and applying efficient hard-core predicates [9], it is possible to show the existence of generators for which $S \cdot T \geq \epsilon^7 N$. By instead applying the ideas of randomized encodings to a pair f, p where f is a random permutation (modeling a one-way permutation) and p is a random predicate (modeling a "hard-core predicate" for p), we prove the existence of length-increasing generators such that for every distinguisher that makes T oracle queries to the generator, and which has advice S and distinguishing probability ϵ, we have

$$S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$$

where N is the number of seeds. Formally, we have the following theorem.

Theorem 4. *Suppose that A is an oracle algorithm that makes T queries, uses a S-bit advice string, and is such that for every length-increasing function $G : [N] \to [N] \times \{0,1\}$ there is an advice string adv such that*

$$|\mathbf{P}[A^G_{adv}(G(x)) = 1] - \mathbf{P}[A^G_{adv}(y) = 1]| \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$

A key intermediate result used here is a lower bound on the following kind of computation. Given any predicate $P : \{0,1\}^n \to \{0,1\}$ we are required to compute $P(x)$ by querying the oracle P on any point but x. For this kind of computation, Yao [18] had established an (optimal) lower bound on the trade-off between length of advice and the number of queries to the oracle P when one is required to compute P correctly at all places. We extend this to the case when we are only required to compute P correctly at $1/2 + \epsilon$ fraction of the places, to get the following result.

Theorem 5. *Suppose that A is an oracle algorithm that makes T queries while never querying its input, uses a S bit advice string, and is such that for every predicate $P : [N] \to \{0,1\}$, there is an advice string adv such that*

$$\mathbf{P}[A^P_{adv}(x) = P(x)] \geq \frac{1}{2} + \epsilon$$

Then $S \cdot T \geq \Omega(\epsilon^2 N)$.

We note that the optimal lower bound in this case seems to be $S \cdot T \geq \Omega(\epsilon N)$ and we do not how to close this gap. This gap is reflected in the gap between our

lower bound for pseudorandom generators of the form $x \to f(x)p(x)$ where f is a permutation, and known constructions of distinguishers for such generators. In particular, the best known algorithm is one of the following (depending on ϵ, S, T) : Use the algorithm for inverting functions which can be at best $S \cdot T \leq \tilde{\Omega}(\epsilon N)$ or use the circuit which we described in the previous subsection. That in particular uses $S = \tilde{\Omega}(\epsilon^2 N)$ and $T = \tilde{\Omega}(1)$.

Finally, we look at the *common random string* model, in which all parties share a common random string k, which they can use to select a permutation $f_k(\cdot)$ from a family of permutations, or a generator $G_k(\cdot)$ from a family of generators. In such a setting, the trivial brute force attack that achieves inverting (and distinguishing) probability $\epsilon = T/N$ with no advice remains possible. Alternatively, one can think of a family of permutations as a single permutation $(k, x) \to (k, f_k(x))$. We show that, for families of permutations, either the trivial uniform algorithm or Hellman's construction applied to the mapping $(k, x) \to (k, f_k(x))$ are best possible, depending on whether the available advice is shorter or longer than the number of keys. More precisely, we prove the following theorem.

Theorem 6. *Suppose that A is an oracle algorithm that makes T queries, uses an S-bit advice string, and is such that for every family of permutations $f : [K] \times [N] \to [N]$ there is an advice string adv such that*

$$\mathbf{P}_{k \in [K],\ x \in [N]}[A^f_{adv}(k, f(k, x)) = x] \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon KN) - \tilde{O}(KT)$

We also prove strong lower bounds for distinguishers for pseudorandom generators in the common random string model. We show the following theorem.

Theorem 7. *Suppose that A is an oracle algorithm that makes T queries, uses an S-bit advice string, and is such that for every family of length-increasing functions $G : [K] \times [N] \to [N] \times \{0, 1\}$ there is an advice string adv such that*

$$|\mathbf{P}_{x \in [N]}[A^G_{adv}(k, G(k, x)) = 1] - \mathbf{P}_{y \in [2N]}[A^G_{adv}(k, y) = 1]| \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon^2 KN) - \tilde{O}(KT)$.

This translates to saying that for generators in the common random string model, either $T \geq \tilde{\Omega}(\epsilon^2 N)$ or $ST \geq \tilde{\Omega}(\epsilon^2 KN)$, where K is the number of keys.

5 Model of Computation

Positive Results. In the time/space trade-offs of Hellman, of Fiat and Naor, and of this paper, an algorithm uses "time T" and "space S" if it runs in time at most T (in a RAM model), uses at most S bits of space, and works correctly upon receiving S bits of advice, in the form of an S-bit data structure that dominates the space requirement of the algorithm.

The "advice," in turn, can be computed in uniform time $\tilde{O}(N)$. In the work of Hellman and of Fiat and Naor, one cannot hope, in general, to have processing time significantly smaller than N in order to generate the data structure used by the algorithm. Otherwise, one would have a uniform algorithm that inverts an arbitrary one-way permutation (or function) in time noticeable smaller than N, which is impossible relative to a random permutation (or function) oracle.

In our paper, the data structure we use is also easily pre-computable in time $\tilde{O}(N)$. Pre-processing time significantly smaller than ϵN should not be expected, because then we would have a uniform algorithm to invert a random function on an ϵ fraction of inputs in time significantly smaller than ϵN. With some care, our data structure can indeed be pre-computed using optimal uniform time $\tilde{O}(\epsilon N)$. We, however, do not describe it here for the sake of simplicity.

Negative Results. When we show that a particular combination of space S and time T is not achievable, our result rules out non-uniform algorithms that make at most T oracle queries to the function (or generator) oracle, and which receive at most S bits of advice. The actual space used by the algorithm, as well as the complexity of the computations performed between oracle queries, can be unbounded. Likewise, the non-uniform advice can have arbitrary complexity.

Acknowledgements

We would like to thank Daniel Wichs for suggesting the study of the common random string model, and Scott Aaronson, Cynthia Dwork, Omer Reingold, Udi Wieder and Hoeteck Wee for pointers to the literature. We also thank the anonymous reviewers for helpful comments and pointing us to [16,13].

References

1. Aaronson, S.: Lower bounds for local search by quantum arguments. SIAM Journal of Computing 35(4), 804–824 (2006)
2. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Structures and Algorithms 3(3), 289–304 (1992)
3. Andreev, A.E., Clementi, A.E.F., Rolim, J.D.P.: Optimal bounds for the approximation of boolean functions and some applications. Theoretical Computer Science 180, 243–268 (1997)
4. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006)
5. Bennett, C., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM Journal on Computing 26(5), 1510–1523 (1997)
6. Capalbo, M.R., Reingold, O., Vadhan, S.P., Wigderson, A.: Randomness conductors and constant-degree lossless expanders. In: Proceedings of the 34th ACM Symposium on Theory of Computing, pp. 659–668 (2002)
7. Fiat, A., Naor, M.: Rigorous time/space trade-offs for inverting functions. SIAM Journal on Computing 29(3), 790–803 (1999)

8. Gennaro, R., Trevisan, L.: Lower bounds on the efficiency of generic cryptographic constructions. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, pp. 305–313 (2000)
9. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: Proceedings of the 21st ACM Symposium on Theory of Computing, pp. 25–32 (1989)
10. Golynski, A.: Cell probe lower bounds for succinct data structures. In: Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms, pp. 625–634 (2009)
11. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 212–219 (1996)
12. Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26(4), 401–406 (1980)
13. Joux, A., Lucks, S.: Improved generic algorithms for 3-collisions. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 347–363. Springer, Heidelberg (2009)
14. Ostlin, A., Pagh, R.: Uniform hashing in constant time and linear space. In: Proceedings of the 35th ACM Symposium on Theory of Computing, pp. 622–628 (2003)
15. Siegel, A.: On universal classes of extremely random constant-time hash functions. SIAM Journal of Computing 33(3), 505–543 (2004)
16. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. Journal of Cryptology 12, 1–28 (1999)
17. Wee, H.: On obfuscating point functions. In: Proceedings of the 37th ACM Symposium on Theory of Computing, pp. 523–532 (2005)
18. Yao, A.: Coherent functions and program checkers. In: Proceedings of the 22nd ACM Symposium on Theory of Computing, pp. 84–94 (1990)