Saving Private Randomness in One-Way Functions and Pseudorandom Generators

Nenad Dedić[1,2,4], Danny Harnik[3,4], and Leonid Reyzin[1,4]

[1] Boston University, Department of Computer Science, 111 Cummington St., Boston, MA 02215. reyzin@cs.bu.edu
[2] Google, Inc., 76 9th Ave., 6th Floor, New York, NY 10011. nenad.dedic@gmail.com
[3] IBM Research, Haifa, Israel. danny.harnik@gmail.com
Research conducted while at the Technion, Haifa, Israel.
[4] Research conducted, in part, at the Institute for Pure and Applied Mathematics at UCLA, whose hospitality the authors gratefully acknowledge.

Abstract. Can a one-way function f on n input bits be used with fewer than n bits while retaining comparable hardness of inversion? We show that the answer to this fundamental question is negative, if one is limited black-box reductions.

Instead, we ask whether one can save on *secret* random bits at the expense of more *public* random bits. Using a shorter secret input is highly desirable, not only because it saves resources, but also because it can yield tighter reductions from higher-level primitives to one-way functions. Our first main result shows that if the number of output elements of f is at most 2^k, then a simple construction using pairwise-independent hash functions results in a new one-way function that uses only k secret bits. We also demonstrate that it is not the knowledge of *security* of f, but rather of its *structure*, that enables the savings: a black-box reduction cannot, for a general f, reduce the secret-input length, even given the knowledge that security of f is only 2^{-k}; nor can a black-box reduction use fewer than k secret input bits when f has 2^k distinct outputs.

Our second main result is an application of the public-randomness approach: we show a construction of a pseudorandom generator based on any *regular* one-way function with output range of *known* size 2^k. The construction requires a seed of only $2n + \mathcal{O}(k \log k)$ bits (as opposed to $\mathcal{O}(n \log n)$ in previous constructions); the savings come from the reusability of public randomness. The secret part of the seed is of length only k (as opposed to n in previous constructions), less than the length of the one-way function input.

1 Introduction

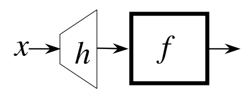
PRG Seed Length It is important to keep the seed required for a pseudorandom generator (PRG) as short as possible, lest the amount of true random bits needed to run it exceed the amount of pseudorandom bits its application requires, thus rendering it pointless. Moreover, in reductions from PRGs (or other constructs) to one-way functions, the blowup in the input length turns out to be the most central parameter in determining the security of the construct. It is therefore a major goal to reduce this parameter (as was addressed in [GIL$^+$90,HL92,HHR06b,Hol06,HHR06a]). The ultimate goal is a linear blowup, a necessary, although not a sufficient, condition to achieve a reduction with tight security preservation, i.e. a linear preserving one [HL92,HILL99].

Consider, therefore, the following problem: when is it possible to build a pseudorandom generator out of a one-way function f while keeping the generator seed length linear in the one-way function input length n? Certainly this is possible if f is a permutation—in fact, in the original PRG construction of [BM82,Yao82] the seed length is equal to the one-way function input length. However, no broader class of one-way functions satisfying this condition is currently known: even one-way bijections, if their output range is not easily mapped to $\{0,1\}^n$, are not known to satisfy this condition (the best constructions for them are the same as for other regular one-way functions, discussed below).

In this paper we demonstrate constructions of PRGs with the linear input length condition for a large class of *known regular* one-way functions. Specifically, if every output of f has α preimages (thus f has 2^k distinct outputs where $k = n - \log \alpha$) and (a lowerbound on) α is known, then we can build a PRG with seed length $2n + \mathcal{O}(k \log k)$. Thus, for functions with high enough degeneracy, where $k = \mathcal{O}(n/\log n)$, our PRG has a linear-length seed, like the Blum-Micali-Yao PRG built from one-way permutations. The construction, described in Section 4, builds upon the techniques of Haitner, Harnik and Reingold [HHR06b], which require longer seed length of $\mathcal{O}(n \log n)$, but assume only regularity rather than *known* regularity.

New Tool: One-Way Functions with Short Secret Inputs We arrive at our pseudorandom generator as part of a study of a more fundamental problem: when is it possible to reduce the input length of a one-way function while maintaining some of its security? In other words, given a one-way function f with input length n, when is it possible to build another function g of input length $\ell(n) < n$ with comparable security? Indeed, if this were possible, then one could, for example, build a pseudorandom generator from g rather than from f, and maintain a reasonable seed length even if the PRG construction blows up the input size. However, we show that in general it is impossible to significantly reduce the input length of one-way function in a black-box manner, even for regular one-way functions (Theorem 5). That is, one must invest essentially the full n random bits when calling a one-way function.

This result, however, does not doom all efforts of using the one-way function with a shorter input. The insight is to use the paradigm introduced by Herzberg and Luby [HL92], which separates *public* randomness from *secret* randomness. It turns out to be possible to reduce the amount of *secret* randomness at the cost of additional *public* randomness. In Theorem 1 we show how to convert any one-way function f with 2^k distinct outputs into a *collection* of one-way functions f_h with inputs of length k, where the index h into the collection is the public randomness. The simple construction uses a pairwise independent family of *expanding* hash functions. The choice of the function from the collection is a choice of a hash function h, and we define $f_h(x) = f(h(x))$. This choice is made using $2n$ *public* random coins, which are available to any potential inverter.



One way to achieve such a result is by using a technical Lemma of Dodis and Smith [DS05, Lemma 12], which shows the same construction secure if it uses $k + 2\log\frac{1}{\varepsilon} + 1$ secret input bits, where ε is the additive security loss. In particular, even if one needs to ensure that extra security loss is exponentially small, the result of [DS05] requires only linearly more input bits. However, the linear improvement we achieve over [DS05] is crucial for building our pseudorandom generator, as we explain shortly. To achieve this improvement, we take a different path from [DS05]: instead of showing that the distributions $(f(x), h)$ and $(f(h(x)), h)$ are statistically close, we show they have polynomially related subset weights, a relation between distributions that we call g-domination.

The secret input to our one-way function need not consist of k uniform independent bits: inputs from any distribution of entropy[5] k suffice (the same is true for our pseudorandom generator construction). This is beneficial, because uniform random bits may be harder to obtain that simply strings of high entropy.[6] Moreover, this enables our pseudorandom generator construction.

Application: The PRG Construction We construct our pseudorandom generator by applying the randomized iterate construction of [HHR06b] (henceforth called "the HHR construction") to f_h for a known regular f. Because f_h is secure even when h is public, the coins for h can be given only once and used for all iterations, resulting in a shorter seed. As compared to the HHR construction, we replace the need for many large hash functions with one large hash function (the $\hat{h}$ used for $f_{\hat{h}}$), and many small ones ($h^1, \ldots, h^k$ used in the randomized iterate construction). Our construction is illustrated in Figure 1.

To get some intuition for the construction, observe that if f is regular, then the number of secret random input bits we require for f_h is the entropy of the output of f_h. This enables iteration, because the output of f_h has enough entropy to be used (after an appropriate transformation) as an input to the next f_h. We could not use the result of [DS05], because it requires more input entropy than is output; nor could we use functions that are not regular, because they produce less output entropy than the input requires. The proof of pseudorandomness is

5 Specifically, Renyi entropy of order 2, i.e., negative logarithm of collision probability.

6 Of course, almost uniform independent bits can be obtained from a distribution of high entropy through the use of a strong extractor (whose seed can be public), but extractors necessarily lose entropy, so this approach would require a secret input with entropy higher than k, which, as we already pointed out, would create difficulties for our PRG construction.

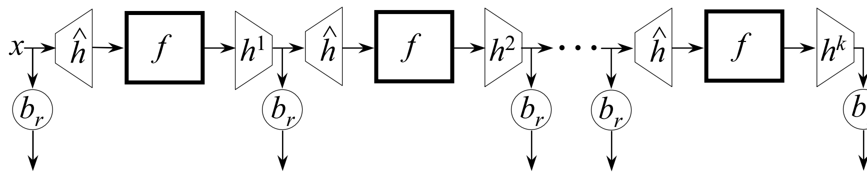


Fig. 1. Our pseudorandom generator on seed x. $\hat{h}$ is a pairwise-independent hash function from k bits to n bits; $h^1, h^2, \ldots, h^k$ are almost-pairwise independent hash functions from the output space of f to k bits, generated by a bounded space generator from a common seed s of length $\mathcal{O}(k \log k)$; b_r is the Goldreich-Levin hardcore bit (the same r is used throughout). $\hat{h}$, s and r are included in the output or, equivalently, are public.

not as simple as applying the HHR result to $f_{\hat{h}}$, because the HHR construction needs to start with a regular one-way function, and $f_{\hat{h}}$ is not necessarily regular even if f is.

In Appendix A we show how one can further exploit the knowledge of the regularity and further shorten the seed of our PRG to $2n + \mathcal{O}(k \log \log k)$, albeit at the cost of lowering its security.

In addition to considering the overall PRG seed length, it is also important to consider how much of the generator seed must be secret, because secret random bits tend to be much harder to obtain than nonsecret ones (again, this was already observed in [HL92]). Our PRG is the first to require a *sublinear* number of secret bits, namely, just k (the HHR generator, like the generators of [BM82,GKL93], requires n secret bits). Moreover, just like for our one-way function, the secret input to our PRG need not consist of uniform independent bits, but can come from any distribution of entropy k.

Example: One-way Function and PRG Based on Factoring Consider the problem of building a one-way function based on the hardness of factoring products of two b-bit randomly chosen primes. If one is willing to assume a trusted party with secret coins, then it is easy: the trusted party chooses two secret random b-bit primes p and q, publishes $N = pq$, and the function can be, for example, squaring modulo N.

However, without trusted setup, there is no such easy construction. In order to work on the domain $\{0,1\}^n$, the one-way function needs to include the process of generating the two random primes. A natural way to do this is to test some number of random integers for primality. To guarantee that two primes are found with probability 2^{-s} for some security parameter s, the number of integers tested should be $\Theta(sb)$ (because the probability that a random b-bit integer is prime is $\Theta(1/b)$). The natural function therefore gets $n = \Theta(sb^2)$ bits as input, splits them into $\Theta(sb)$ integers of length b each, finds the first two such integers p, q that are prime (if they do not exist, output 0), and outputs their product $N = pq$. We call this function f_{mult} (observe that, for sufficiently large s, it is one-way under the assumption that factoring is hard).

For reasonably secure values for b (e.g., 2048) and s (e.g., 64), the input length n of f_{mult} will be on the order of tens of megabytes. To come up with such a long secret input is, naturally, quite costly. Because the output of f_{mult} is short, however, we can apply our result on converting one-way functions to families with shorter secret inputs. Setting $k = 2b = o(\sqrt{n})$, we obtain a family of one-way functions with secret inputs of length only $2b$—as short as the description of the two primes p and q. To sample a function from this family, one still needs $\Theta(n)$ random bits, but they can be public, and are therefore much less expensive to obtain (e.g., from adversarially observable sources such as user behavior or ambient noise). Finally we note that using our techniques, one can generate a product $N = pq$ of two secret b-bit primes p,q using private randomness of entropy $2b$ (and the appropriate amount of public randomness). This can be used, for example, for generating public/secret key pairs for RSA or Paillier functions, from a modest amount of private randomness.

Consider now trying to make a PRG out of f_{mult}. The prior most efficient way (in terms of seed length) to achieve this is to notice that f_{mult} is a regular one-way function (except the negligible 2^{-s} portion that leads to the 0 output) and use the HHR construction, which takes a seed of $\mathcal{O}(n \log n)$ bits with $\mathcal{O}(n)$ of the bits being secret.[7] For reasonable parameter settings, it would be useful only in applications that can afford to gather tens of megabytes of secret randomness and gigabytes of public randomness before invoking the PRG.

Instead, observe that f_{mult} is also a known[8] regular one-way function, with $k < 2b$. Applying our PRG construction, we get a pseudorandom generator with just $2b = o(\sqrt{n})$ secret seed bits (which is roughly what's required to describe the two primes, anyway) and $\mathcal{O}(n)$ seed bits total (which is linear in what's anyway required as an input to f_{mult}).

Impossibility Results As already mentioned, Theorem 5 shows that the total input length of a one-way function cannot be reduced in a black-box manner, thus leading us to use public randomness in order to reduce the amount of secret randomness. It is natural to ask if this approach can also work for one-way functions with a large number of outputs. On the positive side, we show in Theorem 2 that if a sufficiently large portion of the inputs goes to a sufficiently small portion of the outputs, then the answer is yes. In general, however, this appears unlikely to be the case, for the following reasons. In Theorem 6 we show that the number of *secret* random bits used when calling a one-way *permutation* f cannot be reduced to be substantially smaller than n by use of black-box reductions. This theorem is actually more general, and shows that our positive result is indeed tight for regular one-way functions, and the number of secret

[7] It seems fruitless to try to turn f_{mult} into a permutation to order to apply the efficient construction of [BM82,Yao82]. Indeed, a natural way to build a bijection from f_{mult} is to include in the output all the unused bits as well as information on where p and q were in the sequence. However, this does not make it a permutation, because the output range (which includes the product of two primes) is not easily mapped back to the input domain of bit strings. Unfortunately, known solutions for bijections are not any better than those for regular functions.

[8] Our results apply to a weaker notion of "known": α can be a lower bound on the regularity of f, rather than its exact value.

bits cannot be reduced any further in a black-box manner. Moreover, Theorem 7 shows that there is no black-box reduction that takes a one-way function f with hardness 2^s on n input bits and produce a collection of one-way functions on $n - s + \mathcal{O}(\log n)$ input bits. Thus, unless f has hardness very close to 2^n, in general the number of secret inputs bits must remain linear if one wants to have *any* hardness at all.

Discussion Ideally, one would like to use only as many secret bits as the security one gets from the one-way function (it is clear that at least that many bits are necessary: a one-way function with n secret input bits can be easily inverted with probability 2^{-n}). Indeed, typical conjectured one-way functions, for example, RSA or discrete logarithm, are known to provide less security than 2^n (for the above examples, at most roughly $2^{n^{1/3}}$). Our negative results show that this is not possible in general with a black-box reduction (although we do not rule it out for specific functions such as discrete logarithm, of course). Our positive result, however, shows that if this weaker than optimal security manifests itself in a "structural" way, i.e., with the function having fewer outputs (a one-way function with k output bits can be easily inverted with probability 2^{-k}), then reduction in the number of inputs bits is possible.

It is natural to ask, of course, if one can not simply use the same one-way function f on a shorter input. It should be noted that our negative results do not consider such constructions, and hence do not rule them out. However, this option is unavailable when f is a fixed-length function secure in a concrete sense, such as a 128-bit block cipher or a hardware device implementing modular exponentiation for a 2,048-bit modulus. In this case, our impossibility results indicate that if we are given a hardware implementation of a one-way function we should use it with its full input length (unless we can look inside the box and learn something from there). This last observation adds motivation to results that take as input an exponentially hard one-way function and construct from it a pseudorandom generator with *weaker* security (of $n^{\log n}$) (e.g., some of the results stated in [Hol06,HHR06a] and the one in Appendix A in this paper). These results would be less interesting if there was a direct method of trading input length for security.

Even when the one-way function has variable input length, using it on a shorter input will reduce security. Of course, our construction also reduces security, but the security loss (i.e., security of f_h with n-bit f as compared to security of f on n bits) is polynomial. In contrast, simply using f on a shorter input can reduce security more than polynomially when the reduction in input length is superlinear.

Security comparison of the original f and our construction f_h depends on what parameters are set to equal each other. For example, we can compare the security of f on n bits to the security of f_h with a n-bit f (thus equating the input length to f, and hence the output length and likely most of the computational cost). In that case, f_h incurs a polynomial deterioration in security. Herzberg and Luby [HL92] advocate equating the secret input length. In that comparison, our constructions can actually be *more* secure that f, because f needs all n bits to be secret, while f_h and our PRG need only $k < n$ secret bits.

2 Definitions and Notation

If Y is a set, we denote by Y also the uniform distribution over that set, unless another distribution on Y is specified. We denote by U_n the uniform distribution over $\{0,1\}^n$. Given a distribution X and a function $f : X \to Y$, we denote by $f(X)$ the induced distribution on Y.

Let P and Q be distributions over some finite domain X. The collision-probability of P is $CP(P) = \sum_{x \in X} P(x)^2$. P and Q ε-close (or have statistical distance ε) if for every $A \subseteq X$ it holds that $|\Pr_{x \leftarrow P}(A) - \Pr_{x \leftarrow Q}(A)| \leq \varepsilon$ (equivalently, $\frac{1}{2}\sum_{x \in X} |\Pr_P[x] - \Pr_Q[x]| \leq \varepsilon$).

We assume familiarity with the standard notions of computational indistinguishability, one-way functions and pseudorandom generators (with public inputs, or equivalently, as public-coin collections), which are given in the full version of this paper [DHR07].

Definition 1 (Regular functions). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* regular *if for any $x, y \in \{0,1\}^n$, $|f^{-1}(f(x))| = |f^{-1}(f(y))|$. If $k(n) = -\log(|\{f(x) \mid x \in \{0,1\}^n\}|)$ then f is said to be* regular with output entropy *k. When k is also polynomial-time computable on input 1^n, f is* known-regular*.*

It is also customary to say that f *is an α-regular function* (for some $\alpha : \mathbb{N} \to \mathbb{N}$) — this means that f is a regular function with output entropy $k(n) = n - \log \alpha(n)$, i.e. preimage sizes are equal to $\alpha(n)$.

Definition 2 (Family of almost pairwise-independent hash functions). *Let $\{X_n\}_{n\in\mathbb{N}}$,$\{Y_n\}_{n\in\mathbb{N}}$ be two families of subsets of $\{0,1\}^*$. For any $n \in \mathbb{N}$ let $\mathcal{H}_n$ be a collection of functions where each $h \in \mathcal{H}_n$ is from X_n to Y_n. $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ is an* (efficient) family of δ-almost pairwise-independent hash functions *if: 1. there is a polynomial-time sampler which on $n \in \mathbb{N}$ outputs a description of randomly chosen $h \in \mathcal{H}_n$, 2. for any $h \in \mathcal{H}_n$, $|h|$ (i.e., the description length of h) is polynomial in $\log|X_n|$, 3. each $h \in \mathcal{H}_n$ is a polynomially-computable function, and 4. for all $x \neq x' \in X_n$ and all $y, y' \in Y_n$,*

$$\left| \Pr_{h \leftarrow \mathcal{H}_n}[h(x) = y \bigwedge h(x') = y'] - \frac{1}{|Y_n|^2} \right| \leq \delta(n).$$

A 0-almost pairwise independent family is called simply pairwise independent*.*

There are various constructions of efficient families of pairwise-independent hash functions (i.e. $\delta = 0$) for any $X_n = \{0,1\}^n$ and $Y_n = \{0,1\}^{\ell(n)}$ whose description length (i.e., $|h|$) is linear in $max\{n, \ell(n)\}$ (e.g., [CW77]). It is possible to construct δ-almost pairwise independent families for $\delta > 0$ whose description size depends very mildly on the input size. In particular, using [CW77], [WC81] and [NN93] one gets constructions of efficient families of almost pairwise-independent hash functions for $X_n = \{0,1\}^n$ and $Y_n = \{0,1\}^{\ell(n)}$ whose description length is $\mathcal{O}(\log(n) + \ell(n) + \log(1/\delta))$.

Proposition 1. *Let $\{\mathcal{H}_n\}$ be a family of δ-almost pairwise independent hash functions from X_n to Y_n. Then for any n, and any distinct $x_1, x_2 \in X_n$ the following distributions have statistical distance at most $\delta|Y_n|^2/2$: 1. uniform on $Y_n \times Y_n$, 2. $(h(x_1), h(x_2))$ for uniformly random $h \in \mathcal{H}_n$.*

Proof: For any $y_1, y_2 \in Y_n$,
$\left|\Pr_h[(h(x_1), h(x_2)) = (y_1, y_2)] - \Pr_{(z_1,z_2)\in Y_n\times Y_n}[(z_1, z_2) = (y_1, y_2)]\right| \le \delta$ by definition. Summing over all $y_1, y_2 \in Y_n$ and dividing by 2, we get the desired result.
□

To simplify exposition, we will often work with (almost) pairwise independent hash functions on some fixed domain and range X and Y (rather than consider families $\{X_n\}$, $\{Y_n\}$).

Definition 3 (g-Domination). *Let B and C be distributions on the same set Π, and g a real-valued function. We will say that C g*-dominates *B if $\forall S \subseteq \Pi$, $\Pr_C[S] \ge g(\Pr_B[S])$ (this is a generalization of the notion of "dominates" from [Lev86], which contemplated linear g).*

Lemma 1. *If C g-dominates B for a convex function g, then for any distribution D on a set Φ, $D \times C$ g-dominates $D \times B$.*

Proof: Let $E \subset \Phi \times \Pi$. Let $p(\pi)$, for $\pi \in \Pi$, be $\Pr_{\phi \leftarrow D}[(\phi, \pi) \in E]$.

$$\begin{aligned}
\Pr_{D\times C}[E] &= \mathop{\mathbb{E}}_{\pi \leftarrow C} p(\pi) \\
&= \int_0^1 \Pr_{\pi \leftarrow C}[p(\pi) > \alpha]\, d\alpha \quad (\text{using } \mathbb{E}(x) = \textstyle\int \Pr[x > \alpha]\, d\alpha) \\
&\ge \int_0^1 g\left(\Pr_{\pi \leftarrow B}[p(\pi) > \alpha]\right) d\alpha \\
&\ge g\left(\int_0^1 \Pr_{\pi \leftarrow B}[p(\pi) > \alpha]\, d\alpha\right) \quad (\text{Jensen's inequality, since } g \text{ is convex}) \\
&= g\left(\mathop{\mathbb{E}}_{\pi \leftarrow B} p(\pi)\right) = g\left(\Pr_{D\times B}[E]\right). \quad \square
\end{aligned}$$

A common approach in cryptographic reduction is to focus only on the subset of B for which $p(\pi)$ is large, and use Markov's inequality to obtain g'-domination of $D \times B$ by $D \times C$, for $g' \in \omega(g)$. Instead, this lemma, which takes all subsets into account, saves the increase in g and the corresponding loss of tightness in reductions.

3 One-way Functions and Public Randomness

Here we show that a one-way function needs only as many secret input bits as the number of output bits it produces. We state our theorem in terms of bits in order to get a more concise statement; neither the domain nor the range need to be restricted to bit strings of a particular length, as shown in Lemma 2.

Theorem 1. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one-way function that on n-bit inputs has at most 2^k distinct outputs. Let $\mathcal{H}_{k,n}$ be a family of pairwise-independent functions from $\{0,1\}^k$ to $\{0,1\}^n$. Define the domain-sampled f as $f_h(x) \stackrel{def}{=} f(h(x))$ for $h \in \mathcal{H}_{k,n}$ and $x \in \{0,1\}^k$. Then $\{f_h\}_{h\in\mathcal{H}_{k,n}}$ is a public-coin collection of one-way functions.*

The theorem is immediate from the following lemma.

Lemma 2. *Let $f : Y \to Z$ be a function, where $|Z| = K$. Let X be a distribution with collision probability at most $1/K$, and let $\mathcal{H}_{X,Y}$ be a family of pairwise-independent functions from the elements of X to Y. For every $h \in \mathcal{H}_{X,Y}$ define $f_h : X \to Z$ as $f_h(x) \stackrel{def}{=} f(h(x))$. Then any adversary A that inverts f_h with probability at least ε over $x \in X$ and $h \in \mathcal{H}_{X,Y}$ can be used to invert f on uniformly random inputs from Y with probability at least $\varepsilon^4/21 - 1/(4K^2)$ ($\varepsilon^2/2$ if f is regular) in the* same running time *as A (plus the time required to pick and evaluate a random hash function from $\mathcal{H}_{X,Y}$).*

Proof: Suppose that an algorithm A, when given $(f_h(x), h)$ computes x' such that $f_h(x') = f_h(x)$ with probability ε. That is,

$$\Pr_{(x,h) \leftarrow (X,\mathcal{H}_{X,Y})}[f_h(A(f_h(x), h)) = f_h(x)] \geq \varepsilon$$

Consider the following procedure M^A for inverting f: on input $z \in Z$, choose a random $h' \in \mathcal{H}_{X,Y}$, let $x' = A(z, h')$, and output $h'(x')$. Note that the notation h' in M^A, rather than h, emphasizes that the h' does not necessarily have to be consistent with z. While there exist many h with x such that $z = f_h(x)$, the chosen h' might not be one of them.

We will analyze the success probability of M^A as follows. The success of A (and therefore M^A) is determined by its internal coin flips and its input (z, h'). We will show that the distribution of (coinflips, input) pairs that A sees when run within M g-dominates the distribution for which A is designed, for a polynomial g; therefore, the probability of the event that M^A succeeds in inverting f is polynomially related to the probability of the event that A inverts the domain-sampled f. We will first show g-domination for inputs only, ignoring the coinflips, and take care of the coinflips later.

It is worth comparing the following proposition, about g-domination of inputs, to the aforementioned lemma by Dodis and Smith [DS05, Lemma 12], which analyzes the same construction but with longer inputs to h, showing that $(f(y), h')$ is close to $(f(h(x)), h)$. Our proof technique is entirely different and builds on the technique of [HHR06b].

Proposition 2. *For any (not necessarily one-way) $f : Y \to Z$ with K distinct outputs, distribution X with $CP(X) \leq 1/K$, and pairwise-independent hash family $\mathcal{H}_{X,Y}$, the distribution $(f(y), h')$ (where $y \leftarrow Y, h' \leftarrow \mathcal{H}_{X,Y}$) g-dominates $(f(h(x)), h)$ (where $x \leftarrow X, h \leftarrow \mathcal{H}_{X,Y}$), for $g(\delta) = \delta^4/21 - 1/(4K^2)$, or $g(\delta) = \delta^2/2$ if f is regular.*

Proof: We need show that for any $S \subseteq Z \times \mathcal{H}_{X,Y}$,

$$\Pr_{(x,h) \leftarrow X \times \mathcal{H}_{X,Y}}[(f_h(x), h) \in S] \geq \delta \Rightarrow \Pr_{(y,h') \leftarrow Y \times \mathcal{H}_{X,Y}}[(f(y), h') \in S] \geq \frac{\delta^4}{21} - \frac{1}{4K^2}$$

(replace the right-hand-side with $\delta^2/2$ if f is regular).

First we give a one-paragraph outline of the proof of this proposition. Call the points in S *good*. Let $(y,h) \in Y \times \mathcal{H}_{X,Y}$ be called *good* if and only if $(f(y),h)$ is good. We will divide the space Y of inputs to f into K equal-size chunks, producing a set of chunks called C. Call $(c,h) \in C \times \mathcal{H}_{X,Y}$ *good* if $\exists y \in c$ such that (y,h) is good (i.e., a chunk is good if contains a preimage of a good point in Z). We will show, simply using properties of $\mathcal{H}_{X,Y}$, that the fraction of good chunks (under the uniform distribution) is at least $\delta^2/2.125$. This will imply that A works on some portion of sufficiently many chunks. Then, using the fact that f has only K outputs, we will show that A works on a sufficiently large portion of most of these chunks. The actual proof is in in the full version of this paper [DHR07]. □

M^A succeeds whenever A succeeds; in turn, the success or failure of A depends on the point (z,h') chosen, and on the coin flips of A. Let Φ, with probability distribution D, be the space of all coin flips of A. Let $\Pi = Z \times \mathcal{H}_{X,Y}$, let B be the distribution on Π obtained by choosing $x \leftarrow X, h \in \mathcal{H}_{X,Y}$, and $z = f_h(x)$, and let C be the distribution on Π obtained by choosing a uniform $y \in Y$, $h' \in \mathcal{H}_{X,Y}$, and $z = f(y)$. Applying Lemma 1 below to the event E that that A succeeds (here $g(\delta) = \delta^4/21 - 1/4K^2$, or $\delta^2/2$ in the case of regular functions), we obtain the desired statement. □

3.1 The Case of Many Outputs

Theorem 1 can be used to reduce the number of secret input bits to a one-way function provided the function has a sufficiently small output range. As we show in this section, the same technique is useful even if the function has large output range, as long as an appreciable fraction of the inputs falls into a rather small subset of the output range. Namely, suppose there is a set of outputs O_H of size 2^k such that $\Pr_{y \in \{0,1\}^n}[f(y) \in O_H] \geq p_H$. If $k < \sqrt{p_H n}$, then it is possible to reduce the number of secret input bits from n to k^2/p_H, as follows.

Let X be a distribution of collision probability $1/2^k$, and $\mathcal{H}_{X,Y}$ and $f_h(x)$ as above. In the full version [DHR07], we show that $f_h(x)$ is a collection of *weak* one-way functions, i.e., is not invertible with probability appreciably more than $1 - p_H$.

We can then use the standard hardness amplification technique of Yao [Yao82] in order to convert the weak one-way function collection into a strong one. The technique simply concatenates many independent copies of the weak one-way function. The number of repetitions needed to reduce the easily invertible fraction of inputs to (negligibly more than) $1/2^k$ from $1-p_H$ is k/p_H (thus requiring k^2/p_H secret bits) This gives the following result, whose proof is similar to the proof of Theorem 1 and is outlined in the full version of this paper.

Theorem 2. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one-way function and suppose for every n there exists a set $O_H(n)$ of size $k(n)$ such that $\Pr_{y\in\{0,1\}^n}[f(y) \in O_H(n)] \geq p_H(n)$. For every $n \in \mathbb{N}$ let $\mathcal{H}_{k,n}$ be a family of pairwise-independent functions from k bits to n bits. Denote $\ell = k/p_H$ and define $\overline{f_{\overline{h}}}(x_1,\ldots,x_\ell) \stackrel{def}{=} (f_{h_1}(x_1),\ldots,f_{h_\ell}(x_\ell))$ for $\overline{h} = (h_1,\ldots,h_\ell) \in \mathcal{H}^\ell_{k,n}$ and $x_1,\ldots,x_\ell \in \{0,1\}^k$. Then $\{\overline{f_{\overline{h}}}\}_{\overline{h}\in\mathcal{H}^\ell_{k,n}}$ is a public-coin collection of one-way functions.*

4 Pseudorandom Generator Collection from any Known Regular OWF

In this section we show a construction of a pseudorandom generator collection from any regular one-way function. Unlike in the randomized iterate constructions of [GKL93,HHR06b], here the underlying function f has *known* (i.e. efficiently computable) regularity. We use this knowledge to get a PRG collection with particularly short secret input and little security loss.

Namely, suppose f is a regular OWF with output entropy $k(n)$, and that $(t(n),\epsilon(n))$ is the security of f. On secret seed of length $s_S(n) = k(n)$, our PRG collection attains the security of $(\mathsf{poly}(n) + t(n), \mathsf{poly}(\epsilon(n)))$ (Theorem 3). For example, if $k(n) = n^{1/3}$, then we get security comparable to $(t(n),\epsilon(n))$ using only $n^{1/3}$ secret bits. And since, for sufficiently small k, the public index of our PRG collection is of linear size $\mathcal{O}(n)$, one can also view it as a PRG, rather than collection, with good security preservation: on seed length $\mathcal{O}(n)$ it attains security $(\mathsf{poly}(n) + t(n), \mathsf{poly}(\epsilon(n)))$.

Our construction in fact requires a somewhat weaker condition on f than known regularity: f still must be regular, but it is sufficient to have an efficiently computable upper bound $k(n)$ on the output entropy of f. Note that a more accurate bound leads to greater savings in the number of secret seed bits.

Theorem 3. *Let f be a regular one-way function with security $(t(n),\epsilon(n))$ and output entropy at most $k(n)$ (for k computable in time polynomial in n). Then there is a public-coin PRG collection G, which is $(\mathsf{poly}(n) + t(n), \mathsf{poly}(\epsilon(n)))$-indistinguishable on secret seeds of length $s_S(n) = k(n)$ and public seeds of length $s_P(n) = 2n + \mathcal{O}(k(n)\log k(n))$. (In particular $s_P(n) = \mathcal{O}(n)$ if $k = \mathcal{O}(n/\log n)$.)*

Before the actual construction we present the basic tool of the randomized iterate [GKL93,HHR06b]. We define it slightly differently than [GKL93,HHR06b]: theirs outputs a value in $Im(f)$, and ours outputs a hash function image.

Definition 4 (The m^{th} Randomized Iterate of f). *Let $f : \{0,1\}^k \to \{0,1\}^\ell$ and let $\mathcal{H}$ be a family of functions from $\{0,1\}^\ell$ to $\{0,1\}^k$. For input $x \in \{0,1\}^k$ and $\overline{h} = (h^1,\ldots,h^t) \in \mathcal{H}^t$ define the m^{th}* Randomized Iterate *$f^m : \{0,1\}^k \times \mathcal{H}^t \to Im(f)$ for every $m \in [t]$ recursively as:*

$$f^m(x,\overline{h}) = h^m(f(f^{m-1}(x,\overline{h})))$$

where $f^0(x,\overline{h}) = x$.

We first show a construction with public seed length $2n + \mathcal{O}(k^2)$ and then describe how it may be reduced to as low as $2n + \mathcal{O}(k \log k)$, following the same technique as in the HHR construction.

Construction 1. *The generator takes the following as inputs:*
1. *A secret random $x \in \{0,1\}^k$*
2. *A (public) description of one hash function $\hat{h}$ from a family $\mathcal{H}_{k,n}$ of pairwise independent hash functions from k bits to n bits (requires $2n$ bits).*
3. *(Public) descriptions of k hash functions $\overline{h} = (h^1, \ldots, h^k)$ from a family $\mathcal{H}_{\ell,k}$ of* 2^{-3k}-almost *pairwise independent hash functions from ℓ bits to k bits (requires $\mathcal{O}(k)$ bits each).*
4. *A (public) random string $r \in \{0,1\}^k$ for the Goldreich-Levin [GL89] hardcore bit b_r (requires k bits).*

The generator is defined as follows:

$$G_{\hat{h},\overline{h},r}(x) = b_r(x), b_r(f^1_{\hat{h}}(x,\overline{h})), \ldots, b_r(f^k_{\hat{h}}(x,\overline{h}))\,,$$

where $f^i_{\hat{h}}$ denotes the i^{th} randomized iterate of the function $f_{\hat{h}} = \hat{h} \circ f$ (see Figure 1).

Theorem 4. *Suppose f is regular one-way with output entropy at most $k(n)$ and security $(t(n), \epsilon(n))$. Then G in Construction 1 is a public-coin pseudorandom generator collection. It is $(\mathsf{poly}(n) + t(n), \mathsf{poly}(\epsilon(n)))$-indistinguishable on secret seeds of length $s_S(n) = k(n)$ (and public seeds of length $s_P(n) = 2n + \mathcal{O}(k^2)$). (In particular, $s_P(n) = \mathcal{O}(n)$ if $k(n) = \mathcal{O}(\sqrt{n})$).*

Proof: G takes k bits and outputs $k+1$ bits. Thus it is expanding. We must now prove that it is indistinguishable. It is tempting to first fix $\hat{h}$ and since by Theorem 1 $f_{\hat{h}}$ is a one-way function, simply plug $f_{\hat{h}}$ in the HHR construction. However, the HHR construction relies heavily on the fact that the underlying function is regular or at least very close to regular. The function $f_{\hat{h}}$ on the other hand is *not* guaranteed to be regular once $\hat{h}$ is fixed, even if f is regular to begin with. If $\hat{h}$ were from a k-wise independent family (rather than a pairwise independent one) then one can prove that with overwhelming probability $f_{\hat{h}}$ is close to regular. This is not the case with pairwise independent $\hat{h}$ and on the contrary, it is likely that with noticeable probability $f_{\hat{h}}$ will deviate too much from a regular function. Our proof follows the basic structure of the proof of the HHR construction, so we give a sketch, detailing the parts which differ from [HHR06b].

As in the previous iterative constructions (such as [BM82,Yao82,Lev87], [GKL93,HHR06b]), the key to the proof is the unpredictability of the sequence

$$\left(f^k_{\hat{h}}(x,\overline{h}), f^{k-1}_{\hat{h}}(x,\overline{h}), \ldots, f^1_{\hat{h}}(x,\overline{h}), x\right),$$

even for an adversary who is given $(\overline{h}, \hat{h})$. Once this is shown (Lemma 3), it follows from the stronger Goldreich-Levin theorem [Lev93], that the output of the PRG is next-bit unpredictable with essentially the same security. Next-bit unpredictability is equivalent to indistinguishability with a security loss $1/k^{\mathcal{O}(1)}$ (see [Gol01], Theorem 3.3.7). Thus the output of G is indeed pseudorandom, with security essentially the same as of the above sequence. We now turn to the proof of unpredictability.

Let $\mathsf{Supp}(n) = \mathcal{H}^k_{\ell,k} \times \mathcal{H}_{k,n} \times \{0,1\}^k$, and call an element of Supp an *instance*. Let $\Phi = \{0,1\}^{\mathbb{N}}$ denote the set of all coin toss sequences. We say that an algorithm A *inverts i-th iteration (on random coins ω and instance $(\overline{h}, \hat{h}, f^i_{\hat{h}}(x, \overline{h}))$)* if

$$A(\omega, \overline{h}, \hat{h}, f^i_{\hat{h}}(x, \overline{h})) = f^{i-1}_{\hat{h}}(x, \overline{h}).$$

Let $D(n)$ be the distribution of instances produced by the generator, i.e. $(\overline{h}, \hat{h}, f^i_{\hat{h}}(x, \overline{h}))$ for uniform $(\overline{h}, \hat{h}, x)$. Let $Z(n)$ be the uniform distribution of instances, i.e. uniform $(\overline{h}, \hat{h}, z)$.

Lemma 3. *Let A be an algorithm with running time $\leq t(n)$. Suppose that*

$$\Pr[A \text{ inverts } i\text{-th iteration on } (\omega, \overline{h}, \hat{h}, f^i_{\hat{h}}(x, \overline{h}))] \geq \epsilon(n),$$

where ω is uniform and $(\overline{h}, \hat{h}, f^i_{\hat{h}}(x, \overline{h}))$ is distributed according to $D(n)$. Then there is an algorithm B which runs in time $\leq \mathsf{poly}(n) + t(n)$ and inverts $f(x)$ with probability $\geq \epsilon^{2.5}(n)/(16(k+1))$ (for $|x| = n$).

Proof: On input y, the algorithm B generates random $(\overline{h}, \hat{h})$, sets $u \leftarrow A(\overline{h}, \hat{h}, h^i(y))$, and outputs $\hat{h}(u)$.

Fix some n and then we can omit it from the notation. B chooses the hash functions independently of y, i.e. it produces instances distributed according to Z. However, A is guaranteed to invert with probability ϵ on a different distribution D. The bulk of the proof is devoted to proving that A inverts with comparable probability $\approx \epsilon^2$ also on distribution Z. The basic idea of the proof is similar to [HHR06b]: we show that collision probabilities of Z and D are closely related $CP(Z) \geq \mathcal{O}(k) \cdot CP(D)$, and from that we conclude that event probabilities are closely related as well $\Pr_Z[S] \geq (\Pr_D[S])^2/\mathcal{O}(k)$. In particular, the inversion event happens with probability $\epsilon^2/\mathcal{O}(k)$ under Z. The actual proof is more involved than this simple outline, the main complications being: 1. there is a single expanding hash function $\hat{h}$ which is used in every iteration, so the technique of [HHR06b] is not directly applicable, 2. contracting hash functions h^i cause collisions, so an inverse of i-th iteration may be unrelated to y. The details of the proof are given in the full version of this paper [DHR07]. □

Reducing the public seed length. To reduce the public seed length of the above construction from $2n + \mathcal{O}(k^2)$ to $2n + \mathcal{O}(k \log k)$, we follow exactly the same derandomization technique as in the HHR construction. The idea is to not use independent choices of hash functions for $\overline{h} = (h_1, \ldots, h_k)$ but rather choose functions that are correlated yet satisfy the proof of the previous section. The central observation is that the collision probability of a randomized iterate can be computed by a *bounded space* program. More precisely, there is a simple bounded space branching program such that its input tape consists of the choice of $\overline{h}$ and its acceptance probability is precisely the collision probability of $f_{\hat{h}}^{k}$ (the probability is over inputs $x, \overline{h}$) for every fixed $\hat{h}$. Thus replacing the hash functions in the input tape by the output of a generator that fools bounded space programs (such as the generators of [Nis92,INW94]) changes the collision probability only by a small additive error. This is sufficient to make the proof of the previous section go through. Loosely speaking, the bounded space program takes two initial inputs x_1 and x_2.[9] At the first step the program reads the randomizing hash function h^1 and computes $f_{\hat{h}}^{1}(x_1, h^1)$ and $f_{\hat{h}}^{1}(x_2, h^1)$ and stores only these two intermediate values (not storing x_1 and x_2). At each iteration the program reads a new randomizing hash and computes the next randomized iterate of the two values, while not storing the previous one. At the end the program simply compares the two values and outputs 1 only if they are the same value. An accurate account of such a program, bounded space generators and the revisions needed in the proof appears in [HHR06b].

Construction 2. *The generator takes the following as inputs:*

1. *A secret random* $x \in \{0,1\}^k$
2. *Description of one hash function* $\hat{h}$ *from a family* $\mathcal{H}_{k,n}$ *of pairwise independent hash functions from* k *bits to* n *bits (requires* $2n$ *bits).*
3. *Seed* $s \in \{0,1\}^{\mathcal{O}(k \log k)}$ *to a bounded space generator BSG with space bound* $2k$ *and error* 2^{-k}*. The output* $BSG(s) = (h^1, \ldots, h^k)$ *of the generator consists of the descriptions of* k *hash functions from a family* $\mathcal{H}_{\ell,k}$ *of* almost *pairwise independent hash functions from* ℓ *bits to* k *bits.*
4. *A random string* $r \in \{0,1\}^k$ *for the Goldreich-Levin hardcore bit* b_r *(requires* k *bits).*

The generator is defined as follows:

$$G'(x, \hat{h}, s, r) = b_r(x), b_r(f_{\hat{h}}^{1}(x, BSG(s))), \ldots, b_r(f_{\hat{h}}^{k}(x, BSG(s))), \hat{h}, s, r$$

Where $f_{\hat{h}}^{i}$ *denotes the* i^{th} *randomized iterate of the function* $f_{\hat{h}} = \hat{h} \circ f$*.*

The seed length of the aforementioned generators is $\mathcal{O}(\log |\mathcal{H}_{\ell,k}| \cdot \log k)$ (which equals $\mathcal{O}(k \log k)$ with our choice of parameters) and thus the overall construction takes seed length $2n + \mathcal{O}(k \log k)$.

On using secret seeds from non-uniform distributions. A simple modification makes our PRG secure even when used with secret seed drawn from any distribution X as long as $CP(X) \leq 2^{-k}$. The modification can be applied to either Construction 2 or Construction 1. The public seed then increases by only $\mathcal{O}(k)$ bits, therefore it remains unchanged asymptotically. Please see Appendix B for a brief description of the modification.

[9] The program actually computes the collision probability for one fixed pair of inputs x_1, x_2. The actual collision probability is the average over all possible input fixings. But since the generator fools each program separately, it will also fool the average.

5 Black-Box Separations

As discussed in the introduction, it is natural to ask under which conditions one can reduce the input length to a one-way function below its "native" length n. More abstractly, we want to know: *Is there a generic way of securely using a OWF on n-bit inputs, if we are given only $\ell < n$ random bits? How small can ℓ be?*

We formalize these questions using circuits, where it is easy to talk about security on fixed-length input. (It is possible to formulate them in the uniform context, but they become too cumbersome.) We then give some indications that improving upon our results requires non-black-box reductions. Roughly, by "no black-box reduction of P to Q" we mean that the security proof *"if Q is secure then P is too"* is necessarily non-black-box (the construction of P from Q, however, may be black-box). Before elaborating, let us informally summarize the optimality results:

1. For any $l < n$, there is no black-box reduction of l-bit input OWF to regular n-bit-input OWF (and, as a corollary, no black-box reduction to either OWF of known hardness, or arbitrary OWF).
2. For any $l < n - \log \alpha$, there is no black-box reduction of l-bit input one-way-collection to α-regular n-bit-input OWF (and, as a corollary, no black-box reduction to either OWF of known hardness $< 2^n/\alpha$, or arbitrary OWF).
3. For any $s < n$ and $l < n - s$, there is no black-box reduction of l-bit input one-way-collection to an n-bit input OWF of hardness at most s.

5.1 Formal Statements

Let $\mathcal{F}^n$ denote the set of all $f : \{0,1\}^n \to \{0,1\}^n$. Let $\nu(n)$ denote a negligible function (one decaying faster than any inverse polynomial). Note that $1/\nu(n)$ is then a superpolynomial function.

Circuits, oracle circuits. Let $|A|$ denote the size of the circuit A. For an oracle circuit A and a function $f : \{0,1\}^n \to \{0,1\}^m$, A^f denotes the oracle circuit in which each oracle gate with input x outputs $f(x)$. If $\mathcal{G} = \{g_i\}_{i \in \{0,1\}^n}$ is a collection of functions $g_i : \{0,1\}^n \to \{0,1\}^m$ then $A^{\mathcal{G}}$ denotes the oracle circuit in which each oracle gate, on input (i, x) outputs $g_i(x)$.

Inverter. A circuit $A : \{0,1\}^l \to \{0,1\}^n$ is a *p-inverter for $f : \{0,1\}^n \to \{0,1\}^l$* if $\Pr_{x \in \{0,1\}^n}[A(f(x)) \in f^{-1}(f(x))] \geq p$. A 1-inverter is called *perfect*.

Black-box reduction. Let $\mathcal{F} \subseteq \mathcal{F}^n$. A pair of circuits (R, g) is an *(l, p)-reduction to $\mathcal{F}$* if for any $f \in \mathcal{F}$:
1. g has l input wires.
2. If V is a perfect inverter for g^f, then $R^{V,f}$ is a p-inverter for f.

A sequence (R_n, g_n) of (l_n, p_n)-reductions to $\mathcal{H}_n \subseteq \mathcal{F}^n$ is called *$d(n)$-saving* if: 1. $(|R_n| + |g_n|)/p_n$ is polynomial in n, 2. $n - l_n = d(n)$.

Let $\mathcal{F}^{n,\alpha}_{\text{REG}} \subseteq \mathcal{F}_{\text{ALL}}$ denote its subset of all α-regular functions. Let $\mathcal{F}^{n,s}_{\text{LOW}} \subseteq \mathcal{F}_{\text{ALL}}$ denote the subset of all *at most s-hard permutations* (permutations which have a 1/2-inverter of size $< s$).

Black-box collection reduction. A pair of circuits (R, g) is a (l, m, p)-*collection-reduction to* $\mathcal{F}$ if:
1. For any $f \in \mathcal{F}$, and any $(i, x) \in \{0,1\}^m \times \{0,1\}^l$, $g^f(i, x)$ is of the form (i, y).
2. If V is a perfect inverter for g^f, then $R^{V,f}$ is a p-inverter for f.
A sequence (R_n, g_n) of (l_n, m_n, p_n)-reductions to $\mathcal{H}_n \subseteq \mathcal{F}^n$ is called $d(n)$-*saving* if: 1. $m_n(|R_n| + |g_n|)/p_n$ is polynomial in n, 2. $n - l_n = d(n)$.

The following two technical lemmas are at the heart of our separations. Their proofs can be found in the full version of the present article [DHR07].

Lemma 4. *Let $l = n - c$ and $p \geq 2^{-c/2+1}$. If (R, g) is an (l, p)-reduction to $\mathcal{F}^{n,\alpha}_{\text{REG}}$ then $|g| > 2^{c/2}$ or $|R| > p2^{n-a+3}$.*
Lemma 5. *Let $l = n - \log\alpha - d$. If (R, g) is a (l, m, p)-collection-reduction from $\mathcal{F}^{n,\alpha}_{\text{REG}}$, then $|R| > p2^{d-4}/m$.*

Theorem 5. *Let $\alpha(n) = \nu(n)2^n$. There is no $\omega(\log n)$-saving reduction to $\mathcal{F}^{n,\alpha(n)}_{\text{REG}}$.*
Proof: Suppose to the contrary that (R, g) is a $\omega(\log n)$-saving reduction to $\mathcal{F}^{n,\alpha(n)}_{\text{REG}}$. Consider some particular f, and let D be the set of all possible oracle queries that g^f can ask, on any input. Then $|S| \leq |g|2^l$, because on each of the 2^l distinct inputs, g asks at most $|g|$ queries. The basic idea of the lower bound proof is that, for $l < n - \omega(\log n)$, and polynomial-sized g, S occupies a negligible fraction of f's domain. But the one-way f can be easy on S, and g^f is then not one-way.

Formally: apply Lemma 4 to (R_n, g_n) with $c = \omega(\log(n))$ and $p = p(n)$. Since $2^{c/2} = 1/\nu(n)$ and $2^{n-\log\alpha(n)} = 2^n/\alpha(n) = 1/\nu(n)$ we conclude that $|R_n| + |g_n|$ is superpolynomial. $\square$

Theorem 6. *Let $\alpha(n) = \nu(n)2^n$. There is no $(\omega(\log(n)) + \log\alpha(n))$-saving collection-reduction to $\mathcal{F}^{n,\alpha(n)}_{\text{REG}}$.*
Proof: Suppose that (R, g) is the collection-reduction which contradicts the theorem statement, and let l be the number of g's input wires. We show that it is possible to build from (R, g) a circuit B of size about 2^l which inverts any $f \in \mathcal{F}^{n,\alpha(n)}_{\text{REG}}$. To do this, note that R^V inverts any $f \in \mathcal{F}^{n,\alpha(n)}_{\text{REG}}$ as long as it is given an inverter V for g^f. But V can be implemented as a circuit of size $2^l/\nu(n)$. Therefore R^V can be implemented (without any oracle) as a circuit of size about $|R|2^l/\nu(n)$. But this is too small to invert any function $f \in \mathcal{F}^{n,\alpha(n)}_{\text{REG}}$. The formal argument follows.

If $|g_n|$ is superpolynomial we are done. Else suppose $|g_n|$ grows polynomially fast. Apply Lemma 5 with $d = \omega(\log(n))$ (and $\log|I| < |g_n|$ since $\log|I|$ is at most the number of input wires of g_n), to get that $|R_n| > p(n)2^{\omega(\log(n))}/|g_n|$ which is superpolynomial. $\square$

Theorem 7. *Let $s(n) < n$. There is no $(\omega(\log(n)) + s(n))$-saving collection-reduction to $\mathcal{F}^{n,s(n)}_{\text{LOW}}$.*
Proof Sketch: Let f be a random permutation and let $h(p, y)$ output $x = f^{-1}(y)$ if p is an s-bit prefix of x. This ensures that f is "exactly" s-hard. For any construction g^f with input size $l = n - s - d$ (and description of family index m polynomial in n), we can show an oracle V which inverts it, but such that V

does not significantly reduce the hardness of f. Some minor modifications are needed to ensure that (f, h) is a permutation.

V, on input (i, y), simply outputs a random x for which $g_i^f(x) = y$. To see that f is still s-hard, suppose there is a poly-size inverter $A^{(f,h),V}$ for f. From it one can build a circuit B^f which perfectly simulates $A^{(f,h),V}$. Each call to h can be simulated using 2^{n-s} queries to f, and each call to V using $\approx 2^l$ queries to f. So B^f calls f about $|B|(2^l + 2^{n-s}) < |B|(2 \cdot 2^{n-s})$ times. With this many queries, the probability of inverting f cannot exceed $\approx 2^{-s}$, so f is still s-hard. $\square$

Corollary 1 (To Theorem 5). *There is no $\omega(\log n)$-saving reduction to $\mathcal{F}^n$.*

Corollary 2 (To Theorem 6). *There is no $(\omega(\log n) + \log \alpha(n))$-saving reduction to $\mathcal{F}^n$.*

Acknowledgements

We thank anonymous referees for their many helpful suggestions. Research of N.D. and L.R. was supported, in part, by IPAM at UCLA, and by US NSF grants CCF-0515100, CNS-0546614 and CNS-0202067. Research of D.H. was supported by a Lady Davis Fellowship and by a grant from the Israeli Science Foundation.

References

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117, 1982.

[CW77] I. Carter and M. Wegman. Universal classes of hash functions. In *9th ACM Symposium on Theory of Computing*, pages 106–112, 1977.

[DHR07] N. Dedić, D. Harnik, and L. Reyzin. Saving private randomness in one-way functions and pseudorandom generators. Technical Report 2007/458, Cryptology e-print archive, `http://eprint.iacr.org`, 2007.

[DS05] Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *37th STOC*, pages 654–663, 2005.

[GIL+90] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan, and D. Zuckerman. Security preserving amplification of hardness. In *31st IEEE Symposium on Foundations of Computer Science*, pages 318–326, 1990.

[GKL93] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. *SIAM Journal of Computing*, 22(6):1163–1175, 1993.

[GL89] O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[Gol01] O. Goldreich. **Foundations of Cryptography**. Cambridge University Press, 2001.

[HHR06a] I. Haitner, D. Harnik, and O. Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In *33rd ICALP, 2006, Pt. II, LNCS*, volume 4052, pages 228–239. Springer, 2006.

[HHR06b] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. In *CRYPTO '06, LNCS*, volume 4117, pages 22–40. Springer, 2006.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 29(4):1364–1396, 1999.

[HL92] A. Herzberg and M. Luby. Pubic randomness in cryptography. In *CRYPTO '92, LNCS*, volume 740, pages 421–432. Springer, 1992.

[Hol06] T. Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In *TCC '06*, pages 443–461, 2006.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *26th STOC*, pages 356–364, 1994.

[Lev86] Leonid A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Lev87] L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.

[Lev93] Leonid A. Levin. Randomness and nondeterminism. *The Journal of Symbolic Logic*, 58(3):1102–1103, 1993.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[WC81] M. Wegman and J. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 1981.

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

A Further Shortening the PRG Seed

In our pseudorandom generator, the output of the last hash function has, intuitively, almost k bits of entropy. It entropy can be converted to pseudorandomness using an extractor with a public seed (of length k). To get this pseudorandomness to be, e.g., $n^{\log^c n}$-close to uniform for some c, one will lose $\Theta(\log^{c+1} n)$ bits. If we take this approach, then the we need to run the randomized iterate construction not k times, but $\Theta(\log^{c+1} n)$ times; thus, we need the space-bounded generator to produce not k, but $\Theta(\log^{c+1} n)$ hash functions, which can be done in space $\mathcal{O}(k \log(\log^{c+1} n)) = \mathcal{O}(k \log\log k)$. The result is a PRG with seed length $2n+\mathcal{O}(k \log\log k)$ of which only k bits needs to be secret, but security reduced to the bare minimum $n^{\log^c n}$.

B On using secret seeds from non-uniform distributions

Suppose X is a distribution with the only guarantee that $CP(X) \leq 2^{-k}$. We outline the modification which makes our PRG secure even when its seed x is drawn from X. Namely, suppose that the support of X is $\{0,1\}^m$, and let $\mathcal{H}_{m,k}$ be a family of 2^{-3k}-almost pairwise independent hash functions from $\{0,1\}^m$ to $\{0,1\}^k$. The modified generator first pre-processes its seed x by applying a random $h^0 \in \mathcal{H}_{m,k}$ to x, and then uses our PRG (either of Construction 2 or of Construction 1) on secret seed $h^0(x)$. The hash function h^0 need not be secret. As explained in Section 2, h^0 can be specified using $\mathcal{O}(k)$ bits, therefore the public seed length remains essentially unchanged ($\mathcal{O}(k \log k)$ for Construction 2, or $\mathcal{O}(k^2)$ for Construction 1).