On the Power of the Randomized Iterate*

Iftach Haitner† Danny Harnik ‡ Omer Reingold§

January 16, 2013

Abstract

We consider two of the most fundamental theorems in Cryptography. The first, due to Håstad, Impagliazzo, Levin, and Luby (STOC '89, STOC '90, SIAM J. on Computing '99), is that pseudorandom generators can be constructed from any one-way function. The second, due to Yao [34] (FOCS '82), states that the existence of weak one-way functions implies the existence of full-fledged one-way functions. These powerful plausibility results shape our understanding of hardness and randomness in Cryptography, but unfortunately their proofs are not as tight (i.e., security preserving) as one may desire.

This work revisits a technique that we call the *randomized iterate*, introduced by Goldreich et al. [11] (SIAM J. on Computing '93). This technique was used by Goldreich et al. to give a construction of pseudorandom generators from *regular* one-way functions. We simplify and strengthen this technique in order to obtain a similar construction, where the seed length of the resulting generators is as short as $\Theta(n \log n)$ (rather than $\Theta(n^3)$ achieved by Goldreich et al.). Our technique has the potential of implying seed-length $\Theta(n)$, and the only bottleneck for such a result are the parameters of current generators against bounded-space computations. We give a construction with similar parameters for security amplification of regular one-way functions. This improves upon the construction of Goldreich, Impagliazzo, Levin, Venkatesan, and Zuckerman (FOCS '90) in that the construction does not need to "know" the regularity parameter of the functions (in terms of security, the two reductions are incomparable). In addition, we use the randomized iterate to show a construction of a pseudorandom generator based on an exponentially-hard one-way function that has seed length of only $\Theta(n^2)$. This improves a recent result of Holenstein [19] (TCC '06) that shows a construction with seed length $\Theta(n^5)$ based on such one-way functions.

*This paper combines preliminary versions that appeared as [12] and [13].

†School of Computer Science, Tel Aviv University. E-mail: iftachh@cs.tau.ac.il. Research conducted while at Weizmann Institute of Science. Research supported by grant no. 1300/05 from the Israel Science Foundation.

‡IBM Haifa Research Labs. danny.harnik@gmail.com. Research conducted while at the Weizmann Institute and the Technion.

§Microsoft Research, Silicon Valley Campus, and Weizmann Institute of Science. E-mail:omreing@microsoft.com. Supported by grant no. 1300/05 from the Israel Science Foundation.

Finally, we show that the randomized iterate may even be useful in the general context of Håstad, Impagliazzo, Levin, and Luby. In particular, we use the randomized iterate to replace the basic building block of the Håstad et al. construction. Interestingly, this modification improves efficiency by an $\Theta(n^2)$ factor and reduces the seed length to $\Theta(n^7)$ (which also implies improvement in the security of the construction).

Keywords: cryptography, pseudorandom generator, one-way functions, hardness amplification.

Contents

1 Introduction

In this work we address two fundamental problems in cryptography: (1) constructing pseudorandom generators from one-way functions and (2) transforming weak one-way functions into strong one-way functions. The common thread linking the two problems in our discussion is the technique we use. This technique, which we call here the *randomized iterate*, was introduced by Goldreich et al. [11] in the context of constructing pseudorandom generators from regular one-way functions. We revisit this method, simplify existing proofs and utilize our new perspective to achieve significantly better parameters for security and efficiency. We demonstrate that the randomized iterate is also applicable to the construction of pseudorandom generators *from any one-way function*. Specifically, we revisit the seminal paper of Håstad, Impagliazzo, Levin, and Luby and show that the randomized iterate can help improve the parameters in this context. We also give significant improvements to the construction of pseudorandom generators from one-way functions that are exponentially hard to invert. Finally, we use the randomized iterate both to simplify and to strengthen previous results regarding efficient hardness amplification of regular one-way functions.

We start by introducing the randomized iterate in the context of pseudorandom generators, and postpone the discussion on amplifying weak to strong one-way functions to Section 1.3.

1.1 Pseudorandom Generators and the Randomized Iterate

Pseudorandom generators, first introduced by Blum and Micali [2], and stated in its current, equivalent form, by Yao [34], are one of the cornerstones of cryptography. Informally, a pseudorandom generator is a polynomial-time computable function G that stretches a short random string x into a long string $G(x)$ that "looks" random to any efficient (i.e., polynomial-time) algorithm. Hence, there is no efficient algorithm that can distinguish between $G(x)$ and a truly random string of length $|G(x)|$ with more than a negligible probability. Originally introduced in order to convert a small amount of randomness into a much larger number of effectively random bits, pseudorandom generators have since proved to be valuable components for various cryptographic applications such as bit commitments [28], pseudorandom functions [9] and pseudorandom permutations [26], to name a few.

1.1.1 Previous Constructions

The first construction of a pseudorandom generator given in Blum and Micali [2], was based on a particular one-way function and was later generalized in Yao [34] into a construction of a pseudorandom generator based on any one-way permutation (hereafter, the BMY generator). The BMY generator works by iteratively applying the one-way permutation on its own output. More precisely, for a given function f and input x, define the k'th iterate recursively as $f^k(x) = f(f^{k-1}(x))$, where $f^1(x) = f(x)$. To complete the construction, one needs to take a hardcore-bit at each iteration. If we denote by $b(x)$ the hardcore-bit of x (take for

instance the Goldreich and Levin [8] predicate), then the BMY generator on seed x outputs the hardcore-bits $b(f^1(x)), \ldots, b(f^{n+1}(x))$.[1]

The natural question arising from the BMY generator was whether one-way permutations are actually necessary for pseudorandom generators, or can one do with a more relaxed notion. Specifically, is any one-way function sufficient for pseudorandom generators? Levin [25] observed that the BMY construction works for any *one-way function on its iterates*, that is, a one-way function that remains one-way when applied sequentially on its own outputs. A general one-way function, however, does not have this property since the output of f may have very little randomness in it, and a second application of f may be easy to invert. A partial solution was suggested by Goldreich et al. [11], who showed a construction of a pseudorandom generator based on any regular[2] one-way function (hereafter, the GKL generator). The GKL generator introduced the technique at the core of this work, that we call the *randomized iterate*. Rather than simple iterations, an extra randomization step is added between every two applications of f. More precisely,

Definition 1.1 (the randomized iterate (informal))**.** *For function $f\colon \{0,1\}^n \mapsto \{0,1\}^n$, input $x \in \{0,1\}^n$ and a vector $\overline{h} = (h_1, \ldots, h_{k-1})$ of length-preserving hash functions over $\{0,1\}^n$, recursively define the k'th randomized iterate by:*

$$f^k(x, \overline{h}) = f(h_{k-1}(f^{k-1}(x, (h_1, \ldots, h_{k-2})))),$$

where $f^1(x) = f(x)$. For $m > k-1$, we let $f^k(x, h_1, \ldots, h_m) = f^k(x, h_1, \ldots, h_{k-1})$.

The rational is that $\overline{h}_{k-1}(f^{k-1}(x, \overline{h}))$, for a random x and $\overline{h}$, is uniformly distributed in $\{0,1\}^n$, and the challenge is to show that f, when applied to $h_{k-1}(f^{k-1}(x, \overline{h}))$, is hard to invert even when $\overline{h}$ is made public. Once this is shown, the generator is similar in nature to the BMY generator, i.e., it outputs $b(f^1(x, \overline{h})), \ldots, b(f^{n+1}(x, \overline{h})), \overline{h}$.

Finally, Håstad et al. [16] (combining [23, 15]) culminated this line of research by showing a construction of a pseudorandom generator using any one-way function (hereafter, the HILL generator). This result is one of the most fundamental and influential theorems in cryptography. It introduced many new ideas that have since proved useful in other contexts, such as the notion of pseudoentropy, and the implicit use of family of pairwise-independent hash functions as randomness extractors. We mention that HILL departs from GKL in its techniques, taking a significantly different approach.

1.1.2 The Complexity and Security of Previous Constructions

While the HILL generator fully answers the question of the plausibility of a generator based on any one-way function, the construction is highly involved and very inefficient. Other than the evident contrast between the simplicity and elegance of the BMY generator to

[1] We mention that typically the BMY generator is presented as $b(x), b(f^1(x)), \ldots, b(f^n(x))$. For consistency with our results, however, we present it so that the first hardcore bit is taken *after* the first iteration.

[2] A function is regular if every element in its image has the same number of preimages.

the complex construction and proof of the HILL generator, the parameters achieved in the construction are far worse, rendering the construction impractical.

In practice, it is not necessarily sufficient that a reduction translates polynomial security into polynomial security. In order for reductions to be of any practical use, the concrete overhead introduced by the reduction comes into play. There are various factors involved in determining the security of a reduction, and in Section 2.11 we elaborate on the security of cryptographic reductions and the classification of reductions in terms of their security. Here, however, we focus only on one central parameter, which is the length d of the generator's seed compared to the length n of the input to the underlying one-way function. The BMY generator takes a seed of length $\Theta(n)$, the GKL generator takes a seed of length $\Theta(n^3)$ while the HILL construction produces a generator with seed length in the order of $\Theta(n^8)$.[3]

The seed length is of great importance to the security of the resulting generator. While it is not the only parameter, it serves as a lower bound to how good the security may be. For instance, the HILL generator on d bits has security that is at best comparable to the security of the underlying one-way function, but only on $\Theta(\sqrt[8]{d})$ bits. To illustrate the implications of this deterioration in security, consider the following example: suppose that we only trust a one-way function when applied to inputs of at least 100 bits, then the GKL generator can only be trusted when applied to a seed of length of at least one million bits, while the HILL generator can only be trusted on seed lengths of 10^{16} and up (both being highly impractical). Thus, trying to improve the seed length towards a linear one (as it is in the BMY generator) is of great importance in making these constructions practical.

1.1.3 Exponentially Hard One-Way Functions and Improving the Seed Length

The BMY and GKL generators demonstrate that assuming restrictions on the underlying one-way function allows for great improvement of the seed length (or input blowup). The common theme in these restrictions is that they deal with the *structure* of the one-way function. A different approach was recently taken by Holenstein [19], who builds a pseudorandom generator from any one-way function with *exponential hardness*.[4] This approach is different as it discusses *raw hardness* as opposed to structure. The result in [19] is a generalization of the HILL generator that takes into account the parameter stating the hardness of the one-way function. In its extreme case where the hardness is exponential, the pseudorandom generator takes a seed length of $d = \Theta(n^5)$ and has security $2^{\Theta(d^{\frac{1}{5}})}$. The seed length can be reduced to as low as $\Theta(n^4 \log^2 n)$ when the resulting generator is only required to have super-polynomial security (i.e., security of $n^{\log n}$). In its other extreme based on a general one-way function (with super-polynomial hardness), [19] forms a formal proof of the best known seed length for the HILL construction (seed length $\Theta(n^8)$).

[3] The seed length actually proved in [16] is $\Theta(n^{10})$, however it is mentioned that a more careful analysis can get to $\Theta(n^8)$. A formal proof for the $\Theta(n^8)$ seed length construction is given by Holenstein [19].

[4] For some constant $c > 0$, no algorithm of running time at most 2^{cn} inverts the function with probability better than 2^{-cn}.

1.2 Our Results on Pseudorandom Generators

Our improvements to the seed length of pseudorandom generators under the various assumptions are summarized in Figure 1. In the upcoming section we elaborate on each of these constructions and highlight the source of the improvements.

Paper	Type of function	Seed length
[2, 34]	One-way permutation	$\Theta(n)$
[11] This work	Regular one-way function	$\Theta(n^3)$ $\Theta(n \log n)$
[19] This work	One-way function with exponential hardness	$\Theta(n^5)$ $\Theta(n^2)$
This work	Regular one-way function with exponential hardness	$\Theta(n)$
[16, 19] This work	Any one-way function	$\Theta(n^8)$ $\Theta(n^7)$

Figure 1: Summary of results.

1.2.1 Regular One-Way Functions

We give a construction of a pseudorandom generator from any regular one-way function with seed length $\Theta(n \log n)$. We mention that our approach has the potential of reaching a construction with a linear seed, the bottleneck being the efficiency of the currently known bounded-space generators. Our construction follows the randomized iterate method and is achieved in two steps:

- We give a significantly simpler proof that the GKL generator works, allowing the use of a family of hash functions that is pairwise-independent rather than n-wise independent (as used in [11]). This gives a construction with seed length $\Theta(n^2)$ (see Theorem 3.6).
- The new proof allows for the derandomization of the choice of the randomizing hash functions via the *generator against bounded-space adversaries* (for short, bounded-space generator) of Nisan [29], further reducing the seed length to $\Theta(n \log n)$ (see Theorem 3.10).

The proof method. Following is a high-level description of our proof method. For simplicity we focus on the second randomized iteration (i.e., on $f^2(x,h) = f(h(f(x)))$, but the same argument generalizes to the other iterations. The main task at hand is to show that it is hard to find $f^1(x,h) = f(x)$ when given $f^2(x,h)$ and h. This follows by showing that any procedure A for finding $f(x)$ given $(f^2(x,h),h)$ enables to invert the one-way function f (on a random image). Specifically, we show that for a random image $y \in f(U_n)$, if we choose a random and independent hash h' and feed the pair (y,h') to A, then A is likely to

return a value $f(x')$ such that $h'(f(x')) \in f^{-1}(y)$ (and thus we obtain an inverse of y). This is ultimately shown by proving that if A succeeds on the distribution of $(f^2(x,h),h)$, then A is also successful on the distribution of $(f^2(x,h),h')$, where h' is chosen independently of (x,h).

Our proof is inspired by a technique used by Rackoff in his proof of the Leftover Hash Lemma (in [22]). Rackoff proves that a distribution is close to uniform by showing that it has *collision probability* that is very close to that of the uniform distribution.[5] We would have liked to follow this scheme and consider the collision probability of the two aforementioned distributions. In our case, however, the two distributions could actually be very far from each other. Yet, based on our analysis of the collision probabilities, we manage to prove that the probability of any event under the first distribution is *polynomially related* to the probability of the same event under the second distribution. This proof generalizes nicely also to the case of many iterations.

The derandomization using a bounded-space generator, follows directly from the new proof. The point is to introduce a derandomization of the hash functions such that the collision probability of the randomized iterate remains essentially the same. Since the proof centers around the collision probability of $(f^k(x,\overline{h}),\overline{h})$, the proof will hold also for the derandomized version. More precisely, consider the procedure that given inputs x_0, x_1 and $\overline{h} = (h_1,\ldots,h_{k-1})$, outputs '1' if $f^k(x_0,\overline{h})$ equals $f^k(x_1,\overline{h})$ and '0' otherwise. Note that the probability, over a uniform choice of inputs, that the above procedure outputs '1', is exactly the collision probability of $(f^k(x,\overline{h}),\overline{h})$. Also note that the above procedure can run in linear space, since it simply needs to store the two intermediate iterates at each step. Therefore, the probability that the above procedure outputs '1' while replacing $\overline{h}$ with the output of a generator against linear space adversaries, is very close to the collision probability of $(f^k(x,\overline{h}),\overline{h})$. It follows that the collision probability of $(f^k(x,\widetilde{h}),\widetilde{h})$, where $\widetilde{h}$ is now the output of the bounded-space generator is very close to that of $(f^k(x,\overline{h}),\overline{h})$, and the security proof now follows as in the proof when using independent randomizing hash functions. We mention that derandomization of similar spirit was used by Phillips [31], in his efficient amplification of weak one-way permutations (see Section 1.3).

1.2.2 Exponentially Hard One-Way Functions

We give a construction of a pseudorandom generator from *any* exponentially-hard one-way function with seed length $d = \Theta(n^2)$ and security $2^{\Theta(\sqrt{d})}$. If we only require the security of the resulting generator to be super-polynomial, then the construction gives a seed that is only $\Theta(n\log^2 n)$ long. We mention that Holenstein's result applies for *any* one-way function (but is most efficient when the one-way function is exponentially hard). Our construction on the other hand is specialized for one-way functions with exponential hardness, and does not generalize to use significantly weaker one-way functions. More concretely, our construction can use any one-way function with security $2^{\phi(n)}$, *as long as* $\phi(n) \in \Omega(\frac{n}{\log n})$.

[5] The collision probability of a distribution is the probability of getting the same element twice when taking two independent samples from the distribution.

The core technique of our construction is once again the randomized iterate. Trying to apply the randomized iterate to a general one-way function, we encounter the following difficulty: for $k \geq 2$, the k'th randomized iteration of a general one-way function may be easy on a large fraction of the inputs. Our key observation is that the randomized iterate cannot be easy everywhere. Our Lemma 4.1 indicates that for every one-way function f, there exists a set $\mathcal{S}$ of inputs to f^k such that the k'th randomized iteration is hard to invert over inputs taken from this set. Moreover, the density of $\mathcal{S}$ is at least $1/k$. This means that there is some pseudorandomness to be extracted from the k'th randomized iterate; taking a hard-core bit of the k'th randomized iteration gives a bit that with probability $1/k$ looks random (to a computationally bounded observer). Our idea is to collect these bits that contain some pseudoentropy and to then extract from them the pseudorandom output of the generator.

Consider taking t independent copies of the randomized iterate (on t independent inputs) and for each of the t copies taking a hardcore bit from the k'th iteration. This forms a string of t bits, of which t/k are expected to be random looking. Our next step would be to run a *randomness extractor* on this string, to generate $\Theta(t/k)$ pseudorandom bits. The problem, however, is that the total number of pseudorandom bits generated, i.e., $\Theta(\sum_{k=1}^{m} \frac{t}{k})$ where m is the number of iterations, is too low, and in particular insufficient to compensate for the tn bits invested in the random seed.

This problem can be remedied by taking more hardcore bits at each iteration. Specifically, if the one-way function has exponential hardness then a linear number of hardcore bits may be taken at each iteration (Goldreich and Levin [8]). Thus, taking $t = n$ independent copies, the total number of pseudorandom bits generated can be larger than the seed length. The construction gives a seed that is $\Theta(n^2)$ bits long, as each independent copy of the randomized iterate only runs a constant number of iterations.

Remark 1.2 (on randomness extractors and pseudorandomness)**.** *The use of randomness extractors in a computational setting, was initiated in [16]. We give a general "uniform extraction lemma" (Lemma 2.14) for this purpose that is proved using a uniform hardcore Lemma of Holenstein [18]. We mention that a similar proof was given independently in [19].*

1.2.3 Any One-Way Function

The HILL generator takes a totally different path than the GKL generator. The initial step in the HILL construction takes a one-way function f and generates a bit that has significantly more *pseudoentropy* than actual entropy. This gap is then exploited in order to build a full-fledged pseudorandom generator. This initial construction does not use iterations of f at all. We ask whether the technique of randomized iterations can be helpful for the case of any one-way function, and give a positive answer to this question (actually, we are using only the first two iterations). Specifically, this method also improves the efficiency of the overall construction by an $\Theta(n^2)$ factor (ignoring polylog(n) terms) over the original HILL generator and reduces the seed length by a factor of n, which also implies improvement in the security of the construction. All in all, we present (Theorem 5.8) a pseudorandom generator from *any* one-way function with seed length $\Theta(n^7)$.

Our generator replaces the initial step of the HILL generator with a different construction based on the techniques we have developed. We briefly describe the new initial step. Denote the degeneracy of y by $\mathrm{D}_f(y) = \lceil \log |f^{-1}(y)| \rceil$ (this is a measure that divides the images of f to n categories according to their preimage size). Let b denote a hardcore-bit (again we take the Goldreich-Levin hardcore bit [8]). Loosely speaking, we consider the bit $b(f(x))$ when given the value $(f^2(x,h),h)$ and make the following observation: when $\mathrm{D}_f(f(x)) \geq \mathrm{D}_f(f^2(x,h))$, the value $b(f(x))$ is (almost) fully determined by $(f^2(x,h),h)$; as opposed to when $\mathrm{D}_f(f(x)) < \mathrm{D}_f(f^2(x,h))$, where no information about $b(f(x))$ leaks. But in addition, if $\mathrm{D}_f(f(x)) = \mathrm{D}_f(f^2(x,h))$, then $b(f(x))$ is computationally-indistinguishable from uniform (that is, looks uniform to any efficient observer), even though it is actually fully determined. The latter stems from the fact that when $\mathrm{D}_f(f(x)) = \mathrm{D}_f(f^2(x,h))$ the behavior is close to that of a regular function.

As a corollary we get that the bit $b(f(x))$ has entropy of no more than $\frac{1}{2}$ (i.e., the probability of $\mathrm{D}_f(f(x)) < \mathrm{D}_f(f^2(x,h)))$, but has "entropy of at least $\frac{1}{2} + \frac{1}{\Theta(n)}$ in the eyes of any computationally-bounded observer" (i.e., the probability of $\mathrm{D}_f(f(x)) \leq \mathrm{D}_f(f^2(x,h)))$. In other words, $b(f(x))$ has entropy $\frac{1}{2}$ but pseudoentropy of $\frac{1}{2} + \frac{1}{\Theta(n)}$.[6] As in HILL, it is this gap of $\frac{1}{\Theta(n)}$ between the entropy and pseudoentropy that eventually allows the construction of a pseudorandom generator.

Comparing to Håstad et al. [16]. [16] build a pair of function and predicate such that the predicate has entropy p, but pseudoentropy of at least $p + \frac{1}{\Theta(n)}$ (see Appendix A for the description of their pair). Unlike in our construction, however, the entropy threshold p in their construction is unknown (i.e., not efficiently computable). This is a real disadvantage, since knowledge of this threshold is essential for the overall construction. To overcome this, they enumerate all values for p (up to an accuracy of $\Theta(\frac{1}{n})$), run the generator with each of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor n to the seed length of the final generator and an additional factor of n^2 to the number of calls to the underlying function f, and hence our efficiency and security improvements.

Remark 1.3. *[on pseudorandomness in* NC^1*] For the most part, the [16] construction is "depth" preserving. In particular, given two "non-uniform" hints of* $\log n$ *bits each (that specify two different properties of the one-way function*[7]*), the reduction gives generators in* NC^1 *from any one-way function in* NC^1*. Unfortunately, without these hints, the depth of the construction is polynomial (rather than logarithmic). Our construction eliminates the*

[6] It natural to ask why should we consider $b(f(x))$ as the predicate and not simply $b(x)$. Clearly the pseudoentropy of $b(x)$, given $(f^2(x,h),h)$ is at least as large as that of $b(f(x))$ (since $f(x)$ is determined by x). The problem is that the real entropy of $b(x)$ in this case is unknown (and may, in fact, be as high as the pseudoentropy). In other words, when considering $b(f(x))$ rather than $b(x)$, we reduce the conditional entropy to a known bound, while keeping the pseudoentropy larger than this bound.

[7] Consider the random variable induced by applying the one-way function on a uniformly chosen input. One of these hints relates to the entropy of this random variable, where the other hint, p, relates to its variance.

need for one of these hints (we still need to know the entropy of the function) and thus can be viewed as a step towards achieving generators in NC^1 *from any one-way function in* NC^1*. Building pseudorandom generators in* NC^1 *(from one-way functions in* NC^1*) would be highly significant. In particular, since [1] showed that such generators imply pseudorandom generators in* NC^0*.*

1.2.4 The Recent Generator of Haitner, Reingold, and Vadhan

In a very recent result, Haitner, Reingold, and Vadhan [14] presented a new family of pseudorandom generator for "any hardness". In particular, when starting from standard (polynomially secure) one-way functions, their construction achieves seed length $\Theta(n^4)$ (compared with the $\Theta(n^7)$ achieved here). When starting from exponentially-hard one-way functions, their seed length matches the result presented here.[8] An additional advantage of the generators of [14] over this work (and over [16, 19]), is that their generators use non-adaptive calls to the underlying one-way functions. In particular, their results yields that pseudorandom generators in NC^0 can be constructed from one-way functions in NC^1 (see Remark 1.3).

1.3 One-Way Functions – Amplification from Weak to Strong

The existence of one-way functions is essential to almost any task in cryptography (see for example [21]) and also sufficient for numerous cryptographic primitives such as the pseudorandom generators discussed above. In general, for constructions based on one-way functions we use what are called *strong* one-way functions. That is, functions that can only be inverted efficiently with negligible success probability. A more relaxed definition is that of an ε-weak one-way function, where $\varepsilon = \varepsilon(n)$ is a polynomial fraction. This is a function that every efficient algorithm fails to invert on at least a ε fraction of the inputs. This definition is significantly weaker, yet, Yao [34] showed how to convert any weak one-way function into a strong one (see proof in [7]). The new strong one-way function simply consists of many independent copies of the weak function concatenated to each other. The solution of Yao, however, incurs a blow-up factor of $\omega(\log(n)/\varepsilon)$ to the input length of the strong function, which translates to a significant loss in the security (as in the case of pseudorandom generators).

Goldreich et al. [10] pointed out this loss of security problem and gave a solution for one-way permutations that has just a linear blowup in the length of the input. Their solution was also generalized to *known-regular* one-way functions (regular functions whose image size is efficiently computable), where its input length varied according to the required security. The input length is linear when the required security is $2^{O(\sqrt{n})}$, but deteriorates up to $\Theta(n^2)$ when the required security is higher (e.g., security $2^{\Theta(n)}$).[9] Their construction uses a variant of randomized iterates, where the randomization is via one random step on an expander

[8] Their result also generalizes to functions with in between hardness, significantly improving over [19] for every choice of hardness.

[9] Loosely speaking, one can think of the security as the probability of finding an inverse to a random image $f(x)$ simply by choosing a random element in the domain.

graph. Additional attempts to avoid this loss of security problem were given by [31, 5] (see below).

1.3.1 Our Contribution to Hardness Amplification

We present an alternative efficient hardness amplification for regular one-way functions. Specifically, in Theorem 6.3 we show that the k'th randomized iterate of a weak one-way function along with the randomizing hash functions form a strong one-way function (for the right choice of k). Moreover, this holds also for the derandomized version of the randomized iterate (Theorem 6.7), giving an almost linear construction. Our construction is arguably simpler and has the following advantages:

1. While the construction in [10] works only for *known* regular weak one-way functions, our amplification works for any regular weak one-way function (whether its image size is efficiently computable or not).

2. The input length of the resulting strong one-way function is $\Theta(n \log n)$ regardless of the required security. Thus, for some range of the parameters our solution is better than that of [10] (although it is worse than [10] for other ranges).

As in the case of pseudorandom generators discussed above, our method would yield a construction with input length $\Theta(n)$ if bounded-space generators with better parameters become available.

The Idea. At the basis of all hardness amplification lies the fact that for any inverting algorithm, a weak one-way function has a set that the algorithm fails upon (hereafter, the *failing-set* of this algorithm). It follows that a large enough number of randomly chosen inputs are bound to hit every such failing-set and thus to fail every algorithm. Taking independent random samples (i.e., $f'(x_1, \ldots, x_k) = (f(x_1), \ldots, f(x_k))$) works well (this is Yao's construction [34]), but with the price of increasing the input length. An alternative approach (a variant of which was used by [10]) would be to use randomized iterations (i.e., to consider the function $f^k(x, \overline{h})$). This also amounts to applying f to k random inputs and therefore bound to hit every failing set. One obstacle is that when given $f^k(x, \overline{h})$, an adversary may invert f^k to a different input $(x', \overline{h}')$ (with different and carefully chosen hash functions) such that the computation of $f^k(x', \overline{h}')$ avoids applying f to a relevant failing set. To overcome this, the hash functions $\overline{h}$ are also given as part of the output (i.e., we consider the function $g(x, \overline{h}) = (f^k(x, \overline{h}), \overline{h})$), forcing an inverter to invert to the same hash functions. Using our core technique (from the pseudorandom generators section) we show that the hardness of inverting f^k is maintained even when $\overline{h}$ is known.

At a first glance, the aforementioned approach does not help in decreasing the input blowup, since the description of $\overline{h}$ is long. Indeed, choosing fully independent randomizing hash functions requires an input as long as that of Yao's solution (an input of length $\Theta(n \cdot \omega(\log(n))/\varepsilon)$). What makes this approach appealing, is the derandomization of the hash

functions using space-bounded generators, which reduces the input length to only $\Theta(n \log n)$. We mention that since the hardness of f^k stems from the fact that a random input hits with high probability any failing-set, it is required that this is also the case for the derandomized f^k (and not only that the derandomized function maintains low collision probability as in the pseudorandom generator case). Fortunately, the derandomization using bounded-space generators also guarantees this property.

We mention that there have been several attempts to formulate such a construction, using all of the aforementioned tools. Goldreich et al. [10] did actually consider following the GKL methodology, but chose a different (though related) approach. Phillips [31] gives a solution with input length $\Theta(n \log n)$ using bounded-space generators, but only for the simple case of permutations (where [10] has better parameters). Di Crescenzo and Impagliazzo [5] give a solution for regular functions, but only in a model where public randomness is available (in the mold of [17]). Their solution is based on pairwise-independent hash functions that serve as the public randomness. We are able to combine all these ingredients into one general result, perhaps due to our simplified proof.

1.3.2 Additional Issues

On non-length-preserving functions. This work focuses on length-preserving one-way functions. We also demonstrate how our proofs may be generalized, with no penalty in the tightness of the security, to use non-length preserving functions.[10] This generalization requires the use of a construction of a family of *almost* pairwise-independent hash functions (see Sections 2.7.1 and 3.4.1).

The results in the public randomness model. Similarly to previous works, our results also give linear reductions in the public randomness model. This model (introduced by Herzberg and Luby [17]) allows the use of public random coins that are not regarded a part of the input. Our results, however, introduce significant savings in the amount of public randomness that is necessary.

1.4 Organization

Section 2 includes the formal definitions and notations used throughout this work. In Section 3 we present our construction of pseudorandom generators from regular one-way functions. Section 4 presents the construction based on exponentially-hard one-way functions and in particular proves a lemma regarding the hardness of inverting the randomized iterate of a general one-way function (Lemma 4.1). In Section 5 we present our improvement to the HILL pseudorandom generator from any one-way function. Finally, in Section 6 we present our hardness amplification of regular one-way functions.

[10] Some alternative techniques for converting arbitrary one-way functions to length-preserving ones (e.g., padding), incur serious deterioration in the security of the resulting one-way functions.

2 Preliminaries

2.1 Notations

We use capital letters for random variables and matrices, standard letters for values and calligraphic letters for sets. Given two equal length strings x and y, we denote by $\langle x, y \rangle_2$ their inner product modulu two. A set $\mathcal{L} \subseteq \mathcal{S}$ is of *density* (at least) δ with respect to $\mathcal{S}$, if $|\mathcal{L}| \geq |\mathcal{S}| \cdot \delta$. For $f \colon \mathcal{S} \mapsto \{0,1\}^*$ and $\mathcal{L} \subseteq \mathcal{S}$, we let $f(\mathcal{L}) = \{f(x) \colon x \in \mathcal{L}\}$. For $y \in f(\mathcal{S})$, we denote the preimages of y under f by $f^{-1}(y) = \{x \in \mathcal{S} \colon f(x) = y\}$. The *degeneracy* of f on y is defined by $\mathrm{D}_f(y) := \lceil \log |f^{-1}(y)| \rceil$. The characteristic function of a set $\mathcal{S} \subseteq \mathcal{U}$, denoted $\chi_{\mathcal{S}}$, answers 1 on $x \in \mathcal{S}$ and 0 otherwise.

We let poly denote the set of polynomials, where we sometime abuse notation and use it to denote a member of this set, and let PPT denote the set of probabilistic algorithms (i.e., Turing machines) that run in *strict* polynomial time. A function $\mu : \mathbb{N} \mapsto [0,1]$ is *negligible*, if $\mu(n) < 1/p(n)$ for every $p \in \text{poly}$ and large enough n. We denote by $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, the ensemble of functions $\{f_n : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}_{n \in \mathbb{N}}$ (it will be clear from the context whether f denotes an infinite family or a single function). Throughout the paper we let n be the security parameter, and when its value is clear from the context, we sometimes omit it from the notation.

2.2 Distributions and Entropy

We adopt the convention that when the same random variable appears multiple times in an expression, all occurrences refer to the same instantiation. For example, $\Pr[X = X]$ is 1. Let X be a random variable taking values in a finite set $\mathcal{U}$. The *support* of X is $\mathrm{Supp}(X) := \{x \in \mathcal{U} \colon \Pr[X = x] > 0\}$ and we write $x \leftarrow X$ to indicate that x is selected according to X. If $\mathcal{S}$ is a subset of $\mathcal{U}$, then $x \leftarrow \mathcal{S}$ means that x is selected according to the uniform distribution on $\mathcal{S}$. We write U_n to denote the random variable distributed uniformly over $\{0,1\}^n$. Given a function $f : \{0,1\}^n \mapsto \{0,1\}^\ell$, we denote by $f(U_n)$ the distribution over $\{0,1\}^\ell$ induced by f operating on U_n. By a distribution ensemble over $\{\mathcal{S}(n)\}_{n \in \mathbb{N}}$, we mean a series $\{D_n\}_{n \in \mathbb{N}}$, where each D_n is a distribution over $\mathcal{S}(n)$.
Let D be a distribution over some finite domain X, we use the following measures of entropy:

- The Shannon entropy of D, denoted $\mathrm{H}(D)$, is defined as $\sum_{x \in X} D(x) \cdot \log \frac{1}{D(x)}$.
- The collision probability of D, denoted $\mathrm{CP}(D)$, is defined as $\sum_{x \in X} D(x)^2$.
- The min entropy of D, denoted $\mathrm{H}_\infty(D)$, is defined as $\min_{x \in X} \log \frac{1}{D(x)}$.

Two distributions X and Y over $\mathcal{U}$ are ε close, denoted $\Delta(X,Y) \leq \varepsilon$, if $\max_{\mathcal{S} \subseteq \mathcal{U}} |\Pr_{w \leftarrow X}(\mathcal{S}) - \Pr_{w \leftarrow Y}(\mathcal{S})| \leq \varepsilon$. We define the distinguishing advantage of an algorithm D between two distribution ensembles $\{X_n\}$ and $\{Y_n\}$ by

$$\Delta^{\mathsf{D}}(X_n, Y_n) = \left| \Pr[\mathsf{D}(1^n, x) = 1] - \Pr[\mathsf{D}(1^n, y) = 1] \right|,$$

where the probabilities are taken over $x \leftarrow X_n$ and $y \leftarrow Y_n$, and the randomness of D.

2.3 Function Ensembles

We denote an ensemble of function families by $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$, where $f \in \mathcal{F}_n$ maps strings of length n to stings of length $\ell(n)$, (namely $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$). We let $\mathcal{F} = \mathcal{F}_n$, whenever n is clear from the context, where in the special case that $\ell(n) = n$, the family $\mathcal{F}$ is called a length preserving.

To be useful in applications, we need $\mathcal{F}$ to be efficient:

Definition 2.1 (efficient function ensemble)**.** *An ensemble of function families $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ is* efficient*, if the description of the elements of $\mathcal{F}_n$ is the set $\{0,1\}^{d(n)}$, for some $d \in$ poly, and there exists a polynomial-time algorithm Eval with $Eval(f, x) = f(x)$ for every $n \in \mathbb{N}$, $f \in \mathcal{F}_n$ and $x \in \{0,1\}^n$.*

2.3.1 Functions with Partial Domains

While in Section 2 we usually consider function families with full domain: $f \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, which stands for $\{f_n \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$,[11] in the following sections we typically find it more convenient to consider function families with partial domain: $\{f_n \colon \mathcal{S}(n) \mapsto \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$, which we denote by $f \colon \mathcal{S}(n) \mapsto \{0,1\}^{\ell(n)}$. It is easy to see, however, that each of theses partial-domain families admits (via padding) a full-domain family with the same "features", e.g., if the partial-domain family is one way, then its full-domain variant is one-way with exactly the same security guarantee (ignoring constant factors). Hence, one can safely apply (as we do) definitions and lemmas that are stated with respect to full-domain function families, to a partial-domain family, while such application is formally reasoned with respect to the full-domain variant of the family.

2.4 Pairwise-Independent Hash Functions

Definition 2.2 (pairwise-independent hash functions)**.** *An ensemble of function families $\mathcal{H} = \{\mathcal{H}_n = \{h \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}\}_{n\in\mathbb{N}}$ is a* family of pairwise-independent hash functions*, if for every $n \in \mathbb{N}$ and $x \neq x' \in \{0,1\}^n$, it holds that $(h(x), h(x'))_{h\leftarrow\mathcal{H}_n}$ is uniformly distributed over $\{0,1\}^{2\ell(n)}$.*

For any polynomial-time computable function $\ell(n) \leq \text{poly}(n)$, there are various constructions of efficient families of pairwise-independent hash functions whose description length (i.e., $d(n)$ according to Definition 2.1) is linear in $n + \ell(n)$ (e.g., [3]).

In some cases we cannot afford to use hash functions whose description length is linear in the input size, but can only afford a description that is linear in the output size. In such cases we use the following relaxation of pairwise-independent hash functions:

Definition 2.3 (almost pairwise-independent hash functions)**.** *An ensemble of function families $\mathcal{H}_n = \{h \colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ is a* family of $\varepsilon(n)$-almost pairwise-independent hash functions*, if for any $n \in \mathbb{N}$ and $x \neq x' \in \{0,1\}^n$ it holds that $(h(x), h(x'))_{h\leftarrow\mathcal{H}_n}$ is $\varepsilon(n)$-close to the uniform distribution over $\{0,1\}^{2\ell(n)}$.*

[11] One exception is Lemma 2.9, where we consider a function family over $\{\{0,1\}^{2n} \times \mathcal{H}_n\}$.

Due to [3, 33] and [27], for any polynomial-time computable functions ε and ℓ, where $\ell(n)$ is an integer function bounded by $\text{poly}(n)$, there exist efficient families of $\varepsilon(n)$-almost pairwise-independent hash functions whose description length is $\Theta(\log(n)+\ell(n)-\log(\varepsilon(n)))$.

2.5 Randomness Extractors

Randomness extractors, introduced by Nisan and Zuckerman [30], are an information theoretic tool for obtaining true randomness from a "weak" source of randomness. In this work extractors are used in a computational setting to extract pseudorandomness from an imperfect source.

Definition 2.4 (strong extractors)**.** *A function* $\text{Ext} : \{0,1\}^q \times \{0,1\}^n \mapsto \{0,1\}^\ell$ *is* (k,ε)-strong extractor*, if* $\Delta\left((\text{Ext}(U_q, X), U_q), (U_\ell, U_q)\right) \leq \varepsilon$ *for every distribution* X *over* $\{0,1\}^n$ *with* $\text{H}_\infty(X) \geq k$*.*

2.6 Bounded-Space Generators

Bounded-space generators refer to (pseudorandom) generators that fool bounded-space adversaries. Such generators play a central role in derandomization tasks. We are interested in generators for the following type of adversaries:

Definition 2.5 (bounded-width layered branching program - LBP)**.** *An* (s,m,v)-LBP M *is a finite directed acyclic graph whose nodes are partitioned into* $m+1$ *layers indexed by* $\{1,\ldots,m+1\}$*. The first layer has a single node (the source), the last layer has two nodes (sinks) labeled with 0 and 1, and each of the intermediate layers has up to* 2^s *nodes. Each node in the* $i \in [m]$ *layer has exactly* 2^v *outgoing labeled edges to the* $(i+1)^{st}$ *layer, one for every possible string* $z \in \{0,1\}^v$*.*

For a sequence $\overline{z} \in \{0,1\}^{mv}$*, we let* $M(\overline{z})$ *(the output of* M *on input* $\overline{z}$*) be the label reached at the end of the following* m*-step walk: the walk starts at the source node of the first layer, and at each step advances from the* i^{th} *to the* $(i+1)^{st}$ *layer along the edge labeled by* $\overline{z}_i$*.*

An alternative (and somewhat more intuitive) description to the above, associates labels with the graph's nodes (rather than with its edges). Upon "reading" the input $\overline{z}_i$, the program uses an *arbitrary* computation, which depends only on the current node label and $\overline{z}_i$, and advances to a node in the $i+1$ layer.

Definition 2.6. *A generator* $\text{BSG} : \{0,1\}^n \mapsto \{0,1\}^{mv}$ *is said to* ε-fool *an* LBP M*, if*

$$|[M(U_{mv}) = 1] - \Pr[M(\text{BSG}(U_n)) = 1]| < \varepsilon.$$

The following theorem immediately follows from [24, Thm 2].

Theorem 2.7 ([29, 24])**.** *Let* $s(n), m(n), v(n) \in \mathbb{N}$ *and* $\varepsilon(n) \in (0,1)$ *be polynomial-time computable functions. Then there exist a polynomial-time computable function* $q(n) \in \Theta(v(n)+(s(n)+\log(m(n)/\varepsilon(n)))\cdot\log m(n))$ *and a generator* $\text{BSG} : \{0,1\}^{q(n)} \mapsto \{0,1\}^{m(n)\cdot v(n)}$ *that runs in time* $\text{poly}(s(n), m(n), v(n), \log(1/\varepsilon(n)))$*, and* $\varepsilon(n)$*-fools every* $(s(n), m(n), v(n))$-LBP*.*

2.7 One-Way Functions

Definition 2.8 (one-way functions)**.** *A function* $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ *is* $(T(n), \varepsilon(n))$-one-way*, if* f *is polynomial-time computable and*

$$\Pr_{y \leftarrow f(U_n)}[\mathsf{A}(1^n, y) \in f^{-1}(y)] < \varepsilon(n)$$

for any algorithm A *of running time* $T(n)$ *and large enough* n*.*[12] *A* one-way permutation *is a one-way function that is a permutation over* $\{0,1\}^n$ *for every* $n \in \mathbb{N}$*. A function* f *is* regular*, if there exists an integer function* α *such that*

$$\left|f^{-1}(f(x))\right| = \alpha(n)$$

for every $n \in \mathbb{N}$ *and* $x \in \{0,1\}^n$*. In the special case that* α *is polynomial-time computable, we say that* f *is* known *regular.*[13]

In the case that $\varepsilon(n) = 1/T(n)$, we simply write that f is $T(n)$-one-way. f is one-way, if it is $T(n)$-one-way for every $T \in$ poly, where f is exponentially hard (one-way), if it is 2^{cn}-one-way for some constant $c > 0$. Finally, if f is $(T(n) > n^{O(1)}, 1 - \varepsilon(n))$-one-way, it is customary to call it an $\varepsilon(n)$-weak one-way function.

2.7.1 Length Preserving One-Way Functions

In the following we prove the "folklore" fact that a one-way function can be assumed without loss of generality to be length preserving. In the case where the function is length decreasing, one can generate a length preserving one-way function simply by padding the output with extra zeros. In the case that it is length increasing, however, one needs to be more careful in order for the input length to remain of the same order.

Lemma 2.9. *Let* $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ *be a* $(T = T(n), \varepsilon = \varepsilon(n))$*-one-way function and let* $\mathcal{H}$ *be an efficient family of* 2^{-2n}*-almost pairwise-independent hash functions from* $\{0,1\}^{\ell(n)}$ *to* $\{0,1\}^{2n}$*. We define* g *as*

$$g(x_a, x_b, h) = (h(f(x_a)), h),$$

where $x_a, x_b \in \{0,1\}^n$ *and* $h \in \mathcal{H}$*. Then* g *is a length-preserving* $(T - n^{O(1)}, \varepsilon + 2^{-n+1})$*-one-way function.*[14]

Recall that by [3, 33] and [27], we have such hash families whose description length is $\Theta(n)$.

Proof. The function g is length preserving as both input and output are of length $2n$ plus the description of $h \in \mathcal{H}$. Let A be an algorithm that runs in time $T_{\mathsf{A}} = T_{\mathsf{A}}(n)$ and inverts g with probability $\varepsilon_{\mathsf{A}} = \varepsilon_{\mathsf{A}}(n)$. Note that x_b is a dummy input (used just for padding) so the success of A is taken over (x_a, h) and A's randomness. Define M^{A} as follows:

[12] We typically omit the security parameter (i.e., 1^n) from the adversary's parameters list.

[13] In this work we do not require such property, and our results hold for functions with unknown regularity. Thus, when we say regular functions we actually mean unknown-regular functions.

[14] See Section 2.3.1 regarding the fact that g is only defined over some input lengths.

Algorithm 2.10. *(M^{A})*

Input: $y \in f(\{0,1\}^n)$.

Operation:

1. *Choose a uniformly random* $h \in \mathcal{H}$.
2. *Apply* $\mathsf{A}(h(y), h)$ *to get an output* (x_a, x_b, h).
3. *Output* x_a.

Clearly, the running time of M^{A} is (at most) $T_{\mathsf{A}} + n^{O(1)}$. Algorithm M^{A} is sure to succeed on any choice of (y, h) for which the following hold: A succeeds on $(h(y), h)$ and there exists no $y' \neq y \in f(\{0,1\}^n)$ such that $h(y') = h(y)$ (h does not introduce any collision to y). Let $\mathcal{S} := \{(y,h) \in \{0,1\}^n \times \mathcal{H} \ : \ \exists y' \neq y \in f(\{0,1\}^n) \ : \ h(y') = h(y)\}$. Thus,

$$\begin{aligned}\Pr[M^{\mathsf{A}}(f(U_n)) \in f^{-1}(f(U_n))] &\geq \Pr[\mathsf{A}(g(U_n,H)) \in g^{-1}(g(U_n,H)) \wedge g(U_n,H) \notin \mathcal{S}] \\ &\geq \Pr[\mathsf{A}(g(U_n,H)) \in g^{-1}(g(U_n,H))] - \Pr[g(U_n,H) \in \mathcal{S}],\end{aligned}$$

where H is a random variable uniformly distributed over $\mathcal{H}$. The almost pairwise independence of $\mathcal{H}$ assures that for every $y \neq y' \in f(\{0,1\}^n)$, it holds that $\Pr[H(y') = H(y)] \leq 2^{-2n+1}$. Since $|f(U_n)| \leq 2^n$, a union bound implies that $\Pr[\exists y' \neq y \in f(\{0,1\}^n) \ : \ H(y') = H(y)] \leq 2^{-n+1}$ for every $y \in f(U_n)$. Thus, an averaging argument yields that $\Pr[(f(U_n), H) \in \mathcal{S}] \leq 2^{-n+1}$. Putting it all together, we get that $\Pr[M^{\mathsf{A}}(f(U_n)) \in f^{-1}(f(U_n))] \geq \varepsilon_{\mathsf{A}} - 2^{-n+1}$. □

2.8 Hardcore Predicates and Functions

Hard-core predicates/functions have a major role in the construction of one-way function based pseudorandom generators.

Definition 2.11 (hardcore functions)**.** *We call* $hc : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ *a* $(T(n), \varepsilon(n))$-hardcore function *of* $f : \{0,1\}^n \mapsto \{0,1\}^*$ *over* $\mathcal{S} = \{\mathcal{S}(n) \subseteq \{0,1\}^n\}_{n \in \mathbb{N}}$*, if*

$$\Delta^{\mathsf{D}}\left((1^n, f(x), hc(x))_{x \leftarrow \mathcal{S}(n)}, (1^n, f(x), U_{\ell(n)})_{x \leftarrow \mathcal{S}(n)}\right) \leq \varepsilon(n),$$

for any algorithm D *of running-time* $T(n)$ *and large enough* n.

We use the following conventions: in case $\mathcal{S}(n) = \{0,1\}^n$ for every n, we omit it from the above notation. In the case that $\varepsilon = 1/T$, we simply say that hc is a T-hardcore function. Where hc is simply a hardcore function, if it is a $T(n)$-hardcore function for every $T \in$ poly. If hc is a predicate (i.e., $\ell(n) = 1$), it is called a hardcore predicate of f.[15] Finally, it is common to call the value $hc(x)$, the "hardcore-bits" of $f(x)$.

[15] For hardcore predicates, it is more common (and somewhat more convenient) to measure the advantage of a possible "predictor" in guessing $hc(x)$ given $f(x)$, rather than the distinguishing gap as above. Yet, we preferred the above definition to have the same terminology for (hardcore) predicates and functions.

It is easy to see that a $(T(n), \varepsilon(n))$-hardcore predicate of f over $\{\mathcal{S}(n) \subseteq \{0,1\}^n\}_{n\in\mathbb{N}}$ of density δ (i.e., $\mathcal{S}(n)$ is of density $\delta(n)$ with respect to $\{0,1\}^n$, for every $n \in \mathbb{N}$), is a $(T(n), \delta(n)\cdot\varepsilon(n) + \frac{1-\delta(n}{2})$-hardcore predicate of f over the whole domain (i.e., $\{\{0,1\}^n\}_{n\in\mathbb{N}}$).

We use the Goldreich-Levin hardcore functions, and in particular make use of the following theorem whose proof immediately follows from [8, Corollary 1].

Theorem 2.12 ([8]). *There exists a polynomial-time computable function $gl : \{0,1\}^{3v} \mapsto \{0,1\}^v$ such that the following holds: let $f : \{0,1\}^n \mapsto \{0,1\}^{m_f(n)}$ and $g : \{0,1\}^n \mapsto \{0,1\}^{m_g(n)}$ be two polynomial-time computable functions over $\{0,1\}^n$, and let $\mathcal{S} = \{\mathcal{S}(n) \subseteq \{0,1\}^n\}_{n\in\mathbb{N}}$. Assume that*

$$\Pr_{x\leftarrow\mathcal{S}(n)}[\mathsf{A}(1^n, f(x)) = g(x)] \leq \varepsilon(n)$$

for any algorithm A of running time $T(n)$ and large enough n. Then for any $\ell(n) \in [m_g(n)]$, the function hc: $\{0,1\}^n \times \{0,1\}^{2m_g(n)} \mapsto \{0,1\}^{\ell(n)}$, defined as $hc(x,r) = gl(g(x),r)_{1,\ldots,\ell(n)}$, is a $(T(n)\cdot(\varepsilon(n)/n)^{O(1)}, O(2^{\ell(n)}\cdot\varepsilon(n)))$-hardcore function of $f'(x,r) = (f(x),r)$ over $\mathcal{S}' = \{\mathcal{S}(n)\times\{0,1\}^{2m_g(n)}\}_{n\in\mathbb{N}}$.[16]

For the important case of super-polynomial hardness and polynomial-time computable f and g, the above theorem yields the following fact:

Corollary 2.13. *Let f, g, $\mathcal{S}$, hc, f' and $\mathcal{S}'$ be as in Theorem 2.12. Assume that f is polynomial-time computable, that $\ell(n) \in O(\log n)$ and that*

$$\Pr_{x\leftarrow\mathcal{S}(n)}[\mathsf{A}(1^n, f(x)) = g(x)] = \mathrm{neg}(n)$$

for any PPT A*, then hc is a hardcore function of f' over $\mathcal{S}'$.*

2.9 A Uniform Extraction Lemma

The following lemma is a generalization of the (uniform version) of Yao's XOR lemma. Given t independent $(T, (1-\varepsilon)/2)$-hardcore bits, we "extract" approximately εt pseudorandom bits out of them.[17]

The basic statement is that the application of a strong randomness extractor to t independent "weak" hardcore bits, forms a strong hardcore function. In our discussion we denote the weak hardcore predicate by b and the output of the extractor by hc. The statement is proved by showing a reduction from a distinguishing algorithm D that succeeds in distinguishing between the output of hc and true randomness (when seeing the output of f), to an algorithm that predicts the weak hardcore predicate b with high success probability, thus contradicting the hardness guarantees of the weak hardcore predicate.

[16] The computation of the first ℓ output bits of gl only uses the first $m_g + \ell - 1$ bits of r. In particular for $\ell = 1$, the input length of hc (the "Goldreich-Levin predicate") can be made m_g (and not $2m_g$ as stated in the theorem). Yet, to simplify notations we always apply hc on inputs of length $2m_g$.

[17] Lemma 2.14 generalizes [16, Lemma 6.5]. Independently of this work, [20, Thm 7.3] presents a similar generalization of this lemma.

For convenience of presentation, we denote by $\rho_{\delta,t,m}$ the probability that out of t independent binomial random variables, each with probability at least δ of being one, at least m come out ones.

Lemma 2.14. *Let $f : \{0,1\}^n \mapsto \{0,1\}^\ell$, $b\colon \{0,1\}^n \mapsto \{0,1\}$, and let $\mathrm{Ext} : \{0,1\}^q \times \{0,1\}^t \mapsto \{0,1\}^r$ be a $(m, \varepsilon_{\mathrm{Ext}})$-strong extractor over $\{0,1\}^t$ where ℓ, q, t, m and r are functions of n and $\varepsilon_{\mathrm{Ext}} \in (0,1]$. Moreover, assume that f, b and Ext are polynomial-time computable (when Ext is given also given n as an additional advice).*

Define $hc\colon \{0,1\}^q \times \{0,1\}^{t\times n} \mapsto \{0,1\}^r$ as

$$hc(s, x_1, \ldots, x_t) = (s, \mathrm{Ext}(s, b(x_1), \ldots, b(x_t))).$$

Then for any distinguishing algorithm D, there exists a PPT M such that the following holds: If $\delta \in (0,1]$ and $\gamma \in (3t \cdot 2^{-n/3}, 1]$ are such that

$$\varepsilon_{\mathsf{D}} := \Delta^{\mathsf{D}}\big((1^n, f(U_n{}^1), \ldots, f(U_n{}^t), hc(U_q, b(U_n{}^1), \ldots, b(U_n{}^t))), (1^n, f(U_n{}^1), \ldots, f(U_n{}^t), U_q, U_r)\big) \tag{1}$$

$$> \varepsilon_{\mathrm{Ext}} + \rho_{\delta,t,m} + \gamma,$$

then

$$\Pr\left[M(1^n, 1^{T_{\mathsf{D}}(n)}, 1^{\lfloor 1/\delta \rfloor}, 1^{\lfloor 1/\gamma \rfloor}, f(U_n)) = b(U_n)\right] > 1 - \delta/2 + c \cdot \gamma^2 \cdot \delta^5,$$

where $T_{\mathsf{D}}(n)$ bounds the running time of D and $c > 0$ is a universal constant.

Specifically, for the right choice of parameters, if b is a hardcore predicate of f and Ext is a strong extractor, then hc is an hardcore function of $f(x_1, \ldots, x_t) = f(x_1), \ldots, f(x_t)$.

Proof. For a fixed set $\mathcal{S} \subseteq \{0,1\}^n$ of density δ, define the following (not necessarily efficiently computable) randomized predicate $Q : \{0,1\}^n \mapsto \{0,1\}$:

$$Q(x) = \begin{cases} U_1 & x \in \mathcal{S}, \\ b(x) & \text{otherwise.} \end{cases} \tag{2}$$

For $i \in \{0, \ldots, t\}$, define the random variable X^i as

$$X^i = (1^n, f(U_n{}^1), \ldots, f(U_n{}^t), U_q, \mathrm{Ext}(U_q, b(U_n{}^1), \ldots, b(U_n{}^i), Q(U_n{}^{i+1}), \ldots, Q(U_n{}^t)) \tag{3}$$

By definition, the statistical distance between X^0 and $(1^n, f(U_n{}^1), \ldots, f(U_n{}^t), U_q, U_r)$ is bounded by $\varepsilon_{\mathrm{Ext}} + \rho_{\delta,t,m}$. Since $X^t = (1^n, f(U_n{}^1), \ldots, f(U_n{}^t), hc(U_q, U_n{}^1, \ldots, U_n{}^t))$, it follows that

$$\Pr[\mathsf{D}(X^t) = 1] - \Pr[\mathsf{D}(X^0) = 1] > \varepsilon_{\mathsf{D}} - \varepsilon_{\mathrm{Ext}} - \rho_{\delta,t,m} = \gamma, \tag{4}$$

where the absolute value is omitted without loss of generality. A straightforward hybrid argument yields that D distinguishes between X^j and X^{j+1} with advantage γ/t, for some

$j \in \{0,\ldots,t-1\}$. We next use D to define an algorithm D_1 for telling $(f(X), b(X))_{x\leftarrow\mathcal{S}}$ from $f(X), U)_{x\leftarrow\mathcal{S}}$: algorithm D_1 first finds $j \in \{0,\ldots,t-1\}$ with

$$\Pr[\mathsf{D}(X^{j+1}) = 1] - \Pr[\mathsf{D}(X^j) = 1] > \gamma/2t \tag{5}$$

Note that using $4nt^3/\gamma^2$ simulations of D, such index j can be found correctly with probability $1 - 2^{-n}$. Then on input (y, b), algorithm D_1 returns $D(1^n, f(U_n^1), \ldots, f(U_n^j), y, f(U_n^{j+1}), \ldots, f(U_n^t), U_q, \mathrm{Ext}(U_q, b(U_n^1), \ldots, b(U_n^j), b, U^{j+2}, \ldots, U^t))$. It follows that

$$\Pr[\mathsf{D}_1(f(U_n), b(U_n)) = 1] - \Pr[\mathsf{D}_1(f(U_n), U) = 1] > \gamma/2t - 2^{-n} > \gamma/3t \tag{6}$$

and that the running time of D_1 is $\mathrm{poly}(n) \cdot T_{\mathsf{D}}(n)/\gamma^2$. Moreover, since X^j and X^{j+1} are identical when conditioned that ${U_n}^{j+1} \notin \mathcal{S}$, then Equation (6) holds even when conditioning on $U_n \in \mathcal{S}$. Using a standard reduction from distinguishing to predicting (e.g. in [7, Sec 3.3.5]) we get that there exists a predictor P that using a single call to D_1 achieves

$$\Pr_{x\leftarrow\mathcal{S}}[P(f(x)) = b(x)] > \frac{1}{2} + \gamma/3t \tag{7}$$

We complete the proof using the following proposition, which immediately follows from [20, Thm 6.8].[18]

Proposition 2.15. *(Uniform Hardcore Lemma, [20]) Let $f\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ and $b\colon \{0,1\}^n \mapsto \{0,1\}$ be polynomial-time computable functions. Then for any predictor algorithm P there exists a* PPT *M such that the following holds for any $n \in \mathbb{N}$, $\delta \in (0,1]$ and $\gamma \in (2^{-n/3}, 1]$: assuming that*

$$\Pr_{x\leftarrow\mathcal{S}}[P^{\chi_{\mathcal{S}}}(f(x)) = b(x)] > \frac{1}{2} + \gamma$$

for every set $\mathcal{S} \subseteq \{0,1\}^n$ of density δ, where all of P's queries to $\chi_{\mathcal{S}}$ are computed independently of the input $f(x)$, then

$$\Pr[M(1^n, 1^{T_P(n)}, 1^{\lfloor 1/\delta \rfloor}, 1^{\lfloor 1/\gamma \rfloor}, f(U_n)) = b(U_n)] > 1 - \delta/2 + c \cdot \gamma^2 \cdot \delta^5,$$

where $T_P(n)$ bounds the running time of P and $c > 0$ is a universal constant.

Given Equation (7), Proposition 2.15 yields the existence of a PPT M with

$$Pr\left[M(1^n, 1^{T_{\mathsf{D}}(n)}, 1^{\lfloor 1/\delta \rfloor}, 1^{\lfloor 1/\gamma \rfloor}, f(U_n)) = b(U_n)\right] > 1 - \delta/2 + c \cdot \gamma^2 \cdot \delta^5$$

for any such n, m, δ and γ. □

[18] [20, Thm 6.8] requires δ and γ to be polynomial-time computable and noticeable, where here we give them as parameters and give no such restrictions on their values. Nevertheless, the generalization presented here readily follows from the original proof.

2.10 Pseudorandom Generators

Definition 2.16 (pseudorandom generators)**.** *An ensemble of distributions $D = \{D_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^{\ell(n)}$ is $(T(n),\varepsilon(n))$-*pseudorandom*, if*

$$\Delta^{\mathsf{D}}\left(D_n, U_{\ell(n)}\right) < \varepsilon(n)$$

for every algorithm D *of running time $T(n)$ and large enough n.*

*A a function $G : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ is a $(T(n),\varepsilon(n))$-*pseudorandom generator*, if it is length-increasing and polynomial-time computable, and $G(U_n)$ is $((T(n),\varepsilon(n))$-pseudorandom.*

In the case that $\varepsilon(n) = 1/T(n)$, we simply say that G is a $T(n)$-pseudorandom generator, where G is a pseudorandom generator, if it is $T(n)$-pseudorandom generator for any $T \in$ poly.

2.11 The Security of Cryptographic Constructions

Typically, the proof of security for cryptographic constructions is based on reductions. In this paradigm we use a presumably secure implementation of one primitive (or possibly several primitives) in order to implement a second primitive. The proof of security for the second primitive relies on the security assumption for the original one. More precisely, we prove that any efficient adversary that breaks the implementation of the second primitive can be used to efficiently break the original primitive. Note that the meaning of "breaking a primitive" and, furthermore, the definition of the *success probability* of an adversary in breaking the primitive, varies between different primitives. For example, in the case of one-way functions the success probability is the fraction of inputs on which the adversary manages to invert the function. Usually, there is a tradeoff between the running time of an adversary and its success probability (e.g., it may be possible to utterly break a primitive by enumerating all possibilities for the secret key). Therefore, both the running time and success probability of possible adversaries are relevant when analyzing the security of a primitive. A useful, combined parameter is the *time-success ratio* of an adversary which we define next.

Definition 2.17 (time-success ratio)**.** *Let P be a primitive and let A be an adversary running in time $T_A(n)$ and breaking P with probability $\varepsilon_A(n)$. The* time-success ratio *of A in breaking P is defined as $R(n) = \frac{T_A(n)}{\varepsilon_A(n)}$, where n is the security-parameter of the primitive.*

Note that in the above definition, the smaller the R the better A is in breaking P. A quantitative analysis of the security of a reduction is crucial for both theoretical and practical reasons. Given an implementation of primitive P using primitive Q along with a proof of security, let R_P be the security-ratio of a given adversary with respect to P and let R_Q be the security-ratio of the adversary that the proof of security yields. A natural way to measure the security of a reduction is by the relation between R_P and R_Q. Clearly, the smaller the R_Q comparing to R_P, the better the performance of the adversary the reduction yields when trying to break Q comparing to the performance of the adversary trying to break P.

The most desirable reduction is when $R_Q(n) \in n^{O(1)} \cdot O(R_P(O(n)))$. In such reductions, known as *linear-preserving reductions*, we are guaranteed that breaking the constructed primitive is essentially as hard as breaking the original one. Next we find the *polynomial-preserving reductions* when $R_Q \in n^{O(1)} \cdot O(R_P(O(n))^{O(1)})$. Note that a linear/polynomial-preserving reduction typically means that for the same level of security we can take the inputs of Q and P to be of the same length (up to a constant-ratio). The other side of the scale is when $R_Q \in n^{O(1)} \cdot O(R_P(n^{O(1)}))$. In such reductions, known as *weak-preserving reductions*, we are only guaranteed that breaking P is as hard as breaking Q for polynomially smaller security-parameter (e.g., polynomially smaller input length). For a more comprehensive discussion of the above issues the reader may refer to [6, 17]. This quantitative classification of security preserving reduction partly motivates our focus on the input length as the main parameter that our reductions aim to improve. In particular, better space bounded generators would make our reduction polynomial preserving rather than weak preserving.

2.11.1 Black-Box Reductions

It is worth mentioning while the reductions given in Sections 3 and 6 are fully black box, the underlying function and the possible adversary are treated as black boxes (i.e., as oracles), the reductions given in Sections 4 and 5 are not. The latter reductions are all using Holenstein's "uniform hardcore lemma" (specifically, [20, Thm 6.8]) that has a non-black box proof. This usage of non-black-box proof, however, does not seems inherent to the problem, and it is very likely that the proof of [20, Thm 6.8] can be made to be fully black-box. Such a change would also change these reductions to be fully black box.

3 Pseudorandom Generators from Regular One-Way Functions

The following discussion considers only length preserving regular one-way functions, where the extension to general regular one-way functions is described in Section 3.4.1.

3.1 Some Motivation and the Randomized Iterate

Recall that the BMY generator simply iterates a one-way permutation $f\colon \{0,1\}^n \mapsto \{0,1\}^n$ on itself for $n+1$ times and outputs a hardcore bit of each iteration. The rational is that since f is a permutation, the output of each iteration is uniform in $\{0,1\}^n$ and therefore it is hard to invert f on each of the iterations. It follows that output bits of the generator are unpredictable, and by Yao [34] they are also pseudorandom.

We want to duplicate this approach for general one-way functions, but unfortunately the situation changes drastically when the function f is not a permutation. After a single application of f, the output may be very far from uniform, and in fact may be concentrated on a very small and easy fraction of the inputs to f. Thus, reapplying f to this output gives no hardness guarantees at all. In an attempt to salvage the BMY framework, Goldreich et al.

[11] suggested to add a randomization step between every two applications of f, making the next input to f a truly random one. This modification, which we call the randomized iterate, lies at the core of our work and is defined next.

Definition 3.1 (the randomized iterate)**.** *Let $n \in \mathbb{N}$, $f : \{0,1\}^n \mapsto \{0,1\}^n$ and let $\mathcal{H}$ be a family of pairwise-independent length-preserving hash functions over strings of length n. For $k \in \mathbb{N}$, $x \in \{0,1\}^n$ and $\overline{h} = (h_1, \ldots, h_{k-1}) \in \mathcal{H}^{k-1}$, define the k'th* randomized iterate *$f^k : \{0,1\}^n \times \mathcal{H}^{k-1} \mapsto \{0,1\}^n$ recursively as*

$$f^k(x, \overline{h}) = f(h_{k-1}(f^{k-1}(x, (h_1, \ldots, h_{k-2})))),$$

where $f^1(x) = f(x)$. For $m > k-1$, we let $f^k(x, h_1, \ldots, h_m) = f^k(x, h_1, \ldots, h_{k-1})$.

In the following we denote by H^j the random variable uniformly distributed over $\mathcal{H}^j$.

The application of the randomized iterate for pseudorandom generators is a bit tricky. On the one hand, such a randomization costs a large number of random bits, much larger than what can be compensated for by the hardcore-bits generated in each iteration. So in order for the output to actually be longer than the input, we also output the descriptions of the hash functions. But on the other hand, handing out the randomizing hash gives information on intermediate values such as $f^i(x, \overline{h})$, and thus f might no longer be hard to invert when applied to such an input. Somewhat surprisingly, the randomized iterate of a regular one-way function remains hard to invert even when the hash functions are known. This fact, which is central to the whole approach, was proved in [11] when using a family of n-wise independent hash functions. As a first step, we give a simpler proof that extends to pairwise-independent hash functions as well.

3.2 The Last Randomized Iteration is Hard to Invert

In this section we formally state and prove the key observation mentioned above. After applying k randomized iterations of a regular one-way function f, it is hard to invert the last iteration, even if given access to all of the hash functions leading up to this point.

Lemma 3.2. *Let n, f, $\mathcal{H}$, k and f^k be as in Definition 3.1. Assume that f is regular, then for any algorithm A with*

$$\Pr[\mathsf{A}(f^k(U_n, H^{k-1}), H^{k-1}) = f^{k-1}(U_n, H^{k-1})] = \varepsilon_{\mathsf{A}}$$

it holds that

$$\Pr[H_{k-1}^{k-1}(\mathsf{A}(f(U_n), H^{k-1})) \in f^{-1}(f(U_n))] \geq \varepsilon_{\mathsf{A}}^2/k.$$

Assuming that f one-way, we get the following corollary.

Corollary 3.3. *Let $k(n) \in \mathbb{N}$, and for $n \in \mathbb{N}$ let f, $\mathcal{H}$ and f^k be as in Definition 3.1 (that is f, $\mathcal{H}$ and f^k are infinite families of functions, whose input sets are indexed by n). Assuming that f is a regular one-way function, that $\mathcal{H}$ is efficiently computable and that $k \leq \mathrm{poly}(n)$, then the following hold:*

1. $\Pr[\mathsf{A}(f^k(U_n, H^{k-1}), H^{k-1}) = f^{k-1}(U_n, H^{k-1})] = \mathrm{neg}(n)$ *for any* PPT A, *and*

2. *The predicate* $b^k \colon \{0,1\}^n \times \mathcal{H}^{k-1} \times \{0,1\}^{2n} \mapsto \{0,1\}$ *defined as* $b^k(x, \overline{h}, r) = gl(f^{k-1}(x, \overline{h}), r)_1$ *is a hardcore predicate of* $\widehat{f^k}(x, \overline{h}, r) = (f^k(x, \overline{h}), \overline{h}, r)$, *where* gl *is the Goldreich-Levin hardcore function (see Theorem 2.12).*

Proof. Assume there exists a PPT A that violates (1), Lemma 3.2 yields that $\Pr[H^{k-1}_{k-1}(\mathsf{A}(f(U_n), H^{k-1})) \in f^{-1}(f(U_n))] \neq \mathrm{neg}(n)$. Hence, there exists a PPT that inverts f with non-negligible probability, in contradiction to the one-wayness of f. The the second part of the corollary immediately follows from (1) and Theorem 2.12, when plugging $f = f^k$, $g = f^{k-1}$ and $\mathcal{S}(n) = \{0,1\}^n \times \mathcal{H}^{k-1}$. □

Proof. (of Lemma 3.2) Note that $\mathsf{A}(f^k(x, \overline{h}), \overline{h}) = f^{k-1}(x, \overline{h})$ implies that $\overline{h}_{k-1}(\mathsf{A}(f^k(x, \overline{h}), \overline{h})) \in f^{-1}(f^k(x, \overline{h}))$. Since $f^k(U_n, H^{k-1})$ and $f(U_n)$ are identically distributed, the lemma would have trivially followed had H^{k-1} been *independent* of $f^k(U_n, H^{k-1})$. Unfortunately, this is clearly not the case for arbitrary an function f. Rather, Lemma 3.4 states that for our purposes, the variables H^{k-1} and $f^k(U_n, H^{k-1})$ are "close enough" to being independent in the case of regular functions. Details follow.

In the following we assume that A is deterministic, where the proof for randomized A would follow.[19] Let $\mathrm{Good}_{\mathsf{A}} \subset \{0,1\}^n \times \mathcal{H}^{k-1}$ include the inputs on which A is successful upon. Namely,

$$\mathrm{Good}_{\mathsf{A}} = \{(y, \overline{h}) \in \{0,1\}^n \times \mathcal{H}^{k-1} : \overline{h}_{k-1}(\mathsf{A}(y, \overline{h})) \in f^{-1}(y)\} \tag{8}$$

Since $\Pr[(f^k(U_n, H^{k-1}), H^{k-1}) \in \mathrm{Good}_{\mathsf{A}}] = \varepsilon_{\mathsf{A}}$, the proof of Lemma 3.2 follows by the next lemma when plugging in $\mathcal{L} = \mathrm{Good}_{\mathsf{A}}$ and $\delta = \varepsilon_{\mathsf{A}}$.

Lemma 3.4. *Let* f, $\mathcal{H}$, k, *and* f^k *be as in Lemma 3.2. Assume that* f *is regular, then for any set* $\mathcal{L} \subseteq \{0,1\}^n \times \mathcal{H}^{k-1}$ *with*

$$\Pr[(f^k(U_n, H^{k-1}), H^{k-1}) \in \mathcal{L}] \geq \delta,$$

it holds that

$$\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}] \geq \delta^2/k.$$

□

[19] Let N be the number of different values for the coins used by A, let A_r be the instance of A whose random coins are fixed to r and let $\varepsilon_{\mathsf{A}_r}$ be the success probability of A_r. Assuming Lemma 3.2 for deterministic algorithms, it follows that

$$\begin{aligned}\Pr[H^{k-1}_{k-1}(\mathsf{A}(f(U_n), H^{k-1})) \in f^{-1}(f(U_n))] &= \frac{1}{N} \cdot \sum_r \Pr[H^{k-1}_{k-1}(\mathsf{A}_r(f(U_n), H^{k-1})) \in f^{-1}(f(U_n))] \\ &\geq \frac{1}{N} \cdot \sum_r \frac{\varepsilon^2_{\mathsf{A}_r}}{k} = \frac{1}{Nk} \cdot \sum_r \varepsilon^2_{\mathsf{A}_r} \geq \frac{1}{Nk} \cdot N \cdot \varepsilon_A^2 = \varepsilon^2_{\mathsf{A}}/k,\end{aligned}$$

where the first inequality is by Lemma 3.2 (for deterministic algorithms) and last one follows since $\varepsilon_A = \frac{1}{N}\sum_r \varepsilon_{\mathsf{A}_r}$ and using the inequality $\|x\|_1 \leq \sqrt{N} \cdot \|x\|_2$, for any $x \in \mathbb{R}^N$.

Proof. The lemma essentially states that with respect to $\widehat{f^k}(x,\overline{h}) = (f^k(x,\overline{h}),\overline{h})$ (i.e., the function defined by concatenating the input hash functions to the output of f^k), any large subset of inputs induces a large subset of outputs. Thus, there is a fairly high probability of hitting this output set simply by sampling independent y and $\overline{h}$. Intuitively, if a large set of inputs induces a small set of outputs, then there must be many collisions in this set (a collision means that two different inputs lead to the same output). Indeed, to prove the lemma we start by showing that many collisions are impossible, or more precisely, by proving an upper bound on the collision probability of the function $(f^k(x,\overline{h}),\overline{h})$.

Claim 3.5. *It holds that*

$$\mathrm{CP}(f^k(U_n,H^{k-1}),H^{k-1}) \leq \frac{k}{|\mathcal{H}|^{k-1}\cdot|f(\{0,1\}^n)|}.$$

Proof. For every two inputs $(x_0,\overline{h_0})$ and $(x_1,\overline{h_1})$ to f^k, in order to have a collision we must first have that $\overline{h_0} = \overline{h_1}$, which happens with probability $\frac{1}{|\mathcal{H}|^{k-1}}$. Now, given that $\overline{h_0} = \overline{h_1} = \overline{h}$ (with a random $\overline{h} \in \mathcal{H}^{k-1}$), we require also that $f^k(x_0,\overline{h})$ equals $f^k(x_1,\overline{h})$. If $f(x_0) = f(x_1)$ (happens with probability $1/|f(\{0,1\}^n)|$) then a collision is assured. Otherwise, there must be an $i \in [k-1]$ for which $f^i(x_0,\overline{h}) \neq f^i(x_1,\overline{h})$, but $f^{i+1}(x_0,\overline{h}) = f^{i+1}(x_1,\overline{h})$. Since $f^i(x_0,\overline{h}) \neq f^i(x_1,\overline{h})$, due to the pairwise independence of h_i, the values $h_i(f^i(x_0,\overline{h}))$ and $h_i(f^i(x_1,\overline{h}))$ are uniformly random values in $\{0,1\}^n$, and thus $f(h_i(f^i(x_0,\overline{h}))) = f(h_i(f^i(x_1,\overline{h})))$ happens with probability $1/|f(\{0,1\}^n)|$. Altogether, $\mathrm{CP}(f^k(U_n,H^{k-1}),H^{k-1}) \leq \frac{1}{|\mathcal{H}|^{k-1}}\cdot\frac{k}{|f(\{0,1\}^n)|}$. □

Continuing the proof of Lemma 3.4, we now find a lower bound on the probability of getting a collision, namely a lower bound on the quantity $\mathrm{CP}(f^k(U_n,H^{k-1}),H^{k-1})$. We do so by bounding the (possibly smaller) probability of getting a collision $(x_0,\overline{h_0}) = (x_1,\overline{h_1}) \in \mathcal{L}$, for two independent values of $(f^k(U_n,H^{k-1}),H^{k-1})$. We first request that both $(x_0,\overline{h_0}) \in \mathcal{L}$ and $(x_1,\overline{h_1}) \in \mathcal{L}$. This happens with probability at least δ^2. Once inside $\mathcal{L}$, we know that the probability of collision is at least $1/|\mathcal{L}|$. Altogether:

$$\mathrm{CP}(f^k(U_n,H^{k-1}),H^{k-1}) \geq \delta^2\cdot\frac{1}{|\mathcal{L}|}. \tag{9}$$

Combining Claim 3.5 and eq. (9), we get $\frac{|\mathcal{L}|}{|\mathcal{H}|^{k-1}\cdot|f(\{0,1\}^n)|} \geq \frac{\delta^2}{k}$. Since the probability of getting a value in $\mathcal{L}$ when choosing a random element in $f(\{0,1\}^n)\times\mathcal{H}^{k-1}$ is exactly $\frac{|\mathcal{L}|}{|\mathcal{H}|^{k-1}\cdot|f(\{0,1\}^n)|}$ (this follows since for a regular function f the distribution $f(U_n)$ is the uniform distribution over $f(\{0,1\}^n)$). It follows that $\Pr[(y,\overline{h})\in\mathcal{L}] \geq \delta^2/k$ as requested. □

3.3 Pseudorandom Generators from Regular One-Way Functions

After showing that the randomized iterations of a regular one-way function are hard to invert, it is natural to follow the footsteps of the BMY construction to construct a pseudorandom generator. Rather than using simple iterations of the function f, randomized iterations of f

are used instead, with fresh randomness in each application. As in the BMY case, a hardcore-bit(s) of the current input is taken at each stage. In order to keep things more readable, we start by giving our pseudorandom generator based on regular one-way functions with super-polynomial hardness (i.e., standard one-way functions). In Section 3.3.1, we generalize this result to regular one-way functions with arbitrary hardness. In particular, we get more efficient pseudorandom generators, assuming that underlying regular one-way functions are exponentially hard.

Theorem 3.6. *For $n, k \in \mathbb{N}$, let f, $\mathcal{H}$ and f^k be as in Definition 3.1, and let $b(\cdot) := gl(\cdot)_1$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12). We define G over $\{0,1\}^n \times \mathcal{H}^n \times \{0,1\}^{2n}$ as*

$$G(x, \overline{h}, r) = (b(f^1(x, \overline{h}), r), \ldots, b(f^{n+1}(x, \overline{h}), r), \overline{h}, r).$$

Assuming that f is a regular one-way function and that $\mathcal{H}$ is efficient, then G is a pseudo-random generator.

Proof. Let $b^k(x, \overline{h}, r) = b(f^k(x, \overline{h}), r)$. Corollary 3.3 yields that b^k is a hardcore function of $\widehat{f^k}(x, \overline{h}, r) = (f^k(x, \overline{h}), \overline{h}, r)$. In the following we show that any algorithm that breaks the pseudorandomness of G, violates the hardness of $b(f^{k-1})$ for some $k \in [n+1]$.

We do the proof with respect to the reordering of the output bits of G as $(r, \overline{h}, b^{n+1}, \ldots, b^1)$, as this shuffling has no effect the pseudorandomness property. Yao [34] showed using a hybrid argument that it is, up to linear factor, as hard to distinguish a pseudorandom sequence from a random one, as it is to predict the next bit of the sequence for every prefix of the sequence. Thus, it suffices to show that for every $k \in [n+1]$ it is hard to predict b^{k-1} given $(r, \overline{h}, b^{n+1}, \ldots, b^k)$. Assume toward a contradiction the existence of a PPT P for which $\Pr[P(r, \overline{h}, b^{n+1}, \ldots, b^k) = b^{k-1}] > \frac{1}{2} + \phi(n)$ for some non-negligible function $\phi(n)$. Consider the following efficient algorithm M^P for predicting b^{k-1} given $(r, \overline{h}, f^k(x, \overline{h}))$.

Algorithm 3.7. *(M^P)*

Input: $(r, h_1, \ldots, h_{k-1}, f^k(x, h_1, \ldots, h_{k-1}))$.

Operation:

1. *Choose uniformly at random $h_k, \ldots, h_n \in \mathcal{H}$,*
2. *Generate $f^{k+1}, \ldots, f^{n+1}$ from $(f^k, h_{k+1}, \ldots, h_n)$, i.e., $f^{k+j}(x, h_1, \ldots, h_n) = f^j(h_k(f^k(x, h_1, \ldots, h_{k-1})), h_{k+1}, \ldots, h_n)$.*
3. *Output $P(r, h_1 \ldots, h_n, b^{n+1}, \ldots, b^k)$*

By choosing $h_{k+1}, \ldots, h_n$ independently at random, M^P generates a series $(f^1, \ldots, f^{n+1})$ that has the same distribution as in the evaluation of G. Thus, the procedure M^P succeeds in predicting b^{k-1} with probability at least $\frac{1}{2} + \phi(n)$, in contradiction to Corollary 3.3. □

3.3.1 From Any Hardness

The next theorem generalizes Theorem 3.6 for regular one-way function with arbitrary hardness. The construction simply takes many hardcore bits at each iteration (as much as the hardness of the function allows) and the proof (omitted) follows the same lines as the proof of Theorem 3.6, using an "any hardness" variant of Corollary 3.3 (see Corollary 4.2 for a generalized version of this variant).

Theorem 3.8. *Let f, $\mathcal{H}$, f^k and gl be as in Theorem 3.6, let $\ell = \ell(n) \in [n]$ be a polynomial-time computable function and let $m(n) = \lceil n/\ell \rceil + 1$. We define G over $\{0,1\}^n \times \mathcal{H}^m \times \{0,1\}^{2n}$ as*

$$G(x, \overline{h}, r) = (gl(f^1(x, \overline{h}), r)_{1,\ldots,\ell}, \ldots, gl(f^m(x, \overline{h}), r), \overline{h}, r)_{1,\ldots,\ell}, \overline{h}, r).$$

Assuming that f is a $(T(n), \varepsilon(n))$ regular one-way function and that $\mathcal{H}$ is efficient, then G is a $(T(n) \cdot (\varepsilon'(n)/n)^c, 2^\ell \cdot \varepsilon'(n))$, where $\varepsilon'(n) = n^c \cdot \sqrt{\varepsilon(n)}$ and $c > 0$ is a universal constant.

For exponentially hard one-way regular one-way function, the above yields the following "linear stretch" generator:

Corollary 3.9. *Assume there exists a regular, exponentially hard one-way function, then there exists a $2^{\Omega(n)}$-pseudorandom generator with linear stretch.*

3.4 An Almost-Linear-Input Generator from Regular One-Way Functions

Assuming that the underlying function is one way in the usual sense (i.e., of super-polynomial hardness) and that the hash function family has linear description size, the pseudorandom generator presented in the previous section (when using Theorem 3.8) stretches a seed of length $\Theta(n^2/\log(n))$ by $\log(n)$ bits. Although this is an improvement over the GKL generator, it still translates to a rather high loss of security, since the security of the generator on $d(n)$ bits relies on the security of regular one-way function on $\sqrt{d(n)}$ bits. In this section we give a modified construction of the pseudorandom generator of Theorem 3.6, whose seed length is only $d(n) \in \Theta(n \log n)$.

Notice that the input length of the generator of Theorem 3.6 is dominated by the description of the n independent hash functions $\overline{h} = (h_1, \ldots, h_n)$. The idea of the new construction is to derandomize the choice of $\overline{h}$. Thus, $h_1, \ldots, h_n$ are no longer chosen independently, but are chosen in a way that is sufficient for the proof to go through. The derandomization uses generators against bounded-space distinguishers, and specifically we can use the generator of Nisan [29], (or that of Impagliazzo, Nisan, and Wigderson [24]). The key observation is that calculating the randomized iterate of an input can be viewed as a bounded-space algorithm, alternatively presented here as a bounded-width layered branching program. More accurately, at each step the branching program gets a random input h_i and produces $f^{i+1} = f(h_i(f^i))$. We will show that indeed when replacing $h_1, \ldots, h_n$ with the output of a generator that fools such branching programs, then the proof of security still holds (and specifically the proof of Lemma 3.4).

Theorem 3.10. *Let f, $\mathcal{H}$, f^k and b be as in Theorem 3.6. Let $v(\mathcal{H}) = 2n$ be the description length of $h \in \mathcal{H}$ and let $\mathrm{BSG} : \{0,1\}^{q(n) \in \Theta(n \log n)} \mapsto \{0,1\}^{n \cdot v(\mathcal{H})}$ be a bounded-space generator that 2^{-n}-fools every $(2n, n, v(\mathcal{H}))$-LBP.*[20] *We define G over $\{0,1\}^n \times \{0,1\}^{q(n)} \times \{0,1\}^{2n}$ as*

$$G(x, s, r) = (b(f^1(x, \mathrm{BSG}(s)), r), \ldots, b(f^{n+1}(x, \mathrm{BSG}(s)), r), s, r).$$

Assuming that f is a regular one-way function and that $\mathcal{H}$ is efficient, then G is a pseudo-random generator.

Proof outline. The proof of the derandomized version follows in the steps of the proof of Theorem 3.6. We give a high-level outline of this proof, focusing only on the main technical lemma that changes slightly. The proof first shows that given the k'th randomized iterate $f^k(x, \overline{h})$ and $\overline{h}$, it is hard to compute $f^{k-1}(x, \overline{h})$ (analogously to Lemma 3.2), only now this also holds when the hash functions are chosen as the output of the bounded-space generator. The proof is identical to the proof of Lemma 3.2, only replacing appearances of $\overline{h}$ with the seed s. Again, the key to the proof is the following technical lemma (slightly modified from Lemma 3.4):

Lemma 3.11. *For every set $\mathcal{L} \subseteq \{0,1\}^n \times \{0,1\}^{q(n)}$ with*

$$\Pr[(f^k(U_n, \mathrm{BSG}(U_{q(n)})), U_{q(n)}) \in \mathcal{L}] \geq \delta,$$

it holds that

$$\Pr[(f(U_n), U_{q(n)}) \in \mathcal{L}] \geq \delta^2/(k+1).$$

Once we know that $f^{k-1}(x, \mathrm{BSG}(s))$ is hard to compute given $f^k(x, \mathrm{BSG}(s))$ and s (for random x and s), we deduce that one cannot predict a hardcore bit $b(f^{k-1}(x, \mathrm{BSG}(s)), r)$ given $f^k(x, \mathrm{BSG}(s))$ and the seed s to the bounded-space generator. From here, the proof follows just as the proof of Theorem 3.6 in showing that the output of G is an unpredictable sequence and therefore a pseudorandom sequence.

Proof. (of Lemma 3.11) Let $q = q(n)$ and denote by $g : \{0,1\}^n \times \{0,1\}^q \mapsto \{0,1\}^n \times \{0,1\}^q$ the function taking inputs of the form (x, s) to outputs of the form $(f^k(x, \mathrm{BSG}(s)), s)$. We proceed by giving bounds on the collision probability of g. For every two inputs (x_0, s_0) and (x_1, s_1) to g, in order to have a collision we must first have that $s_0 = s_1$ which happens with probability $1/2^q$. Now, given that $s_0 = s_1 = s$ (with a random s), we analyze the probability of the event that $f^k(x_0, \mathrm{BSG}(s))$ equals $f^k(x_1, \mathrm{BSG}(s))$.

Consider the following $(2n, n, v(\mathcal{H}))$-LBP M for the input pair (x_0, x_1): the source node is labeled by $(y_1^0 = f(x_0), y_1^1 = f(x_1))$, and being on node labeled by (y_i^0, y_i^1) in the i'th layer, it does the following on input $h \in \mathcal{H}$: let $y_{i+1}^0 = f(h(y_i^0))$ and $y_{i+1}^1 = f(h(y_i^1))$. If $i < n$, it moves to the node (y_{i+1}^0, y_{i+1}^1) in the $i+1$ layer. Otherwise (i.e., $i = n$), it moves to the 1-labeled node (in the $n+1$ layer) in case $y_{n+1}^0 = y_{n+1}^1$ and to the 0-labeled node otherwise.

[20] Such generators follow from Theorem 2.7.

The LBP described above accepts (outputs 1) with probability that is exactly the desired collision probability, that is, the probability that $f^k(x_0, \overline{h}) = f^k(x_1, \overline{h})$ over *any* distribution on $\overline{h} = (h_1, \ldots, h_{k-1})$. For every pair (x_0, x_1) with $f(x_0) \neq f(x_1)$, this probability over random $\overline{h}$ was bounded in the proof of Lemma 3.4 by:

$$\Pr_{\overline{h} \leftarrow \mathcal{H}^{k-1}}[f^k(x_0, \overline{h}) = f^k(x_1, \overline{h})] \leq \frac{k-1}{|f(\{0,1\}^n)|}.$$

Since the generator fools the above LBP, then replacing the random inputs $\overline{h}$ with the output of the bounded-space generator does not change the probability of acceptance by more than $\varepsilon_{\mathrm{BSG}}(n) = 2^{-n}$. Therefore, assuming $f(x_0) \neq f(x_1)$, we have that

$$\Pr_{s \leftarrow U_q}[f^k(x_0, \mathrm{BSG}(s)) = f^k(x_1, \mathrm{BSG}(s))] \leq \frac{k-1}{|f(\{0,1\}^n)|} + \frac{1}{2^n} \tag{10}$$

When taking the probability over random (x_0, x_1), we also add the probability that $f(x_0) = f(x_1)$. Thus,

$$\Pr_{(x_0,x_1) \leftarrow U_{2n}, s \leftarrow U_q}[f^k(x_0, \mathrm{BSG}(s)) = f^k(x_1, \mathrm{BSG}(s))] \leq \frac{k}{|f(\{0,1\}^n)|} + \frac{1}{2^n} \leq \frac{k+1}{|f(\{0,1\}^n)|}.$$

Plugging the above into the calculation of the collision probability of the function g (recall that $g(x, s) = (f^k(x, \mathrm{BSG}(s)), s)$), we get:

$$\mathrm{CP}(g(U_n, U_q)) \leq \frac{k+1}{2^q \cdot |f(\{0,1\}^n)|} \tag{11}$$

Continuing the proof, we now find a lower bound on the probability of getting a collision inside the set $\mathcal{L}$, which is a lower bound on the probability of getting a collision at all. We first request that both $f^k(x_0, \mathrm{BSG}(s_0)) \in \mathcal{L}$ and $f^k(x_1, \mathrm{BSG}(s_1)) \in \mathcal{L}$, which happens with probability at least δ^2. Once inside $\mathcal{L}$, we know that the probability of collision is at least $1/|\mathcal{L}|$. Altogether:

$$\mathrm{CP}(g(U_n, U_q)) \geq \delta^2 / |\mathcal{L}| \tag{12}$$

Combining Equations (11) and (12), we get

$$\frac{|\mathcal{L}|}{2^q \cdot |f(\{0,1\}^n)|} \geq \frac{\delta^2}{k+1}.$$

But the probability of getting a value in $\mathcal{L}$ when choosing a random element in $f(\{0,1\}^n) \times \{0,1\}^q$ is exactly $\frac{|\mathcal{L}|}{2^q \cdot |f(\{0,1\}^n)|}$. Thus, $\Pr[(f(U_n), U_{q(n)}) \in \mathcal{L}] \geq \delta^2/(k+1)$ as requested. □

Remark 3.12. *It is tempting to think that one should replace Nisan generator in the above proof with the generator of Nisan and Zuckerman [30]. That generator may have seed of size $\Theta(n)$ (rather than $\Theta(n \log n)$), where $s = 2n$ as in our case. Unfortunately, with such*

a short seed, that generator incurs an error $\varepsilon_{\mathrm{BSG}}(n) = 2^{-n^{1-\gamma}}$ for some constant γ, which is too high for our proof to work. In order for the proof to go through we need that $\varepsilon_{\mathrm{BSG}}(n) < poly(n)/|f(\{0,1\}^n)|$. Interestingly, this means that we get a linear-input construction when the image size is significantly smaller than 2^n; in order to achieve a linear-input construction in the general case, we need better generators against LBPs (that have both short seed and small error).

3.4.1 Dealing with Non Length-Preserving Functions

The pseudorandom generators presented in this section assumed that the underlying regular one-way function is length preserving. We mention that this is not a necessity and outline how any regular one-way function can be used. For the simple case that f is shrinking, simply padding the output to the same length is sufficient.[21] The more interesting case is of a length-expanding one-way function f. The important point is that we want the generator to be almost linear in the length of the input to f rather than its output. In Lemma 2.9 we show how to transform an expanding one-way function f from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ into a length preserving one-way function from $\{0,1\}^{d(n)}$ to $\{0,1\}^{d(n)}$ for some $d(n) \in \Theta(n)$. This construction, however, does not maintain the regularity of the one-way function (it maintains only an approximate regularity).

For the regular case we suggest a different solution; rather than changing the underlying one-way function to be length preserving, we change the randomizing hash functions to be shrinking. That is, given a regular one-way function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, define the randomized iterate of this function with respect to a family of hash functions from $\{0,1\}^{\ell(n)}$ to $\{0,1\}^n$. The randomized iterate is now well defined, and all the proofs given above for length preserving one-way functions, readily hold for this new construction. The only problem with this approach is that the description of such hash functions is too long (it is $\Theta(\ell(n))$ instead of $\Theta(n)$). This is overcome by using an efficient family of *almost* pairwise-independent hash functions from $\ell(n)$ bits to n bits with error 2^{-n} (see Definition 2.3), which requires a description of only $\Theta(n)$ bits.

4 Pseudorandom Generators from Exponentially Hard One-Way Functions

4.1 Overview

4.1.1 The Randomized Iterate and General One-Way Functions

As mentioned in the introduction, the last randomized iteration of a general one-way function is not necessarily hard to invert, and in fact may be easy to invert. This hardness, however, is not totally diminished, it simply deteriorates in every additional iteration. By refining

[21] Dedic et al. [4] showed that in the case of a shrinking function one can actually build a generator with seed length that is a function of the output length of f rather than the (longer) input length.

the techniques used in the case of regular one-way functions, we manage to give a lower bound on this deterioration. More precisely, we show (Section 4.2) that there exists a set $\mathcal{S}$ of inputs to f^k with density at least $1/k$, such that the k'th randomized iteration is hard to invert over inputs taken from $\mathcal{S}$. As a result, a hard core bit of the k'th randomized iteration has the guarantee that with probability at least $1/k$ it is pseudorandom.

4.1.2 The Multiple Randomized Iterate

Our goal is to get a string of pseudorandom bits, and the idea is to run t independent copies of the randomized iterate (on t independent inputs). We call this the *multiple randomized iterate*. From each of the t copies, we output a hardcore bit of the k'th iteration. For this iteration this forms a string of t bits, of which t/k are expected to be random looking. The next step is to run a randomness extractor on such a string (where the output of the extractor is of length $\Omega(t/k)$). This ensures that with very high probability, the output of the extractor is a pseudorandom string of bits.

4.1.3 The Pseudorandom Generator – A First Attempt

A first attempt for the pseudorandom generator runs the multiple randomized iterate on t independent inputs, for m (to be determined later) iterations. For each $k \in [m]$, we extract $\frac{t}{2k}$ bits at the k'th iteration. These bits are guaranteed to be pseudorandom (even when given all of the values at the $(k+1)$'th iterate and all of the randomizing hash functions). Thus, outputting the concatenation of the pseudorandom strings for the different values of k forms a long pseudorandom output (by a standard hybrid argument).

This concatenation, however, is still not long enough. It is required that the output of the generator is longer than its input, which is not the case here. The input contains t strings $x_1, \ldots, x_t$ and tm hash functions. The hash functions are included in the output, so the rest of the output needs to make up for the tn bits of $x_1, \ldots, x_t$. At each iteration we output $\frac{t}{2k}$ bits which adds up to $\sum_{k=1}^{m} \frac{t}{2k}$ bits. This harmonic progression is bounded by $t \cdot \frac{\log m}{2}$, Thus, in order to exceed the tn bits of the input we need $m > 2^n$ which is far from being efficient.

4.1.4 The Pseudorandom Generator and Exponential Hardness

The failed generator from above can be remedied when the exponential hardness comes into play. It is known that if a function is 2^{cn}-one-way (for some constant $c \in (0,1)$), then it has a $2^{c'n}$-hardcore function of $c'n$ bits (for another constant c'). Thus, if the original hardness was exponential, then in the k'th iteration we can actually extract $c'n$ random looking strings, each of length $\frac{t}{2k}$. Altogether, we get that the output length is $c'n \cdot \sum_{k=1}^{m} \frac{t}{2k} \geq c'tn \cdot \log m$. Thus, for a choice of m such that $\log m > \frac{1}{c'}$, we get that the overall output is a pseudorandom string of length greater than the input. It follows that to get a $T(n)$-pseudorandom generator, the input length of the above generator is $\Theta(tn)$, where t can be taken in $\Theta(\log(m \cdot T(n)))$. In particular, in order to get a $2^{\Omega(n)}$-pseudorandom generator, one needs to take a seed of length $\Theta(n^2)$.

To sum up, we describe the full construction in a slightly different manner: one first creates a matrix of size $t \times m$, where each *row* in the matrix is generated by computing the first m randomized iterates of f (each row takes independent inputs). Now from each entry in the matrix $\Theta(cn)$ hardcore bits are computed (thus generating a matrix of hardcore bits). The final stage runs a randomness extractor on each of the *columns* of the hardcore bits matrix.[22] Moreover, the number of pseudorandom bits extracted from a column deteriorates from one iteration to another (t/k pseudorandom bits are taken at the columns associated with the k'th randomized iterate).

4.1.5 Some Remarks

- Our method also works for $2^{\phi(n)}$-one-way functions only as long as $\phi(n) \in \Omega(\frac{n}{\log n})$, but fails to handle smaller values of ϕ (as m grows, the value $\frac{1}{m}$ becomes too small, requiring t to grow substantially).
- The description in this section focuses on length-preserving one-way functions, the results can be generalized, using Lemma 2.9, to use non-length preserving functions.

4.2 The Last Randomized Iterate is (sometimes) Hard to Invert

We now formally state and prove the key observation of this section; there exists a set of inputs of significant weight for which it is hard to invert the k'th randomized iteration, even if given access to all of the hash functions leading up to this point. This statement is a generalization of Lemma 3.2 to all functions rather than just regular ones. In the following Lemma it is instructive to ignore the parameter gap (set gap $= 0$), as it will only be used in Section 5.

Lemma 4.1. *Let f, $\mathcal{H}$, k and f^k be as in Definition 3.1, let* $\text{gap} \in \{0,\ldots,n\}$ *and let* $\mathcal{S} \subseteq \{0,1\}^n \times \mathcal{H}^{k-1}$ *be defined as*

$$\mathcal{S} = \{(x,\overline{h}) \in \{0,1\}^n \times \mathcal{H}^{k-1} \colon \mathrm{D}_f(f^k(x,\overline{h})) + \text{gap} \geq \max_{j\in[k]} \mathrm{D}_f(f^j(x,\overline{h}))\}.$$

Then,

1. $\Pr[(U_n, H^{k-1}) \in \mathcal{S}] \geq 1/k$.
2. *For any algorithm* A *with*

$$\Pr[\mathsf{A}(f^k(U_n,H^{k-1}),H^{k-1}) = f^{k-1}(U_n,H^{k-1}) \wedge (U_n,H^{k-1}) \in \mathcal{S}] = \varepsilon_{\mathsf{A}}$$

it holds that

$$\Pr[H_{k-1}^{k-1}(\mathsf{A}(f(U_n),H^{k-1})) \in f^{-1}(f(U_n))] \geq \frac{\varepsilon_{\mathsf{A}}^2}{2nk\cdot 2^{\text{gap}}}.$$

[22] Note that each execution of the extractor runs on a column in which each entry consists of a single bit (rather than $\Theta(cn)$ bits). In other words, we translate each column of $\Theta(cn)$ bit-strings to $\Theta(cn)$ bit columns. This is a requirement of the proof technique.

Note that by considering a function $\text{gap} > 0$, one can increase the size of $\mathcal{S}$ at the price of weakening the hardness of f^k over it. We will use this feature in Section 5, in which we will set $\text{gap} = \Theta(\log n)$. In the current section we only make use of the choice $\text{gap} = 0$.

As in Section 3, assuming that f is a one-way function yields the following corollary:

Corollary 4.2. *Let $k = k(n), \text{gap} = \text{gap}(n) \in \mathbb{N}$, and for $n \in \mathbb{N}$ let f, $\mathcal{H}$, f^k and $\mathcal{S} = \mathcal{S}(n)$ be as in Lemma 4.1. Assuming that f is $(T(n), \varepsilon(n))$-one-way, that $\mathcal{H}$ is efficient and that $k \leq \text{poly}(n)$, then there exists a constant $c > 0$ such that the following hold:*

1. $\Pr_{(x,\overline{h}) \leftarrow \mathcal{S}(n)}[\mathsf{A}(f^k(x,\overline{h}),\overline{h}) = f^{k-1}(x,\overline{h})] \leq \varepsilon'(n) = n^c \cdot \sqrt{2^{\text{gap}} \cdot \varepsilon(n)}$ *for any algorithm A of running time $T(n) - n^c$ and large enough n, and*

2. *For $\ell = \ell(n) \in [n]$, let $hc^k \colon \{0,1\}^n \times \mathcal{H}^{k-1} \times \{0,1\}^{2n} \mapsto \{0,1\}^\ell$ be defined as $hc^k(x,\overline{h},r) = gl(f^{k-1}(x,\overline{h}),r)_{1,\ldots,\ell}$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12). Then hc^k is a $(T(n) \cdot (\varepsilon'(n)/n)^c, 2^\ell \cdot \varepsilon'(n))$-hardcore function of $\widehat{f^k}(x,\overline{h},r) = (f^k(x,\overline{h}),\overline{h},r)$ over $\{\mathcal{S}(n) \times \{0,1\}^{2n}\}_{n\in\mathbb{N}}$.*

Proof. Lemma 4.1(1) states that $\Pr[(U_n, H^{k-1}) \in \mathcal{S}(n)] \geq 1/k$. Hence, for any algorithm A with $\Pr_{(x,\overline{h}) \leftarrow \mathcal{S}(n)}[\mathsf{A}(f^k(x,\overline{h}),\overline{h}) = f^{k-1}(x,\overline{h})] > \varepsilon'(n)$, it holds that

$$\Pr[\mathsf{A}(f^k(U_n,H^{k-1}),H^{k-1}) = f^{k-1}(U_n,H^{k-1}) \wedge (U_n,H^{k-1}) \in \mathcal{S}(n)] > \frac{\varepsilon'}{k} = \frac{n^c}{k} \cdot \sqrt{2^{\text{gap}} \cdot \varepsilon}.$$

The corollary follows by applying Lemma 4.1(2) and getting

$$\begin{aligned}\Pr[H_{k-1}^{k-1}(\mathsf{A}(f(U_n),H^{k-1})) \in f^{-1}(f(U_n))] &\geq \left(\frac{n^c}{k} \cdot \sqrt{2^{\text{gap}} \cdot \varepsilon}\right)^2 / 2nk \cdot 2^{\text{gap}} \\ &= n^{2c-1}\varepsilon/2 \cdot k^3,\end{aligned} \tag{13}$$

which, for large enough c, contradicts the one-wayness of f. The the second part of the corollary immediately follows from the above and Theorem 2.12, when plugging $f = f^k$ and $g = f^{k-1}$. □

Proof. (of Lemma 4.1) Since $f^1(U_n,H^k),\ldots,f^k(U_n,H^k)$ are i.i.d. over $f(U_n)$, then by symmetry the k'th iteration has the heaviest preimage size with probability at least $1/k$. This proves the first part of the lemma. For the second part of the lemma, we assume for simplicity that A is deterministic (this can be generalized as explained in the proof of Lemma 3.2). Let

$$\text{Good}_\mathsf{A} = \{(y,\overline{h}) \in \{0,1\}^n \times \mathcal{H}^{k-1} \colon \overline{h}_{k-1}(\mathsf{A}(y,\overline{h})) \in f^{-1}(y)\} \tag{14}$$

By this notation, the assumption is that $\Pr[(f^k(U_n,H^{k-1}),H^{k-1}) \in \text{Good}_\mathsf{A} \wedge (U_n,H^{k-1}) \in \mathcal{S}] = \varepsilon_\mathsf{A}$, and the proof of Lemma 4.1 follows by the next lemma when plugging in $\mathcal{L} = \text{Good}_\mathsf{A}$ and $\delta = \varepsilon_\mathsf{A}$.

Lemma 4.3. *Let f, $\mathcal{H}$, k, f^k, gap and $\mathcal{S}$ be as in Lemma 4.1. Then for any set $\mathcal{L} \subseteq \{0,1\}^n \times \mathcal{H}^{k-1}$ with*

$$\Pr[(f^k(U_n, H^{k-1}), H^{k-1}) \in \mathcal{L} \wedge (U_n, H^{k-1}) \in \mathcal{S}] \geq \delta,$$

it holds that

$$\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}] \geq \frac{\delta^2}{2nk \cdot 2^{\text{gap}}}.$$

□

Proof. (of Lemma 4.3) Divide the outputs of the function f into n slices according to their preimage size: for every $j \in [n]$, define the j'th slice $\mathcal{S}_j = \{(x, \overline{h}) \in \mathcal{S} \mid \mathrm{D}_f(f^k(x, \overline{h})) = j\}$. Note that since $\mathcal{S}_j \subseteq \mathcal{S}$, for each $(x, \overline{h}) \in \mathcal{S}_j$ and $i \in [k]$, it holds that $\mathrm{D}_f(f^i(x, \overline{h})) \leq \mathrm{D}_f(f^k(x, \overline{h})) + \text{gap} = j + \text{gap}$. The proof of Lemma 4.3 follows the methodology of the analogous lemma for the regular case (see Lemma 3.4). More precisely, the proof studies the collision probability of f^k, only here we look at f^k when restricted to $\mathcal{S}_j$ (i.e., we work separately on each slice). Denote this as:

$$\mathrm{CP}(f^k(U_n, H^{k-1}) \wedge \mathcal{S}_j) := \Pr[(f^k(x_0, \overline{h}_0), \overline{h}_0) = (f^k(x_1, \overline{h}_1), \overline{h}_1) \wedge (x_0, \overline{h}_0), (x_1, \overline{h}_1) \in \mathcal{S}_j],$$

where $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$ are i.i.d. over $\{0,1\}^n \times \mathcal{H}^{k-1}$. We first give an upper-bound on this collision probability.

Claim 4.4.

$$\mathrm{CP}(f^k(U_n, H^{k-1}) \wedge \mathcal{S}_j) \leq \frac{k}{|\mathcal{H}|^{k-1} \cdot 2^{n-gap-j}}$$

Proof. For every two random inputs $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$ in $\{0,1\}^n \times \mathcal{H}^{k-1}$, in order to have a collision we must first have that $\overline{h}_0 = \overline{h}_1$, which happens with probability $(1/|\mathcal{H}|)^{k-1}$. Given that $\overline{h}_0 = \overline{h}_1 = \overline{h}$ (with $\overline{h} \in \mathcal{H}^{k-1}$ being uniform), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$. In the following we upper bound this probability.

If $f(x_0) = f(x_1)$, such a collision is assured. Since $(x_1, \overline{h}) \in \mathcal{S}_j$, it holds that $\mathrm{D}_f(f(x_1)) \leq \mathrm{D}_f(f^k(x_1, \overline{h})) + \text{gap} = j + \text{gap}$. Therefore, $\Pr[f(x_0) = f(x_1)] \leq 2^{j+\text{gap}-n}$. For the other cases (i.e., $f(x_0) \neq f(x_1)$), there must be an index $j \in [k-1]$ for which $f^j(x_0, \overline{h}) \neq f^j(x_1, \overline{h})$, but $f^{j+1}(x_0, \overline{h}) = f^{j+1}(x_1, \overline{h})$. Since $f^j(x_0, \overline{h}) \neq f^j(x_1, \overline{h})$, the pairwise independence of $\mathcal{H}$ yields that $\overline{h}_j(f^j(x_0, \overline{h}))$ is uniform over $\{0,1\}^n$ and independent of $\overline{h}_j(f^j(x_1, \overline{h}))$. Thus (as in the proof of the first part), it holds that $\Pr[f(\overline{h}_j(f^j(x_0, \overline{h}))) = f(\overline{h}_j(f^j(x_1, \overline{h})))] \leq 2^{j+\text{gap}-n}$. Namely, $\Pr[f^{j+1}(x_0, \overline{h}) = f^{j+1}(x_1, \overline{h}) \mid f^j(x_0, \overline{h}) \neq f^j(x_1, \overline{h})] \leq 2^{j+\text{gap}-n}$. Altogether, it holds that

$$\mathrm{CP}(f^k(U_n, H^{k-1}) \wedge \mathcal{S}_j) \leq k \cdot \frac{2^{j+\text{gap}-n}}{|\mathcal{H}|^{k-1}} = \frac{k}{|\mathcal{H}|^{k-1} \cdot 2^{n-\text{gap}-j}}.$$

□

Continuing the proof Lemma 4.3, we now give a lower-bound for the above collision probability. We seek the probability of getting a collision inside $\mathcal{S}_j$ and further restrict our calculation to collisions whose output lie in the set $\mathcal{L}_j = \{(z, \overline{h}) \in \mathcal{L} \mid \mathrm{D}_f(z) = j\}$ (this further restriction may only reduce the collision probability and thus the lower bound holds also without the restriction). In order to have such a collision, we first request that both inputs are in $\mathcal{S}_j$ and generate outputs in $\mathcal{L}_j$. Letting $\delta_j = \Pr[(f^k(U_n, \mathcal{H}^{k-1}), \mathcal{H}^{k-1}) \in \mathcal{L}_j \wedge (U_n, \mathcal{H}^{k-1}) \in \mathcal{S}_j]$, the above happens with probability δ_j^2. Once inside $\mathcal{L}_j$, we require that both outputs collide (which happens with probability at least $\frac{1}{|\mathcal{L}_j|}$). Altogether:

$$\mathrm{CP}(f^k(U_n, H^{k-1}) \wedge \mathcal{S}_j) \geq \delta_j^2 / |\mathcal{L}_j| \tag{15}$$

Combining Claim 4.4 and eq. (15), we get:

$$\frac{|\mathcal{L}_j| \cdot 2^{j+\mathrm{gap}-n}}{|\mathcal{H}|^{k-1}} \geq \delta_j^2/k \tag{16}$$

Note that when drawing a random value from $(f(U_n), H^{k-1})$, the probability of hitting an element in $\mathcal{L}_j$ is at least $2^{j-n-1}/|\mathcal{H}|^{k-1}$ (since each output in $\mathcal{L}_j$ has preimage at least 2^{j-1}). This means that

$$\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}_j] \geq |\mathcal{L}_j| \cdot 2^{j-n-1}/|\mathcal{H}|^{k-1} \tag{17}$$

and by Equation (16) we deduce that $\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}_j] \geq \frac{\delta_j^2}{k \cdot 2^{\mathrm{gap}+1}}$. Finally, the probability of hitting $\mathcal{L}$ is $\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}] = \sum_j \Pr[(f(U_n), H^{k-1}) \in \mathcal{L}_j] \geq \sum_j \frac{\delta_j^2}{k \cdot 2^{\mathrm{gap}+1}}$. Since $\sum_j \delta_j^2 \geq (\sum_j \delta_j)^2/n$ and (by definition) $\sum_j \delta_j = \delta$, it holds that $\Pr[(f(U_n), H^{k-1}) \in \mathcal{L}] \geq \frac{\delta^2}{nk \cdot 2^{\mathrm{gap}+1}}$, as claimed. □

4.3 The Multiple Randomized Iterate

In this section we consider the function $f^{t \times k}$ that consists of t independent copies of the randomized iterate f^k.

Definition 4.5 (the multiple randomized iterate)**.** *Let f, $\mathcal{H}$, k and f^k be as in Definition 3.1. For $t \in \mathbb{N}$, we define the k'th* Multiple Randomized Iterate *$f^{t \times k} : \{0,1\}^{t \times n} \times \mathcal{H}^{t \times (k-1)} \mapsto \{0,1\}^{t \times n}$ as:*

$$f^{t \times k}(\overline{x}, V) = (f^k(\overline{x}_1, V_1), \ldots, f^k(\overline{x}_t, V_t)),$$

where $\overline{x} \in \{0,1\}^{t \times n}$ and $V \in \mathcal{H}^{t \times (k-1)}$.

For each of the t outputs of $f^{t \times k}$, we look at its hardcore function hc^k. Corollary 4.2 yields that t/k of these t hardcore strings are expected to fall inside the "hard-set" of f^k (and thus are indeed pseudorandom given $(f^{t \times k}(\overline{x}, V), V)$). The next step is to invoke a randomness extractor on a concatenation of one bit from each of the different independent hardcore strings. The output of the extractor is taken to be of length $\lfloor \frac{t}{4k} \rfloor$. The intuition

being that with high probability, the concatenation of these bits from the different outputs of hc^k contains "pseudoentropy" of at least $\frac{t}{3k}$ bits. Thus, the output of the extractor forms a pseudorandom string and might serve as a hardcore function of the multiple randomized iterate $f^{t\times k}$.

Definition 4.6 (hardcore function for the multiple randomized iterate)**.** *Let k, t, f, $\mathcal{H}$ and $f^{t\times k}$ be as in Definition 4.5, let $\ell \in [n]$, $hc^k : \{0,1\}^n \times \mathcal{H}^{k-1} \times \{0,1\}^{2n} \mapsto \{0,1\}^\ell$ be as in Corollary 4.2 and let $\mathrm{Ext}_k \colon \{0,1\}^q \times \{0,1\}^t \mapsto \{0,1\}^{\lfloor \frac{t}{4k} \rfloor}$. We define $hc^{t\times k} \colon \{0,1\}^{t\times n} \times \mathcal{H}^{t\times(k-1)} \times \{0,1\}^{2n} \times \{0,1\}^q \mapsto \{0,1\}^{\ell\cdot\lfloor \frac{t}{4k} \rfloor}$ as*

$$hc^{t\times k}(\overline{x}, V, r, s) = (hc_1^{t\times k}(\overline{x}, V, r, s), \ldots, hc_\ell^{t\times k}(\overline{x}, V, r, s)),$$

where $\overline{x} \in \{0,1\}^{t\times n}$, $V \in \mathcal{H}^{t\times(k-1)}$, $r \in \{0,1\}^{2n}$, $s \in \{0,1\}^q$ and $hc_i^{t\times k}(\overline{x}, V, r, s) = \mathrm{Ext}_k(s, (hc^k(\overline{x}_1, V_1), r)_i, \ldots, (hc^k(\overline{x}_t, V_t), r)_i))$. Finally, we let $\widehat{f^{t\times k}}(\overline{x}, V, r, s) = (f^{t\times k}(\overline{x}, V), V, r, s)$.

Lemma 4.7. *Let $k = k(n), t = t(n), q = q(n), \ell = \ell(n) \in \mathbb{N}$ with $\ell(n) \leq n$, and for $n \in \mathbb{N}$ let f, $\mathcal{H}$, Ext_k, $hc^{t\times k}$ and $\widehat{f^{t\times k}}$ be as in Definition 4.6. Assume that f is $(T(n), \varepsilon(n))$-one-way, that $\mathcal{H}$ is efficient, Ext_k is a $(\lfloor \frac{t}{3k} \rfloor, \varepsilon_{\mathrm{Ext}_k})$-strong extractor and that $2(\varepsilon_{\mathrm{Ext}_k} + \rho_{\frac{1}{2k}, t, \lfloor \frac{t}{3k} \rfloor}) \leq \varepsilon(n) < 2^{-2\ell} \cdot n^{-O(1)}$, where ρ is taken as in Lemma 2.14[23]. Further, assume that $k, t, q \leq \mathrm{poly}(n)$, that ε is polynomial-time computable and that Ext_k is polynomial-time computable given k. Then $hc^{t\times k}$ is a $(T \cdot \varepsilon^{O(1)}, \ell\varepsilon)$-hardcore function of $\widehat{f^{t\times k}}$.*

Proof. We prove that for any $i = i(n) \in [\ell(n)]$, the function $hc_i^{t\times k}$ is a $(T \cdot \varepsilon^{O(1)}, \varepsilon)$-hardcore function of $g_i^{t\times k}$ defined as $g_i^{t\times k}(\overline{x}, V, r, s) = (\widehat{f^{t\times k}}(\overline{x}, V, r, s), hc^k(\overline{x}_1)_{1,\ldots,i-1}, \ldots, hc^k(\overline{x}_t)_{1,\ldots,i-1})$, and the proof of the lemma follows by a straightforward hybrid argument.

Assume there exist a distinguisher D of running-time $T' = T \cdot \varepsilon^{O(1)}$ and an index function $i = i(n) \in \mathbb{N}$, that contradict the hardness of $hc_i^{t\times k}$ with respect to $g_i^{t\times k}$ as stated above. Let $g_i^k(\cdot) = (\widehat{f^k}(\cdot), hc^k(\cdot)_{1,\ldots,i-1})$ and let $hc_i^k(\cdot) = hc^k(\cdot)_i$. Lemma 2.14 yields (plugging $m = \lfloor \frac{t}{3k} \rfloor$, $\delta = 1/2k$, $\gamma = \varepsilon/2$, $f = g_i^k$, $b = hc_i^k$ and $\mathrm{Ext} = \mathrm{Ext}_k$)[24] that there exists a PPT M with

$$\Pr\left[M(1^n, 1^{T'}, 1^k, 1^{\lfloor 2/\varepsilon \rfloor}, g_i^k(X)) = hc_i^k(X)\right] > 1 - 1/4k \tag{18}$$

for infinitely many n's, where X is uniformly distributed over $\{0,1\}^n \times \mathcal{H}^{k-1} \times \{0,1\}^{2n}$.

On the other hand, Corollary 4.2 yields that hc^k is a $(T'' = T \cdot (\varepsilon'/n)^{O(1)}, \varepsilon'' = 2^\ell \cdot \varepsilon')$ hardcore function of $\widehat{f^k}$ over a family of sets $\mathcal{S}$ of density $1/k$, where $\varepsilon' = \varepsilon'(n) = n^{O(1)} \cdot \sqrt{\varepsilon(n)}$. It easily follows that hc_i^k is a $(T''' \in \Omega(T'' \cdot \varepsilon''/n), \varepsilon''' \in O(\varepsilon''))$ hardcore predicate of g_i^k over $\mathcal{S}$,

[23] $\rho_{\delta,t,m}$ is the probability that out of t independent binomial random variables, each with probability at least δ of being one, at least m come out ones.

[24] To use Lemma 2.14, we assume without loss of generality that $\varepsilon(n) > 2^{-n/3}$ (otherwise the proof is trivial). We also assume without loss of generality that k can be efficiently extracted from the input length of g_i^k, yielding that Ext_k is polynomial-time computable given this value.

for any $i = i(n) \in [\ell(n)]$.[25] Since (by assumption) $\varepsilon < 2^{-2\ell} \cdot n^{-O(1)}$, it holds that $\varepsilon''' \in o(\frac{1}{k})$ and therefore $T''' = T \cdot \varepsilon^{O(1)}$. Thus, hc_i^k is a $(T \cdot \varepsilon^{O(1)}, \frac{1}{2} - \frac{1}{3k})$ hardcore predicate of g_i^k (over $\{0,1\}^n$) for any such i. A standard equivalence between distinguishing and prediction algorithms yields that no algorithm of running time T''' predicts $hc_i^k(\cdot)$ from $g_i^k(\cdot)$ with probability better than $1 - \frac{1}{3k}$, which, for the right constant in the definition of T' above, contradicts Equation (18). □

4.4 A Pseudorandom Generator from Exponentially Hard One-Way Functions

We are now ready to present our pseudorandom generator. After deriving a hardcore function for the multiple randomized iterate, the generator is similar to the construction from regular one-way function. That is, run randomized iterations and output the hardcore bits. The major difference in our construction is that, for starters, it uses hardcore functions rather than hardcore bits. More importantly, the amount of hardcore bits extracted at each iteration is not constant and deteriorates with every additional iteration. As in Section 3, our generator applies the standard BMY principle on the multiple randomized iterate, outputting the hardcore bits of each iteration.

Our generator is given in the following theorem, whose proof immediately follows by Lemma 4.7 and a standard BMY like indistinguishability argument (cf., the proof of Theorem 3.6).

Theorem 4.8. *Let $m = m(n) \in \mathbb{N}$, and for $k = k(n), \ell = \ell(n), t = t(n), q = q(n) \in \mathbb{N}$ with $k \leq m+1$ and $\ell \leq n$, let f, $\mathcal{H}$, Ext_k, $hc^{t\times k}$ and $f^{t\times k}$ be as in Definition 4.6. We define $G\colon \{0,1\}^{t\times n} \times \mathcal{H}^{t\times m} \times \{0,1\}^{2n} \times \{0,1\}^q \mapsto \{0,1\}^{\ell \cdot \sum_{k\in[m+1]} \lfloor \frac{t}{4k} \rfloor} \times \mathcal{H}^{t\times m} \times \{0,1\}^{2n} \times \{0,1\}^q$ as*

$$G(\overline{x}, V, r, s) = (hc^{t\times 1}(\overline{x}, V, r, s) \ldots, hc^{t\times m}(\overline{x}, V, r, s), V, r, s),$$

where $\overline{x} \in \{0,1\}^{t\times n}$, $V \in \mathcal{H}^{t\times m}$, $r \in \{0,1\}^{2n}$ and $s \in \{0,1\}^q$.

Assume that f is a $(T = T(n), \varepsilon = \varepsilon(n))$-one-way, $\mathcal{H}$ is efficient, and that Ext_k is a $(\lfloor \frac{t}{3k} \rfloor, \varepsilon_{\mathrm{Ext}_k} = \varepsilon_{\mathrm{Ext}_k}(n))$-strong extractor and $2(\varepsilon_{\mathrm{Ext}_k} + \rho_{\frac{1}{2k}, t, \lfloor \frac{t}{3k} \rfloor}) \leq \varepsilon < 2^{-\ell} \cdot n^{-O(1)}$ (where ρ is taken as in Lemma 2.14) for every $k \in [m+1]$. Further, assume that $m, t, q \leq \mathrm{poly}(n)$, that ε is polynomial-time computable, that Ext_k is polynomial-time computable given k and that G is length increasing. Then G is a $(T \cdot \varepsilon^{O(1)}, m \cdot \ell \cdot \varepsilon)$-pseudorandom generator.

The above theorem yield the following efficient generator from exponentially hard one-way functions:

[25] Let D be a distinguisher of running time T_D, which given $g_i^k(\cdot)$ tells $hc_i^k(\cdot)$ from uniform with advantage ε_D for some (infinite) sequence of indices $i = i(n)$. Allowing to err with probability $\varepsilon_D/8$, it takes $O(T_D \cdot \log(1/\varepsilon_D)/\varepsilon_D)$ time to find a value of $i \in [\ell(n)]$ on which D distinguishes the above with advantage $\varepsilon_D/2$. Given such good value of i and $\widehat{f^k}(\cdot)$, it is easy to use D to distinguish $hc^k(\cdot)$ from uniform with advantage $\varepsilon_D/4$. Hence, the above process runs in time $O(T_D \cdot \log(1/\varepsilon_D)/\varepsilon_D)$, and distinguishes $hc^k(\cdot)$ from uniform with advantage $\varepsilon_D/8$ for infinitely many n's.

Corollary 4.9. *Assuming there exists an exponentially hard one-way function, then there exists a $2^{\Omega(\sqrt{n})}$-pseudorandom generator with linear stretch.*

Proof. Let $\mathcal{H}$ and $\mathcal{W}$ be efficient families of length-preserving pairwise independent hash functions with linear description size, and for $t \in \mathbb{N}$ and $k \in [t]$, define $\mathrm{Ext}_k : \mathcal{W}_t \times \{0,1\}^t \mapsto \{0,1\}^{\lfloor \frac{t}{4k} \rfloor}$ as $\mathrm{Ext}_k(w,x) = w(x)_{1,\ldots,\lfloor \frac{t}{4k} \rfloor}$. For any such t and k, the Leftover Hash Lemma ([16, Lemma 4.8]) yields that Ext_k is a $(\lfloor \frac{t}{3k} \rfloor, 2^{-c_1 \cdot t/k})$-strong extractor, where a Chernoff bound yields that $\rho_{\frac{1}{2k},t,\lfloor \frac{t}{3k} \rfloor} < 2^{-c_2 \cdot t/k^2}$, where $c_1, c_2 > 0$ are universal constants. Let f be a 2^{cn}-one-way function, where we assume without loss of generality that $c \leq \frac{1}{2} \cdot \min\{c_1, c_2\}$ (otherwise, consider a smaller value for c). Note that $2^{-cn} > 2(\varepsilon_{\mathrm{Ext}_k} + \rho_{\frac{1}{k},t,\lfloor \frac{t}{2k} \rfloor})$ for every $t > nk^2$ and $n > 1/c$. In the following we let $t = t(n) = nk^2$, $\ell = \ell(n) = cn/2$ and let m be a constant such that $\sum_{k \in [m+1]} \frac{1}{5k} \geq 4/c$. Note that for such m and large enough t (e.g., $t = 20(m+1)$), it holds that $\ell \cdot \sum_{k \in [m+1]} \lfloor \frac{t}{4k} \rfloor \geq 2tn$.

Now let $\varepsilon = \varepsilon(n) \geq 2^{-cn}$ to be determined later by the analysis. Clearly, f is a $(T(n) = 2^{cn}, \varepsilon)$-one-way function. By plugging the above f, $\mathcal{H}$, Ext_k, t, ℓ and m into Theorem 4.8, we get a $(T \cdot \varepsilon^{O(1)}, \Omega(n \cdot \varepsilon))$-pseudorandom generator mapping strings of length $r(n) = t(n) \cdot (n + mq(n)) + 2n + q(n)$ to strings of length $r'(n) = t(n)(2n + mq(n)) + 2n + q(n)$. Since $q(n) \in \Theta(n)$, it follows that $r(n) \in \Theta(n^2)$ and that $r'(n) = r(n)(1 + \Theta(1))$. Finally, by taking $\varepsilon(n) = 2^{-c'n}$ for small enough c', we have that G is a $2^{c'n}$-pseudorandom generator. □

5 Pseudorandom Generator from Any One-Way Function

Our implementation of a pseudorandom generator from any one-way function follows the footsteps of Håstad et al. [16] (more precisely, we follow the new proof to [16] due to Holenstein [19]), though we take a totally different approach in the implementation of the initial step.

The basic building block of the [16] generator is a *pseudoentropy pair*.[26] A distribution is said to have *pseudoentropy* at least k if it is computationally indistinguishable from some distribution that has entropy k. Informally, a pair (f, b) of a function and predicate is a pseudoentropy pair if the following hold: the pseudoentropy of b's output given the output of f is noticeably larger than the real (conditional) entropy of this bit. In their construction, [16] exploit this gap between real entropy and pseudoentropy to construct a pseudorandom generator. We show that the second randomized iterate of a one-way function together with a standard hardcore predicate, forms a pseudoentropy pair with better properties than the [16] one. Hence, plugging our pseudoentropy pair as the first step of the [16] construction, results in a better overall construction. Let us now turn to a more formal discussion. A pseudoentropy pair is defined as follows:

[26] The notion of pseudoentropy pair is implicit in [16], and was formally defined in [12].

Definition 5.1 (pseudoentropy pair (PEP))**.** *Let* $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ *and* $b : \{0,1\}^n \mapsto \{0,1\}$ *be polynomial-time computable functions. The pair* (f, b) *is a* $(\lambda(n), \gamma(n))$-PEP*, if*

1. $\mathrm{H}(b(U_n) \mid f(U_n)) \leq \lambda(n)$*, and*
2. b *is a* $(T(n), \frac{1-\lambda(n)-\gamma(n)}{2})$*-hardcore predicate of* f*, for any* $T \in \mathrm{poly}$*.*

Håstad et al. [16] show how to use any one-way function in order to construct a $(\lambda, 1/2n)$-PEP, where $\lambda \in [0,1]$ is an *unknown* value. They then present a construction of a pseudorandom generator using a $(\lambda, 1/\Theta(n))$-PEP where λ is *known*. To overcome this gap, Håstad et al. [16] enumerate all possible values of λ (up to an accuracy of $1/4n$), invoke the generator with every one of these values and eventually combine all of the potential generators using an XOR of their outputs. This enumeration costs an additional factor of n to the seed length as well as n^2 times more calls to the underlying one-way function.

In Section 5.1 we prove that the second randomized iterate of a one-way function can be used to construct a $(\frac{1}{2}, \log n/n)$-PEP. In Section 5.2 we show that by combining our PEP with the second part of the construction in [19], we get a pseudorandom generator that is more efficient and has better security than the original construction of [16]/[19]. For comparison, we present the PEP used by [16] in Appendix A.

5.1 A Pseudoentropy Pair Based on the Randomized Iterate

Recall that for a given function f, we have defined (Definition 3.1) its second randomized iterate as $f^2(x,h) = f(h(f(x)))$. We would like to prove that the second iterate of a one-way function gives rise to a PEP. Indeed, since the randomized iterate maintains some of the hardness of a function (Lemma 4.1), we have that with probability $\frac{1}{2} + \Omega(\log n/n)$ it is hard to compute the value of $f(x)$ given the output $f^2(x,h)$. We want to complement this fact by saying that with probability $\frac{1}{2}$ the value of $f(x)$ can essentially be determined from the output. The latter statement is almost true, except that there typically remains a small amount of uncertainty regarding $f(x)$. To overcome this, we add to the output a small amount of random information about $f(x)$. Specifically, we define the *extended randomized iterate* to include some additional random information about $f(x)$. This information (of $\Theta(\log(n)$ bits) is small enough to diminish any entropy left in $f(x)$ (at least on $\frac{1}{2}$ of the inputs), yet is not significant enough to change its pseudoentropy.

5.1.1 The Extended Randomized Iterate

For simplicity, in the following we focus on length-preserving (one-way) functions, the adaptation to general functions is done via Lemma 2.9. We use the following extended version of the second randomized iterate:

Definition 5.2 (the extended randomized iterate)**.** *Let* $f : \{0,1\}^n \mapsto \{0,1\}^n$ *and let* $\mathcal{H}$ *and* $\mathcal{H}_E$ *be two families of pairwise-independent hash functions over* $\{0,1\}^n$*, outputting strings*

of length n and $t = t(n) = \lceil 3\log(n) + 7 \rceil$ respectively. We define $g\colon \{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E \mapsto \{0,1\}^n \times \mathcal{H} \times \{0,1\}^t \times \mathcal{H}_E$, the extended randomized iterate of *f, as:*

$$g(x, h, h_E) = (f^2(x, h), h, h_E(f(x)), h_E).$$

In the following we denote by H and H_E the random variables uniformly distributed over $\mathcal{H}$ and $\mathcal{H}_E$ respectively, and let $\mathrm{Dom}(g) = \{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E$.

The heart of this section is the following two lemmata: in Lemma 5.3 we show that for any one-way function f, with probability $\frac{1}{2} + \Omega(\frac{\log n}{n})$ it is hard to compute the value of $f(x)$ given a random output $g(x, h, h_E)$. While in Lemma 5.4, we show that the value of $f(x)$ is determined with high probability by the value of $g(x, h, h_E)$. It follows that there is more "computational entropy" in b given f than there is real entropy.

Lemma 5.3. *Let $t = t(n) \in \mathbb{N}$, and for $n \in \mathbb{N}$ let f, H, H_E and g be as in Definition 5.2. Assume that f is one way, and that H and H_E are efficient. Then for every constant $c > 0$ there exists a family of sets $\{\mathcal{S}(n) \subseteq \{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E\}_{n\in\mathbb{N}}$ of density at least $\frac{1}{2} + c \cdot \log n/n$, such that*

$$\Pr_{(x,h,h_E)\leftarrow\mathcal{S}(n)}[\mathsf{A}(g(x, h, h_E)) = f(x)] = \mathrm{neg}(n)$$

for any PPT A.

Proof. For $n \in \mathbb{N}$, define

$$\mathcal{S}(n) = \{(x, h, h_E) \in \mathrm{Dom}(g)\colon \mathrm{D}_f(f(x)) \leq \mathrm{D}_f(f^2(x, h)) + \mathrm{gap}\},$$

where $\mathrm{gap} = \mathrm{gap}(n) = 2 \cdot \lceil c \cdot \log n \rceil$.

The hardness of g over $\{\mathcal{S}(n)\}$ follows from Corollary 4.2(1) with the additional observation that since $H_E(f(x)$ is short, then $(H_E(f(x), H_E)$ can be guessed correctly with probability $1/\operatorname{poly}(n)$ when given $(f^2(x, h), h)$.

In order to lower bound the density of $\mathcal{S}(n)$, we denote by R^0 and R^1 the random variables distributed according to $\lceil \mathrm{D}_f(f(U_n))/\mathrm{gap} \rceil$ and $\lceil \mathrm{D}_f(f^2(U_n, H))/\mathrm{gap} \rceil$ respectively. Since $f(U_n)$ and $f^2(U_n, H) = f(H(f(U_n)))$ are i.i.d. over $f(U_n)$, it follows that R^0 and R^1 are i.i.d. over $[\lceil n/\mathrm{gap} \rceil]$. By symmetry,

$$\Pr[R^0 \lneq R^1] = \Pr[R^0 \gneq R^1] \tag{19}$$

and since the collision probability of a random variable X is at least $1/\operatorname{Supp}(X)$, it holds that

$$\Pr[R^0 = R^1] \geq 1/\lceil n/\mathrm{gap} \rceil \tag{20}$$

Combining the above equations yields that $\Pr[R^0 \leq R^1] \geq \frac{1}{2} + \frac{1}{\lceil n/\text{gap} \rceil}$, and therefore

$$\begin{aligned}\frac{|\mathcal{S}(n)|}{|\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E|} &= \Pr[\mathrm{D}_f(f(U_n)) \leq \mathrm{D}_f(f^2(U_n, H)) + \text{gap}] \\ &\geq \Pr[R^0 \leq R^1] \\ &\geq \frac{1}{2} + \frac{1}{\lceil n/\text{gap} \rceil} \\ &\geq \frac{1}{2} + c \cdot \log n/n,\end{aligned}$$

where the last inequality holds for large enough n. □

The following Lemma is the basis for showing that with high probability, the value of $f(x)$ is determined by the value of $g(x, h, h_E)$.

Lemma 5.4. *Let f and g be as in Definition 5.2. For $(x, h, h_E) \in \mathrm{Dom}(g)$, let $\alpha(x, h, h_E) = \Pr[f(U_n) = f(x) \mid g(U_n, H, H_E) = g(x, h, h_E)]$ and let $\mathcal{L} = \{(x, h, h_E) \in \mathrm{Dom}(g) \colon \alpha(x, h, h_E) \geq 1 - 1/16n^2\}$. Then, $|\mathcal{L}| \,/\, |\mathrm{Dom}(g)| \geq \frac{1}{2} + \frac{1}{4n}$.*

Proof. Let $\text{FirstIterIsHeavier} = \{(x, h, h_E) \in \mathrm{Dom}(g) : \mathrm{D}_f(f(x)) \geq \mathrm{D}_f(f^2(x, h)\}$. By symmetry (cf., the proof of Lemma 5.3) it holds that

$$|\text{FirstIterIsHeavier}| \,/\, |\mathrm{Dom}(g)| \geq \frac{1}{2} + \frac{1}{2n} \tag{21}$$

In addition, for any $(x, h, h_E) \in \text{FirstIterIsHeavier}$ we have that:

$$\begin{aligned}\alpha(x, h, h_E) &= \frac{\Pr[g(U_n, H, H_E) = g(x, h, h_E) \wedge f(U_n) = f(x)]}{\Pr[g(U_n, H, H_E) = g(x, h, h_E)]} \\ &= \frac{\Pr[g(U_n, h, h_E) = g(x, h, h_E) \wedge f(U_n) = f(x)]}{\Pr[g(U_n, h, h_E) = g(x, h, h_E)]} \\ &= 1 - \frac{\Pr[g(U_n, h, h_E) = g(x, h, h_E) \wedge f(U_n) \neq f(x)]}{\Pr[g(U_n, h, h_E) = g(x, h, h_E)]} \\ &\geq 1 - \frac{\Pr[g(U_n, h, h_E) = g(x, h, h_E) \wedge f(U_n) \neq f(x)]}{\Pr[f(U_n) = f(x)]} \\ &\geq 1 - \frac{\Pr[g(U_n, h, h_E) = g(x, h, h_E) \wedge f(U_n) \neq f(x)]}{2^{\mathrm{D}_f(f(x))-n-1}} \\ &\geq 1 - \underbrace{\frac{\Pr[g(U_n, h, h_E) = g(x, h, h_E) \wedge f(U_n) \neq f(x)]}{2^{\mathrm{D}_f(f^2(x,h))-n-1}}}_{\beta(x,h,h_E)},\end{aligned} \tag{22}$$

where the last inequality follows by the definition of FirstIterIsHeavier. We next show that $\beta(x, h, h_E) \leq 1/16n^2$ for *most* elements of $\mathrm{Dom}(g)$. This will conclude the proof of the lemma,

as it would follow that most elements of FirstIterIsHeavier are inside $\mathcal{L}$ (and therefore we can set $\mathcal{L}$ to be almost all of FirstIterIsHeavier). Let $\text{Typical} \subseteq \text{Dom}(g)$ be defined as

$$\text{Typical} = \{(x,h,h_E) \colon \Pr[g(U_n,h,h_E) = g(x,h,h_E) \wedge f(U_n) \neq f(x)] \leq 4n \cdot 2^{\text{D}_f(f^2(x,h))-n-t}\}$$

Note that $\beta(x,h,h_E) \leq 8n/2^t \leq 1/16n^2$ for every $(x,h,h_E) \in \text{Typical}$. Hence,

$$\alpha(x,h,h_E) \geq 1 - 1/16n^2 \tag{23}$$

for every $(x,h,h_E) \in \text{Typical} \cap \text{FirstIterIsHeavier}$. It is left to show that most elements of FirstIterIsHeavier are inside Typical.

Claim 5.5. *It holds that* $|\text{Typical}|/|\text{Dom}(g)| \geq 1 - 1/4n$.

Hence,

$$\begin{aligned}\Pr[\alpha(U_n,H,H_E) \geq 1 - 1/16n^2] &\geq |\text{FirstIterIsHeavier} \cap \text{Typical}| \,/\, |\text{Dom}(g)| \\ &\geq \frac{1}{2} + \frac{1}{2n} - \frac{1}{4n} = \frac{1}{2} + \frac{1}{4n},\end{aligned}$$

concluding the proof of the lemma. □

Proof of Claim 5.5. For $(x,z_1,z_2) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^t$, define $\mathcal{M}(x,z_1,z_2) \subseteq \mathcal{H} \times \mathcal{H}_E$ as

$$\mathcal{M}(x,z_1,z_2) = \{(h,h_E) : (f^2(x,h), h_E(f(x)) = (z_1,z_2)\} \tag{24}$$

I.e., $\mathcal{M}(x,z_1,z_2)$ contains the pairs (h,h_E) that map x to (z_1,z_2). The pairwise independence of $\mathcal{H}$ and $\mathcal{H}_E$ implies that $\Pr_{(h,h_E) \leftarrow \mathcal{M}(x,z_1,z_2)}[g(x',h,h_E) = g(x,h,h_E)] \leq 2^{\text{D}_f(z_1)-n-t}$ for any $x' \notin f^{-1}(f(x))$. Hence, a Markov inequality yields that

$$\Pr_{(h,h_E) \leftarrow \mathcal{M}(x,z_1,z_2)}[(x,h,h_E) \in \text{Typical}] \geq 1 - 1/4n \tag{25}$$

for any (x,z_1,z_2). Consider the following randomized process to generate elements inside $\text{Dom}(g)$: 1. Choose a uniform $(x,h,h_E) \in \text{Dom}(g)$, 2. Choose a uniform $(h',h'_E) \in \mathcal{M}(x,f^2(x,h),h_E(f(x)))$, and 3. Output (x,h',h'_E). Since the output of this process is uniform in $\text{Dom}(g)$, Equation (25) yields that

$$\Pr_{(x,h,h_E) \leftarrow \text{Dom}(g)}[(x,h,h_E) \in \text{Typical}] \geq 1 - 1/4n.$$

□

5.1.2 Constructing the Pseudoentropy Pair

Recall that a PEP consists of a function and a corresponding predicate. For the function we use the extended randomized iterate, and for the predicate we basically take the Goldreich-Levin hardcore bit. The twist here is that unlike a standard GL predicate, we take the hardcore bit from the intermediate value $f(x)$ rather than from the actual input x. While a hardcore bit taken from x has the required computational hardness, it may also have entropy to it. By applying the predicate on $f(x)$, we achieve the desired gap between entropy and pseudoentropy. In the following formal theorem we use a slightly modified version of g to incorporate the randomness required by the hardcore predicate. The function and predicate are proved to form a $(\frac{1}{2}, \log n/n)$-PEP.

Theorem 5.6. *Let $t = t(n) \in \mathbb{N}$, and for $n \in \mathbb{N}$ let f, $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 5.2. For $r \in \{0,1\}^{2n}$, let $g'(x,h,h_E,r) = (g(x,h,h_E),r)$ and $b(x,h,h_E,r) = gl(f(x),r)_1$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12).*

Assuming that f is one-way and that $\mathcal{H}$ and $\mathcal{H}_E$ are efficient, then (g',b) is a $(\frac{1}{2}, \log n/n)$-PEP.

Proof. The computational side of the theorem follows by Lemma 5.3 and Corollary 2.13: for $n \in \mathbb{N}$, let $\mathcal{S}(n)$ be the set of density $\frac{1}{2} + 2 \cdot \log n/n$ guaranteed by Lemma 5.3 (we choose the constant $c = 2$ when applying the lemma). By plugging $f = g$, $f' = g'$, $g(x,h,h_E) = f(x)$, $hc_1 = b$, $\mathcal{S} = \{\mathcal{S}(n)\}_{n\in\mathbb{N}}$ and $\mathcal{S}' = \{\mathcal{S}(n) \times \{0,1\}^{2n}\}_{n\in\mathbb{N}}$ to Corollary 2.13, we have that b is a hardcore predicate of g' over $\mathcal{S}'$. Since $\mathcal{S}'$ is of density $\frac{1}{2} + 2 \cdot \log n/n$, this concludes the proof of this part.

It is left to prove that $\mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) \leq \frac{1}{2}$ for any $n \in \mathbb{N}$, where W_n is uniformly distributed over $\mathrm{Dom}(g)$. By Lemma 5.4 there exists a set $\mathcal{L} \subseteq \mathrm{Dom}(g)$ of density $\frac{1}{2} + \frac{1}{4n}$, such that $\Pr_{(x',h',h'_E)\leftarrow\mathrm{Dom}(g)}[f(x') = f(x) \mid g(x',h',h'_E) = g(x,h,h_E)] > 1 - \frac{1}{16n^2}$ for every $(x,h,h_E) \in \mathcal{L}$. Hence,

$$\begin{aligned}
\mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) &= \mathrm{H}(b(W_n,U_n) \mid g(W_n),U_n) \qquad (26)\\
&= \Pr[g(W_n) \in g(\mathcal{L})] \cdot \mathrm{E}_{(z,r)\leftarrow(g(W_n),U_n),y\leftarrow g^{-1}(z)}[\mathrm{H}(b(y,r)) \mid z \in g(\mathcal{L})]\\
&\quad + \Pr[g(W_n) \notin g(\mathcal{L})] \cdot \mathrm{E}_{(z,r)\leftarrow(g(W_n),U_n),y\leftarrow g^{-1}(z)}[\mathrm{H}(b(y,r)) \mid z \notin g(\mathcal{L})]\\
&\leq (\frac{1}{2} + \frac{1}{4n}) \cdot \mathrm{H}(1/16n^2) + (\frac{1}{2} - \frac{1}{4n}),
\end{aligned}$$

where $\mathrm{H}(q)$, for $q \in [0,1]$, is the entropy of the Boolean random variable that equals 1 with probability q. Let $v = \frac{1}{16n^2}$, it follows that

$$\begin{aligned}
\mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) &\leq (\frac{1}{2} + \frac{1}{4n}) \cdot 2\left(v \cdot (1-v)\right)^{1/\ln(4)} + (\frac{1}{2} - \frac{1}{4n})\\
&< (\frac{1}{2} + \frac{1}{4n}) \cdot \left(v \cdot (1-v)\right)^{1/2} + (\frac{1}{2} - \frac{1}{4n})\\
&< \frac{1}{8n} + v + (\frac{1}{2} - \frac{1}{4n}) < \frac{1}{2}
\end{aligned}$$

for large enough n, where the first inequality follows since $\mathrm{H}(q) \leq 2(q(1-q))^{1/\ln(4)}$ (cf., [32, Thm 1.2]), and the second one follows since $2(q(1-q))^{1/\ln(4)} < (q(1-q))^{1/2}$ for small enough q (i.e., $q < 1/100$). □

5.2 The Pseudorandom Generator

The following fact is adapted from [19, Lemma 5].

Proposition 5.7. *Let $\gamma(n) > 1/n$ be polynomial-time computable function, and let (g, b) be a $(\lambda(n), \gamma(n))$-PEP over $\{0,1\}^n$. Suppose we are given two non-uniform advices α_n and β_n satisfying $\alpha_n \leq \lambda \leq \alpha_n + \gamma/4$ and $\beta_n \leq \mathrm{H}(g(U_n)) \leq \beta_n + \gamma/4$, then there exists a pseudorandom generator G over $\{0,1\}^{d(n)}$, where $d(n) \in \Theta(n^3 \cdot \log^2 n/\gamma(n)^2)$. Further, on input of length $d(n)$, G only makes oracle calls to f and b on input length n.*

Combining the above proposition and Theorem 5.6, we get the main result of this section.

Theorem 5.8. *Assume there exists a one-way function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, then there exists a pseudorandom generator over $\{0,1\}^{d(n)}$, where $d(n) \in \Theta(n^7/\log n)$. Further, on input of length $d(n)$, G only makes oracle calls to f on input length n.*

Proof. Let $\mathcal{H}$ and $\mathcal{H}_E$ be two families of efficient pairwise-independent hash functions over $\{0,1\}^n$, outputting strings of length n and $t(n)$ respectively, where $t(n) \in O(\log n)$ is as in Definition 5.2. Theorem 5.6 guarantees the existence of a $(1/2, \log n/n)$-PEP (g, b) over $\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E \times \{0,1\}^{2n}$, which makes oracle calls to f on inputs of length n. Taking $\mathcal{H}$ and $\mathcal{H}_E$ to be of linear description size and using simple padding, it follows that there exists a $(1/2, \log n/n)$-PEP (g', b') over $\{0,1\}^n$, whose oracle calls to f are on input of length $\Omega(n)$.

We would like to apply Proposition 5.7 on (g', b'), $\lambda(n) = 1/2$ and $\gamma(n) = \log n/n$, but in order to do that we need a knowledge of proper values for (α_n, β_n). Choosing $\alpha_n = \lambda(n) = 1/2$ satisfies the requirement for α_n. Still, we require a good choice of value for β_n.[27] This is overcome by enumerating the possible values of β_n at a granularity of $\gamma/4 = \log n/4n$ (that is $\frac{4n^2}{\log n}$ possible values for β_n since $\mathrm{H}(g(U_n)) \in [0, n]$) and using the following "combiner":[28]

For any $\nu \in [0,1]$, let G_ν over $\{0,1\}^{d'(n) \in \Theta(n^5)}$ be the generator resulting from applying Proposition 5.7 with $\beta_n = \nu$. Let G'_ν be the result of applying the [9] length-extension method to G_ν such that the output length of G'_ν is $t = \lceil 4n/\gamma(n) \rceil \in O(n^2/\log n)$ times longer than the input length of G_ν. Finally let $G(x_1, \ldots, x_t) = \bigoplus_{i=0}^{t} G'_{i/t}(x_i)$.

Overall, G is length expanding, polynomial-time computable and has input length $\Theta(n^7/\log n)$. Furthermore, since for some $i \in [t]$ the distribution $G'_{i/t}(U_{d'(n)})$ is pseudorandom, a straightforward hybrid argument yields that the output of G is pseudorandom as well. □

[27] Note that $\mathrm{H}(g(W_n))$ is not necessarily efficiently approximable.

[28] Such a combiner was previously used by other applications, e.g., [16, Proposition 4.17] and [19, Thm 1], and therefore we only give a high-level overview of its construction and proof.

Remark 5.9 (The efficiency improvement)**.** *The improvement in seed length by a factor of n with respect to the generator of [16, 19], is also accompanied by an efficiency improvement; first we need to generate only $\Theta(n^2/\log n)$ generator candidates. Second, in order to exceed the seeds of these candidate pseudorandom generators, the output of each of these candidates is stretched (only) by a factor of $\Theta(n^2/\log n)$. In the [16, 19] construction for comparison, the enumeration is over $\Theta(n^3/\log n)$ possible generators, and the output of each of these candidates is stretched by a factor of $\Theta(n^3/\log n)$. All in all, the efficiency difference adds up to a factor of $\Theta(n^2)$ less calls to f.*

6 Hardness Amplification of Regular One-Way Functions

In this section we present an efficient hardness amplification of any regular weak one-way function.

6.1 Overview

As mentioned in the introduction, the key to one-way function hardness amplification lies in the fact that every $(T, 1-\varepsilon)$-one-way function (i.e., ε-weak one-way, in the terminology of the introduction) has a *failing set* of density ε for every algorithm of running time (essentially) T, where the latter is a set that the algorithm fails to invert f upon. Sampling sufficiently many independent inputs to f is bound to hit *every* failing set, and thus fails every algorithm. Indeed, the basic hardness amplification of Yao [34] does exactly this. Since independent sampling requires a long input, we turn to use the randomized iterate that, together with the derandomization method, reduces the input length to $\Theta(n \log n)$.

We start, Section 6.2, by formally defining failing sets and showing that weak one-way functions admit such a set for any inversion algorithm. In Section 6.3 we show that the randomized iterate of a weak one-way function is strongly one-way, where in Section 6.4, we use a similar derandomization idea to that used in Section 3.4, to get an efficient one-way function.

6.2 Failing Sets

Definition 6.1 (failing set)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ and let A be an algorithm trying to invert f. We say that f has a $(\delta(n), \varepsilon(n))$-*failing-set* for A, if for large enough n, there exists a set $\mathcal{S}(n) \subseteq \{0,1\}^{\ell(n)}$ such that*

1. $\Pr[f(U_n) \in \mathcal{S}(n)] \geq \delta(n)$*, and*
2. $\Pr[\mathsf{A}(y) \in f^{-1}(y)] < \varepsilon(n)$ *for every $y \in \mathcal{S}(n)$.*

Claim 6.2. *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $(T(n), 1-\varepsilon(n))$-one-way function, and let $\gamma(n)$ be polynomial-time computable. Then any algorithm of running time $T(n) \cdot \gamma(n)/n^c$, where $c > 0$ is a universal constant, has a $(\varepsilon(n) - 2^{-n}, \gamma(n))$-failing-set.*

Proof. Let A be an algorithm of running time $n^c \cdot T(n) \cdot \gamma(n)$, and assume toward a contradiction that for infinitely many n's there exists a set $\mathcal{L}(n) \subseteq \{0,1\}^{\ell(n)}$ such that

1. $\Pr[f(U_n) \in \mathcal{L}(n)] \geq 1 - \varepsilon(n) + 2^{-n}$, and
2. $\Pr[\mathsf{A}(y) \in f^{-1}(y)] \geq \gamma(n)$ for every $y \in \mathcal{L}(n)$.

We next show that the above contradicts the one-wayness of f, and the claim follows by taking $\mathcal{S}(n) = \{y \in \{0,1\}^{\ell(n)} : \Pr[\mathsf{A}(y) \in f^{-1}(y)] < \gamma(n)\}$ as A's failing set. Let M^{A} be the algorithm that on input $y \in \{0,1\}^n$ invokes A for $n/\gamma(n)$ times (with independent random coins), and returns a preimage of y if any of these invocations finds one. For large enough c, the running time of M^{A} is bounded by $T(n)$, and it inverts any $y \in \mathcal{L}(n)$ with probability $1 - 2^{-n}$. Hence, M^{A} inverts f with probability $\Pr[f(U_n) \in \mathcal{L}(n)] \cdot (1 - 2^{-n}) > 1 - \varepsilon(n)$ for infinitely many n's,[29] contradicting to the one-wayness of f. □

6.3 The Basic Construction

In the following we assume for simplicity that the underlying weak one-way function is length-preserving, where the adaptation to any regular one-way function is the same as in Section 3.

As a first step, we show that the function $g(x, \overline{h}) = (f^{k+1}(x, \overline{h}), \overline{h})$, for the proper choice of k, is (strongly) one-way.[30]

Theorem 6.3. *Let $\varepsilon(n) > 1/\operatorname{poly}(n)$ be a polynomial-time computable function and let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular $(T(n), 1-\varepsilon(n))$-one-way function. Let $k = k(n) = \lceil 4n/\varepsilon(n) \rceil$, and for $n \in \mathbb{N}$ let $\mathcal{H}$ and f^{k+1} be as in Definition 3.1. Then the function $g : \{0,1\}^n \times \mathcal{H}^k \mapsto f(\{0,1\}^n) \times \mathcal{H}^k$ defined as*

$$g(x, \overline{h}) = (f^{k+1}(x, \overline{h}), \overline{h}),$$

is a $(T'(n) = T(n) \cdot (\varepsilon'(n)/n)^c, \varepsilon'(n))$-one-way function for any polynomial-time computable function ε', where $c > 0$ is a universal constant.

Proof. Assume towards a contradiction the existence of an algorithm A and a polynomial-time computable function ε_{A} that violate the one-wayness of g. Namely, A runs in time $T_{\mathsf{A}}(n) \leq T(n) \cdot (\varepsilon_{\mathsf{A}}(n)/n)^c$, where $c > 0$ is a universal constant to be determined by the analysis, and inverts g with probability larger than $\varepsilon_{\mathsf{A}}(n)$ for infinitely many n's. We consider the following algorithm M^{A} that inverts the last iteration of f^{i+1} for some $i \in [k]$:

Algorithm 6.4 (M^{A})**.**

Input: $(y, h_1 \ldots, h_i) \in f(\{0,1\}^n) \times \mathcal{H}^i$, *where* $i \in [k]$.

Operation:

[29] We assume without loss of generality that $\varepsilon(n) < 1 - 2^{-n}$, as stronger one-way functions cannot exist.

[30] The focus on f's $(k+1)$ iteration, and not on its k'th iteration as in the earlier sections, is merely for notational convenience.

1. *If $i = k$, let $w = y$.*

   Otherwise, choose uniformly at random $h_{i+1}, \ldots, h_k \in \mathcal{H}$ and let $w = f^{k-i}(h_{i+1}(y), h_{i+2}, \ldots, h_k)$.[31]

2. *Apply $\mathsf{A}(w, h_1, \ldots, h_k)$ to get $(x, h'_1, \ldots, h'_k)$.*

3. *Return $f^i(x, h_1, \ldots, h_{i-1})$.*

The proof of the theorem proceeds as follows: in Claim 6.5 we show that for every dense set $\mathcal{S} \subseteq \{0,1\}^n$, there exists $i \in [k]$ such that M^{A} inverts the last iteration of f^{i+1} with high probability, even when conditioning that the output of this iteration is inside $\mathcal{S}$. We then use a generalization of Lemma 3.2, to show that M^{A} yields an algorithm for which f has no failing set, in contradiction to the one-wayness of f.

In the following we assume for simplicity that $\varepsilon_{\mathsf{A}}(n) \in \omega(2^{-n})$. The heart of the proof lies in the following claim:

Claim 6.5. *For any $n \in \mathbb{N}$ and $\mathcal{S} \subseteq \{0,1\}^n$ with $\Pr[f(U_n) \in \mathcal{S}] \geq \varepsilon(n)/2$, there exists $i \in [k]$ such that*

$$\Pr[M^{\mathsf{A}}(f^{i+1}(U_n, H^i), H^i) = f^i(U_n, H^i) \wedge f^{i+1}(U_n, H^i) \in \mathcal{S}] \geq \frac{\varepsilon_{\mathsf{A}}(n)^2}{9k(n)^2}.$$

Before proving Claim 6.5, let us first use it for proving Theorem 6.3. For that, we use the following immediate generalization of Lemma 3.2.

Lemma 6.6. *Let f, $\mathcal{H}$, i, and f^i be as in Definition 3.1. Assume that f is regular, then for any $n \in \mathbb{N}$, any set $\mathcal{L} \subseteq \{0,1\}^n$ and any algorithm A with*

$$\Pr[\mathsf{A}(f^{i+1}(U_n, H^i), H^i) = f^i(U_n, H^i) \wedge f^i(U_n, H^i) \in \mathcal{L}] = \varepsilon_{\mathsf{A}},$$

it holds that

$$\Pr[H_i^i(\mathsf{A}(f(U_n), H^i)) \in f^{-1}(f(U_n)) \wedge f(U_n) \in \mathcal{L}] \geq \varepsilon_{\mathsf{A}}^2/(i+1).$$

Consider the following algorithm for inverting f: on input $y \in \{0,1\}^n$, algorithm B^A chooses a random $i \in [k]$ and $\overline{h} \in \mathcal{H}^i$, and returns $\overline{h}_i(M^A(y, \overline{h}))$. For any set $\mathcal{S}(n) \in \{0,1\}^n$ with $\Pr[f(U_n) \in \mathcal{S}(n)] \geq \varepsilon(n)/2$, it holds that

$$\begin{aligned}
&\Pr[B^{\mathsf{A}}(f(U_n)) \in f^{-1}(f(U_n)) \ \wedge \ f(U_n) \in \mathcal{S}(n)] \\
&= \mathrm{E}_{i \leftarrow [k]} \left[\Pr[H_i(M^{\mathsf{A}}(f(U_n), H^i)) \in f^{-1}(f(U_n)) \ \wedge \ f(U_n) \in \mathcal{S}(n)]\right] \\
&\geq \mathrm{E}_{i \leftarrow [k]} \left[(\Pr[M^{\mathsf{A}}(f^{i+1}(U_n, H^i), H^i) = f^i(U_n, H^i) \wedge f^{i+1}(U_n, H^i) \in \mathcal{S}])^2/(i+1)\right] \\
&\geq \frac{1}{k} \cdot \left(\frac{\varepsilon_{\mathsf{A}}(n)^2}{9k^2}\right)^2 \in \Omega(\varepsilon_{\mathsf{A}}^4/k^5),
\end{aligned}$$

[31] Namely, if $y = f^{i+1}(x, h_1 \ldots, h_i)$, then we set that $w = f^{k+1}(x, h_1, \ldots, h_k)$.

where the first inequality is by Lemma 6.6, and the second one is by Claim 6.5. It follows that B^A has no $(\varepsilon/2, \gamma(n) = (\varepsilon_{\mathsf{A}}/n)^{O(1)})$-failing-set. Since B^{A} runs in time $\mathrm{poly}(n) + T_{\mathsf{A}}(n)$, which by taking large enough c in the theorem statement, is smaller than $T(n) \cdot \gamma(n)/n^{c'}$ for any $c' > 0$, this is in contradiction to the one-wayness of f and Claim 6.2. □

Proof of Claim 6.5. For $(y, \overline{h}) \in f(\{0,1\}^n) \times \mathcal{H}^k$, let $\mathrm{Sib}(y, \overline{h}) = \{y' \neq y \in f(\{0,1\}^n) \colon g(f^{-1}(y'), \overline{h}) = g(f^{-1}(y), \overline{h})\})$, where $g(f^{-1}(y), \overline{h}) = g(x, \overline{h})$ for an arbitrary $x \in f^{-1}(y)$.[32] The pairwise independence of $\mathcal{H}$ yields that $\Pr[f^{k+1}(f^{-1}(y), H^k) = f^{k+1}(f^{-1}(y'), H^k)] \leq k/|f(\{0,1\}^n)|$ for any $y \neq y' \in f(\{0,1\}^n)$ (cf., the proof of Claim 3.5). Thus, for any $y \in f(\{0,1\}^n)$ it holds that $\mathrm{E}[|\mathrm{Sib}(y, H^k))|] = \sum_{y' \neq y \in f(\{0,1\}^n)} \mathrm{E}\left[f^{k+1}(f^{-1}(y'), H^k) = f^{k+1}(f^{-1}(y), H^k)\right] \leq k$, and a Markov's inequality yields that $\Pr[|\mathrm{Sib}(y, H^k))| > 2k/\varepsilon_{\mathsf{A}}] \leq \varepsilon_{\mathsf{A}}/2$ for any $y \in f(\{0,1\}^n)$. The above and our assumption about A yields that

$$\Pr[\mathrm{Success}_{\mathsf{A}}(U_n, H^k) \wedge \left|\mathrm{Sib}(f(U_n), H^k)\right| \leq 2k/\varepsilon_{\mathsf{A}}] \geq \varepsilon_{\mathsf{A}}/2, \tag{27}$$

where $\mathrm{Success}_{\mathsf{A}}(x, \overline{h}) = 1$ if $\mathsf{A}(g(x, \overline{h})) \in g^{-1}(g(x, \overline{h}))$ and zero otherwise. Let $\mathcal{S} \subseteq \{0,1\}^n$ be a set with $\Pr[f(U_n) \in \mathcal{S}] \geq \varepsilon/2$. The pairwise independence of $\mathcal{H}$ (actually one-wise independence suffices for this part) yields that $\Pr[f^{i+1}(x, (\overline{h}, H)) \in \mathcal{S}] \geq \varepsilon/2$ for every $i \in [k]$, $x \in \{0,1\}^n$ and $\overline{h} \in \mathcal{H}^{i-1}$. Hence, $\Pr[\exists i \in [k] \ : \ f^{i+1}(x, H^i) \in \mathcal{S}] > 1 - 2^{-2n}$ for any $x \in \{0,1\}^n$ (recall that $k = \lceil 4n/\varepsilon \rceil$), and a union bound yields that

$$\begin{aligned} \Pr[\mathrm{Hit}_{\mathcal{S}}(H^k)] &> 1 - |f(\{0,1\}^n)| \cdot 2^{-2n} \\ &\geq 1 - 2^{-n} \\ &\geq 1 - \varepsilon_{\mathsf{A}}/6, \end{aligned} \tag{28}$$

where $\mathrm{Hit}_{\mathcal{S}}(\overline{h}) = 1$ if for each $x \in \{0,1\}^n$ there exists $i \in [k]$ with $f^{i+1}(x, \overline{h}) \in \mathcal{S}$, and equals zero otherwise. Combining Equations (27) and (28) yields that

$$\Pr\left[\mathrm{Success}_{\mathsf{A}}(U_n, H^k) \wedge |\mathrm{Sib}(f(U_n), H^k)| \leq 2k/\varepsilon_{\mathsf{A}} \ \wedge \ \mathrm{Hit}_{\mathcal{S}}(H^k)\right] > \varepsilon_{\mathsf{A}}/2 - \varepsilon_{\mathsf{A}}/6 \geq \varepsilon_{\mathsf{A}}/3 \tag{29}$$

Let $\mathsf{A}(\cdot)_1$ be lhs of $\mathsf{A}(\cdot)$ (i.e., x). It follows that

$$\begin{aligned} &\Pr[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \wedge \ \mathrm{Hit}_{\mathcal{S}}(H^k)] \\ &\geq \Pr\left[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \ \wedge \ \mathrm{Hit}_{\mathcal{S}}(H^k) \wedge \ |\mathrm{Sib}(f(U_n), \mathcal{H}^i))| \leq 2k/\varepsilon_{\mathsf{A}}\right] \\ &\geq \Pr\left[\mathrm{Success}_{\mathsf{A}}(U_n, H^k) \ \wedge \ \mathrm{Hit}_{\mathcal{S}}(H^k) \wedge \ |\mathrm{Sib}(f(U_n), \mathcal{H}^i))| \leq 2k/\varepsilon_{\mathsf{A}}\right] \\ &\quad \cdot \Pr\left[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \mid \mathrm{Success}_{\mathsf{A}}(U_n, H^k) \wedge \mathrm{Hit}_{\mathcal{S}}(H^k) \wedge \ |\mathrm{Sib}(f(U_n), \mathcal{H}^i))| \leq 2k/\varepsilon_{\mathsf{A}}\right] \\ &\geq \frac{\varepsilon_{\mathsf{A}}}{3} \cdot \frac{1}{\frac{2k}{\varepsilon_{\mathsf{A}}} + 1} \geq \frac{\varepsilon_{\mathsf{A}}^2}{9 \cdot k}, \end{aligned} \tag{30}$$

[32] Since $g(x, \overline{h})$ is the same for every $x \in f^{-1}(y)$, the above is well defined.

where the last inequality is by Equation (29) and the regularity of f (i.e., all images of f in $\mathrm{Sib}(f(U_n), \mathcal{H}^i)) \cup \{f(U_n)\}$ occur with the same probability). It follows that

$$\Pr\left[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \ \wedge \ \exists i \in [k] \ : \ f^{i+1}(U_n, H^k) \in \mathcal{S}\right] \geq \frac{\varepsilon_{\mathsf{A}}^2}{9 \cdot k}$$

and in particular there exists $i \in [k]$ such that

$$\Pr\left[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \ \wedge \ f^{i+1}(U_n, H^k) \in \mathcal{S}\right] \geq \frac{\varepsilon_{\mathsf{A}}^2}{9 \cdot k^2} \tag{31}$$

We conclude that

$$\begin{aligned}
&\Pr[M^{\mathsf{A}}(f^{i+1}(U_n, H^i), H^i) = f^i(U_n, H^i) \wedge f^{i+1}(U_n, H^i) \in \mathcal{S}] \\
&\geq \Pr[\mathsf{A}(g(U_n, H^k))_1 \in f^{-1}(f(U_n)) \wedge \ f^{i+1}(U_n, H^k) \in \mathcal{S}] \\
&\geq \frac{\varepsilon_{\mathsf{A}}^2}{9 \cdot k^2}.
\end{aligned}$$

□

6.4 An Almost-Linear-Input Construction

In this section we derandomize the randomized iterate used in the basic construction above, to get a one-way function with input length $\Theta(n \log n)$. For that we use the bounded-space generator of either [29] or [24] (see Theorem 2.7).

Theorem 6.7. *Let f, k, $\mathcal{H}$ and f^{k+1} be as in Theorem 6.3. Let $v(\mathcal{H}) = 2n$ be the description length of $h \in H$ and let $\mathrm{BSG} : \{0,1\}^{q(n) \in \Theta(n \log n)} \mapsto \{0,1\}^{mv(\mathcal{H})}$ be a bounded-space generator that 2^{-2n}-fools every $(2n, k, v(\mathcal{H}))$-LBP.*

Then $g : \{0,1\}^n \times \{0,1\}^{q(n)} \mapsto \{0,1\}^n \times \{0,1\}^{q(n)}$ defined as $g(x,s) = (f^{k+1}(x, \mathrm{BSG}(s)), s)$, is a $(T(n) \cdot (\varepsilon'(n)/n)^{O(1)}, \varepsilon'(n))$-one-way function for any polynomial-time computable function ε'.

For polynomially weak one-way functions, the above yields the following result.

Corollary 6.8. *Let $\varepsilon(n) > 1/\mathrm{poly}(n)$ be a polynomial-time computable function and let $f\colon \{0,1\}^n \mapsto \{0,1\}^n$ be a polynomial-time computable regular function, which is $(T(n), 1 - \varepsilon(n))$-one-way for every $T \in \mathrm{poly}$. Then g of Theorem 6.7 is one way.*

Proof. If g is not one-way, then it is not $(1/p(n))$-one-way for some $p \in \mathrm{poly}$, in contradiction to Theorem 6.7 (taking $\varepsilon'(n) = 1/p(n)$). □

Proof idea of Theorem 6.7: The proof of the derandomized version follows the proof of Theorem 6.3. Through the proof of Theorem 6.3, the structure of $\mathcal{H}^k$ is used to derive the following facts:

1. For any $i \in [k+1]$ and $\mathcal{L} \subseteq \{0,1\}^n$

$$\Pr[(f(U_n), H^k) \in \mathcal{L}] \geq \Pr[(f^i(U_n, H^k), H^k) \in \mathcal{L}]^2/i$$

2. For any $y \neq y' \in f(\{0,1\}^n)$

$$\Pr[f^{k+1}(f^{-1}(y), H^k) = f^{k+1}(f^{-1}(y'), H^k)] \leq k/\left|f(\{0,1\}^n)\right|$$

3. For any $x \in \{0,1\}^n$ and $\mathcal{S} \subseteq \{0,1\}^n$ with $\Pr[f(U_n) \in \mathcal{S}] \geq \varepsilon/2$

$$\Pr[\exists i \in [k] \ : \ f^{i+1}(x, H^k) \in \mathcal{S}] \geq 1 - 2^{-2n}.$$

Fact 1. was used to prove Lemma 6.6, and facts 2. and 3. were used to prove Claim 6.5 (specifically, to derive Equations (27) and (28) respectively). As in the proof of Theorem 3.10, the following hold with respect to the derandomized version of $\mathcal{H}^k$:

1' For any $i \in [k+1]$ and $\mathcal{L} \subseteq \{0,1\}^n$

$$\Pr[(f(U_n), \mathrm{BSG}(U_{q(n)})) \in \mathcal{L}] \geq \Pr[(f^i(U_n, \mathrm{BSG}(U_{q(n)})), \mathrm{BSG}(U_{q(n)})) \in \mathcal{L}]^2/(i+1)$$

2' For any $y \neq y' \in f(\{0,1\}^n)$

$$\begin{aligned}\Pr[f^{k+1}(f^{-1}(y), \mathrm{BSG}(U_{q(n)})) = f^{k+1}(f^{-1}(y'), \mathrm{BSG}(U_{q(n)}))] \\ \leq \frac{k}{|f(\{0,1\}^n)|} + 2^{-2n} < \frac{k+1}{|f(\{0,1\}^n)|}\end{aligned}$$

3' For any $x \in \{0,1\}^n$ and $\mathcal{S} \subseteq \{0,1\}^n$ with $\Pr[f(U_n) \in \mathcal{S}] \geq \varepsilon/2$

$$\Pr[\exists i \in [k] \ : \ f^{i+1}(x, \mathrm{BSG}(U_{q(n)})) \in \mathcal{S}] \geq 1 - 2^{-2n} - 2^{-2n} = 1 - 2^{1-2n}$$

Item 1.′ follows from Lemma 3.11, item 2.′ is an immediate conclusion of Equation (10) (appears in the proof of Lemma 3.11) and the proof item 3.′ is an easy variant of the proof of (2′).[33] Since the proofs of Lemma 6.6 and Equations (27) and (28) still hold even under the weaker guarantees above (in the case of Lemma 6.6, there is an insignificant loss in the parameters), then the proof of the main theorem (Theorem 6.3) goes through.

Acknowledgments

We are grateful to Oded Goldreich, Moni Naor, Asaf Nussbaum, Eran Ofek and Ronen Shaltiel for helpful conversations. We thank Tal Moran, Ariel Gabizon and the anonymous referees for useful comments on preliminary versions of this paper.

[33] Assuming that (3′) does not hold with respect to some $y \in f(\{0,1\}^n)$ and $\mathcal{S} \subseteq \{0,1\}^n$, the (bounded-width layered) branching program that outputs 1 iff one of the iterations of $f^{k+1}(f^{-1}(y), \overline{h})$ hits $\mathcal{S}$, distinguishes between H^k and $\mathrm{BSG}(U_{q(n)})$ with advantage larger than 2^{-2n}. (Recall that $k = \lceil 4n/\varepsilon(n) \rceil$.)

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36, 2006.

[2] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 112–117, 1982.

[3] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, Apr. 1979.

[4] N. Dedic, D. Harnik, and L. Reyzin. Saving private randomness in one-way functions and pseudorandom generators. In *Theory of Cryptography, Fourth Theory of Cryptography Conference, TCC 2008*, pages 607–625, 2008.

[5] G. Di Crescenzo and R. Impagliazzo. Security-preserving hardness-amplification for any regular one-way function. In *31st STOC*, pages 169–178, 1999.

[6] O. Goldreich. On security preserving reductions — revised terminology. Technical Report 2000/001, Cryptology ePrint Archive, 2000.

[7] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

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

[9] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[10] O. Goldreich, R. Impagliazzo, L. A. Levin, R. Venkatesan, and D. Zuckerman. Security preserving amplification of hardness. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 318–326, 1990.

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

[12] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. In *Advances in Cryptology – CRYPTO 2006*, 2006.

[13] I. Haitner, D. Harnik, and O. Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In *Automata, Languages and Programming, 24th International Colloquium, ICALP*, 2006.

[14] I. Haitner, O. Reingold, and S. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, 2010.

[15] J. Håstad. Pseudo-random generators under uniform assumptions. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 387–394, 1990.

[16] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions in *STOC'89* and *STOC'90*.

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

[18] T. Holenstein. Key agreement from weak bit agreement. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2005.

[19] T. Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, 2006.

[20] T. Holenstein. Strengthening key agreement using hard-core sets – PhD thesis, 2006.

[21] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 230–235, 1989.

[22] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[23] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 12–24, 1989.

[24] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–364, 1994.

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

[26] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[27] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 1993.

[28] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version in *CRYPTO'89*.

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

[30] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[31] S. J. Phillips. Security preserving hardness amplification using PRGs for bounded space. Preliminary Report, Unpublished, 1992.

[32] F. Topsøe. Bounds for entropy and divergence for distributions over a two-element set. *Journal of Inequalities in Pure and Applied Mathematics*, 2001.

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

[34] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

A The Pseudoentropy Pair of Håstad et al.

In the following we complete the picture of Section 5, presenting the pseudoentropy pair used in [16, 19].[34]

Construction A.1 (the Håstad et al. PEP)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a one-way function and let $\mathcal{H}$ be an efficient family of length-preserving pairwise-independent hash functions over $\{0,1\}^n$. We define the function and predicate f_{H} and b_{H} over $\{0,1\}^n \times \mathcal{H} \times [n]$ as*

- $f_{\mathrm{H}}(x,h,i) = (f(x), h(x)_{1,\ldots,i+2\lceil\log(n)\rceil}, h, i)$*, and*
- $b_{\mathrm{H}}(x,h,i) = gl(x)_1$*,*

where gl is the Goldreich-Levin hardcore function (see Theorem 2.12).

[16, 19] proved that $(f_{\mathrm{H}}, b_{\mathrm{H}})$ is a $(\rho + 1/2n, 1/2n)$-PEP, for $\rho = \Pr_{(x,i)\leftarrow(U_n,[n])}[\mathrm{D}_f(f(x)) < i]$. The proof first shows that if $i \leq \mathrm{D}_f(f(x))$, then it is hard to predict $b_{\mathrm{H}}(x,h,i)$.[35] Where

[34] The following is implicit in [16], and is formally shown in [19].

[35] In such a case, $(h_{i+2\lceil\log(n)\rceil}(x), h, i)$ does not contain any noticeable information about x. It follows that it is essentially as hard to predict $b_{\mathrm{H}}(x,h,i) = gl(x)_1$ given $f_{\mathrm{H}}(x,h,i)$, as it is to predict $gl(x)_1$ given $f(x)$.

since the probability that $i = \mathrm{D}_f(x)$ is $1/n$, it follows that b_{H} is a $\frac{1-\rho-1/n}{2}$-hardcore predicate of f_{H}.

On the other hand, the pairwise independence of $\mathcal{H}$ yields that if $i \geq \mathrm{D}_f(x)$, then there is almost no entropy (i.e., less then $1/2n$) in $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$. Thus, the entropy of $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$ is not more than $\rho + \frac{1}{2n}$.