Electronic Colloquium on Computational Complexity, Report No. 135 (2005)

On the Power of the Randomized Iterate

Iftach Haitner[*] Danny Harnik[*] Omer Reingold[*]

November 19, 2005

Abstract

We consider two of the most fundamental theorems in Cryptography. The first, due to Håstad et. al. [HILL99], is that pseudorandom generators can be constructed from any one-way function. The second due to Yao [Yao82] states that the existence of weak one-way functions (i.e. functions on which every efficient algorithm fails to invert with some noticeable probability) implies the existence of full fledged one-way functions. These powerful plausibility results shape our understanding of hardness and randomness in Cryptography. Unfortunately, the reductions given in [HILL99, Yao82] are not as security preserving as one may desire. The main reason for the security deterioration is the input blow up in both of these constructions. For example, given one-way functions on n bits one obtains by [HILL99] pseudorandom generators with seed length $\Omega(n^8)$.

This paper revisits a technique that we call the *Randomized Iterate*, introduced by Goldreich, et. al. [GKL93]. This technique was used in [GKL93] to give a construction of pseudorandom generators from *regular* one-way functions. We simplify and strengthen this technique in order to obtain a similar reduction where the seed length of the resulting generators is as short as $\mathcal{O}(n \log n)$ rather than $\Omega(n^3)$ in [GKL93]. Our technique has the potential of implying seed-length $\mathcal{O}(n)$, and the only bottleneck for such a result is the parameters of current generators against space bounded computations. We give a reduction with similar parameters for security amplification of *regular* one-way functions. This improves upon the reduction of Goldreich et al. [GIL$^+$90] in that the reduction does not need to know the regularity parameter of the functions (in terms of security, the two reductions are incomparable). Finally, we show that the randomized iterate may even be useful in the general context of [HILL99]. In Particular we use the randomized iterate to replace the basic building block of the [HILL99] construction. Interestingly, this modification improves efficiency by an n^3 factor and reduces the seed length by a factor of n (which also implies improvement in the security of the construction).

[*]Dept. of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: {`Iftach.haitner, danny.harnik, omer.reingold`}`@weizmann.ac.il`.

ISSN 1433-8092

1 Introduction

In this paper we address two fundamental problems in cryptography: constructing pseudorandom generators from one-way functions and 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 that we call the *Randomized Iterate* was introduced by Goldreich, Krawczyk and Luby [GKL93] in the context of constructing pseudorandom generators from regular one-way functions. We revisit this method, both simplify existing proofs and utilize our new view to achieve significantly better parameters for security and efficiency. We further expand the application of the randomized iterate to constructing pseudorandom generators from any one-way function. Specifically we revisit the seminal paper of Håstad, Impagliazzo, Levin and Luby [HILL99] and show that the randomized iterate can help improve the parameters within. Finally, we use the randomized iterate method to both simplify and 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 function to subsection 1.2.

1.1 Pseudorandom Generators and the Randomized Iterate

Pseudorandom Generators, a notion first introduced by Blum and Micali [BM82] and stated in its current, equivalent form by Yao [Yao82], 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 [Nao91], pseudorandom functions [GGM86] and pseudorandom permutations [LR88], to name a few.

The first construction of a pseudorandom generator was given in [BM82] based on a particular one-way function and was later generalized in [Yao82] into a construction of a pseudorandom generator based on any one-way permutation. We refer to the resulting construction as the BMY construction. 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 i^{th} iterate recursively as $x^i = f(x^{i-1})$ where $x^0 = f(x)$. 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-Levin [GL89] predicate), then the BMY generator on seed x outputs the hardcore-bits $b(x^0), \ldots, b(x^\ell)$.

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 [Lev87] 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. However, a general one-way function 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. [GKL93] that showed a construction of a pseudorandom generator based on any *regular one-way function* (referred to as the GKL generator). A regular function is a function such that every element in its image has the same number of preimages. The GKL generator uses the technique at the core of this paper, 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 (Informal): (The Randomized Iterate) *For function f, input x and random hash functions $h_1, \ldots, h_\ell$, recursively define the i^{th} randomized iterate (for $i \leq \ell$) by:*

$$f^i(x, h_1, \ldots, h_\ell) = x^i = f(h_i(x^{i-1}))$$

where $x^0 = f(x)$.

The rational is that $h_i(x^i)$ is now uniformly distributed, and the challenge is to show that f, when applied to $h_i(x^i)$, is hard to invert even when the randomizing hash functions $h_1, \ldots, h_\ell$ are made public. Once this is shown, the generator is similar in nature to the BMY generator (the generator outputs $b(x^0), \ldots, b(x^\ell), h_1, \ldots, h_\ell$).

Finally, Håstad et al. [HILL99] (combining [ILL89, Hås90]), culminated this line of research by showing a construction of a pseudorandom generator using any one-way function (called here 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 pseudo-entropy, and the implicit use of family of pairwise-independent hash functions as randomness extractors. We note that HILL departs from GKL in its techniques, taking a significantly different approach.

1.1.1 The Complexity and Security of the 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 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 2.8 we elaborate on the security of cryptographic reductions and the classification of reductions in terms of their security. In this discussion, however, we focus only on one central parameter, which is the length m 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 $m = \mathcal{O}(n)$, the GKL generator takes a seed of length $m = \Omega(n^3)$ while the HILL construction produces a generator with seed length on the order of $m = \Omega(n^8)$.[1]

The length of the seed 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 m bits has security that is at best comparable to the security of the underlying one-way function, but on only $\mathcal{O}(\sqrt[8]{m})$ 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]The seed length actually proved in [HILL99] is $\mathcal{O}(n^{10})$, however it is mentioned that a more careful analysis can get to $\mathcal{O}(n^8)$.

1.1.2 Our Results on Pseudorandom Generators

Regular One-Way Functions: We give a construction of a pseudorandom generator from any regular one-way function with seed length $\mathcal{O}(n \log n)$. We note that our approach has the potential of reaching a construction with a linear seed, the bottleneck being the efficiency of the current 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 which is pairwise-independent rather than n-wise independent (as used in [GKL93]). This gives a construction with seed length $m = \mathcal{O}(n^2)$.
- The new proof allows for the derandomization of the choice of the randomizing hash functions via the *bounded-space generator* of Nisan [Nis92], further reducing the seed length to $m = \mathcal{O}(n \log n)$.

The proof method: Following is a high-level description of our proof method. For simplicity we focus on a single randomized iteration, that is on $x^1 = f^1(x, h) = f(h(f(x)))$. In general, the main task at hand is to show that it is hard to find $x^0 = f(x)$ when given $x^1 = f^1(x, h)$ and h. This follows by showing that any procedure A for finding x^0 given (x^1, h) enables to invert the one-way function f. Specifically, we show that for a random image $z = f(x)$, if we choose a random and independent hash h' and feed the pair (z, h') to A, then A is likely to return a value $f(x')$ such that $h'(f(x')) \in f^{-1}(z)$ (and thus we obtain an inverse of z).

Ultimately, we assume that A succeeds on the distribution of (x^1, h) where h is such that $x^1 = f^1(x, h)$, and want to prove A is also successful on the distribution of (x^1, h') where h' is chosen independently. Our proof is inspired by a technique used by Rackoff in his proof of the Leftover Hash Lemma (in [IZ89]). Rackoff proves that a distribution is close to uniform by showing that it has *collision-probability*[2] that is very close to that of the uniform distribution. We would like to follow this scheme and consider the collision-probability of the two aforementioned distributions. However, in our case the two distributions could actually be very far from each other. Yet, with the 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 bounded-space follows directly from the new proof. In particular, consider the procedure that takes two random inputs x_0 and x_1 and random $h_1, \ldots, h_\ell$, and compares $f^\ell(x_0, h_1, \ldots, h_\ell)$ and $f^\ell(x_1, h_1, \ldots, h_\ell)$. This procedure can be run in bounded-space since it simply needs to store the two intermediate iterates at each point. Also, this procedure accepts with probability that is exactly the collision-probability of $(f^\ell(x, h_1, \ldots, h_\ell), h_1, \ldots, h_\ell)$. Thus, replacing $h_1, \ldots, h_\ell$ with the output of a bounded-space generator cannot change the acceptance rate by much, and the collision-probability is thus unaffected. The proof of security of the derandomized pseudorandom generator now follows as in the proof when using independent randomizing hash functions.

Any One-Way Function: The HILL generator takes a totally different path than the GKL generator. 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. Interestingly, this method also

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

improves the efficiency by an n^3 factor and reduces the seed length by a factor of n (which also implies improvement in the security of the construction) over the original HILL generator.

Unlike in the case of regular functions, the hardness of inverting the randomized iterate deteriorates quickly when using any one-way function. Therefore we use only the first randomized iterate of a function, that is $x^1 = f(h(f(x)))$. Denote the degeneracy of y by $D_f(y) = \lceil \log |f^{-1}(y)| \rceil$ (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 [GL89]). Loosely speaking, we consider the bit $b(x^0)$ when given the value (x^1, h) (recall that $x^0 = f(x)$) and make the following observation: When $D_f(x^0) \geq D_f(x^1)$ then $b(x^0)$ is (almost) fully determined by (x^1, h), as opposed to when $D_f(x^0) < D_f(x^1)$ where $b(x^0)$ is essentially uniform. But in addition, when $D_f(x^0) = D_f(x^1)$ then $b(x^0)$ 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 $D_f(x^0) = D_f(x^1)$ the behavior is close to that of a regular function.

As a corollary we get that the bit $b(x^0)$ has entropy of no more than $\frac{1}{2}$ (the probability of $D_f(x^0) < D_f(x^1)$), but has entropy of at least $\frac{1}{2} + \frac{1}{\mathcal{O}(n)}$ in the eyes of any computationally-bounded observer (the probability of $D_f(x^0) \leq D_f(x^1)$). In other words, $b(x^0)$ has entropy $\frac{1}{2}$ but *pseudo-entropy* of $\frac{1}{2} + \frac{1}{\mathcal{O}(n)}$. It is this gap of $\frac{1}{\mathcal{O}(n)}$ between the entropy and pseudo-entropy that eventually allows the construction of a pseudorandom generator.

Indeed, a function with similar properties lies at the basis of the HILL construction. HILL give a different construction that has entropy p but pseudo-entropy of at least $p + \frac{1}{\mathcal{O}(n)}$. However, in the HILL construction the entropy threshold p is unknown (i.e., not efficiently computable), while with the randomized iterate the threshold is $\frac{1}{2}$. This is a real advantage since knowledge of this threshold is essential for the overall construction. To overcome this, the HILL generator enumerates all values for p (up to an accuracy of $\Omega(\frac{1}{n})$), runs the generator with every one 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 as well an additional factor of n^3 to the number of calls to the underlying function f.

On pseudorandomness in NC^1: For the most part, the HILL 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), 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 need for one of these hints, and thus can be viewed as a step towards achieving generators in NC^1 from any one-way function in NC^1 (see [AIK04] for the significance of such a construction).

1.2 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 [IL89]) 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 α-weak one-way function where $\alpha(n)$ is a polynomial fraction. This is a function that any efficient algorithm fails to invert on almost an $\alpha(n)$ fraction of the inputs. This definition is significantly weaker, however, Yao [Yao82] showed how to convert any weak one-way function into a strong one. 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 at least $\omega(1)/\alpha(n)$ to the input length of the strong function[3], which translates to a significant loss in the security (as in the case of pseudorandom generators).

With this security loss in mind, several works have tried to present an efficient method of amplification from weak to strong. Goldreich et al. [GIL+90] give a solution for one-way permutations that has just a linear blowup in the length of the input. This solution generalizes to "known-regular" one-way functions (regular functions whose image size is efficiently computable), where its input length varies according to the required security. The input length is linear when security is at most $2^{\Omega(\sqrt{n})}$, but deteriorates up to $\mathcal{O}(n^2)$ when the required security is higher (e.g., security $2^{\mathcal{O}(n)}$).[4] Their construction uses a variant of randomized iterates where the randomization is via one random step on an expander graph.

1.2.1 Our Contribution to Hardness Amplification

We present an alternative efficient hardness amplification for regular one-way functions based on the randomized iterate. Our construction is arguably simpler and has the following advantages:

1. While the [GIL+90] construction works only for known regular weak one-way functions, our amplification works for any regular weak one-way functions (whether its image size is efficiently computable or not).

2. The input length of the resulting strong one-way function is $\mathcal{O}(n \log n)$ regardless of the required security. Thus, for some range of the parameters our solution is better than that of [GIL+90] (although it is worse than [GIL+90] for other ranges).

Note that our method may yield an $\mathcal{O}(n)$ input construction 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, called here the *failing-set* of this algorithm. The idea is that a large enough number of randomly chosen inputs are bound to hit every such failing-set and thus fail every algorithm. Taking independent random samples works well, but when trying to generate the inputs to f sequentially this rationale fails. The reason is that sequential applications of f are not likely to give random output, and hence are not guaranteed to hit a failing-set. Instead, the natural solution is to use randomized iterations. However, it might be easy for an inverter to find some choice of randomizing hash functions so that all the iterates are outside of the required failing-set. To overcome this, the randomizing hash functions are also added to the output, and thus the inverter is required to find an inverse that includes the original randomizing hash functions. In the case of permutations it is obvious that outputting the randomizing hash functions is harmless, and thus the k^{th} randomized iterate of a weak one-way permutation is a strong one-way permutation. However, the case of regular functions requires our analysis that shows that the randomized iterate of a regular one-way function remains hard to invert when the randomizing hash functions are public. We also note that the proof for regular functions has another subtlety. For permutations the randomized iterate remains a permutation and therefore has only a single inverse. Regular functions, on the other hand, can have many inverses. This comes into play in the proof, when an inverting algorithm might not return the right inverse that is actually needed by the proof.

[3]The $\omega(1)$ factor stands for the logarithm of the required security. For example, if the security is $2^{\mathcal{O}(n)}$ then this factor of order n.

[4]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.

A major problem with the randomized iterate approach is that choosing fully independent randomizing hash functions requires an input as long as that of Yao's solution (an input of length $\mathcal{O}(n \cdot \omega(1)/\alpha(n))$). What makes this approach appealing after all, is the derandomization of the hash functions using space-bounded generators, which reduces the input length to only $\mathcal{O}(n \log n)$. Note that in this application of the derandomization, it is required that the bounded-space generator not only approximate the collision-probability well, but also maintain the high probability of hitting any failing-set.

We note that there have been several attempts to formulate such a construction, using all of the tools mentioned above. Goldreich et al. [GIL$^+$90] did actually consider following the GKL methodology, but chose a different (though related) approach. Phillips [Phi93] gives a solution with input length $\mathcal{O}(n \log n)$ using bounded-space generators but only for the simple case of permutations (where [GIL$^+$90] has better parameters). Di Crescenzo and Impagliazzo [DI99] give a solution for regular functions, but only in a model where public randomness is available (in the mold of [HL92]). Their solution is based on pairwise-independent hash functions that serve as the public randomness. We are able to combine all of these ingredients into one general result, perhaps due to our simplified proof.

Additional Issues:

- **On Non-Length-Preserving Functions:** The paper focuses on length-preserving one-way functions. We also demonstrate how our proofs may be generalized to use non-length preserving functions. This generalization requires the use of a construction of a family of *almost* pairwise-independent hash functions.

- **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 [HL92]) allows the use of public random coins that are not regarded a part of the input. However, our results introduce significant savings in the amount of public randomness that is necessary.

Paper Organization: Section 2 includes the formal definitions and notations used throughout this paper. In Section 3 we present our construction of pseudorandom generators from regular one-way functions. In Section 4 we present our improvement to the HILL construction of pseudorandom generators from any one-way function, for completeness we give in Appendix A a high-level overview of the original construction. Finally, in Section 5 we present our hardness amplification of regular one-way functions.

2 Preliminaries

2.1 Notations

A function $\mu : \mathbb{N} \to [0,1]$ is *negligible* if for every polynomial p we have that $\mu(n) < 1/p(n)$ for large enough n. To denote that μ is negligible we simply write $\mu(n) \in neg(n)$.

Given a function $f : \{0,1\}^* \to \{0,1\}^*$, we denote by $Dom(f)$ and $Im(f)$ the domain and image of f respectively. Let $y \in Im(f)$, we denote the preimages of y under f by $f^{-1}(y)$. The **degeneracy** of f on y is defined by $D_f(y) \overset{\text{def}}{=} \lceil \log |f^{-1}(y)| \rceil$.

We denote by $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, where ℓ is a function from $\mathbb{N}$ to $\mathbb{N}$, the ensemble of functions $\{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n \in \mathbb{N}}$.

Finally, PPT stands for probabilistic-polynomial time Turing machine.

2.2 Distributions and Entropy

We denote by U_n the uniform distribution over $\{0,1\}^n$. Given a function $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, we denote by $f(U_n)$ the distribution over $\{0,1\}^{\ell(n)}$ induced by f operating on the uniform distribution. Let D be a distribution over some finite domain X, we use the following "measures" of entropy:

- The Shannon-entropy of D is $H(D) = \sum_{x \in X} D(x) \log \frac{1}{D(x)}$.
- The collision-probability of D is $CP(D) = \sum_{x \in X} D(x)^2$.
- The min-entropy of D is $H_\infty(D) = min_{x \in X} \log \frac{1}{D(x)}$.

Two distributions P and Q over Ω are ε-close (or have statistical distance ε) if for every $A \subseteq \Omega$ it holds that $|\Pr_{x \leftarrow P}(A) - \Pr_{x \leftarrow Q}(A)| \le \varepsilon$.

By a *Distribution Ensemble* we mean a series $\{D_n\}_{n \in \mathbb{N}}$ where D_n is a distribution over $\{0,1\}^n$. Let $\{X_n\}$ and $\{Y_n\}$ be distribution ensembles. $\{X_n\}$ and $\{Y_n\}$ are *computationally-indistinguishable* if for every PPT M,

$$|\Pr[M(1^n, X_n) = 1] - \Pr[M(1^n, Y_n) = 1]| \in neg(n)$$

where the probability is taken over the distributions X_n and Y_n, and the randomness of M.

2.3 Family Of Pairwise-Independent Hash Functions

Definition 2.1 *(Efficient family of pairwise-independent hash functions) Let $\mathcal{H}$ be a collection of functions where each function $h \in \mathcal{H}$ is from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$. $\mathcal{H}$ is an* efficient family of pairwise-independent hash functions *if $|h|$ (i.e., the description length of h) and $\ell(n)$ are polynomials in n, each $h \in \mathcal{H}$ is a polynomially-computable function, and for all n, for all $x \neq x' \in \{0,1\}^n$ and all $y, y' \in \{0,1\}^{\ell(n)}$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = y \bigwedge h(x') = y'] = \frac{1}{2^{2\ell(n)}}$$

There are various constructions of efficient families of pairwise-independent hash functions for any values of n and $\ell(n)$ whose description length (i.e., $|h|$) is linear in n (e.g., [CW77]). In this paper we also make use of the special case in which $\ell(n) = n$ (the hash is length preserving). In such a case $\mathcal{H}$ is called an *efficient family of pairwise-independent length-preserving hash functions.*

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

Definition 2.2 *(Efficient family of almost pairwise-independent hash functions) Let $\mathcal{H}$ be a collection of functions where each function $h \in \mathcal{H}$ is from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$. $\mathcal{H}$ is an* efficient family of δ-almost pairwise-independent hash functions *if $|h|$ and $\ell(n)$ are polynomials in n, each $h \in \mathcal{H}$ is a polynomially-computable function, and for all n, for all $x \neq x' \in \{0,1\}^n$ and all $y, y' \in \{0,1\}^{\ell(n)}$,*

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

Due to [CW77], [WC81] and [NN93] there exist constructions of efficient families almost pairwise-independent hash functions for any values of n, δ and $\ell(n)$ whose description length is $\mathcal{O}(log(n) + \ell(n) + log(\delta))$.

2.4 One-Way Functions

Definition 2.3 (One-way functions) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is* one-way *if for every* PPT *A, the function*

$$\mu(n) = \Pr_{x \leftarrow \{0,1\}^n}[A(1^n, f(x)) \in f^{-1}(f(x))]$$

is negligible. A one-way permutation *is a one-way function that is a permutation over any input length n.*

Definition 2.4 (Regular one-way functions) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one-way function. f is regular if there exist a function $\alpha : \mathbb{N} \to \mathbb{N}$ such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ we have:*

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

In the special case that α is also polynomial-time computable, f is known-regular. *In our paper we do not require this property and our results hold for functions with unknown-regularity. Thus in our paper when we say* regular *functions we actually mean* unknown-regular *functions.*

Definition 2.5 (α-Weak one-way functions) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is α-*weak one-way *if for every* PPT *A,*

$$\Pr_{x \leftarrow \{0,1\}^n}[A(1^n, f(x)) \in f^{-1}(f(x))] < 1 - \alpha(n)$$

Few convention remarks: When the value of the security-parameter (i.e., 1^n) is clear, we allow ourselves to omit it from the adversary's parameters list. Since any one-way function is w.l.o.g. length-regular (i.e., inputs of same length are mapped to outputs of the same length), it can be viewed as an ensemble of functions mapping inputs of a given length to outputs of some polynomial (in the input) length. Therefore we can write: let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a one-way function, where $\ell(n)$ is some polynomial-computable function.

2.5 Hardcore predicates

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

Definition 2.6 (Hardcore predicate) *Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ and $b : \{0,1\}^n \to \{0,1\}$ be a polynomial-time computable functions. We call b a* hardcore predicate *of f if the distribution $(f(U_n), U_1)$ and $(f(U_n), b(U_n))$ are computationally-indistinguishable.*

It is custom to call the value $b(x)$, the "hardcore-bit" of $f(x)$. In our applications we use the general hardcore predicate of Goldreich and Levin [GL89]. Let $x, r \in \{0,1\}^n$ and denote $b_r(x) = \langle x, r \rangle \bmod 2$ (that is $b_r(x)$ is the inner product of x and r modulo 2). The proof of the following hardcore-bit theorem is an immediate extension of the proof given in [GL89].

Theorem 2.7 *Let f and g be functions defined over $\{S_n \subseteq \{0,1\}^n\}_{n\in\mathbb{N}}$ and suppose that for all* PPT *algorithms A:*

$$\Pr_{x \leftarrow S_n}[A(f(x)) = g(x)] \in neg(n)$$

Then for X and R uniformly chosen in S_n and $\{0,1\}^n$ respectively we have that $(f(X), R, U_1)$ and $(f(X), R, b_R(g(X))$ are computationally indistinguishable. More precisely, let A be an algorithm that distinguishes between the above distributions with probability $\varepsilon(n)$ and running-time $T_A(n)$, then there exist an algorithms that computes $g(x)$ given $f(x)$ over S_n with probability $\frac{\varepsilon(n)}{4}$ and running-time $\mathcal{O}(\frac{n^3 T_A(n)}{\varepsilon(n)^4})$ (see [Gol01]).

The following is a relaxation of the hard-core bit notion.

Definition 2.8 *[δ-hard predicate] Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ and $b : \{0,1\}^n \to \{0,1\}$ be a polynomial-time computable functions, and let δ be some real fraction. b is a δ-*hard predicate *of f if for any* PPT *algorithm A the following holds,*

$$\Pr_{x \leftarrow U_n}[A(f(x)) = b(x)] < 1 - \delta/2$$

2.6 Pseudorandom Generators

Definition 2.9 (Pseudorandom-Generator (PRG)) *Let $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a polynomial-time computable function where $\ell(n) > n$. We say that G is a* Pseudorandom-Generator *if $G(U_n)$ is computationally-indistinguishable from $U_{\ell(n)}$.*

2.7 Bounded-Space Generators

Bounded-Space generators are pseudorandom generators against bounded-space adversaries. Such generators plays a central role in derandomization tasks. We are interested in generator for the following type of adversaries.

Definition 2.10 ((Bounded Width) Layered Branching Program (LBP)) *A (S, t, ℓ)-LBP M is a directed graph with $S \cdot (t+1)$ vertices, partitioned into $t+1$ layers with S vertices in each layer. Each vertex in the i^{th} layer has exactly 2^ℓ outgoing labelled edges to the $(i+1)^{st}$ layer, one for every possible string $h \in \{0,1\}^\ell$. The vertices in layer t (the last layer) are labelled by 0 or 1.*

Denote by M_x such a LBP with starting vertex $x \in \{1, \ldots, S\}$ in the 0^{th} level. For a sequence $h_1, \ldots, h_t \in B^\ell$ we define the output of the LBP $M_x(h_1, \ldots, h_t)$ by a walk on the graph starting at vertex x in layer 0 and advancing to the i^{th} layer along the edge labelled by h_i. $M_x(h_1, \ldots, h_t)$ accepts if it reaches a vertex labelled by 1 and rejects otherwise.

Definition 2.11 *A generator $G : \{0,1\}^m \to \{0,1\}^{t \cdot \ell}$ is said to ε-*fool *a LBP M if for every $x \in \{1, \ldots, S\}$ we have:*

$$|\Pr_{h_1,\ldots,h_t \in U_{t \cdot \ell}}[M_x(h_1, \ldots, h_t) \text{ accepts}] - \Pr_{\tilde{h} \in U_m}[M_x(G(\tilde{h})) \text{ accepts}]| < \varepsilon$$

Theorem 2.12 ([Nis92, INW94]) *For every S, t, ℓ there exist a generator $BSG : \{0,1\}^{\mathcal{O}(\ell + (S + \log(\frac{t}{\varepsilon})) \log t)} \to \{0,1\}^{t(S) \cdot \ell}$ running in time $poly(S, t, \ell)$ that ε-fools every (S, t, ℓ)-LBP.*

2.8 The Security of Cryptographic Constructions

Typically the proof of security for cryptographic constructions is based on reduction. 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 next define.

Definition 2.13 (Time-success ratio) *Let P be a primitive and let A be an adversary running in time $T(n)$ and breaking P with probability $\delta(n)$. The* time-success ratio *of A in breaking P is defined as $R(n) = \frac{T(n)}{\delta(n)}$, where n is the security-parameter* [5] *of the primitive.*

Note that 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 w.r.t. 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 reductions is when $R_Q(n) \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n))$. 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 $R_Q \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n)^{\mathcal{O}(1)})$. Note that a linear/polynomial-preserving reduction typically means that the inputs of Q and P are of the same length (up to a constant-ratio). The other side of the scale is when $R_Q \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n^{\mathcal{O}(1)}))$. 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 about the above issues the reader may refer to [HL92]. 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 (see Section 3.4.1).

3 Pseudorandom Generators from Regular One-Way Functions

The following discussion considers only length preserving regular one-way functions. We justify this assumption and describe how to deal with all regular one-way functions in Appendix 3.4.2.

3.1 Some Motivation and the Randomized Iterate

Recall that the BMY generator simply iterates the one-way permutation f on itself, and outputs a hardcore-bit of the intermediate step at each iteration. The crucial point is that the output of the

[5]It is convenient to define the security-parameter of a primitive as its input length. This is in particular the convention for the primitives discussed in this paper.

function is also uniform in $\{0,1\}^n$ since f is a permutation. Hence, when applying f to the output, it is hard to invert this last application of f, and therefore hard to predict the new hardcore-bit (Yao shows [Yao82] that the unpredictability of bits implies pseudorandomness). Since the seed is essentially just an n bit string and the output is as long as the number of iterations, then the generator actually stretches the seed.

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. [GKL93] suggested to add a randomization step between every two applications of f, thus making the next input to f a truly random one. This modification that we call randomized iterates lies at the core of our work and is defined next:

Definition 3.1 (The k^{th} Randomized Iterate of f) *Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ and let $\mathcal{H}$ be an efficient family of pairwise-independent hash functions from $\{0,1\}^n$ to $\{0,1\}^n$. For input $x \in \{0,1\}^n$ and $h_1,\dots,h_{k-1} \in \mathcal{H}$ define the k^{th}* Randomized Iterate *$f^k : \{0,1\}^n \times \mathcal{H}^k \rightarrow Im(f)$ recursively as:*

$$f^k(x,h_1,\dots,h_k) = f(h_k(f^{k-1}(x,h_1,\dots,h_{k-1})))$$

where $f^0(x) = f(x)$. For convenience we denote by $x^k \overset{def}{=} f^k(x,h_1,\dots,h_k)$.[6]

Another handy notation is the k^{th} explicit randomized iterate $\widehat{f^k} : \{0,1\}^n \times \mathcal{H}^k \rightarrow Im(f) \times \mathcal{H}^k$ defined as:

$$\widehat{f^k}(x,h_1,\dots,h_k) = (f^k(h_1,\dots,h_k),h_1,\dots,h_k)$$

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 (in other words, use the explicit randomized iterate $\widehat{f^k}$). But on the other hand, handing out the randomizing hash gives information on intermediate values such as $h_i(x^i)$ and f might no longer be hard to invert when applied to such an input. Somewhat surprisingly, the last 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 [GKL93] when using a family of n-wise independent hash functions. We give a simpler proof that extends to pairwise-independent hash functions as well.

Remark: In the definition randomized iterate we define $f^0(x) = f(x)$. This was chosen for ease of notation and consistency with the results for general OWFs (Section 4). For the regular OWF construction it suffices to define $f^0(x) = x$, thus saving a single application of the function f.

3.2 The Last Randomized Iteration is Hard to Invert

In this section we formally state and prove the key observation mentioned above, that is, that 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.

[6] We make use the notation x^k only when the values of $h_1,\dots,h_k$ and x are clear by the presentation.

Lemma 3.2 *Let f be a length-preserving regular one-way function, $\mathcal{H}$ be an efficient family of pairwise-independent length-preserving hash functions and x^k be the k^{th} randomized iterates of f (Definition 3.1). Then for any* PPT *A and every $k \in poly(n)$ we have:*

$$\Pr_{(x,h_1,\ldots,h_k) \leftarrow (U_n,\mathcal{H}^k)}[A(x^k, h_1, \ldots, h_k) = x^{k-1}] \in neg(n)$$

where the probability is also taken over the random coins of A.

More precisely, if such a PPT *A succeeds with probability ε then there exists a probabilistic polynomial time oracle machine M^A that succeeds in inverting f with probability at least $\varepsilon^3/8(k+1)$ with essentially the same running time as A.*

We briefly give some intuition to the proof, illustrated with regard to the first randomized iterate. Suppose that we have an algorithm A that *always* finds x^0 given $x^1 = f^1(x, h)$ and h. In order to invert the one-way function f on an element $z \in Im(f)$, we simply need to find a hash h' that is consistent with z, in the sense that there exists an x' such that $z = f^1(x', h')$. Now we simply run $y = A(z, h')$, and output $h'(y)$ (and indeed $f(h'(y)) = z$). The point is that if f is a regular function, then finding a consistent hash is easy, simply because a random and independent h' is likely to be consistent with z. The actual proof follows this framework, but is far more involved due to the fact that the reduction starts with an algorithm A that has only a *small* (yet polynomial) success probability.

Proof: Suppose for sake of contradiction that there exists an efficient algorithm A that given $(x^k, h_1, \ldots, h_k)$ computes x^{k-1} with probability ε for some polynomial fraction $\varepsilon(n) = 1/poly(n)$ (for simplicity we simply write ε). In particular A inverts the last-iteration of $\widehat{f^k}$ with probability at least ε, that is

$$\Pr_{(x,h_1,\ldots,h_k) \leftarrow (U_n,\mathcal{H}^k)}[f(h(A(\widehat{f^k}(x, h_1, \ldots, h_k)))) = f^k(x, h_1, \ldots, h_k)] \geq \varepsilon$$

Our goal is to use this procedure A in order to break the one-way function f. Consider the procedure M^A for this task:

```
$M^A$ on input $z \in Im(f)$:
  1. Randomly (and independently) choose $h_1, \ldots, h_k \in \mathcal{H}$.
  2. Apply $A(z, h_1, \ldots, h_k)$ to get an output $y$.
  3. If $f(h_k(y)) = z$ output $h_k(y)$, otherwise abort.
```

The rest of the proof of Lemma 3.2 shows that M^A succeeds with probability at least $\varepsilon^3/8(k+1)$ on inputs $z \in Im(f)$.

We start by focusing our attention only on those inputs for which A succeeds reasonably well. Recall that the success probability of A is taken over the choice of inputs to A as induced by the choice of $x \in \{0,1\}^n$ and $h_1, \ldots, h_k \in \mathcal{H}$ and the internal coin-tosses of A. The following Markov argument implies that the probability of getting an element in the set that A succeeds on is not very small:

Claim 3.3 *Let $S_A \subseteq Im(\widehat{f^k})$ be the subset defined as:*

$$S_A = \left\{(y, h_1, \ldots, h_k) \in Im(\widehat{f^k}) \mid \Pr[f(h_k(A(y, h_1, \ldots, h_k))) = y] > \frac{\varepsilon}{2}\right\}$$

Then

$$\Pr_{(x,h_1,\ldots,h_k) \leftarrow (U_n,\mathcal{H}^k)}[\widehat{f^k}(x,h_1,\ldots,h_k) \in S_A] \geq \frac{\varepsilon}{2}$$

.

Proof: Suppose that $\Pr[\widehat{f^k}(x,h_1,\ldots,h_k) \in S_A] < \frac{\varepsilon}{2}$. Then we have:

$$Pr[f(h(A(\widehat{f^k}(x,h_1,\ldots,h_k)))) = f^k(x,h_1,\ldots,h_k)] <$$
$$\frac{\varepsilon}{2} \cdot Pr[\widehat{f^k}(x,h_1,\ldots,h_k) \notin S_A] + 1 \cdot Pr[\widehat{f^k}(x,h_1,\ldots,h_k) \in S_A] < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon$$

which contradicts the assumption about the success probability of A. ▮

Now that we identified a subset of polynomial weight of the inputs that A succeeds upon, we want to say that M^A has a fair (polynomially large) chance to hit outputs induced by this subset. This is formally shown in the following lemma.

Lemma 3.4 *For every set $T \subseteq Im(\widehat{f^k})$, if*

$$\Pr_{(x,h_1,\ldots,h_k) \leftarrow (U_n,\mathcal{H}^k)}[\widehat{f^k}(x,h_1,\ldots,h_k) \in T] \geq \delta$$

then

$$\Pr_{(z,h_1,\ldots,h_k) \leftarrow (f(U_n),\mathcal{H}^k)}[(z,h_1,\ldots,h_k) \in T] \geq \delta^2/(k+1),$$

We stress that the probability in the latter inequality is over z drawn from $f(U_n)$ and an independently *chosen $h_1,\ldots,h_k \in \mathcal{H}$.*

Assuming Lemma 3.4 we may conclude the proof of Lemma 3.2. By Claim 3.3 we have that $\Pr[(x^k,h_1,\ldots,h_k) \in S_A] \geq \frac{\varepsilon}{2}$. By Lemma 3.4 taking $T = S_A$ and $\delta = \varepsilon/2$ we get that $\Pr[(z,h_1,\ldots,h_k) \in S_A] \geq \varepsilon^2/4(k+1)$. Thus M^A has a $\varepsilon^2/4(k+1)$ chance of hitting the set S_A on which it will succeed with probability at least $\varepsilon/2$. Altogether, M^A succeeds in inverting f with the polynomial probability $\varepsilon^3/8(k+1)$, contradicting the one-wayness of f. ▮

Proof: (of Lemma 3.4) The lemma essentially states that with respect to $\widehat{f^k}$, 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 z and $h_1,\ldots,h_k$. 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). However, we show that this is impossible by proving that the collision-probability of the function $\widehat{f^k}$ is small. The proof therefore follows by analyzing the collision-probability of $\widehat{f^k}$. For every two inputs $(x_0,{h_1}^0,\ldots,{h_k}^0)$ and $(x_1,{h_1}^1,\ldots,{h_k}^1)$ to $\widehat{f^k}$, in order to have a collision we must first have that ${h_i}^0 = {h_i}^1$ for every $i \in [k]$, which happens with probability $(1/|\mathcal{H}|)^k$. Now, given that ${h_i}^0 = {h_i}^1 = h_i$ for all i (with a random $h_i \in \mathcal{H}$), we require also that $x_0^k = f^k(x_0,h_1,\ldots,h_k)$ equals $x_1^k = f^k(x_1,h_1,\ldots,h_k)$. If $f(x_0) = f(x_1)$ (happens with probability $1/|Im(f)|$) then a collision is assured. Otherwise, there must be an $i \in [k]$ for which $x_0^{i-1} \neq x_1^{i-1}$ but $x_0^i = x_1^i$ (where x_0 denotes the input x). Since $x_0^{i-1} \neq x_1^{i-1}$, due to the pairwise-independence of h_i, the values $h_i(x_0^{i-1})$ and $h_i(x_1^{i-1})$ are uniformly random values in $\{0,1\}^n$, and thus $f(h_i(x_0^{i-1})) = f(h_i(x_1^{i-1}))$ happens with probability $1/|Im(f)|$. Altogether:

$$CP(\widehat{f^k}(U_n,\mathcal{H}^k)) \leq \frac{1}{|\mathcal{H}|^k} \sum_{i=0}^{k} \frac{1}{|Im(f)|} \leq \frac{k+1}{|\mathcal{H}|^k\,|Im(f)|} \qquad (1)$$

On the other hand, we check the probability of getting a collision inside the set T, which is a lower bound on the probability of getting a collision at all. We first request that both $(x_0, {h_1}^0, \dots, {h_k}^0) \in T$ and $(x_1, {h_1}^1, \dots, {h_k}^1) \in T$. This happens with probability at least δ^2. Then, once inside T, we know that the probability of collision is at least $1/|T|$. Altogether:

$$CP(\widehat{f^k}(U_n, \mathcal{H}^k)) \geq \delta^2 \frac{1}{|T|} \tag{2}$$

Combining (1) and (2) we get $\frac{|T|}{|\mathcal{H}|^k |Im(f)|} \geq \frac{\delta^2}{k+1}$.
But the probability of getting a value in T when choosing a random element in $Im(f) \times \mathcal{H}^k$ is exactly $\frac{|T|}{|\mathcal{H}|^k |Im(f)|}$. Thus $\Pr[(z, h_1, \dots, h_k) \in T] \geq \delta^2/(k+1)$ as requested. ▮

Remark: The proof of Lemma 3.4 is where the regularity of the one-way function is required. The proof fails for general OWFs where the preimage size of different elements may vary considerably. More accurately, the collision-probability at the heart of this lemma remains considerably small only as long as the last element in a sequence of applications is at least as heavy as all the elements along the sequence. While for regular functions this requirement is always true, for general OWFs this occurs with probability that deteriorates linearly (in the length of the sequence). Thus using a long sequence of iterations is likely to lose the hardness of the original OWF.

3.3 A Pseudorandom Generator from a Regular One-Way Function

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 of the current input is taken at each stage. Formally:

Theorem 3.5 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a regular length-preserving one-way function and let $\mathcal{H}$ be an efficient family of pairwise-independent length preserving hash functions, let G be:*

$$G(x, h_1 \dots, h_n, r) = (b_r(x^0), \dots, b_r(x^n), h_1, \dots, h_n, r),$$

where:

- *$x \in \{0,1\}^n$ and $h_1 \dots, h_n \in \mathcal{H}$.*
- *Recall that $x^0 = f(x)$ and for $1 \leq i \leq n$ $x^i = f(h_i(x^{i-1}))$.*
- *$b_r(x^i)$ denotes the GL-hardcore bit of x^i.*

Then G is a pseudorandom generator.

Note that the above generator does not require the knowledge of the preimage size of the regular one-way function. The generator requires just $n+1$ calls to the underlying one-way function f (each call is on an n bit input). The generator's input is of length $m = \mathcal{O}(n^2)$ and it stretches the output to $m+1$ bits.
Proof: First notice that the generator G indeed stretches a m bit string into a $m+1$ bit string. Since all of $h_1, \dots, h_n \in \mathcal{H}$ and r appear in the output as well, the only difference is that the n bits of x are replaced by $n+1$ hardcore bits.

The proof of pseudorandomness follows from the unpredictability property of this generator. Yao showed [Yao82] using a hybrid argument that a sequence is pseudorandom if and only if it is hard to predict the next bit of the sequence for every prefix of the sequence. We show that the sequence $G(x, h_1, \ldots, h_n)$ has this property in the following reordering: $(r, h_1, \ldots, h_n, b(x^n), \ldots, b(x^0))$ (for ease of notation we omit the subscript r from the hardcore bits, thus $b(x) = b_r(x)$). Since $(r, h_1, \ldots, h_n)$ is a truly random string then the unpredictability property holds vacuously for these bits. It is left to show that for every k, one cannot predict $b(x^{k-1})$ given $(r, h_1, \ldots, h_n, b(x^n), \ldots, b(x^k))$. Suppose there exists such a predictor B with $\Pr[B(r, h_1, \ldots, h_n, b(x^n), \ldots, b(x^k)) = b(x^{k-1})] > \frac{1}{2} + 1/poly(n)$ for some polynomial $poly(\cdot)$. Then there is also a B' for predicting $b(x^{k-1})$ given $(r, h_1, \ldots, h_k, x^k)$. The algorithm B' does the following:

1. Choose random $\bar{h}_{k+1}, \ldots, \bar{h}_n \in \mathcal{H}$.
2. Generate $\bar{x}^{k+1}, \ldots, \bar{x}^n$ from $(x^k, \bar{h}_{k+1}, \ldots, \bar{h}_n)$.
3. Output $B(r, h_1 \ldots, h_k, \bar{h}_{k+1}, \ldots, \bar{h}_n, b(\bar{x}^n), \ldots, b(\bar{x}^{k+1}), b(x^k))$

Choosing $(\bar{h}_{k+1}, \ldots, \bar{h}_n)$ independently from $(h_1, \ldots, h_k)$ generates a series $(\bar{x}^{k+1}, \ldots, \bar{x}^n)$ that has the same distribution as $(x^{k+1}, \ldots, x^n)$. Thus the procedure B' succeeds in predicting $b(x^{k-1})$ with the same success probability as B, i.e., with probability at least $\frac{1}{2} + 1/poly(n)$. This yields a contradiction since by Lemma 3.2, it is computationally hard to find x^{k-1} given $(x^k, h_1, \ldots, h_k)$ and due to the Goldreich-Levin Theorem (Theorem 2.7), it is also impossible to predict a hardcore-bit of this value with a polynomial advantage. ∎

3.4 An Almost-Linear-Input Construction from a Regular One-Way Function

The pseudorandom generator presented in the previous section (Theorem 3.5) stretches a seed of length $\mathcal{O}(n^2)$ by one bit. 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 m bits relies on the security of regular one-way function on $\sqrt{m}$ bits. In this section we give a modified construction of the pseudorandom generator that takes a seed of length only $m = \mathcal{O}(n \log n)$.

Notice that the input length of the generator is dominated by the description of the n independent hash functions $h_1, \ldots, h_n$. The idea of the new construction is to give a derandomization of the choice of the n hash functions. 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 [Nis92], (or that of Impagliazzo, Nisan and Wigderson [INW94]). An important 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 $x^{i+1} = f(h_i(x^i))$. We will show that indeed when replacing $h_1, \ldots, h_n$ with the output of a generator that fools related branching programs, then the proof of security still holds (and specifically the proof of Lemma 3.4).

For our application we use the bounded-space generator with parameters $t = n$, $S = 2n$ and $\ell = 2n$ (or more generally, ℓ is taken to be the description length of a hash function in $\mathcal{H}$). Finally, the error is chosen to be $\varepsilon = 2^{-n}$. The generator therefore stretches $\mathcal{O}(n \log n)$ bits to $n \cdot 2n$ bits. Denote the bounded-space generator by $BSG : \{0,1\}^{cn \log n} \to \{0,1\}^{2n^2}$ where c is a universal constant. For convenience denote $\tilde{n} = cn \log n$.

3.4.1 The New Pseudorandom Generator

Theorem 3.6 *For any regular length-preserving one-way function f, let G' be:*

$$G'(x,\tilde{h},r) = (b_r(x^0),\dots,b_r(x^n),\tilde{h},r),$$

where:

- $x \in \{0,1\}^n$ *and* $\tilde{h} \in \{0,1\}^{\tilde{n}}$.
- $(h_1,\dots,h_n) = BSG(\tilde{h})$.
- *Recall that* $x^0 = f(x)$ *and for* $1 \le i \le n$, $x^i = f(h_i(x^{i-1}))$.
- $b_r(x^i)$ *denotes the GL-hardcore bit of* x^i.

Then G' is a pseudorandom generator.

Proof outline: The proof of the derandomized version follows in the steps of the proof of Theorem 3.5. 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 x^k and all of the randomizing hash functions, it is hard to compute x^{k-1} (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 3.2, only replacing appearances of $(h_1,\dots,h_k)$ with the seed $\tilde{h}$. Again, the key to the proof is the following technical lemma (slightly modified from Lemma 3.4).

Lemma 3.7 *For every set* $T \subseteq Im(f) \times \{0,1\}^{\tilde{n}}$, *if*

$$\Pr_{(x,\tilde{h}) \leftarrow (U_n,U_{\tilde{n}})}[(x^k,\tilde{h}) \in T] \ge \delta$$

then

$$\Pr_{(z,\tilde{h}) \leftarrow (f(U_n),U_{\tilde{n}})}[(z,\tilde{h}) \in T] \ge \delta^2/(k+2)$$

where probability is over $z \in f(U_n)$ *and an* independently *chosen* $\tilde{h} \in \{0,1\}^{\tilde{n}}$.

Once we know that x^{k-1} is hard to compute, we deduce that one cannot predict a hardcore-bit $b_r(x^{k-1})$ given x^k and the seed to the bounded-space generator. From here, the proof follow just as the proof of Theorem 3.5 in showing that the output of G' is an unpredictable sequence and therefore a pseudorandom sequence. ▮

Proof: (of Lemma 3.7) Denote by $g : \{0,1\}^n \times \{0,1\}^{\tilde{n}} \to Im(f) \times \{0,1\}^{\tilde{n}}$ the function taking inputs of the form $(x,\tilde{h})$ to outputs of the form $(x^k,\tilde{h})$ (recall that x^k is the k^{th} randomized iterate of f under seed $\tilde{h}$). We proceed by giving bounds on the collision-probability of g. For every two inputs to g, $(x_0,\tilde{h_0})$ and $(x_1,\tilde{h_1})$, in order to have a collision we must first have that $\tilde{h_0} = \tilde{h_1}$ which happens with probability $1/2^{\tilde{n}}$. Now, given that $\tilde{h_0} = \tilde{h_1}$ (with a random $\tilde{h}$), we analyze the probability of the event that $x_0^k = f^k(x_0,h_1,\dots,h_k)$ equals $x_1^k = f^k(x_1,h_1,\dots,h_k)$.

Consider the following (S,t,ℓ)-LBP for the input pair (x_0,x_1):

- Input: The LBP starts at a node labelled by (x_0,x_1).

- Layer i (for $i \in [n]$): Get random input $h_i \in \mathcal{H}$ and move from node (x_0^{i-1}, x_1^{i-1}) to the node labelled (x_0^i, x_1^i) with $x_0^i = f(h_i(x_0^{i-1}))$ and $x_1^i = f(h_i(x_1^{i-1}))$.
- The LBP accepts if $x_0^n = x_1^n$ and rejects otherwise.

The LBP described above has parameters $S = 2n$, $t = n$ and $\ell = 2n$. Furthermore, it accepts with probability that is exactly the desired collision probability, that is, the probability that $f^k((x_0, h_1, \ldots, h_k)) = f^k((x_1, h_1, \ldots, h_k))$ over any distribution on $(h_1, \ldots, h_k)$. For every pair (x_0, x_1) with $x_0 \neq x_1$ this probability over random $(h_1, \ldots, h_n)$ was bounded in the proof of Lemma 3.4 by:

$$\Pr_{h_1,\ldots,h_n \leftarrow \mathcal{H}}[f^k(x_0, h_1, \ldots, h_k) = f^k(x_1, h_1, \ldots, h_k)] \leq \frac{k}{|Im(f)|}$$

Since the generator fools the above LBP, then replacing the random inputs $h_1, \ldots, h_k$ with the output of the bounded-space generator does not change the probability of acceptance by more than $\varepsilon = 2^{-n}$. Therefore:

$$\Pr_{h_1,\ldots,h_n \leftarrow BSG(U_{\tilde{n}})}[f^k(x_0, h_1, \ldots, h_k) = f^k(x_1, h_1, \ldots, h_k)] \leq \frac{k}{|Im(f)|} + \frac{1}{2^n} \leq \frac{k+1}{|Im(f)|}$$

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

$$\Pr_{x_0,x_1 \leftarrow U_n, h_1,\ldots,h_n \leftarrow BSG(U_{\tilde{n}})}[f^k(x_0, h_1, \ldots, h_k) = f^k(x_1, h_1, \ldots, h_k)] \leq \frac{k+1}{|Im(f)|} + \frac{1}{2^n} \leq \frac{k+2}{|Im(f)|}$$

When the above is plugged into the calculation of the collision-probability of $\widehat{f^k}$, we get:

$$CP(\widehat{f^k}(U_n, U_{\tilde{n}})) \leq \frac{k+2}{2^{\tilde{n}}\,|Im(f)|} \tag{3}$$

On the other hand, we check the probability of getting a collision inside the set T. We first request that both $(x_0, \tilde{h_0}) \in T$ and $(x_1, \tilde{h_1}) \in T$, which happens with probability at least δ^2. Then once inside T, we know that the probability of collision is at least $1/|T|$. Altogether:

$$CP(\widehat{f^k}(U_n, U_{\tilde{n}}) \geq \delta^2 \frac{1}{|T|} \tag{4}$$

Combining (3) and (4) we get:

$$\frac{|T|}{2^{\tilde{n}}\,|Im(f)|} \geq \frac{\delta^2}{k+2}$$

But the probability of getting a value in T when choosing a random element in $Im(f) \times \{0,1\}^{\tilde{n}}$ is exactly $\frac{|T|}{2^{\tilde{n}}|Im(f)|}$. Thus $\Pr[(z, \tilde{h}) \in T] \geq \delta^2/(k+2)$ as requested. ▮

Remark: It is tempting to think that one should replace Nisan/INW generator in the above proof with the generator of Nisan and Zuckerman [NZ96]. That generator may have seed of size $\mathcal{O}(n)$ (rather than $\mathcal{O}(n \log n)$) when S=2n as in our case. Unfortunately, with such a short seed, that generator will incur an error $\varepsilon = 2^{-n^{1-\gamma}}$ for some constant γ, which is too high for our proof to work. In order for the proof to go through we need that $\varepsilon < poly(n)/|Im(f)|$. 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).

The Overall Time-Success Ratio: Combining all of the reductions involved from breaking the security of the generator to inverting the one-way function (that is, a hybrid argument, the Goldreich-Levin proof and Lemma 3.2) we get the following time-success ratio (see Section 2.8): Given a distinguisher for G' that runs in time $T_D(m)$ and has polynomial success $\varepsilon(m)$ (where m is the seed length to the generator), then there exists an algorithm for inverting the underlying f on n bits, that runs in time $\mathcal{O}(\frac{n^7 \log^4 n}{\varepsilon^4(n \log n)} T_D(n \log n))$ and has success $\Omega(\frac{\varepsilon^3(n \log n)}{n^4 \log^4 n})$.

3.4.2 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. 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 Appendix B we show how to transform an expanding one-way function f from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ (for simplicity we write just ℓ) into a length preserving one-way function from $\{0,1\}^{2n}$ to $\{0,1\}^{2n}$. However, this construction 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 $f : \{0,1\}^n \to \{0,1\}^\ell$, define the randomized iterate of this function with respect to a family of hash functions from $\{0,1\}^\ell$ to $\{0,1\}^n$. The randomized iterate is now well defined, and moreover, we can show that the collision probability hardly changes. Previously the only way to introduce a new collision was during the application of f (a collision happened with probability $1/Im(f)$ in each iteration). In the new construction a collision can also be introduced when applying a hash function. But such a collision during hashing happens with probability only 2^{-n} (by the pairwise independence of the hash). Thus the overall collision probability at most doubles and the proof of security follows.

The only problem with this approach is that the description of such hash functions is too long (it is $\mathcal{O}(\ell)$ instead of $\mathcal{O}(n)$). This is overcome by using an efficient family of *almost* pairwise-independent hash functions from ℓ bits to n bits with error 2^{-n} (see Definition 2.2), which requires a description of only $\mathcal{O}(n)$ bits.

4 Pseudorandom Generator from Any One-Way Function

Our implementation of a pseudorandom generator from any one-way function follows the route of the HILL construction, but takes a totally different approach in the implementation of its initial step. The resulting construction is more efficient and more secure than the original one.

The "pseudo-entropy" of a distribution is at least k if it is computationally-indistinguishable from some distribution that has entropy k. The basic building block of the HILL generator is a "pseudo-entropy pair".[7] Informally, the latter is a pair of a function and predicate on the same input with the following property: The pseudo-entropy of the predicate's output when given the output of the function is noticeably larger than the real (conditional) entropy of this bit. In their construction [HILL99] exploit this gap between real and pseudo entropy to construct a pseudorandom generator. We show that the first explicit randomized iterate of a one-way function together with a standard

[7]We note that [HILL99] used this notion implicitly without giving it an explicit definition.

hardcore predicate forms a pseudo-entropy pair. Moreover, this new PEP has better properties than the original implementation and hence "plugging" it as the first step of the HILL construction results in a better overall construction. Let us now turn to a more formal discussion. We define the pseudo-entropy pair as follows:

Definition 4.1 *[Pseudo-entropy pair (*PEP*)] Let δ and γ be some positive functions over $\mathbb{N}$ and let $g : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ and $b : \{0,1\}^n \to \{0,1\}$ be polynomial-time computable functions. (g, b) is a (δ,γ)-*PEP *if*

1. *$H(b(U_n) \mid g(U_n)) \leq \delta(n)$.*
2. *b is a $(\delta(n) + \gamma(n))$-hard predicate of g.*

[HILL99] show how to construct a (δ, α)-PEP, where $\delta \in [0, 1]$ is some **unknown** value and α is any fraction noticeably smaller than $\frac{1}{2n}$, using any one-way function.[8] They then present a construction of a pseudorandom generator using a $(\delta, \frac{1}{\mathcal{O}(n)})$-PEP where δ is **known**. To overcome this gap, the HILL generator enumerates all values for δ (up to an accuracy of $\Omega(\frac{1}{n})$), runs the generator with every one of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor of n to the seed length as well as n^3 times more calls to the underlying one-way function.

In the rest of this section we prove that the first explicit randomized iterate of a one-way function can be used to construct a $(\frac{1}{2}, \alpha)$-PEP, where α is any fraction noticeably smaller than $\frac{1}{2n}$. We note that the [HILL99] and ours implementations of a PEP have the same efficiency (constant number of calls to the underlying one-way function). Therefore the resulting construction achieved by combining our PEP with the other parts of the HILL construction is more efficient and has better security[9] than the original construction (the efficiency improves by a factor of n^3 and the security by a factor of n). We note that in order to use our PEP instead of the original one a somewhat generalize of the other parts of the HILL construction is needed. For completeness we present in Appendix A a high-level overview of this generalized version. For comparison, we also present there the PEP used by [HILL99].

4.1 A Pseudo-Entropy Pair Based On The Randomized Iterate

For a given one-way function f, we have defined (Definition 3.1) its first explicit randomized iterate as $\widehat{f^1}(x, h) = (f(h(f(x))), h)$. In Section 4.1.1 we present an "extended" version of the above function with the following properties: First, it maintains some hardness of the original one-way function. The hardness is maintained in a sense that with probability $\frac{1}{2} + \frac{1}{2n}$ it is hard to compute the value of $x^0 = f(x)$ given the output. Second, we show that with probability $\frac{1}{2}$ the value of x^0 can be determined w.h.p. from the output. Notice the gap between the computational-knowledge and information about x^0 given the output element. In Section 4.1.2 we show how to take advantage of this gap, in a straight forward manner, to achieve a PEP for the above randomized iterate.

[8][HILL99] actually prove somewhat stronger result. Not only that the predicate of their PEP is $(\delta + \alpha)$-hard, but the hardness comes from the existence of a "hardcore-set" of density $\delta + \frac{1}{2n}$. Where the latter is a subset of the input such that the value of b is computationally unpredictable over it. This additional property was used by [HILL99] original implementation of pseudorandom generator, but it is not required by the implementation presented in this paper (see Appendix A). We stress that our PEP, presented next, also has such a hardcore-set

[9]The proof of security of the reduction to the underlying one-way function has better parameters, see Section 2.8 for a discussion on the security of cryptographic reductions.

4.1.1 The Extended Randomized Iterate Of Any One-Way Function

For simplicity we assume that the one-way function is length-preserving, the adaptation to any one-way function is similar to the one done in Section 3. [10] We use the following extended version of the first explicit randomized iterate:

Definition 4.2 (The Extended Randomized Iterate) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function, let $m = \lceil 3\log(n) + 8 \rceil$, and let $\mathcal{H}$ and $\mathcal{H}_E$ be two families of pairwise-independent hash functions from $\{0,1\}^n$ to $\{0,1\}^n$ and $\{0,1\}^n$ to $\{0,1\}^m$ respectively. We define g,* the extended randomized iterate of f, *as:*

$$g(x, h, h_E) = (\widehat{f^1}(x, h), h_E(f(x)), h_E)$$

where $x \in \{0,1\}^n$, $h \in \mathcal{H}$ and $h_E \in \mathcal{H}_E$.

Remark: We note that while the role of $\mathcal{H}_E$ in the above construction and the role of $\mathcal{H}$ in the HILL PEP (see Appendix A.4) are syntactically similar, their actual role is different. In the above construction the purpose of using $\mathcal{H}_E$ is to reveal a small amount of information (i.e., $\mathcal{O}(log(n))$) about $f(x)$, which in turn *slightly* reduces the uncertainty of $f(x)$ when given $g(x, h, h_E)$. Thus $\mathcal{H}_E$ is used to widen the gap between the real and the pseudo-entropy of $f(x)$ given $g(x, h, h_E)$. This extra information is not of major importance and indeed even without using $\mathcal{H}_E$ (i.e. using $\widehat{f^1}$ instead of g), we can still construct a PEP, though parameters will not be as good. On the other hand, in the HILL PEP there are settings where the hash function reveals a significant amount of information (i.e., $\Omega(n)$) about the input of the function. Thus, removing the hash function altogether guarantees no gap between the real and the pseudo-entropy, and in particular the resulting pair is not likely to be a PEP.

In the rest of this section we show (Lemma 4.3) that with probability $\frac{1}{2} + \frac{1}{2n}$ it is hard to compute the value of $x^0 = f(x)$ given a random output $g(x, h, h_E)$. While on the other hand we show (Lemma 4.6) that in the same settings the value of x^0 is information-theoretically determined with probability at least $\frac{1}{2}$.

Lemma 4.3 *Let f, $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 4.2 and let*

$$G \overset{def}{=} \left\{(x, h, h_E) \in (\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E) \mid D_f(x^0) \le D_f(x^1)\right\}$$

where $x^0 = f(x)$ and $x^1 = f(h(x^0))$. Then,

1. $\frac{|G|}{|\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E|} \ge \frac{1}{2} + \frac{1}{2n}$.
2. *For every* PPT *A,*
$$\Pr_{(x,h,h_E) \leftarrow G}[A(x^1, h, h_E(x^0), h_E) = x^0] \in neg(n)$$
where the probability is also taken over the random coins of A. More precisely, any algorithm that given $(x^1, h, h_E(x^0), h_E)$ computes x^0 over G with probability ε can be used to construct an algorithm that inverts f over $\{0,1\}^n$ with probability $\frac{\varepsilon^3}{64n2^{3m}}$ and the same order of running-time.

[10] A different approach would be to assume w.l.o.g. that the one-way function is length-preserving. This assumption is justified by Corollary B.2 that states that any one-way function can be modified to be a length-preserving with the same (up to constant factor) security ratio.

(Proving 1): Note that for a random (x, h) it holds that x^0 and x^1 are two independent instances of a random variable distributed over $Im(f)$. Hence $D_f(x^0)$ and $D_f(x^1)$ are two independent instances of a random variable distributed over $[n]$ and thus by symmetry,

$$\Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[D_f(x^0) \lneq D_f(x^1)] = \Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[D_f(x^0) \gneq D_f(x^1)] \tag{5}$$

Since the collision-probability of a random variable is minimal when the distribution is uniform,

$$\Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[D_f(x^0) = D_f(x^1)] \geq \frac{1}{n} \tag{6}$$

Combining Equations 5 and 6 we get the following,
$\frac{|G|}{|\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E|} = \Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[D_f(x^0) \leq D_f(x^1)] \geq \frac{1}{2} + \frac{1}{2n}$ as claimed. ∎

(Proving 2): Let A be an algorithm that given $g(x, h, h_E)$ computes x^0 with probability ε over G. In particular A inverts the last-iteration of g with probability at least ε over G, that is

$$\Pr_{(x,h,h_E) \leftarrow G}[f(h(A(g(x, h, h_E)))) = g(x, h, h_E)] \geq \varepsilon$$

Therefore, there exists an algorithm A' that with probability at least $\varepsilon' \stackrel{\text{def}}{=} \varepsilon/2^m$ inverts the last-iteration of g without even seeing the second part of the output (i.e., without seeing $(h_E(f(x)), h_E)$). (A' on input $\widehat{f^1}(x, h)$ chooses a random value for $(h_E, h_E(f(x)))$ and returns $A(\widehat{f^1}(x, h), h_E(x^0), h_E)$). In other words, A' inverts with noticeable success probability the last-iteration of $\widehat{f^1}$ over $G' \stackrel{\text{def}}{=} \{(x, h) \in (\{0, 1\}^n \times \mathcal{H}) \mid D_f(x^0) \leq D_f(x^1)\}$ (i.e., G' is the restriction of G to $\widehat{f^1}$ domain). We now use algorithm A' to invert f, consider the following procedure for this task:

```
$M^{A'}$ on input $z \in Im(f)$:
    1. Choose a random $h \in \mathcal{H}$.
    2. Apply $A'(z, h)$ to get an output $y$.
    3. If $f(h(y)) = z$ output $h(y)$, otherwise abort.
```

We prove that $M^{A'}$ succeeds in inverting f with probability bounded away from zero. We focus on the following set of outputs, which A' inverts their last-iteration with bounded-away from zero probability.

$$S_{A'} = \left\{(y, h) \in Im(\widehat{f^1}) \mid \Pr[f(h(A'(y, h)) = y] > \varepsilon'/2\right\}$$

as in Claim 3.3 it holds that $\Pr_{(x,h) \leftarrow G'}[\widehat{f^1}(x, h) \in S_{A'}] \geq \varepsilon'/2$. Moreover, since the density of G' is big (note that following Lemma 4.3[1], the density of G' w.r.t. $\widehat{f^1}$ input domain is at least $\frac{1}{2}$), it follows that $\Pr_{(x,h) \leftarrow (U_n,\mathcal{H})}[\widehat{f^1}(x, h) \in S_{A'} \bigwedge (x, h) \in G'] \geq \varepsilon'/4$. We now make use of the following Lemma (similar to Lemma 3.4), that relates sets of outputs of $\widehat{f^1}$ to sets of outputs of f.

Lemma 4.4 *For every set $T \subseteq Im(\widehat{f^1})$, if*

$$\Pr_{(x,h) \leftarrow (U_n,\mathcal{H})}[\widehat{f^1}(x, h) \in T \bigwedge (x, h) \in G'] = \delta$$

then

$$\Pr_{(z,h)\,\leftarrow\,(f(U_n),\mathcal{H})}[(z,h)\in T]\geq \delta^2/4n$$

To conclude the proof of Lemma 4.3, take $T=S_{A'}$ and $\delta=\varepsilon'/4$, and Lemma 4.4 yields that $\Pr_{(z,h)\,\leftarrow\,(f(U_n),\mathcal{H})}[(z,h)\in S_{A'}]\geq \frac{\varepsilon'^2}{32n}$. On each of these inputs A' succeeds with probability $\varepsilon'/2$, thus altogether $M^{A'}$ succeeds with probability $\frac{\varepsilon'^3}{64n}=\frac{\varepsilon^3}{2^{3m}64n}$ in inverting f. ▮

Proof: (of Lemma 4.4) Divide the outputs of the function f into n slices according to their preimage size. The set T is divided accordingly into n subsets. For every $i\in[n]$ define the i^{th} slice $T_i=\{(z,h)\in T\mid D_f(z)=i\}$. We divide G' into corresponding slices as well, define the i^{th} slice as $G_i=\left\{(x,h)\in G'\mid D_f(f^1(x,h))=i\right\}$ (note that since $G_i\subseteq G'$, for each $(x,h)\in G_i$ it holds that $D_f(f(x))\leq D_f(f^1(x,h))=i$). The proof of Lemma 4.4 follows the argument for the case of regular functions (Lemma 3.4) but works on each slice separately. We compute the collision-probability of $\widehat{f^1}$ when restricted to G_i. Denote this as:

$$CP(\widehat{f^1}(U_n,\mathcal{H})\bigwedge G_i)=\Pr_{(x_1,x_2,h_1,h_2)\,\leftarrow\,(U_{2n},\mathcal{H}^2)}[\widehat{f^1}(x_1,h_1)=\widehat{f^1}(x_2,h_2)\bigwedge(x_1,h_1),(x_2,h_2)\in G_i]$$

We first give an upper-bound on this collision-probability (we note that the following upper-bound also holds when only one of the input pairs, e.g. (x_1,h_1), is required to be in G_i). Recall that $\widehat{f^1}(x,h)=(f(h(f(x))),h))$, hence in order for a collision to happen we must have $h_1=h_2$ which happens with probability $\frac{1}{|\mathcal{H}|}$ along with a collision in the first part of the output. We divide the collisions in the first part of the output into two types: The first type is a collision that happens either in the inputs or after the first invocation of f (i.e, $x_1=x_2$ or $f(x_1)=f(x_2)$), and the second type is a collision that happened only in the output (i.e., $f(x_1)\neq f(x_2)$ and $f^1(x_2,h)=f^1(x_1,h)$). Since it is required that $(x_1,h_1)\in G_i$ it holds that $D_f(f(x_1))\leq D_f(f^1(x_1,h))=i$ and therefore $\left|f^{-1}(f(x_1))\right|\leq 2^i$. Thus, the probability for a first-type collision is at most 2^{i-n}. If $f(x_1)\neq f(x_2)$ then by the pairwise independence of $\mathcal{H}$ it follows that given the value of $f^1(x_1,h)=f(h(f(x_1)))$, the value of $h(f(x_2))$ is uniformly distributed in $\{0,1\}^n$ and thus the value of $f^1(x_2,h)$ is a random element in $f(U_n)$. Thus, the probability that $f^1(x_2,h)=f^1(x_1,h)$ is at most 2^{i-n} as well (since $\left|f^{-1}(f^1(x_1,h))\right|\leq 2^i$). Altogether:

$$CP(\widehat{f^1}(U_n,\mathcal{H})\bigwedge G_i)\leq\frac{1}{|\mathcal{H}|}\left(2^{i-n}+2^{i-n}\right)=\frac{2^{i-n+1}}{|\mathcal{H}|}\tag{7}$$

We now give a lower-bound for the above collision-probability. The probability of getting a collision inside G_i together with being in T_i is clearly a lower-bound on the above collision-probability. For each slice, denote $\delta_i=\Pr[\widehat{f^1}(x,h)\in T_i\bigwedge(x,h)\in G_i]$. In order to have this kind of collision, we first request that both inputs are in G_i and yield outputs in T_i, which happens with probability δ_i^2. Then once inside T_i we required that both outputs collide, which happens with probability at least $\frac{1}{|T_i|}$. Altogether:

$$CP(\widehat{f^1}(U_n,\mathcal{H})\bigwedge G_i)\geq\delta_i^2\frac{1}{|T_i|}\tag{8}$$

Combining Equations 7 and 8 we get:

$$\frac{|T_i|\,2^{i-n-1}}{|\mathcal{H}|}\geq\frac{\delta_i^2}{4}\tag{9}$$

When taking a random output z and an independent h, the probability of hitting an element in T_i is at least $2^{i-n-1}/\left|\mathcal{H}\right|$ (since each output in T_i has preimage at least 2^{i-1}). But this means that $\Pr[(z,h) \in T_i] \geq |T_i|\, 2^{i-n-1}/\,|\mathcal{H}| \geq$ (by Equation 9) $\delta_i^2/4$. Finally the probability of hitting T is $\Pr[(z,h) \in T] = \sum_i \Pr[(z,h) \in T_i] \geq \sum_i \delta_i^2/4$. Since $\sum_i \delta_i^2 \geq (\sum_i \delta_i)^2/n$ and (by definition) $\sum_i \delta_i = \delta$, it holds that $\Pr[(z,h) \in T] \geq \delta^2/4n$ as claimed. ▮

We are now about to prove the second property of g. That is, we prove that with probability at least $\frac{1}{2}$, the value of $x^0 = f(x)$ is information-theoretically determined by the output of g. Let us start with some intuition. Given an output element $z = g(x, h, h_E)$ we define its zero-iterations as all elements $y = f(x)$ such that $g(x, h, h_E) = z$. We call a zero-iteration y "dominating" if conditioned on $g(x', h, h_E) = z$, the probability that $f(x') = y$ is close to one. When z has such a dominating zero-iterations, it holds w.h.p. that this is its originating zero-iteration. The proof follows by showing that a random output element has such a dominating zero-iteration with probability at least $\frac{1}{2}$.

In more detail, we call a zero-iteration y of $z = g(x, h, h_E)$ "heavy" if $D_f(y) \geq D_f(f^1(x,h))$ and make the following observations. When "ignoring" the second part of g's output, the "random nature" of the first part (i.e., $\widehat{f^1}(x,h) = (f(h(f(x))), h)$) implies that an output element is not likely to have too many heavy zero-iterations (though it might have lots of non-heavy ones). Therefore, the additional information given in the second part of g about its zero-iterations (i.e., $h_E(f(x))$) leaves, if any, only a single heavy zero-iteration and eliminates most of the non-heavy ones. Thus, conditioning that an output element has a heavy zero-iteration (which happens with probability at least $\frac{1}{2}$, due to a symmetry argument), it has w.h.p. only a single heavy zero iteration. Moreover, the above single heavy-zero-iteration is also a dominating one. Let us now turn to a more formal discussion.

Definition 4.5 (Zero-Iteration) *Let $z = (z_1, h, z_2, h_E) \in Im(g)$ and let $y = f(x) \in Im(f)$. We say that y is a* zero-iteration *of z if $g(x, h, h_E) = z$, where y is a* heavy-zero-iteration *of z if it is a zero-iteration and $D_f(y) \geq D_f(z_1)$. Finally, y is a* α-dominating-zero-iteration *of z if it is a zero-iteration and*

$$\frac{\Pr_{(x,h,h_E) \leftarrow (U_n, \mathcal{H}, \mathcal{H}_E)}[g(x,h,h_E) = z \bigwedge f(x) = y]}{\Pr_{(x,h,h_E) \leftarrow (U_n, \mathcal{H}, \mathcal{H}_E)}[g(x,h,h_E) = z]} \geq \alpha$$

Lemma 4.6 *Let z be a random element in $Im(g)$ conditioned that it has a heavy-zero-iteration. Then with probability at least $1 - \frac{1}{4n}$ it has a single heavy-zero-iteration which is a $(1 - \frac{1}{16n^2})$-dominating-zero-iteration.*

Proof: Let $(z_1, z_2) \in Im(f) \times \{0,1\}^m$ and $x \in \{0,1\}^n$, by the definition of g and D_f,

$$\Pr_{(h,h_E) \leftarrow (\mathcal{H}, \mathcal{H}_E)}[g(x,h,h_E) = (z_1, h, z_2, h_E)] \leq 2^{D_f(z_1)-n-m}$$

Now let $x' \in \{0,1\}^n$ such that $f(x') \neq f(x)$ and let S_{x',z_1,z_2} be the set of pairs $(h, h_E) \in \mathcal{H} \times \mathcal{H}_E$ that map x' to (z_1, h, z_2, h_E) (i.e., $g(x', h, h_E) = (z_1, h, z_2, h_E)$). By the pairwise independence of $\mathcal{H}$ and $\mathcal{H}_E$, the above equation still holds even when we restrict ourselves to $(h, h_E) \in S_{x',z_1,z_2}$. That is,

$$\Pr_{(h,h_E) \leftarrow S_{x',z_1,z_2}}[g(x,h,h_E) = g(x',h,h_E)] \leq 2^{D_f(z_1)-n-m}$$

and since the above is true for any x such that $f(x) \neq f(x')$,

$$\Pr_{(h,h_E) \leftarrow S_{x',z_1,z_2}, x \leftarrow U_n}[g(x,h,h_E) = g(x',h,h_E) \bigwedge f(x) \neq f(x')] < 2^{D_f(z_1)-n-m} \qquad (10)$$

Let I_{x',h,h_E} be the indicator random-variable that equals one if $\Pr_{x \leftarrow U_n}[g(x,h,h_E) = g(x',h,h_E) \bigwedge f(x) \neq f(x')] < 8n2^{D_f(f^1(x',h))-n-m}$. By Markov's inequality and Equation 10 we get,

$$\Pr_{(h,h_E) \leftarrow S_{x',z_1,z_2}}[I_{x',h,h_E}] \geq 1 - \frac{1}{8n} \tag{11}$$

The uniform distribution over $\mathcal{H} \times \mathcal{H}_E$ might be viewed as the distribution induced by choosing a random subset $S_{x',z_1,z_2} \subset \mathcal{H} \times \mathcal{H}_E$ according to its density (i.e., S_{x',z_1,z_2} is chosen with probability $\left|S_{x',z_1,z_2}\right| / |\mathcal{H} \times \mathcal{H}_E|$) and then choosing a uniform pair (h,h_E) in S_{x',z_1,z_2}. Since Equation 11 holds for any subset S_{x',z_1,z_2} and the above observation, it follows that,

$$\Pr_{(h,h_E) \leftarrow (\mathcal{H},\mathcal{H}_E)}[I_{x',h,h_E}] \geq 1 - \frac{1}{8n}$$

By a symmetry argument we have that the probability that a random output has a heavy-zero-iteration is at least $\frac{1}{2}$. Thus, by averaging over all $x' \in \{0,1\}^n$,

$$\Pr_{(x',h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[I_{x',h,h_E} \mid g(x',h,h_E) \text{ has a heavy-zero-iteration}] \geq 1 - \frac{1}{4n}$$

Conditioned on I_{x',h,h_E}, we have that for any zero-iteration y of $g(x',h,h_E)$ other than $f(x')$ it holds that $\Pr_{x \leftarrow U_n}[f(x) = y] \leq \Pr_{x \leftarrow U_n}[g(x,h,h_E) = g(x',h,h_E) \bigwedge f(x) \neq f(x')] < 8n2^{D_f(f^1(x',h))-n-m}$. On the other hand by the definition of D_f we have that $\Pr_{x \leftarrow U_n}[f(x) = y] \geq 2^{D_f(y)-n-1}$ and therefore $D_f(y) \leq D_f(f^1(x',h)) + log(8n) - m + 1 < D_f(f^1(x',h))$ (recall that $m = \lceil 3\log(n) + 8 \rceil$). Thus any zero-iteration other than $f(x')$ is not a heavy-zero-iteration of $f^1(x',h)$ and therefore,

$$\Pr_{(x',h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[I_{x',h,h_E} \bigwedge f(x') \text{ is the only heavy-zero-iteration of } g(x',h,h_E) \tag{12}$$
$$\mid g(x',h,h_E) \text{ has a heavy-zero-iteration}] \geq 1 - \frac{1}{4n}$$

Hence, with probability $1 - \frac{1}{4n}$ over a random choice of $z = g(x',h',h'_E)$ conditioned that z has a heavy-zero-iteration,

$$\frac{\Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[g(x,h,h_E) = g(x',h',h'_E) \bigwedge f(x) = f(x')]}{\Pr_{(x,h,h_E) \leftarrow (U_n,\mathcal{H},\mathcal{H}_E)}[g(x,h,h_E) = g(x',h',h'_E)]} = \frac{\Pr_{x \leftarrow U_n}[f(x) = f(x')]}{\Pr_{x \leftarrow U_n}[g(x,h',h'_E) = g(x',h',h'_E)]} =$$
$$\frac{\Pr_{x \leftarrow U_n}[f(x) = f(x')]}{\Pr_{x \leftarrow U_n}[g(x,h',h'_E) = g(x',h',h'_E) \bigwedge f(x) \neq f(x')] + \Pr_{x \leftarrow U_n}[f(x) = f(x')]} \geq$$
$$\frac{\Pr_{x \leftarrow U_n}[f(x) = f(x')]}{8n2^{D_f(f^1(x',h))-n}2^{-m} + \Pr_{x \leftarrow U_n}[f(x) = f(x')]} \geq \frac{2^{D_f(f(x'))-n-1}}{8n2^{D_f(f^1(x',h))-n}2^{-m} + 2^{D_f(f(x'))-n-1}} \geq$$
$$\frac{2^{D_f(f(x'))-n-1}}{8n2^{D_f(f(x'))-n}2^{-m} + 2^{D_f(f(x'))-n-1}} = \frac{1}{16n2^{-m}+1} \geq \frac{1}{\frac{1}{16n^2}+1} \geq 1 - \frac{1}{16n^2}$$

where the first inequality is by Equation 12, the second inequality is since (by definition) $\Pr_{x \leftarrow U_n}[f(x) = f(x')] \geq 2^{D_f(f(x'))-n-1}$ and the third inequality in since $D_f(f(x')) \geq D_f(f^1(x',h))$ (recall that $f(x')$ is a heavy-zero-iteration of $z = g(x',h',h'_E) = (f^1(x',h), h'_E(f(x')), h'_E)$). ▮

4.1.2 Constructing A Pseudo-Entropy Pair

The following theorem states that (a slightly modified version of) g along with the appropriate predicate is a $(\frac{1}{2}, \alpha)$-PEP for any α that is noticeably smaller than $\frac{1}{2n}$, where the modification is just in order to incorporate the GL hardcore predicate randomness in the input of g.

Theorem 4.7 *Let $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 4.2. For $r \in \{0,1\}^n$ let $g'(x, h, h_E, r) = (g(x, h, h_E), r)$ and let $b(x, h, h_E, r) = b_r(f(x))$, where b_r is the Goldreich-Levin predicate (see Theorem 2.7). Let $U_n, \mathcal{H}, \mathcal{H}_E$ and R_n be random variables uniformly distributed over $\{0,1\}^n, \mathcal{H}, \mathcal{H}_E$ and $\{0,1\}^n$ respectively, then the following hold:*

1. *$H(b(U_n, \mathcal{H}, \mathcal{H}_E, R_n) \mid g'(U_n, \mathcal{H}, \mathcal{H}_E, R_n)) \leq \frac{1}{2}$.*
2. *b is a $\left(\frac{1}{2} + \alpha\right)$-hard predicate of g', for any α that is noticeably smaller than $\frac{1}{2n}$.*

Hence (g', b) is a $(\frac{1}{2}, \alpha)$-PEP*, for any α that is noticeably smaller than $\frac{1}{2n}$.*

(Proving 1): Let β be the probability that a random output element of g has a $(1 - \frac{1}{16n^2})$-dominating-zero-iteration. Hence,

$$H(b(U_n, \mathcal{H}, \mathcal{H}_E, R_n) \mid g'(U_n, \mathcal{H}, \mathcal{H}_E, R_n)) = H(b(U_n, \mathcal{H}, \mathcal{H}_E, R_n) \mid (g(U_n, \mathcal{H}, \mathcal{H}_E), R_n))$$
$$\leq \beta H(\frac{1}{16n^2}, 1 - \frac{1}{16n^2}) + (1 - \beta) < \beta(\frac{1}{16n^2}(1 - \frac{1}{16n^2}))^{1/2} + (1 - \beta) < 1 - \beta(1 - \frac{1}{4n})$$

where the second inequality is due to the fact that $H(p, q) \leq (pq)^{1/\ln 4} < (pq)^{1/2}$. The proof of Lemma 4.3[1] also yields that the probability that x^0 is "heavier" than x^1 is at least $\frac{1}{2} + \frac{1}{2n}$ and therefore, the probability that z has a heavy-zero-iteration is at least $\frac{1}{2} + \frac{1}{2n}$. Where Lemma 4.6 states that conditioned that z has a heavy-zero-iteration, z has a $(1 - \frac{1}{16n^2})$-dominating-zero-iteration with probability at least $1 - \frac{1}{4n}$. Hence $\beta \geq \frac{1}{2} + \frac{1}{2n} - \frac{1}{4n}$ and thus $H(b(U_n, \mathcal{H}, \mathcal{H}_E, U_n) \mid g'(U_n, \mathcal{H}, \mathcal{H}_E, R_n)) < 1 - (\frac{1}{2} + \frac{1}{4n})(1 - \frac{1}{4n}) < \frac{1}{2}$. ∎

(Proving 2): By Lemma 4.3 it is hard to compute x^0 given $g(x, h, h_E)$ over $\frac{1}{2} + \frac{1}{2n}$ fraction of g's inputs. Hence, by Theorem 2.7 is hard to predict the Goldreich-Levin predicate invoked on x^0 with probability non-negligibly better than $\frac{1}{2} + \frac{1}{2n}$. Thus, b is a $\left(\frac{1}{2} + \alpha\right)$-hard predicate of g' for any α that is noticeably smaller than $\frac{1}{2n}$. ∎

5 Hardness Amplification Of Regular One-Way Functions

In this section we present an efficient hardness amplification of any regular weak one-way function. As mentioned in the introduction (Section 1.2), the key to hardness amplification lies in the fact that every α-weak one-way function has a *failing-set* for every efficient algorithm. This is a set of density almost α that the algorithm fails to invert f upon. Sampling sufficiently many independent inputs to f is bound to hit *every* failing set and thus fail every algorithm. Indeed, the basic hardness amplification of Yao [Yao82] does exactly this. Since independent sampling requires a long input, we turn to use the randomized iterate, which together with the derandomization method, reduces the input length to $\mathcal{O}(n \log n)$.

As a first step, we show that $\widehat{f_m}$ (the m^{th} explicit randomized iterate of f) is a strong one-way function (for the proper choice of m). The basic intuition is that every iteration of the randomized iterate gives a random element in $Im(f)$ and thus these iterations are bound to hit every significantly large failing-set. However, the proof is more subtle than this. Details follow:

5.1 The Construction

We show that the m^{th} explicit randomized iterate of an $\alpha(n)$-weak (unknown) regular one-way function, where m equals $\lceil \frac{4n}{\alpha(n)} \rceil$, is a (strong) one-way function. For simplicity we assume 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.

Theorem 5.1 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a regular $\alpha(n)$-weak one-way function, let $m = \lceil \frac{4n}{\alpha(n)} \rceil$ and let $\widehat{f^m}$ and $\mathcal{H}$ be as in Definition 3.1. Then $\widehat{f^m}$ is a one-way function.*

More precisely, let $n_s = nm$, let $T_f(n)$ be the computing-time of f and let A be an algorithm that inverts $\widehat{f^m}$ with probability $\varepsilon(n_s)$ and running-time $T_A(n_s)$. Then, algorithm A can be used to construct an algorithm that inverts f with probability $1 - \alpha/2 - 2^{-n}$ and running-time $\mathcal{O}\left((T_f(n) + T_A(n_s))n^{11}/\alpha(n)^7\varepsilon(n_s)^6\right)$.

Proof: Let A be an algorithm that inverts $\widehat{f^m}$ with probability $\varepsilon(n_s)$, that is $\Pr_{\widehat{z} \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z})] > \varepsilon$. (For clarity we write ε and α instead of $\varepsilon(n_s)$ and $\alpha(n)$, and mark the output elements of $\widehat{f^m}$ with $\widehat{\ }$). We prove that the existence of A implies the existence of an efficient algorithm B^A that inverts f with non-negligible success probability over any set of density $\alpha/2$. Thus B^A has no failing-set of density $\alpha/2$ w.r.t. f, a contradiction to the hardness of f. Formally,

Definition 5.2 (Failing-Set) *Let $g : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ and let δ be some positive function over $\mathbb{N}$. We say that $\{S_n \subseteq Im(g(\{0,1\}^n)\}_{n\in\mathbb{N}}$ is a δ-*failing-set *of g w.r.t. an algorithm D, if $\Pr_{x \leftarrow U_n}[f(x) \in S_n] \geq \delta(n)$ and,*

$$\Pr_{y \leftarrow g(U_n)}[D(y) \in g^{-1}(y) \mid y \in S_n] \in neg(n)$$

We first prove (in Lemma 5.3) that f has a $\frac{\alpha}{2}$-failing-set for *any* efficient algorithm. Then we prove (in Lemma 5.4) that the existence of A implies the existence of an efficient algorithm B^A that inverts f with non-negligible probability over *any* set of density $\alpha/2$, thus the algorithm B^A has no $\frac{\alpha}{2}$-failing-set and a contradiction is derived.

Lemma 5.3 *Let g be an γ-weak one-way function, then for any* PPT *D there exists a $\frac{\gamma}{2}$-failing-set of g.*

More precisely, let $T_g(n)$ and $T_D(n)$ be the running-time of g and D respectively. Given that D inverts g with probability δ over any subset of density $\gamma/2$, then algorithm D can be used to construct an algorithm that inverts g with success probability $1 - \gamma/2 - 2^{-n}$ and running-time $\mathcal{O}(nT_g(n)T_D(n)/\delta)$.

Proof: For a fixed n, Let $L \subseteq Im(g)$ be the set of elements that D inverts with probability less than δ (i.e., for each $y \in L$ it holds that $\Pr[D(y) \in g^{-1}(y)] < \delta$). By the definition of D if follows that $\Pr_{x \leftarrow U_n}[g(x) \in L] < \gamma/2$. Now let M^D be the algorithm that on input $y \in Im(g)$ invokes D on input y repeatedly n/δ times and checks for success. Hence, M^D inverts any input $y \in Im(g)\backslash L$ with probability $1 - 2^{-n}$ and thus, M^D inverts g with probability $1 - \gamma/2 - 2^{-n}$. Contradicting the hardness of g. ▌

We next show how to use the inverting algorithm A to construct an algorithm B^A that has no $\frac{\alpha}{2}$-failing-set of f. For a given $i \in [m]$ let M_i^A be the following procedure for inverting the last-iteration of $\widehat{f^i}$:

```
$M_i^A$ on input $(y, h_1 \dots, h_i) \in Im(\widehat{f^i})$:
    1. Choose uniformly and independently at random $h_{i+1}, \dots, h_m \in \mathcal{H}$ and let $w = f^{m-i}(y, h_{i+1}, \dots, h_m)$.
    2. Apply $A(w, h_1, \dots, h_m)$ to get an output $(x, h'_1, \dots, h'_m)$.
    3. If $f^i(x, h_1, \dots, h_i) = y$ output $f^{i-1}(x, h_1, \dots, h_{i-1})$, otherwise abort.
```

and let B^A be the following procedure for inverting f:

```
$B^A$ on input $y \in Im(f)$:
    1. Choose a random $h_1, \dots, h_m \in \mathcal{H}$.
    2. For each $i \in [m]$ set $x_i = M_i^A(y, h_1, \dots, h_i)$.
    3. If there exists an $i$ such that $f(x_i) = y$ output $x_i$, otherwise abort.
```

Lemma 5.4 *B^A inverts f over any set of density $\alpha/2$ with probability $\Omega(\frac{\varepsilon^6}{m^7})$ and running-time $\mathcal{O}(m^2(T_f(n) + T_A(n_s)))$.*

Note that combining Lemma 5.3 and Lemma 5.4 yields the existence of an algorithm that inverts f with probability $1 - \alpha/2 - 2^{-n}$ and running-time $\mathcal{O}\left((T_f(n) + T_A(n_s))nm^{10}/\varepsilon^6\right)$, which proves Theorem 5.1. ▮

Proof: [of Lemma 5.4] The heart of the proof is in the following lemma that states that for any sufficiently dense subset $S \subseteq Im(f)$ there exists an index i such that M_i^A inverts the last-iteration of $\widehat{f^i}$ over $S \times \mathcal{H}^i$ with non-negligible success probability.

Lemma 5.5 *For any subset $S \subseteq Im(f)$ of density, w.r.t. $f(U_n)$, at least $\alpha/2$, there exists an index $i \in [m]$ such that the following holds,*

$$\Pr_{\widehat{z}=(w,h_1,\dots,h_i) \leftarrow \widehat{f^i}(U_n,\mathcal{H}^i)}[M_i^A \text{ inverts the last-iteration of } \widehat{z} \mid w \in S] > \varepsilon^2/16m^2$$

Let $S \subseteq Im(f)$ be a subset of density $\beta \geq \alpha/2$ w.r.t. $f(U_n)$. For each $i \in [m]$ let $L_i \subseteq S \times \mathcal{H}^i$ be the set of elements that algorithm M_i^A inverts their last iteration with probability at least $\frac{\varepsilon^2}{32m^2}$.

Claim 5.6 *$\exists i \in [m]$ such that L_i is of density $\frac{\beta\varepsilon^2}{32m^2}$ w.r.t. $\widehat{f^i}(U_n, \mathcal{H}^i)$.*

Proof: By Lemma 5.5 there exists an $i \in [m]$ such that the following holds (all of the following probabilities are over $\widehat{z} = (w, h_1, \dots, h_i) \leftarrow \widehat{f^i}(U_n, \mathcal{H}^i)$),

$$\Pr[f(h_i(M_i^A(\widehat{z}))) = w \mid w \in S] > \varepsilon^2/16m^2$$

On the other hand,

$$\Pr[f(h_i(M_i^A(\widehat{z}))) = w \mid w \in S] =$$
$$\Pr[f(h_i(M_i^A(\widehat{z}))) = w \bigwedge \widehat{z} \in L_i \mid w \in S] + \Pr[f(h_i(M_i^A(\widehat{z}))) = w \bigwedge \widehat{z} \in \overline{L_i} \mid w \in S] \leq$$
$$\Pr[\widehat{z} \in L_i \mid w \in S] + \frac{\varepsilon^2}{32m^2}$$

where the last inequality is by the definition of L_i. Thus $\Pr[\widehat{z} \in L_i \mid w \in S] > \frac{\varepsilon^2}{32m^2}$ and therefore

$$\Pr[\widehat{z} \in L_i] \geq \Pr[\widehat{z} \in L_i \mid w \in S] \cdot \Pr[w \in S] > \frac{\beta\varepsilon^2}{32m^2}$$

▮

Let $j \in [m]$ be the index guaranteed by the above claim to have

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_j) \leftarrow \widehat{f^j}(U_n,\mathcal{H}^j)}[\widehat{z} \in L_j] \geq \Pr[\widehat{z} \in L_j \mid w \in S] \cdot \Pr[w \in S] > \frac{\beta\varepsilon^2}{32m^2}$$

By Lemma 3.4 (letting $T = L_j$) we have that,

$$\Pr_{(w,h_1,\ldots,h_j) \leftarrow (f(U_n),\mathcal{H}^j)}[(w,h_1,\ldots,h_j) \in L_j] \geq (\frac{\beta\varepsilon^2}{64m^2})^2/(j+1) \in \Omega(\frac{\beta^2\varepsilon^2}{m^4})$$

and thus conditioning on $w \in S$,

$$\Pr_{(w,h_1,\ldots,h_j) \leftarrow (f(U_n),\mathcal{H}^j)}[(w,h_1,\ldots,h_j) \in L \mid w \in S] \in \Omega(\frac{\beta\varepsilon^2}{m^4}) = \Omega(\frac{\alpha\varepsilon^2}{m^4})$$

Since M_j^A inverts each of the elements of L_j with probability $\frac{\varepsilon^2}{32m^2}$, then

$$\Pr_{(w,h_1,\ldots,h_j) \leftarrow (f(U_n),\mathcal{H}^j)}[M_j^A \text{ inverts the last-iteration of } (w,h_1,\ldots,h_j) \mid w \in S] \in \Omega(\frac{\alpha\varepsilon^6}{m^7})$$

Finally, since B^A enumerates all possible $i \in [m]$, it inverts f over S with probability $\mathcal{O}(\frac{\alpha\varepsilon^6}{m^7})$. Note that each invocation of M_i^A involves m invocation of f and one invocation of A, hence the running-time of B^A is $\mathcal{O}(m^2(T_f(n) + T_A(n_s)))$. ▮

Proof: [of Lemma 5.5] Through the rest of this section we allow ourselves to view $\widehat{f^m}$ and f^m also as functions over $Im(f) \times \mathcal{H}^m$ rather than over $\{0,1\}^n \times \mathcal{H}^m$, where $f^m(y,h_1,\ldots,h_m)$ for $y \in Im(f)$ stands for $f^m(x,h_1,\ldots,h_m)$ for some $x \in f^{-1}(y)$. (Note that the above is well defined since for any $x,x' \in f^{-1}(y)$ it holds that $f^m(x,h_1,\ldots,h_m) = f^m(x',h_1,\ldots,h_m)$).

By Lemma 3.4 the collision-probability of f^m equals $\frac{m}{|Im(f)|}$, actually the proof yields the following: For any $y \neq y' \in Im(f)$

$$\Pr_{(h_1,\ldots,h_m) \leftarrow \mathcal{H}^m}[f^m(y,h_1,\ldots,h_m) = f^m(y',h_1,\ldots,h_m)] = \frac{m}{|Im(f)|} \quad (13)$$

For any $\widehat{z} = (w,h_1,\ldots,h_m) \in Im(\widehat{f^m})$ we define the set $B_{\widehat{z}} = \left\{y \in Im(f) \mid \widehat{f^m}(y,h_1,\ldots,h_m) = \widehat{z}\right\}$. Therefore, for any $y \in Im(f)$,

$$\operatorname*{Ex}_{(h_1,\ldots,h_m) \leftarrow \mathcal{H}^m}[\left|B_{\widehat{f^m}(y,h_1,\ldots,h_m)}\right|] =$$
$$1 + \sum_{y' \neq y \in Im(f)} \operatorname*{Ex}_{(h_1,\ldots,h_m) \leftarrow \mathcal{H}^m}[f^m(y',h_1,\ldots,h_m) = f^m(y,h_1,\ldots,h_m)] < m+1$$

where the inequality is due to Equation 13. Hence, by Markov's inequality $Pr_{\widehat{z} \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[|B_{\widehat{z}}| > 4m/\varepsilon] < \varepsilon/2$, and since we assumed that A is inverts $\widehat{f^m}$ with probability ε,

$$\Pr_{\widehat{z} \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z}) \bigwedge |B_{\widehat{z}}| \leq 4m/\varepsilon] > \varepsilon/2 \tag{14}$$

Now let $S \subseteq Im(f)$ be a set of density $\alpha/2$. By the pairwise independence of $\mathcal{H}$ (actually a one-wise family suffices), we have that for any $y \in Im(f)$, $i \in [m]$ and $h_1, \ldots, h_{i-1} \in \mathcal{H}$, it holds that $\Pr_{h_i \leftarrow \mathcal{H}}[f^i(y, h_1, \ldots, h_i) \in S] \geq \alpha/2$. Since each h is chosen independently, then for any $y \in Im(f)$ it holds that $\Pr_{(h_1,\ldots,h_m) \leftarrow \mathcal{H}^m}[\exists i \in [m] \text{ such that } f^i(y, h_1, \ldots, h_i) \in S] > 1 - 2^{-2n}$ (recall that $m = \lceil \frac{4n}{\alpha} \rceil$). Hence by a union bound,

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_m) \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[\forall y \in B_{\widehat{z}}\ \exists i \in [m] \text{ such that } f^i(y, h_1, \ldots, h_i) \in S] > 1 - |B_{\widehat{z}}|\, 2^{-2n} \geq 1 - 2^{-n}$$

Combining the above Equation and Equation 14 we have that,

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_m) \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z}) \bigwedge |B_{\widehat{z}}| \leq 4m/\varepsilon \bigwedge \forall y \in B_{\widehat{z}}\ \exists i \in [m] \text{ such that } f^i(y, h_1, \ldots, h_i) \in S]$$
$$> \varepsilon/2 - 2^{-n} > \varepsilon/4$$

Let $A(\widehat{z})_1$ be the first output element of $A(\widehat{z})$. Clearly if $A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z})$ then $f(A(\widehat{z})_1) \in B_{\widehat{z}}$. Thus,

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_m) \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z}) \bigwedge |B_{\widehat{z}}| \leq 4m/\varepsilon \bigwedge \exists i \in [m] \text{ such that } f^i(A(\widehat{z})_1, h_1, \ldots, h_i) \in S]$$
$$> \varepsilon/4$$

and by an averaging argument there exists an index $k \in [m]$ such that,

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_m) \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m)}[A(\widehat{z}) \in \widehat{f^m}^{-1}(\widehat{z}) \bigwedge |B_{\widehat{z}}| \leq 4m/\varepsilon \mid f^k(A(\widehat{z})_1, h_1, \ldots, h_k) \in S] > \varepsilon/4m$$

Since we have required that $B_{\widehat{z}}$ is small (i.e., less than $4m/\varepsilon$) and since by the regularity of f any elements in $B_{\widehat{z}}$ happens with the same probability, then

$$\Pr_{\widehat{z}=(w,h_1,\ldots,h_m) \leftarrow \widehat{f^m}(U_n,\mathcal{H}^m), y \leftarrow B_{\widehat{z}}}[f(A(\widehat{z})_1) = y \mid f^i(A(\widehat{z})_1, h_1, \ldots, h_i) \in S] > \varepsilon/4m \cdot \varepsilon/4m = \varepsilon^2/16m^2$$

or equivalently,

$$\Pr_{(x,h_1\ldots,h_m) \leftarrow (U_n,\mathcal{H}^m)}[f(A(\widehat{f^m}(x, h_1 \ldots, h_m))_1) = f(x) \mid f^i(x, h_1 \ldots, h_i) \in S] > \varepsilon^2/16m^2$$

To conclude, we note that the above probability is a lower-bound on the probability of the following event: M_i^A inverts the last-iteration of $\widehat{f^i}(x, h_1 \ldots, h_i)$ conditioned that $f^i(x, h_1 \ldots, h_i) \in S$. ▮

5.2 An Almost-Linear-Input Construction

In this section we derandomize the randomized iterate used in Section 5.1 to get a (strong) one-way function with input length $\mathcal{O}(n \log n)$. We use the bounded-space generator of either [Nis92] or [INW94] (see Theorem 2.12).

Theorem 5.7 *Let f, m and f^m be as in Theorem 5.1, let BSG be a generator against $(2n, n+1, 2n)$-LBP with seed length $\tilde{n} \in \mathcal{O}(n \log n)$ and error 2^{-2n}. Define $f' : \{0,1\}^n \times \{0,1\}^{\tilde{n}} \to \{0,1\}^n \times \{0,1\}^{\tilde{n}}$ as*

$$f'(x, \tilde{h}) = (f^m(x, h_1, \ldots, h_m), \tilde{h})$$

where $x \in \{0,1\}^n$, $\tilde{h} \in \{0,1\}^{\tilde{n}}$ and $h_1, \ldots, h_m = BSG(\tilde{h})$. Then f' is a one-way function.

Proof idea: The proof of the derandomized version follows the proof of Theorem 5.1. In the proof we used the following properties of the family $\mathcal{H}$.

1. Collision-probability - for any $y \neq y' \in Im(f)$ and $k \in \{0, \ldots, m\}$

$$\Pr_{(h_1,\ldots,h_k) \leftarrow \mathcal{H}^k}[f^k(y, h_1, \ldots, h_k) = f^k(y', h_1, \ldots, h_k)] = \frac{k}{|Im(f)|}$$

2. Hitting - for all $y \in Im(f)$ and $S \subseteq Im(f)$ of density $\alpha/2$

$$\Pr_{(h_1,\ldots,h_m) \leftarrow \mathcal{H}^m}[\exists i \in [m] \text{ such that } f^m(y, h_1, \ldots, h_m) \in S] > 1 - 2^{-2n}$$

Note that the above two properties can be verified by an $(2n, n+1, 2n)$-LBP. Thus, when choosing the h's as the output of a 2^{-2n}-error BSG against $(2n, n+1, 2n)$-LBP's, the above properties hold with deviation at most 2^{-2n}. Going through the proof of Theorem 5.1, it is not hard to verify that the proof remains valid also when the above deviations are taking into account. (See the proof of Theorem 3.6 for a more detailed proof on a similar derandomization).

The Overall Time-Success Ratio: Combining the above and Theorem 5.1 we get the following: Let $n_s = \tilde{n} + n$, let $T_f(n)$ be the computing-time of f and let A be an algorithm that inverts $\widehat{f^m}$ with probability $\varepsilon(n_s)$ and running-time $T_A(n_s)$. Then, algorithm A can be used to construct an algorithm that inverts f with probability $1 - \alpha/2 - 2^{-n}$ and running-time $\mathcal{O}\left((T_f(n) + T_A(n_s))n^8/\alpha(n)^7\varepsilon(n_s)^6\right)$.

References

[AIK04] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in nc^0. In *45th IEEE Symposium on Foundations of Computer Science*, pages 166–175, 2004.

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23th Annual Symposium on Foundations of Computer Science*, pages 112–117, 1982.

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

[DI99] G. Di Crescenzo and R. Impagliazzo. Security-preserving hardness-amplification for any regular one-way function. In *31st ACM Symposium on the Theory of Computing*, pages 169–178, 1999.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(2):792–807, 1986.

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

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

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

[Gol01] O. Goldreich. *Foundations of Cryptography - Volume 1.* Cambridge University Press, 2001.

[Hås90] J. Håstad. Pseudo-random generators under uniform assumptions. In *22nd ACM Symposium on the Theory of Computing*, pages 387–394, 1990.

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

[HL92] A. Herzberg and M. Luby. Pubic randomness in cryptography. In *Advances in Cryptology - CRYPTO '92, Lecture Notes in Computer Science*, volume 740, pages 421–432. Springer, 1992.

[Hol05] T. Holenstein. Key agreement from weak bit agreement. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 664–673, 2005.

[IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *30th IEEE Symposium on Foundations of Computer Science*, pages 230–235, 1989.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *21st ACM Symposium on the Theory of Computing*, pages 12–24, 1989.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *26th ACM Symposium on the Theory of Computing*, pages 356–364, 1994.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

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

[LR88] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing*, 17(2):373–386, 1988.

[Nao91] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

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

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 1993.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences (JCSS)*, 52(1):43–52, 1996.

[Phi93] Security preserving hardness amplification using PRGs for bounded space. Preliminary Report, Unpublished, 1993.

[Sha02] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[SZ99] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal of Computing*, 1999.

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

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

Appendix

A Using a Pseudo-Entropy Pair to Implement a Pseudorandom Generator

Following is a high-level overview of the reduction between a pseudo-entropy pair (PEP) (Definition 4.1) and a pseudorandom generator (Definition 2.9). This overview is mainly based on [HILL99], though it proves a more general statement than the original one. Specifically, the original proof of [HILL99] was tailored for the specific PEP used in their paper and the proof here is modified to enable the use of any PEP. For this modification we also generalize the [HILL99] "uniform extraction lemma" ([HILL99, Lemma 6.5]) and give it a new proof based on the Holenstein's "uniform hardcore lemma" [Hol05, Lemma 2.5]. [11] We note that the latter might be of an independent interest.

Throughout this section we use somewhat different terminology than the one used by [HILL99] and in particular the notion of the pseudo-entropy pair was only implicit in [HILL99]. We also use general explicit-extractors (see Definition A.3) rather than the efficient family of pairwise-independent hash functions that was used as an extractor in several steps in the original construction.

A.1 High-Level Overview

The pseudo-entropy of a distribution is at least k if it is computationally-indistinguishable from some samplable distribution that has entropy k. Informally, a pseudo-entropy pair (PEP) is a pair of function and predicate (g, b) such that the conditional *pseudo-entropy* of the distribution $b(x)$ when given $g(x)$, is noticeably larger than the conditional *entropy* of this distribution. Namely, (g, b) is a (δ, γ)-PEP, if the conditional pseudo-entropy of b given g is at least $\delta + \gamma$ while the conditional entropy is not more than δ.[12] The reduction is achieved in three steps as follows.

[11] The fact that [HILL99, Lemma 6.5] can be proved using [Hol05, Lemma 2.5], was mentioned in the oral presentation of [Hol05].

[12] For some technical reasons, it does not suffice that b has a conditional pseudo-entropy $\delta + \gamma$, but rather we need that b is a $(\delta + \gamma)$-hard predicate of g (see Definition 2.8).

A.1.1 Using a Pseudo-Entropy Pair to Implement a False-Entropy Generator

A "false-entropy generator" is a function where the pseudo-entropy of the *output's* distribution is noticeably larger than it's real entropy. The basic idea of the construction is to use extractor in order to extract the conditional pseudo-entropy of b and then concatenate the extractor's output to g. More precisely, let (g, b) be a (δ, γ)-PEP, for the proper value of m we let g^m be a concatenation of m independent copies of g and let Ext be an extractor for min-entropy $(\delta + \gamma/2)m$ with output length $(\delta + \gamma/4)m$. We let $g'(x) \overset{\text{def}}{=} (g^m(x_1, \ldots, x^m), Ext(b(x_1), \ldots, b(x_m)))$ and look at the conditional distribution of the extractor's input (i.e., $b(x_1), \ldots, b(x_m)$) given the value of $g^m(x_1, \ldots, x^m)$. When choosing m properly, we are guaranteed by the computational unpredictability of b (recall that b is a $(\delta+\gamma)$-hard predicate of g) that the latter (conditional) distribution is computationally-indistinguishable from some distribution with min-entropy $(\delta + \gamma/2)m$. (The latter claim is the crux of this step, further details are given in Section A.2). Thus, the output of the extractor seems uniform to any efficient algorithm that possess $(b(x_1), \ldots, b(x_m))$. Therefore, the output distribution of g' is computational-indistinguishable from the output distribution of g^m concatenated with the uniform distribution over strings of length $(\delta + \gamma/4)m$. Hence, the output pseudo-entropy of g' is at least the output entropy g^m plus $(\delta + \gamma/4)m$, which is larger than the output entropy of g' (i.e., at most the output entropy of g^m plus δm) and thus g' is a false-entropy generator. Since the above implementation diverts from the original one, we describe it fully in Section A.2.

A.1.2 Using a False-Entropy Generator to Implement a Pseudo-Entropy Generator

A "pseudo-entropy generator" is a function whose *output's* pseudo-entropy is noticeably larger than its *input's* (unconditional) entropy. The basic idea of the construction is similar to the one used in the previous section. Let g be a false-entropy generator and let ℓ be the conditional entropy of g's input given its output. For the proper value of m we let g^m be a concatenation of m independent copies of g. We use an extractor in order to extract (a bit less than) ℓm bits from the input of g^m and then concatenate the extractor's output to the output of g^m. Thus, the output distribution of the new primitive is indistinguishable (information-theoretically) from the output distribution of g^m concatenated with the uniform distribution over strings of length ℓm. Therefore, the new primitive output pseudo-entropy is more than its (unconditional) input entropy.

In the above step, however, we encounter an additional difficulty in comparison to the first step of the reduction (see Section A.1.1). Namely the value of ℓ (i.e., the conditional entropy of g's input) is not given and might not be efficiently computable. Note that if we underestimate ℓ, that is we extract too few bits, then the output pseudo-entropy of the new primitive might be less than the its input entropy. On the other hand, if we extract too many bits, then the pseudo-entropy of the underlying false-entropy generator might disappear altogether and thus the pseudo-entropy of the new primitive is simply its input entropy.[13] It turns out, however, that a rather "rough" estimation of the value of ℓ suffices (i.e., an estimation that can be expressed using logarithmic number of bits). Such a construction that requires a short hint to go through was called a *mildly* non-uniform construction in [HILL99]. Combining the above non-uniform construction with the reduction to pseudorandom generator's (see Section A.1.3), gives a mildly non-uniform construction of a pseudorandom generator. The non-uniform pseudorandom generator is then used to construct a uniform one by the following "combiner".

1. For each possible value of the non-uniform advice construct a candidate pseudorandom gen-

[13]Consider for example the extreme case where we extract all the (unconditional) entropy of the input, clearly the output of the new primitive is easily distinguished from any distribution with more entropy.

erator.

2. Increase the output length of each candidate using the standard transformation of Goldreich and Micali (c.f. [Gol01, Theorem 3.3.3]), such that the output of each candidate is longer than the total input length of all the candidates.
3. Combine the outputs of all generators into one output using an XOR of the outputs to get a uniform pseudorandom generator.

Note that the above enumeration requires an altogether longer input as well as sequential applications of the underlying one-way function and thus reduces the efficiency and security of the resulting construction.

Remark: Note that in the [HILL99] implementation of a PEP (see Section A.4), the upper-bound of the (real) conditional entropy of b given g (i.e., δ) is unknown (unlike in the PEP presented in this paper, see Section 4.1.2, where it is guaranteed to be $\frac{1}{2}$). Thus, when the HILL PEP is used in the first step of the reduction (see Section A.1.1), it yields a mildly non-uniform false-entropy generator. We stress that in such a case the two non-uniform hints (in the two different steps of the HILL construction) are different hints.[14] Therefore, each such non-uniformity should be resolved separately or alternatively the non-uniformity used is of double length. In particular, the deterioration in the security and efficiency of the resulting pseudorandom generator is accumulated (i.e., doubled).

A.1.3 Using a Pseudo-Entropy Generator to Implement a Pseudorandom Generator

In the final step of the reduction, an extractor is applied to the output of the pseudo-entropy generator. The resulting output distribution is computationally-indistinguishable from the uniform distribution of strings of length longer than the input length. Therefore, we have constructed a pseudorandom generator. Again, in the actual construction we consider the concatenation of many independent copies of the pseudo-entropy generator.

Remark: In each of the above steps a concatenation of many independent copies of the underlying primitive is required. It turns out that doing this repetition only once, say in the first step, suffices for all the other steps as well.

A.2 A False-Entropy Generator from a Pseudo-Entropy Pair

As previously mentioned, the first step of the [HILL99] reduction from pseudorandom generator to a PEP (that is, a false-entropy generator using a PEP) was tailored for their specific implementation and does not suit the general definition of a PEP (Definition 4.1) used in this paper. Following is a new reduction that suits any PEP. We start with some definitions.

[14]Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be the underlying one-way function of the HILL PEP, and let R_n be the random variable defined by $D_f(f(U_n))$. The two non-uniform hints required by the first and second steps of the HILL construction are the expectation of R_n and the expectation of R_n^2 respectively. Interestingly, when using our new PEP (Section 4.1.2) to construct the false-entropy generator, the only non-uniform hint needed is the expectation of R_n^2 (where now R_n is defined w.r.t. the one-way function used in our construction).

Definition A.1 (Pseudo-Entropy) *Let $\{X_n\}$ be a distribution ensemble and let k be a positive function over $\mathbb{N}$. The* pseudo-entropy *of $\{X_n\}$ is at least k, if there exists a polynomial-time* samplable *distribution ensemble $\{Y_n\}$ which is computationally-indistinguishable from $\{X_n\}$ and $H(Y_n) \geq k(n)$.*

Definition A.2 (False-entropy generator) *Let $g : \{0,1\}^n \rightarrow \{0,1\}^{\ell(n)}$ be a polynomial-time computable function. g is a* false-entropy generator *if the pseudo-entropy of $g(U_n)$ is noticeably larger than n.*

Definition A.3 (explicit strong extractors) *[NZ96] A polynomial-computable function $Ext : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ is an* explicit (k, ε)-strong extractor *if for every distribution X over $\{0,1\}^n$ with $H_\infty(X) \geq k$, the distribution $(Ext(X,Y), Y)$ is ε-close to (U_m, Y) where Y is uniform over $\{0,1\}^d$.*

We find that the presentation of the construction and proof is clearer when taking the second parameter of the PEP to be a specific real function (this parameter measures the gap between real and pseudo entropy). In particular, we use the function $\frac{1}{3n}$ that fits both the HILL and our constructions. The extension to the general case is straightforward.

Construction A.4 (Constructing a False-entropy-generator using a PEP) *Let (f, b) be a $(\delta, \frac{1}{3n})$-PEP. Let $r = 8(\delta n^3 + \frac{n^2}{12})$, let $d \in \theta(n)$ and let $Ext : \{0,1\}^{8n^3} \times \{0,1\}^d \rightarrow \{0,1\}^{r+d}$ be an explicit $(8(\delta n^3 + \frac{n^2}{6}), 2^{-n})$-strong extractor.*[15] *We define g as:*

$$g(x_1, \ldots, x_{8n^3}, y) = (f(x_1), \ldots, f(x_{8n^3}), y, Ext(b(x_1), \ldots, b(x_{8n_3}), y))$$

where $x_1, \ldots, x_{8n^3} \in \{0,1\}^n$ and $y \in \{0,1\}^d$.

Lemma A.5 *There exists a samplable distribution ξ with the following properties.*

1. *$H(\xi) \geq H(g(U_{8n^4+d})) + \frac{2}{3}n^2$.*
2. *ξ and $g(U_{8n^4+d})$ are computationally indistinguishable.*

Hence, g is a false-entropy generator.

Proof: Let ξ be the output distribution of the following function:

$$g'(x_1, \ldots, x_{8n^3}, y, z) = (f(x_1), \ldots, f(x_{8n^3}), y, z) \tag{15}$$

where $x_1, \ldots, x_{8n^3} \in \{0,1\}^n$, $y \in \{0,1\}^d$ and $z \in \{0,1\}^r$. Clearly ξ is polynomial-time samplable, we prove that it also fulfills the other properties of Lemma A.5.

(Proving 1): Since ξ and $g(U_{8n^4+d})$ are identical when excluding the last r bits, then their entropy only varies in the additional entropy of the last r bits. In ξ the last r bits are uniformly and independently chosen and thus have entropy r. On the other hand, the additional entropy in the last r bits of $g(U_{8n^4+d})$ is no more than the conditional entropy of the bits $(b(x_1), \ldots, b(x_{8n_3}))$ when given the first part of $g(U_{8n^4+d})$ (i.e., $(f(x_1), \ldots, f(x_{8n_3}))$). Since $(b(x_1), \ldots, b(x_{8n_3}))$ is a concatenation of independent random-variables, the above additional entropy equals $8n^3$ times the additional entropy of $b(x)$ given $f(x)$. Finally, since (f, b) is a $(\delta, \frac{1}{3n})$-PEP, the above additional entropy is no larger than $\delta 8n^3$ and thus $H(\xi) - H(g(U_{8n^4+d})) \geq r - \delta 8n^3 = \frac{8n^2}{12} = \frac{2}{3}n^2$. ▮

[15]For example we take the extractor of [SZ99].

(Proving 2): Let $L \subseteq \{0,1\}^n$ be any subset of density $\delta + \frac{1}{3n}$, therefore the expectation of the number of inputs in $\{x_1, \ldots, x_{8n^3}\}$ that hit L is $8(\delta n^3 + \frac{n^2}{3})$. By a Chernoff bound it holds that the probability that at least $8(\delta n^3 + \frac{n^2}{6})$ out of $\{x_1, \ldots, x_{8n^3}\}$ are in L is more than $1 - 2e^{-n}$. The proof is concluded by applying the uniform extraction lemma (Lemma A.6) that is stated and proved in the next section. The Lemma implies that no PPT can distinguish between $g(U_{8n^4+d})$ and ξ with probability (non-negligibly) better than $2e^{-n} + 2^{-n} < 3 \cdot 2^{-n}$. ∎

A.3 A Uniform Extraction Lemma

The following lemma is a generalization of the (uniform version) of Yao's XOR lemma. Given m independent "δ-hard" bits (i.e., it is hard to predict each bit with probability better than $1-\delta/2$), we would like to extract approximately δm pseudorandom bits out of them. The version we present here generalizes [HILL99, Lemma 6.5]). In particular the original lemma required the hardcore predicate to have a hardcore-set (i.e., a subset of inputs such that the value of the predicate is unpredictable (computationally) over this subset), where in the following lemma this property is no longer required. In addition, the original lemma was tailored for the specific function and predicate it was used with, where the following lemma suits any hard predicate. Finally, the original lemma is stated using an efficient family of pairwise-independent hash functions, where the following lemma is stated using explicit extractors. The lemma is proven using Holenstein's "uniform hardcore lemma" [Hol05].

Lemma A.6 *[A uniform extraction lemma] Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a polynomial-time computable function, let δ be some polynomial fraction and let $b : \{0,1\}^n \to \{0,1\}$ be a δ-hard predicate of f. Let $m, k, r \in \mathbb{N}$ and let ρ be the probability that when taking m independent samples in $\{0,1\}^n$ less than k samples are in some* fixed *set of density δ. Let $Ext : \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^r$ be a (k, ϵ)-strong-extractor.*[16] *Finally, let*

$$D = (f(x_1), \ldots, f(x_m), y, Ext(y, b(x_1), \ldots, b(x_m))),$$
$$\xi = (f(x_1), \ldots, f(x_m), y, z),$$

where $x_1, \ldots, x_m \in \{0,1\}^n$, $y \in \{0,1\}^d$ and $z \in \{0,1\}^r$. Then no PPT *can distinguish between D and ξ with probability (non-negligibly) better than $\rho + \epsilon$.*

Proof: Informally, [Hol05, Lemma 2.5] states that given a δ-hard predicate $b : \{0,1\}^n \to \{0,1\}$ and an inverting algorithm M, there exists some samplable subset $S \subseteq \{0,1\}^n$ of density δ such that the following holds: M cannot predict b over S (with success probability better than flipping a random coin) even when given access to χ^S, the characteristic function of S. More precisely, there is no PPT algorithm M that for any subset $S \subseteq \{0,1\}^n$ of density δ, the algorithm gets χ^S and outputs a circuit that given $f(x)$, predicts $b(x)$ with probability noticeably better than $\frac{1}{2}$. Where the probability is over a uniform choice of x in S and the random coins of M.

We assume towards a contradiction that there exists an efficient algorithm A that distinguishes between D and ξ with probability non-negligibly better than $\rho + \epsilon$ and use this algorithm to contradict [Hol05, Lemma 2.5]. W.l.o.g. let A be an efficient algorithm such that, $\Pr_{x \leftarrow D}[A(x)] - \Pr_{x \leftarrow \xi}[A(x)] > \rho + \epsilon + \nu$ for some polynomial fraction $\nu(n) = 1/poly(n)$ (for simplicity we simply

[16] Ext can be a strong-extractor, a Renyi entropy extractor or even a bit-fixing source extractor. For more details on different type of extractors c.f. [Sha02].

write ν). For a given set $S \subseteq \{0,1\}^n$ of density δ, we define the following, not necessarily efficiently computable, predicate $Q : \{0,1\}^n \to \{0,1\}$.

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

For any $i \in \{0, \dots, m\}$, the i^{th} hybrid of D is defined as:

$$D^i = (f(x_1), \dots, f(x_m), y, Ext(y, b(x_1), \dots, b(x_i), Q(x_{i+1}), \dots, Q(x_m))),$$

where $x_1, \dots, x_m \in \{0,1\}^n$ and $y \in \{0,1\}^d$. Note that D^m is simply D. On the other hand, D^0 is ϵ-close to ξ as long as at least k of the random inputs are in S. Thus altogether the statistical difference between D^0 and ξ is at most $\rho + \epsilon$. Hence, by a standard hybrid argument we deduce that there exists a $j \in \{0, \dots, m-1\}$ such that A distinguishes between D^j and D^{j+1} with success probability ν/m. Moreover, Since D^j and D^{j+1} are identical given that $x_{j+1} \notin S$, it follows that A achieves this distinguishing probability between D^j and D^{j+1} also when it is given that $x_{j+1} \in S$. Finally, note that the only difference between D^j and D^{j+1} given that $x_{j+1} \in S$ is whether the $(j+1)^{th}$ input to Ext is $b(x_j)$ or a random input.

Having the above we are ready to define an oracle aided algorithm that contradicts [Hol05, Lemma 2.5]. We will use the characteristic function χ^S in order to sample D^j, note that in the non-uniform model χ^S is not needed since we could simply hardwire into the algorithm a good sample of D^j. Also note that the circuit returned by the following algorithm does not predict $b(x)$ given $f(x)$, but rather distinguishes between $(f(x), b(x))$ and $(f(x), U_1)$. The two tasks, however, are equivalent due to standard reduction.

M^A with oracle χ^S:

1. Using χ^S, find a $j \in \{0, \dots, m-1\}$ such that, with probability at least $1 - 2^{-n}$, A distinguishes between D^j and D^{j+1} with success probability at least $\nu/2m$.
2. Sample B, an instance of D^j (again using χ^S).
3. Return the circuit that on input (y, b) replaces $f(x_{j+1})$ and $b(x_{j+1})$ in B by y and b respectively and outputs $A(B)$.

Hence, the output circuit of M^A distinguishes between $(f(x), b(x))$ and $(f(x), U)$ with probability at least $\nu/m - 2^{-n}$. ∎

A.4 The HILL Implementation of a Pseudo-Entropy Pair

Construction A.7 (The HILL PEP) *Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a one-way function and let b be a hardcore predicate of f (note that by Theorem 2.7 such a predicate is guaranteed to exist). Let $\mathcal{H}$ be an efficient family of pairwise-independent hash functions from $\{0,1\}^n$ to $\{0,1\}^n$. Let $f_H(x, h, i) = (f(x), h_{i+2\lceil log(n) \rceil}(x), h, i)$ and $b_H(x, h, i) = b(x)$, where $x \in \{0,1\}^n$, $h \in \mathcal{H}$, $i \in [n]$ and $h_k(x)$ stands for the first k bits of $h(x)$.*

[HILL99] have, implicitly, proved that (f_H, b_H) is a $(p + \frac{1}{2n}, \alpha)$-PEP, where $p \stackrel{\text{def}}{=} \Pr_{(x,i) \leftarrow (U_n,[n])}[D_f(f(x)) < i]$ and α is any fraction noticeably smaller than $\frac{1}{n}$. The proof goes by showing that it is hard to predict the hardcore-bit of an input element $(x, h, i) \in Im(f_H)$

whenever $i \leq D_f(f(x))$. Roughly speaking, the reason is that in such a case, $(h_{i+2\lceil log(n) \rceil}(x), h, i)$ does not contain any noticeable information about x. Thus, it is essentially as hard to predict $b_H(x, h, i) = b(x)$ given $f_H(x, h, i)$ as it is to predict $b(x)$ given $f(x)$. Since the probability that $i = D_f(x)$ is $\frac{1}{n}$, it follows that for any α noticeably smaller than $\frac{1}{n}$ it holds that b_H is a $(p+\alpha)$-hard predicate of f_H.

On the other hand, by the pairwise independence of $\mathcal{H}$, whenever $i \geq D_f(x)$ there is almost no entropy (i.e., less then $1/2n$) in $b_H(x, h, i)$ given $f_H(x, h, i)$. Thus, the entropy of $b_H(x, h, i)$ given $f_H(x, h, i)$ is not more than $p + \frac{1}{2n}$.

B Length Preserving One-Way Functions

We prove the folklore fact that when given a one-way function it can be assumed w.l.o.g. that it is a length-preserving one. More precisely, we prove that any one-way function can be used to construct a length-preserving one-way function with the same (up to linear factor) security-ratio.

Lemma B.1 *Let $f : \{0,1\}^n \rightarrow \{0,1\}^{\ell(n)}$ be a 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 $\{0,1\}^{2n}$. Define f' as follows:*

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

where $x_a, x_b \in \{0,1\}^n$ and $h \in \mathcal{H}$. Then f' is a length-preserving one-way function.

More precisely, if a PPT *A inverts f' with probability $\nu(n)$ over the choice of (x, h) and the randomness of A, then there exists a* PPT *that inverts f with probability at least $\nu(n) - 2^{-n+1}$.*

Proof: The function f' is length preserving as both input and output are of length $2n + \log |\mathcal{H}|$. Suppose A inverts f' with probability ν, 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 B^A as follows:

```
$B^A$ on input $y \in Im(f)$:
  1. Choose a random $h \in \mathcal{H}$.
  2. Apply $A(h(y), h)$ to get an output $(x_a, x_b, h)$.
  3. If $f(x_a) = y$ output $x_b$, otherwise abort.
```

The PPT B^A is sure to succeed on any choice of (y, h) so that A succeeds on $(h(y), h)$ and in addition there exists no $z \in Im(f)$ such that $h(z) = h(y)$ (h does not introduce any collision to y). Denote by G_A the set that A succeeds on and by Col the set of (y, h) such that there exists $z \in Im(f)$ such that $h(z) = h(y)$. Thus:

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[B^A \text{ inverts } f(x)] &\geq \Pr_{(x,h) \leftarrow (U_n, \mathcal{H})}[(h(f(x)), h) \in G_A \bigwedge (f(x), h) \notin Col] \\
&= \Pr_{(x,h) \leftarrow (U_n, \mathcal{H})}[(h(f(x)), h) \in G_A] - \\
&\quad \Pr_{(x,h) \leftarrow (U_n, \mathcal{H})}[(h(f(x)), h) \in G_A \bigwedge (f(x), h) \in Col] \\
&\geq \Pr_{(x,h) \leftarrow (U_n, \mathcal{H})}[(h(f(x)), h) \in G_A] - \Pr_{(x,h) \leftarrow (U_n, \mathcal{H})}[(f(x), h) \in Col]
\end{aligned}$$

▮

For all $y, z \in Im(f)$ the almost pairwise-independence of h yields that the probability that $h(z) = h(y)$ is at most 2^{-2n+1}. Since the size of $Im(f)$ in $\{0,1\}^{\ell(n)}$ is at most 2^{n+1} then a union bound over all possible $z \in Im(f)$ gives that for any y:

$$\Pr_{h \leftarrow \mathcal{H}}[\exists z \in Im(f) \text{ such that } h(z) = h(y)] \leq 2^{-n+1}$$

Therefore, by an averaging argument:

$$\Pr_{(x,h) \leftarrow (U_n,\mathcal{H})}[(f(x), h) \in Col] \leq 2^{-n+1}$$

Putting it together we get that:

$$\Pr_{x \leftarrow U_n}[B^A \text{ inverts } f(x)] \geq \nu(n) - 2^{-n+1}$$

▮

Since the length of the representation of the hash functions in the 2^{-2n}-almost pairwise-independent family is $\mathcal{O}(n)$ bits then we get the following useful corollary:

Corollary B.2 *Let $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a one-way function, then there exists a length-preserving one-way function with the same (up to constant factor) security-ratio as f.*