On the Power of the Randomized Iterate

Iftach Haitner[1], Danny Harnik[2], Omer Reingold[3]

[1] Dept. of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot, Israel. iftach.haitner@weizmann.ac.il.⋆⋆⋆

[2] Dept. of Computer Science, Technion, Haifa, Israel. harnik@cs.technion.ac.il.†

[3] Incumbent of the Walter and Elise Haas Career Development Chair, Dept. of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot, Israel. omer.reingold@weizmann.ac.il.†

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 to $\mathcal{O}(n^7)$ (which also implies improvement in the security of the construction).

⋆⋆⋆ Research supported in part by a grant of the Israeli Science Foundation (ISF).

† This research was conducted at the Weizmann Institute.

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 $b(x^0), \dots, b(x^\ell)$.

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], 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.

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. 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 $m = \Omega(n^8)$.[4]

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.

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 $m = \mathcal{O}(n^2)$ (see Theorem 5).
- 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 $m = \mathcal{O}(n \log n)$ (see Theorem 6).

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

[4] The seed length actually proved in [HILL99] is $\mathcal{O}(n^{10})$, however it is mentioned that a more careful analysis can get to $\mathcal{O}(n^8)$. A formal proof for the $\mathcal{O}(n^8)$ seed length construction is given by Holenstein [Hol06].

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*[5] 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 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. All in all, we present a pseudorandom generator from *any* one-way function with seed length $\mathcal{O}(n^7)$ (Corollary 10) which is the best known to date.

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 $D_f(y) = \lceil \log \left| f^{-1}(y) \right| \rceil$ (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.

---

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

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).

Related Work: Recently, Holenstein [Hol06] gave a generalized proof to the HILL construction. His proof formally proves the best known seed length for the HILL construction $\mathcal{O}(n^8)$, and further shows that if the underlying one-way function has exponential security (e.g. 2^{-Cn} for a constant C) then the seed length can be as low as $\mathcal{O}(n^5)$, or even $\mathcal{O}(n^4 \log^2 n)$ if the security of the PRG is not required to be exponential (but rather superpolynomial). In subsequent work [HHR06], we show a construction of a PRG based on exponentially strong one-way functions with seed length of only $\mathcal{O}(n^2)$ or respectively $\mathcal{O}(n \log^2 n)$ for a PRG with just superpolynomial security. The new construction follows by further developing the techniques introduced in this paper.

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[6], 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 $2^{\mathcal{O}(n)}$).[7] Their construction uses a variant of randomized iterates where the randomization is via one random step on an expander graph.

Our Contribution to Hardness Amplification: We present an alternative efficient hardness amplification for regular one-way functions. Specifically, we show that the m^{th} randomized iterate of a weak one-way function along with the randomizing hash functions form a strong one-way function (for the right parameter m). Moreover, in Theorem 13 we show that the latter holds also for the derandomized version of the randomized iterate, giving an almost linear construction. 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,

[6] 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.

[7] 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.

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 m^{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.

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:** Throughout the paper we focus on length preserving one-way functions. In the full version we 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: In Section 2, we present our construction of pseudorandom generators from regular one-way functions. In Section 3, we present our improvement to the HILL construction of pseudorandom generators from any one-way function. Finally, in Section 4, we present our hardness amplification of regular one-way functions. Due to space limitations, we only give the outlines of some of the proofs. For the same reasons, we omit the standard definitions and notations. Both the full proofs and the definitions can be found in the paper's full version [HHR05].

2 Pseudorandom Generators from Regular One-Way Functions

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 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, 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 1 (The k^{th} Randomized Iterate of f) *Let $f : \{0,1\}^n \to \{0,1\}^n$ 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 $h_1, \ldots, h_{k-1} \in \mathcal{H}$ define the k^{th}* Randomized Iterate *$f^k : \{0,1\}^n \times \mathcal{H}^k \to Im(f)$ recursively as:*

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

where $f^0(x) = f(x)$. For convenience we denote by $x^k \stackrel{def}{=} f^k(x, h_1, \ldots, h_k)$.[8]

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

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

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

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 $h_i(x^i)$. Hence, 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 of 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 3). For the regular OWF construction it suffices to define $f^0(x) = x$, thus saving a single application of the function f.

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.

Lemma 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 1). Then for any* PPT *A and every $k \in poly(n)$, we have:*

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

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, 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 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 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 (proof omitted) implies that the probability of getting an element in the set that A succeeds on is not very small:

Claim 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

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

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 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 4, we may conclude the proof of Lemma 2. By Claim 3 we have that $\Pr[(x^k, h_1, \ldots, h_k) \in S_A] \geq \frac{\varepsilon}{2}$. By Lemma 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 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:

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

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 $(x_0, {h_1}^0, \ldots, {h_k}^0) \in T$ and $(x_1, {h_1}^1, \ldots, {h_k}^1) \in T$. 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, $\Pr[(z, h_1, \ldots, h_k) \in T] \geq \delta^2/(k+1)$ as requested. ▮

Remark: The proof of Lemma 4 is where the regularity of the one-way function is required. In the case of general one-way functions, we cannot apply the above proof, since the collision-probability at the heart of the proof (i.e.,

$CP(\widehat{f^k}(U_n, \mathcal{H}^k)))$ might be much larger than the collision probability of the uniform distribution over $Im(f)$. Alternatively, we could prove the lemma in the case that the last element in a sequence of applications is at least as heavy as all the elements along the sequence. Unfortunately, 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. [9]

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 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. The proof of security follows by a standard hybrid argument and is given in the full version of the paper [HHR05].

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

The pseudorandom generator presented in the previous section (Theorem 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. That is, 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 $h_1, \dots, h_n$. The idea of the new construction is to give a derandomization of the choice of the n hash functions. Thus, $h_1, \dots, h_n$ are no longer chosen independently, but are chosen in a way that

[9] In [HHR06] we use a variant of the latter idea to get an efficient pseudorandom from exponentially hard one-way functions.

is sufficient for the proof to go through. The derandomization uses generators against bounded-space distinguishers. 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 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 $\tilde{n} = cn \log n$.

The New Pseudorandom Generator

Theorem 6 *For any regular length-preserving one-way function f, let G' be:*

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

where:

- $x \in \{0,1\}^n$ *and* $\tilde{h} \in \{0,1\}^{\tilde{n}}$.
- $(h_1, \ldots, h_n) = BSG(\tilde{h})$.
- *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 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 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 2, only replacing appearances of $(h_1, \ldots, h_k)$ with the seed $\tilde{h}$. Again, the key to the proof is the following technical lemma (slightly modified from Lemma 4, the proof of the lemma is given in the full version [HHR05]).

Lemma 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 5 in showing that the output of G' is an unpredictable sequence and therefore a pseudorandom sequence. ∎

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).

3 Pseudorandom Generator from Any One-Way Function

Our implementation of a pseudorandom generator from any one-way function follows the route of [HILL99] (we follow the presentation and proof of the HILL generator given in [Hol06]), but takes a totally different approach in the implementation of its initial step.

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".[10] Informally, the latter is a pair of a function and predicate on the same input with the following property: When given the output of the function, the pseudo-entropy of the predicate's output 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 hardcore predicate forms a pseudo-entropy pair. Moreover, this pair has better properties than the original one 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 8 *[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. We say that (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.*

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

[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.[11] Then they 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.

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 $\frac{1}{2n}$. By combining our PEP with the second part of the [Hol06] construction, we get a pseudorandom generator that is more efficient and has better security than the original construction due to [HILL99]/[Hol06] (the efficiency improves by a factor of n^3 and the security by a factor of n). Formally, we get the following theorem.

Theorem 9 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable function and let $\varepsilon, \gamma : \mathbb{N} \to [0,1]$. Then there is an efficiently computable function G with the following properties:*

- *G is length expanding.*
- *G has input length $\mathcal{O}(n^6 \log(\frac{1}{\varepsilon(n)}))$.*
- *Any algorithm A that distinguishes the output of G from the uniform distribution with advantage γ, can be used to construct an algorithm that inverts f with probability $\Omega(\frac{1}{n^{13}})$ and runs in time poly$(\frac{1}{\Omega(\frac{\gamma}{n^7 \log(\frac{1}{\varepsilon})}) - \varepsilon}, n, T_A(n))$, where T_A is the running time of A.*

As a corollary we deduce the main statement of this section:

Corollary 10 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function, then there exists a pseudorandom generator with seed length $\mathcal{O}(n^7)$.*

A Pseudo-Entropy Pair Based on the Randomized Iterate

For a given one-way function f, we have defined (Definition 1) its first explicit randomized iterate as $\widehat{f^1}(x,h) = (f(h(f(x))), h)$. 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. Formally,

[11] [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 proof, but it is not required by the new proof due to [Hol06]. We note that our PEP, presented next, also has such a hardcore-set

Definition 11 (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 $h_E \in \mathcal{H}_E$.

Lemma 12 *Let $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 11. 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. Let W be a random variable uniformly distributed over $Dom(g)$ and let α be noticeably smaller than $\frac{1}{2n}$, then the following hold:*

1. *$H(b(W) \mid g'(W)) \leq \frac{1}{2}$.*
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.

4 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 $\mathcal{O}(n \log n)$.

Theorem 13 *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 1. Let BSG be a bounded-space generator against $(2n, n+1, 2n)$-layered-branching-programs 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 (strong) a one-way function.

Acknowledgments

We are grateful to Oded Goldreich, Moni Naor, Asaf Nussbaum, Eran Ofek and Ronen Shaltiel for helpful conversations. We also thank Tal Moran and Ariel Gabizon for reading a preliminary version of this manuscript.

References

[AIK04] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175, 2004.

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

[DI99] G. Di Crescenzo and R. Impagliazzo. Security-preserving hardness-amplification for any regular one-way function. In *31st STOC*, 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 FOCS*, 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 STOC*, pages 25–32, 1989.

[HHR05] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. ECCC, TR05-135, 2005.

[HHR06] I. Haitner, D. Harnik, and O. Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In ICALP 2006, 2006.

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

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

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

[IL89] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *30th FOCS*, pages 230–235, 1989.

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

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th FOCS*, 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.

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

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

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *23rd FOCS*, pages 80–91, 1982.