Efficient Pseudorandom Generators from Exponentially Hard One-Way Functions

Iftach Haitner[1,⋆], Danny Harnik[2,⋆⋆], and 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] Dept. of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot, Israel
omer.reingold@weizmann.ac.il

Abstract. In their seminal paper [HILL99], Håstad, Impagliazzo, Levin and Luby show that a pseudorandom generator can be constructed from any one-way function. This plausibility result is one of the most fundamental theorems in cryptography and helps shape our understanding of hardness and randomness in the field. Unfortunately, the reduction of [HILL99] is not nearly as efficient nor as security preserving as one may desire. The main reason for the security deterioration is the blowup to the size of the input. In particular, given one-way functions on n bits one obtains by [HILL99] pseudorandom generators with seed length $\mathcal{O}(n^8)$. Alternative constructions that are far more efficient exist when assuming the one-way function is of a certain restricted structure (e.g. a permutations or a regular function). Recently, Holenstein [Hol06] addressed a different type of restriction. It is demonstrated in [Hol06] that the blowup in the construction may be reduced when considering one-way functions that have *exponential* hardness. This result generalizes the original construction of [HILL99] and obtains a generator from any exponentially hard one-way function with a blowup of $\mathcal{O}(n^5)$, and even $\mathcal{O}(n^4 \log^2 n)$ if the security of the resulting pseudorandom generator is allowed to have weaker (yet super-polynomial) security.

In this work we show a construction of a pseudorandom generator from any exponentially hard one-way function with a blowup of only $\mathcal{O}(n^2)$ and respectively, only $\mathcal{O}(n \log^2 n)$ if the security of the resulting pseudorandom generator is allowed to have only super-polynomial security. Our technique does not take the path of the original [HILL99] methodology, but rather follows by using the tools recently presented in [HHR05] (for the setting of regular one-way functions) and further developing them.

⋆ Research supported in part by grant no. 1300/05 from the Israel Science Foundation.

⋆⋆ Part of this research was conducted at the Weizmann Institute. Research was supported in part at the Technion by a fellowship from the Lady Davis Foundation.

⋆⋆⋆ Incumbent of the Walter and Elise Haas Career Development Chair.

M. Bugliesi et al. (Eds.): ICALP 2006, Part II, LNCS 4052, pp. 228–239, 2006.

1 Introduction

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 seminal paper of Håstad et al. [HILL99] introduced 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. While the HILL generator fully answers the question of the plausibility of a generator based on any one-way function, the construction is quite involved and very inefficient. This inefficiency also plays a crucial role in the deterioration of the security within the construction.

The seed length and security of the construction: There are various factors involved in determining the security and efficiency 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 HILL construction produces a generator with seed length on the order of $m = \mathcal{O}(n^8)$ (a formal proof of this seed length does not actually appear in [HILL99] and was given in [Hol06]). An alternative construction was recently suggested in [HHR05] which improves the seed length to $\mathcal{O}(n^7)$.

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 HILL generator can only be trusted on seed lengths of 10^{16} (ignoring constants) and up (or 10^{14} using the construction of [HHR05]). Thus, trying to improve the seed length towards a linear one is of great importance in making these constructions practical.

Pseudorandom generators from restricted one-way functions: On the other hand, there are known constructions of pseudorandom generators from one-way functions that are by far more efficient when restrictions are made on the type of one-way functions at hand. Most notable is the so called BMY generator (of [BM82, Yao82]) based on any one-way *permutation*. This construction gives a generator with seed length of $\mathcal{O}(n)$ bits. A generator based on any *regular*

one-way function of seed length $\mathcal{O}(n \log n)$ was presented in [HHR05] (improving the original such construction of seed length $\mathcal{O}(n^3)$ from [GKL93]). Basing generators on one-way functions with *known preimage-size* [ILL89] also yield constructions that are significantly more efficient than the general case.

The common theme in all of the above mentioned restrictions is that they deal with the *structure* of the one-way function. A different approach was taken by Holenstein [Hol06], that builds a pseudorandom generator from any one-way function with *exponential hardness*. This approach is different as it discusses *raw hardness* as opposed to structure. The result in [Hol06] is essentially a generalization of the HILL generator that also takes into account the parameter stating the hardness of the one-way function. In its extreme case where the hardness is exponential (i.e. 2^{-Cn} for some constant C), then the pseudorandom generator takes a seed length of $\mathcal{O}(n^5)$. Alternatively, the seed length can be reduced to as low as $\mathcal{O}(n^4 \log^2 n)$ when the resulting generator is only required to have super-polynomial security (i.e. security of $n^{\log n}$).

This Work: We give a construction of a pseudorandom generator from any exponentially hard one-way function with seed length $\mathcal{O}(n^2)$. If the resulting generator is allowed to have only super-polynomial security then the construction gives seed length of only $\mathcal{O}(n \log^2 n)$.

Unlike Holenstein's result, our constructions is specialized for one-way functions with exponential hardness. If the security parameter is $2^{-\phi n}$ then the result holds only when $\phi > \Omega(\frac{1}{\log n})$, and does not generalize for use of weaker one-way functions. The core technique of our construction is the *randomized iterate* that was introduced by Goldreich, Krawczyk and Luby [GKL93], and is the focal point in [HHR05].

Paper Organization: Due to space limitations, we provide formal proofs only for the core technique, namely the randomized iterate (in Section 3). In Section 2 we provide an overview of the construction and techniques. Section 4 presents the multiple randomized iterate and its properties, while Section 5 presents the actual construction of the generator. The proofs of the theorems in these sections appear in the full version of the paper.

2 Overview of the Construction

As a motivating example we start by briefly describing the BMY generator. This generator works by iteratively applying the one-way permutation on its own output. More precisely, for a given function f and input x define the k^{th} iterate recursively as $f^k(x) = f(f^{k-1}(x))$ where $f^0(x) = f(x)$.[1] To complete the construction, one needs to take a hardcore-bit at each iteration. If we denote by $b(z)$ the hardcore-bit of z (take for instance the Goldreich-Levin [GL89] predicate), then the BMY generator on seed x outputs the hardcore-bits $b(f^0(x)), \ldots, b(f^\ell(x))$. The rationale behind this technique is that for all k, the k^{th} iteration

[1] We take $f^0(x) = f(x)$ rather than $f^0(x) = x$ for consistency with [HHR05] (see also remark in Section 3).

of f is hard to invert (it is hard to compute $f^{k-1}(x)$ given $f^k(x)$). Indeed, Levin [Lev87] showed that the same generator works with any function that is "one-way on its iterates". However, a general one-way function does not have this guarantee, and in fact, may lose all of its hardness after just one iteration (since there may be too little randomness in the output of f).

The randomized Iterate and regular one-way functions: With the above problem in mind, Goldreich et al. [GKL93] suggested to add a randomizing step between every two iterations. This idea is central in our work and we define it next (following [HHR05]):

Definition (Informal): (The Randomized Iterate). *For function f, input x and random pairwise-independent hash functions $\overline{h} = (h_1, \ldots, h_\ell)$, recursively define the i^{th} randomized iterate (for $i \leq \ell$) by:*

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

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

The rational is that $h_i(f^i(x, \overline{h}))$ is now uniformly distributed, and the challenge is to show that f, when applied to $h_i(f^i(x, \overline{h})$, is hard to invert even when the randomizing hash functions $\overline{h}$ are made public. Indeed, in [HHR05] it was shown that the last randomized iteration is hard to invert even when $\overline{h}$ is known, when the underlying one-way function is *regular*[2] (a regular function is a function such that every element in its image has the same preimage size). Once this is shown, a generator from regular one-way function is similar in nature to the BMY generator, replacing iterations with randomized iterations (the generator outputs $b(f^0(x, \overline{h})), \ldots, b(f^\ell(x, \overline{h})), \overline{h})$.

The randomized iterate and general one-way functions: Unfortunately, the last randomized iteration of a general one-way function is not necessarily hard to invert. It may in fact be easy on a large fraction of the inputs. However, following the proof method presented in [HHR05], we manage to prove the following statement regarding the k^{th} randomized iteration (Lemma 1): There exists a set S^k of inputs to f^k such that the k^{th} randomized iteration is hard to invert over inputs taken from this set. Moreover, the density of S^k is at least $\frac{1}{k}$ of the inputs.

So taking a hard core bit of the k^{th} randomized iteration is beneficial, in the sense that this bit will look random (to a computationally bounded observer) just that this will happen only $\frac{1}{k}$ of the time.

The multiple randomized iterate: Our goal is to get a string of pseudorandom bits, and the idea is to run m independent copies of the randomized iterate (on m independent inputs). We call this the *multiple randomized iterate*. From each of the m copies we output a hardcore bit of the k^{th} iteration. This forms a string of m bits, of which $\frac{m}{k}$ are expected to be random looking. The next

[2] Such a statement was originally proved in [GKL93] for n-wise independent hash functions rather than pairwise independent hash.

step is to run a *randomness extractor* on such a string (where the output of the extractor is of length, say, $\frac{m}{2k}$). This ensures that with very high probability, the output of the extractor is a pseudorandom string of bits.

The use of randomness extractors in a computational setting, was initiated in [HILL99]. We give a general "uniform extraction lemma" for this purpose that is proved using a uniform hardcore Lemma of Holenstein from [Hol05]. Note that similar proofs were given previously [Hol06, HHR05]. In the full paper we give a new version since we require a more careful analysis of the security parameters.

The pseudorandom generator – a first attempt: A first attempt for the pseudorandom generator runs the multiple randomized iterate (on m independent inputs) for ℓ iterations. For each $k \in [\ell]$ we extract $\frac{m}{2k}$ bits at the k^{th} iteration. These bits are guaranteed to be pseudorandom (even when given all of the values at the $(k+1)^{st}$ iterate and all of the randomizing hash functions). Thus outputting the concatenation of the pseudorandom strings for the different values of k forms a long pseudorandom output (by a standard hybrid argument).

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

The pseudorandom generator and exponential hardness: The failed generator from above can be remedied when the exponential hardness comes into play. It is known that if a function has hardness 2^{-Cn} (for some constant C), then it has a hardcore function of $C'n$ bits (for another constant C'). Such a general hardcore function appears in the original Goldreich-Levin paper [GL89]. Thus, if the original hardness was exponential, then in the k^{th} iteration we can actually extract $C'n$ random looking strings, each of length $\frac{m}{2k}$. Altogether we get that the output length is $C'n\sum_{k=1}^{\ell} \frac{m}{2k} \geq C'mn \log \ell$. Thus for a choice of ℓ such that $\log \ell > C'$ we get that the overall output is a pseudorandom string of length greater than the input.

The input length of the construction is $\mathcal{O}(nm)$, where m can be taken to be approximately $\mathcal{O}(\log \frac{\ell}{\varepsilon(n)})$ where $\varepsilon(n)$ is the security of the resulting generator. In particular, in order to get an exponentially strong generator, one needs to take a seed of length $\mathcal{O}(n^2)$.

To sum up, we describe the full construction in a slightly different manner: One first creates a matrix of size $m \times \ell$, where each *row* in the matrix is generated by computing the first ℓ randomized iterates of f (each row takes independent inputs). Now from each entry in the matrix $\mathcal{O}(\phi n)$ hardcore bits are computed (thus generating a matrix of hardcore bits). The final stage runs a randomness

extractor on each of the *columns* of the hardcore bits matrix.[3] Moreover, the number of pseudorandom bits extracted from a column deteriorates from one iteration to another ($\frac{m}{k}$ pseudorandom bits are taken at the columns associated with the k^{th} randomized iterate).

Some Notes

- Our method works for one-way functions with hardness $2^{-\phi n}$ as long as $\phi > \Omega(\frac{1}{\log n})$. Loosely speaking, this is because for large values of ℓ, the value $\frac{1}{\ell}$ becomes too small to overcome with limited repetition (and thus requires m to grow substantially).
- The paper focuses on length-preserving one-way functions, however, the results may be generalized to use non-length preserving functions (see [HHR05]).

Notations: We denote by $Im(f)$ the image of a function f. 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 $D_f(y) \stackrel{\text{def}}{=} \lceil \log \left| f^{-1}(y) \right| \rceil$. Due to space limitations we omit standard definitions (provided in the full version).

3 The Randomized Iterate of a One-Way Function

As mentioned in Section 2, the use of randomized iterations lies at the core of our generator. We formally define this notion:

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*[4] *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 $\overline{h} = (h^1, \ldots, h^k)$.

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:

$$\widehat{f^k}(x, \overline{h}) = (f^k(x, \overline{h}), \overline{h})$$

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 in [HHR05]. For the construction presented in this paper one can also define $f^0(x) = x$, thus saving a single application of the function f.

[3] Note that each execution of the extractor runs on a column in which each entry consists of a single bit (rather than $\mathcal{O}(\phi n)$ bits). This is a requirement of the proof technique.

[4] Pairwise independent hash functions where defined in, e.g. [CW77].

3.1 The Last Randomized Iterate Is (sometimes) Hard to Invert

We now formally state and prove the key observation, that there exists a set of inputs of significant weight for which it is hard to invert the k^{th} randomized iteration even if given access to all of the hash functions leading up to this point.

Lemma 1. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function with security $2^{-\phi n}$, and let f^k and $\mathcal{H}$ be as defined in Definition 1.*

Let

$$S^k \stackrel{def}{=} \left\{ (x, \overline{h}) \in (\{0,1\}^n \times \mathcal{H}^k) \mid D_f(f^k(x,\overline{h}) = \max_{j \in [k]} D_f(f^j(x,\overline{h})) \right\}$$

Then,

1. *The set S^k has density at least $\frac{1}{k}$.*
2. *For every* PPT *A,*

$$\Pr_{(x,\overline{h}) \leftarrow S^k}[A(f^k(x,\overline{h}),\overline{h}) = f^{k-1}(x,\overline{h})] \leq 2^{-\mathcal{O}(\phi n)}$$

where the probability is also taken over the random coins of A.
More precisely, given a PPT *A that runs in time T_A and inverts the last iteration over S^k with probability $\varepsilon(n)$ one can construct an algorithm that runs in time $T_A + poly(n)$ and inverts the OWF f with probability $\frac{\varepsilon(n)^3}{32k(k+1)n}$.*

Proof

Proving (1). By the pairwise independence of the randomizing hash functions $\overline{h} = (h^1, \ldots, h^k)$ we have that for each $0 \leq i \leq k$, the value $f^i(x,\overline{h})$ is independently and randomly chosen from the distribution $f(U_n)$. Thus, simply by a symmetry argument, the k^{th} (last) iteration is has the heaviest preimage size with probability at least $\frac{1}{k}$. Thus $\Pr_{(x,\overline{h}) \leftarrow (U_n,\mathcal{H}^k)}[(x,\overline{h}) \in S^k] \geq \frac{1}{k}$.

Proving (2). Suppose for sake of contradiction that there exists an efficient algorithm A that given $(f^k(x,\overline{h}),\overline{h})$ computes $f^{k-1}(x,\overline{h})$ with probability $\varepsilon(n)$ over S^k (for simplicity we simply write ε). In particular A inverts the last-iteration of $\widehat{f^k}$ with probability at least ε, that is

$$\Pr_{(x,\overline{h}) \leftarrow S^k}[f(h(A(\widehat{f^k}(x,\overline{h})))) = f^k(x,\overline{h})] \geq \varepsilon$$

Our goal is to use this procedure A in order to break the one-way function f. This goal s achieved by the following procedure:

M^A **on input** $z \in Im(f)$**:**

1. Choose a random $\overline{h} = (h^1, \ldots, h^k) \in \mathcal{H}^k$.
2. Apply $A(z,\overline{h})$ to get an output y.
3. If $f(h^k(y)) = z$ output $h^k(y)$, otherwise abort.

We prove that M^A succeeds in inverting f with sufficiently high probability. We focus on the following set of outputs, on which A manages to invert their last-iteration with reasonably high probability.

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

A simple Markov argument shows that the set T_A has reasonably large density (the proof is omitted).

Claim 1

$$\Pr_{(x,\overline{h}) \leftarrow (U_n,\mathcal{H}^k)}[\widehat{f^k}(x,\overline{h}) \in T_A] \geq \frac{\varepsilon}{2}.$$

Moreover, since the density of S^k is at least $\frac{1}{k}$, it follows that $\Pr_{(x,\overline{h}) \leftarrow (U_n,\mathcal{H}^k)}$ $[\widehat{f^1}(x,\overline{h}) \in T_A \bigwedge (x,\overline{h}) \in S^k] \geq \varepsilon/2k$. We now make use of the following Lemma, that relates the density of a set with respect to pairs $(f^k(x,\overline{h}),\overline{h})$ where the value of $f^k(x,\overline{h})$ is actually generated using the given randomizing hash functions $\overline{h}$ (i.e. the pair is an output of $\widehat{f^k}$) as opposed to the density of the same set with respect to pairs consisting of a random output of f concatenated with an *independently* chosen hash functions.

Lemma 2. *For every set* $T \subseteq Im(\widehat{f^k})$*, if*

$$\Pr_{(x,\overline{h}) \leftarrow (U_n,\mathcal{H}^k)}[\widehat{f^k}(x,\overline{h}) \in T \bigwedge (x,\overline{h}) \in S^k] \geq \delta$$

then

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

To conclude the proof of Lemma 1, take $T = T_A$ and $\delta = \varepsilon/2k$, and Lemma 2 yields that $\Pr_{(z,\overline{h}) \leftarrow (f(U_n),\mathcal{H}^k)}[(z,\overline{h}) \in T_A] \geq \frac{\varepsilon^2}{16k(k+1)n}$. On each of these inputs A succeeds with probability $\varepsilon/2$, thus altogether M^A manages to invert f with probability $\frac{\varepsilon^3}{32k(k+1)n}$. ∎

Proof. (of Lemma 2) 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 $T_i = \left\{(z,\overline{h}) \in T \mid D_f(z) = i\right\}$. We divide S^k into corresponding slices as well, define the i^{th} slice as $S_i^k = \left\{(x,\overline{h}) \in S^k \mid D_f(f^k(x,\overline{h})) = i\right\}$ (note that since $S_i^k \subseteq S^k$, for each $(x,\overline{h}) \in S_i^k$ and thus for each $0 \leq j < k$ it holds that $D_f(f^j(x,\overline{h})) \leq D_f(f^k(x,\overline{h})) = i$). The proof of Lemma 2 follows the methods from [HHR05], used to obtain a similar argument in the case of regular functions. The method follows by studying the collision-probability of $\widehat{f^k}$ when restricted to S_i^k (we work separately on each slice). Denote this as:

$$CP(\widehat{f^k}(U_n,\mathcal{H}^k) \bigwedge S_i^k) = \Pr_{(x_0,\overline{h}_0),(x_1,\overline{h}_1)}[\widehat{f^k}(x_0,\overline{h}_0) = \widehat{f^k}(x_1,\overline{h}_1) \bigwedge (x_0,\overline{h}_0),$$

$$(x_1,\overline{h}_1) \in S_i^k]$$

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. $(x_0, \overline{h}_0)$, is required to be in S_i^k). Recall that $\widehat{f^k}(x, \overline{h})$ includes the hash functions $\overline{h}$ in its output, thus, for every two inputs $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$, in order to have a collision we must first have that $\overline{h}_0 = \overline{h}_1$ which happens with probability $(1/|\mathcal{H}|)^k$. Now, given that $\overline{h}_0 = \overline{h}_1 = \overline{h}$ (with $\overline{h} \in \mathcal{H}^k$ being uniform), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$.

If $f(x_0) = f(x_1)$ then a collision is assured. Since it is required that $(x_0, \overline{h}_0) \in S_i^k$ it holds that $D_f(f(x_0)) \leq D_f(f^k(x_1, \overline{h})) = i$ and therefore $\left|f^{-1}(f(x_0))\right| \leq 2^i$. Thus, the probability for that $x_1 \in f^{-1}(f(x_0))$ (and thus of $f(x_0) = f(x_1)$) is at most 2^{i-n}. Otherwise, there must be an $i \in [k]$ for which $f^{i-1}(x_0, \overline{h}) \neq f^{i-1}(x_1, \overline{h})$ but $f^i(x_0, \overline{h}) = f^i(x_1, \overline{h})$. Since $f^{i-1}(x_0, \overline{h}) \neq f^{i-1}(x_1, \overline{h})$, then due to the pairwise-independence of h_i, the values $h_i(f^{i-1}(x_0, \overline{h}))$ and $h_i(f^{i-1}(x_1, \overline{h}))$ are uniformly random values in $\{0,1\}^n$, and thus $f(h_i(f^{i-1}(x_0, \overline{h}))) = f(h_i(f^{i-1}(x_1, \overline{h})))$ also happens with probability at most 2^{i-n}. Altogether:

$$CP(\widehat{f^k}(U_n, \mathcal{H}^k) \bigwedge S_i^k) \leq \frac{1}{|\mathcal{H}|^k} \sum_{i=0}^{k} 2^{i-n} \leq \frac{k+1}{|\mathcal{H}|^k 2^{n-i}} \tag{1}$$

On the other hand, we give a lower-bound for the above collision-probability. We seek the probability of getting a collision inside S_i^k and further restrict our calculation to collisions whose output lies in the set T_i (this further restriction may only reduce the collision probability and thus the lower bound holds also without the restriction). For each slice, denote $\delta_i = \Pr[\widehat{f^k}(x, \overline{h}) \in T_i \bigwedge (x, \overline{h}) \in S_i^k]$. In order to have this kind of collision, we first request that both inputs are in S_i^k and generate outputs in T_i, which happens with probability δ_i^2. Then once inside T_i we require that both outputs collide, which happens with probability at least $\frac{1}{|T_i|}$. Altogether:

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

Combining Equations (1) and (2) we get:

$$\frac{|T_i|\, 2^{i-n-1}}{|\mathcal{H}|^k} \geq \frac{\delta_i^2}{2(k+1)} \tag{3}$$

However, note that when taking a random output z and independent hash functions $\overline{h}$, the probability of hitting an element in T_i is at least $2^{i-n-1}/|\mathcal{H}|^k$ (since each output in T_i has preimage at least 2^{i-1}). But this means that $\Pr[(z,h) \in T_i] \geq |T_i|\, 2^{i-n-1}/|\mathcal{H}|$ and by Equation (3) we deduce that $\Pr[(z,h) \in T_i] \geq \delta_i^2/2(k+1)$. Finally, the probability of hitting T is $\Pr[(z,h) \in T] = \sum_i \Pr[(z,h) \in T_i] \geq \sum_i \delta_i^2/2(k+1)$. Since $\sum_i \delta_i^2 \geq (\sum_i \delta_i)^2/n$ and (by definition) $\sum_i \delta_i = \delta$, it holds that $\Pr[(z,h) \in T] \geq \delta^2/2(k+1)n$ as claimed. ∎

A Hardcore Function for the Randomized Iterate. A hardcore function of the k^{th} randomized iteration is simply taken as the GL hardcore function

([GL89]). The number of bits taken in this construction depends on the hardness of the function at hand (that is the last iteration of the randomized iterate). Thus combining Lemma 1 regarding the hardness of inverting the last iteration, and the Goldreich-Levin Theorem on hardcore functions we get the following lemma:

Lemma 3. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function with security $2^{-\phi n}$, and let f^k and $\mathcal{H}$ be as defined in Definition 1. Let $s = \lceil \frac{\phi}{20} n \rceil$ and take $hc = gl_s$ to be the Goldreich-Levin hardcore function which outputs s hardcore bits.*

Then, for every polynomial k, there exist a set $S_k \subseteq \{0,1\}^n \times \mathcal{H}^k$, of density at least $\frac{1}{k}$ such that for any PPT *A,*

$$\Pr[A(\widehat{f^k}(x,\overline{h}),r) = hc(f^{k-1}(x,\overline{h},r) \mid (x,\overline{h}) \in S_k] < 2^{-\mathcal{O}(\phi n)}.$$

In other words, hc is a hardcore function for the k^{th} randomized iterate over the set S^k with $\mu_{hc} \leq 2^{-\phi n/20}$ security.[5]

4 The Multiple Randomized Iterate

In this section we consider the function $\overline{f^k}$ which consists of m independent copies of the randomized iterate f^k.

Construction 1 (The k^{th} Multiple Randomized Iterate of f). *Let $m, k \in \mathbb{N}$, and let f^k and $\mathcal{H}$ be as in Construction 1. We define the k^{th}* Multiple Randomized Iterate *$\overline{f^k} : \{0,1\}^{mn} \times \mathcal{H}^{mk} \to Im(f)^m$ as:*

$$\overline{f^k}(\overline{x}, H) = f^k(\overline{x}_1, H_1), \ldots, f^k(\overline{x}_m, H_m),$$

where $\overline{x} \in \{0,1\}^{mn}$ and $H \in \mathcal{H}^{m \times k}$. We define the k^{th} explicit multi randomized iterate $\widehat{\overline{f^k}}$ as:

$$\widehat{\overline{f^k}}(\overline{x}, H) = \overline{f^k}(\overline{x}, H), H$$

For each of the m outputs of $\overline{f^k}$ we look at its hardcore function hc. By Lemma 3 it holds that m/k of these m hardcore strings are expected to fall inside the "hard-set" of $\widehat{f^k}$ (and thus are indeed pseudorandom given $\widehat{\overline{f^k}}(\overline{x}, H)$). The next step is to invoke a randomness extractor on a concatenation of one bit from each of the different independent hardcore strings. The output of the extractor is taken to be of length $\frac{m}{4k}$. The intuition being that with high probability, the concatenation of single bits from the different outputs of hc contains at least $m/2k$ "pseudoentropy". Thus, the output of the extractor should form a pseudorandom string. Therefore, the output of the extractor serves as a hardcore function of the multiple randomized iterate $\widehat{\overline{f^k}}$.

[5] The constant 1/20 in the security is an arbitrary choice. It was chosen simply as a constant of the form $1/a \cdot b$ where $a > 3$ and $b > 6$ (which are the constants from Lemma 1 and the GL Theorem.

Construction 2 (Hardcore Function for the Multiple Randomized Iterate). *Let f be a one-way function with security μ_f and let $m, k \in \mathbb{N}$. Let s, $\widehat{f^k}$ and hc be as Construction 1 and Lemma 3 yield w.r.t. f, m and k. Let $\varepsilon_{Ext^k} : \mathbb{N} \to [0,1]$ and let $Ext^k : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{\lceil \frac{m}{4k} \rceil}$ be a $(\lfloor \frac{m}{k} \rfloor, \varepsilon_{Ext^k})$-strong extractor. We define $\overline{hc^k} : Dom(\overline{f^k}) \times \{0,1\}^{2n} \to \{0,1\}^{\lceil \frac{m}{4k} \rceil}$ as*

$$\overline{hc^k}(\overline{x}, H, r, y) = w_1^k(\overline{x}, H, r, y), \ldots, w_s^k(\overline{x}, H, r, y),$$

where $\overline{x} \in \{0,1\}^{mn}$, $H \in \mathcal{H}^{m \times k}$, $r \in \{0,1\}^{2n}$ and $y \in \{0,1\}^n$, and for any $i \in [s]$

$$w_i^k(\overline{x}, H, r, y) = Ext^k((hc(f^k(\overline{x}_1, H_1), r)_i, \ldots, (hc(f^k(\overline{x}_m, H_m), r)_i, y).$$

The following lemma implies that, for the proper choice of m and k, it holds that $\overline{hc^{k-1}}$ is a hardcore function of $\widehat{f^k}$. The proof of Lemma 4 appears in the full version. At the heart of this proof lies the uniform extraction Lemma (see discussion in Section 2).

Lemma 4. *Let hc be a hardcore function of the randomized iterate f^k over the set S^k (as in Lemma 3), and denote its security by μ_{hc}. Let $\overline{hc^{k-1}}, \rho^k$ and ε_{Ext^k} be as in Construction 2 and suppose that ρ^k and ε_{Ext^k} are such that $2s(\rho^k + \varepsilon_{Ext^k}) < \mu_{hc}$. Then $\overline{hc^{k-1}}$ is a hardcore function of the multiple randomized iterate $\widehat{f^k}$ with security $\mu_{\overline{hc^{k-1}}} < poly(m,n)\mu_{hc}^{\alpha}$ for some constant $\alpha > 0$.*

5 A Pseudorandom Generator from Exponentially Hard One-Way Functions

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

Construction 3 (The Pseudorandom Generator). *Let $m, \ell \in \mathbb{N}$ and let f be a one-way function with security μ_f. Let s and $\overline{hc^k}$ be as Construction 2 yields w.r.t. f and m. We define G as*

$$G(\overline{x}, H, r, y) = \overline{hc^1}(\overline{x}, H, r, y) \ldots, \overline{hc^\ell}(\overline{x}, H, r, y), H, r, y$$

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

Theorem 1. *Let $\phi : \mathbb{N} \to [0,1]$ and let f be a one-way function with security $\mu_f = 2^{-\phi(n)n}$. Let $\delta > 2^{-\frac{\phi(n)n}{20}}$. Let $\ell \in poly(n)$ be such that $\sum_{j=1}^{\ell} \frac{1}{j} > \frac{\phi(n)}{80}$ and let $m = 8\ell \log(\frac{\phi(n)n}{\delta})$. Then G presented in Construction 3 is a pseudorandom generator with input length $\mathcal{O}(n\ell m)$ and security $poly(n)\delta^{\alpha}$, where $\alpha > 0$ is a constant.*

With the appropriate choice of parameters we get the statements mentioned in the introduction, as summarized in the following Corollary:

Corollary 1. *Let $C > 0$ be a constant, Theorem 1 yields the following pseudorandom generators,*

- *For $\delta = 2^{-\frac{C}{20}n}$ and $\mu_f = 2^{-Cn}$ - G is pseudorandom generator with security $2^{-C'n}$ (where $C' > 0$ is a constant) and input length $\mathcal{O}(n^2)$.*
- *For $\delta = 2^{-log^2(n)}$ and $\mu_f = 2^{-Cn}$ - G is pseudorandom generator with security $2^{-\Omega(log^2(n))}$ and input length $\mathcal{O}(n \log(n)^2)$.*
- *For $\delta = 2^{-log^2(n)}$ and $\mu_f = 2^{-Cn/log(n)}$, G is pseudorandom generator with security $2^{-\Omega(log^2(n))}$ and input length $\mathcal{O}(n^{1+\frac{160}{C}} \log(n)^2)$.*[6]

References

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

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

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

[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 STOC*, pages 25–32, 1989.

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

[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.

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

[Hol06] Thomas Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In *3rd Theory of Cryptography Conference – (TCC '06)*, LNCS. Springer-Verlag, 2006.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *21st ACM STOC*, pages 12–24, 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.

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

[6] Thus this choice of parameters is only useful when $C > \frac{160}{6}$.