Efficient Pseudorandom Generators from Exponentially Hard One-Way Functions

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] 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.

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 corner-

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

stones 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].

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 $f^k(x) = f(f^{i-1}(x))$ where $f^0(x) = f(x)$.[4] 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 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

[4] We take $f^0(x) = f(x)$ rather than $f^0(x) = x$ for consistency with [HHR05] (see also remark in Section 4).

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*[5] (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 42): 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 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 Appendix A we

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

give a new version of this Lemma since we require a more careful 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.[6] 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).

[6] 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.

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]).

Paper Organization: In Section 3 we provide general definitions and notations. Section 4 gives the core results regarding the randomized iterate. Section 5 presents the multiple randomized iterate and its properties, while Section 6 presents the actual construction of the generator. Appendix A discusses the uniform extraction Lemma.

3 Notations and Definitions

We denote by $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, where ℓ is a function from $\mathbb{N}$ to $\mathbb{N}$, the ensemble of functions $\left\{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\right\}_{n\in\mathbb{N}}$. PPT stands for probabilistic-polynomial time Turing machine. A function $\mu : \mathbb{N} \to [0,1]$ is *negligible* if for every polynomial p we have that $\mu(n) < 1/p(n)$ for large enough n. To denote that μ is negligible we simply write $\mu(n) \in neg(n)$. 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 $D_f(y) \stackrel{\text{def}}{=} \lceil \log \left|f^{-1}(y)\right| \rceil$.

3.1 Distributions and Entropy

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

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

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

By a *Distribution Ensemble* we mean a series $\{D_n\}_{n\in\mathbb{N}}$ where D_n is a distribution over $\{0,1\}^n$. Let $\{X_n\}$ and $\{Y_n\}$ be distribution ensembles. Define the distinguishing advantage of an algorithm A between the ensembles $\{X_n\}$ and $\{Y_n\}$ denoted as $\Delta^A(\{X_n\},\{Y_n\})$, by:

$$\Delta^A(\{X_n\},\{Y_n\}) = |\Pr[A(1^n, X_n) = 1] - \Pr[A(1^n, Y_n) = 1]|$$

where the probabilities are taken over the distributions X_n and Y_n, and the randomness of A.

3.2 Family Of Pairwise-Independent Hash Functions

Definition 31 *(Efficient family of pairwise-independent hash functions) Let $\mathcal{H}$ be a collection of functions where each function $h \in \mathcal{H}$ is from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$. $\mathcal{H}$ is an* efficient family of pairwise-independent hash functions *if $|h|$ (i.e., the description length of h) and $\ell(n)$ are polynomials in n, each $h \in \mathcal{H}$ is a polynomially-computable function, and for all n, for all $x \neq x' \in \{0,1\}^n$ and all $y, y' \in \{0,1\}^{\ell(n)}$,*

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

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

3.3 Randomness Extractors

Randomness extractors, introduced by Nisan and Zuckerman [NZ96] are an information theoretic tool for obtaining true randomness from a "weak" source of randomness. In this work, extractors are used in a computational setting to extract *pseudorandomness* from an imperfect source.

Definition 32 (Strong Extractors) *[NZ96] A polynomial-time computable function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is an (explicit) (k,ε)-*strong extractor *if for every distribution X over $\{0,1\}^n$ with $H_\infty(X) \geq k$, the distribution $(Ext(X,Y),Y)$ is ε-close to (U_m,Y) where Y is uniform over $\{0,1\}^d$.*

3.4 One-Way Functions

Definition 33 (One-way functions) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function and let $\mu : \mathbb{N} \to [0,1]$. f is* one-way *with hardness μ if for every* PPT *A with running time T_A the following holds,*

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

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

3.5 Hardcore Predicates and Functions

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

Definition 34 *[Hardcore Function] Let f and g be functions defined over $\{S_n \subseteq \{0,1\}^n\}_{n\in\mathbb{N}}$ and let $hc : Im(g) \to \{0,1\}^s$ be a polynomial-time computable function, and let $\mu : \mathbb{N} \to [0,1]$. We call hc a* hardcore function *of f with hardness μ w.r.t. $\{S_n\}$ if for any algorithm A with running time T_A the following holds,*

$$\Delta^A\left((f(x), hc(g(x)))_{x \leftarrow S_n}, (f(x), y)_{x \leftarrow S_n, y \leftarrow U_s}\right) / T_A(n) < \mu(n).$$

If hc is a predicate (i.e., $s = 1$) then it is a hardcore predicate *of f*

It is custom to call the value $hc(x)$, the "hardcore-bits" of $f(x)$.

Construction 35 (The GL hardcore function) *Let $x \in \{0,1\}^n$, and $r \in \{0,1\}^{2n}$ and let $\ell \in [n]$. For every $i \in [n]$, denote by $b_i(x,r)$ the inner product of x and the n bit substring of r starting at the i^{th} location (i.e. $b_i(x,r) = \sum_{j\in[n]} x_i \cdot r_{i+j-1}$). Define the function $gl_\ell : \{0,1\}^{3n} \to \{0,1\}^\ell$ as follows: $gl_\ell(x,r) = b_1(x,r), b_2(x,r), \ldots, b_\ell(x,r)$.*

Theorem 36 ([GL89]) *Let f and g be functions defined over $\{S_n \subseteq \{0,1\}^n\}_{n\in\mathbb{N}}$ and suppose that for all* PPT *algorithms A, for all sufficiently large n:*

$$\Pr_{x \leftarrow S_n}[A(f(x)) = g(x)] \leq 2^{-\phi n}$$

Then for X and R uniformly chosen in S_n and $\{0,1\}^{2n}$ respectively we have that no PPT *can distinguish between $(f(X), R, gl_{\phi n/6}(g(X), R))$ and $(f(X), R, U_{\phi n/6})$ with advantage better than $\mathcal{O}(2^{-\phi/6})$. More precisely, let D be an algorithm that distinguishes between the above distributions with probability $2^{-\phi/6}$ and running-time $T_D(n)$, then there exist an algorithms that computes $g(x)$ given $f(x)$ over S_n with probability $\mathcal{O}(2^{-\phi n})$ and running-time $\mathcal{O}(\phi^3 n^5 T_D(n))$*

3.6 Pseudorandom Generators

Definition 37 (Pseudorandom-Generator (PRG)) *Let $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a polynomial-time computable function where $\ell(n) > n$. We say that G is a* Pseudorandom-Generator *with security μ, if for any algorithm A with running time T_A the following holds,*

$$\Delta^A\left(G(U_n)), U_{\ell(n)}\right) / T_A(n) < \mu(n).$$

3.7 The Security of Cryptographic Constructions

We note that the security μ of primitives in this paper is generally measured as the best "success-time" ratio of an adversary. That is, μ is the maximal ratio of the success probability of an adversary A, divided by the A's running time. The extent to which a reduction is security preserving is measured by the relation between the security of the original primitive and the security of the resulting one. For a more comprehensive discussion about the security of reductions one may refer to [HL92] (and also to [HHR05].

It is worth noting that the results in this paper may be adapted to work with respect to a less strict security measure. That is, if the security is taken to be the highest success probability of a polynomially bounded adversary. The later notion of security is weaker in the sense that the first notion implies the second but not vice versa.

4 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 41 (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*[7] *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.

4.1 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.

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

Lemma 42 *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 41.*

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 43

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

.

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 44 *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 42, take $T = T_A$ and $\delta = \varepsilon/2k$, and Lemma 44 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 44) 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 44 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 42 regarding the hardness of inverting the last iteration, and the Goldreich-Levin Theorem on hardcore functions we get the following lemma:

Lemma 45 *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 41. 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.[8]

5 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 51 (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 41. 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 45 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 $\widehat{\overline{f^k}}$.

Construction 52 (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, $\widehat{\overline{f^k}}$ and hc be as Construction 51 and Lemma 45 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.*

[8] 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 42 and the GL Theorem.

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 $\widehat{\overline{f^k}}$.

Lemma 53 *Let hc be a hardcore function of the randomized iterate* $\overline{f^k}$ *over the set* S^k *(as in Lemma 45), and denote its security by* μ_{hc}. *Let* $\overline{hc^{k-1}}, \rho^k$ *and* ε_{Ext^k} *be as in Construction 52 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* $\widehat{\overline{f^k}}$ *with security* $\mu_{\overline{hc^{k-1}}} < poly(m,n)\mu_{hc}^{\alpha}$ *for some constant* $\alpha > 0$.

Lemma 53 is derived as a corollary of the following Lemma 54. This derivation is similar to the proof of Corollary A2.

Lemma 54 *Let* f *be a one-way function with security* μ_f. *For choices of* $m, k \in \mathbb{N}$, *let* $Ext^k, \varepsilon_{Ext^k}, \widehat{\overline{f^k}}$ *and* $\overline{hc^k}$ *be as in Construction 52. Denote by* ρ^k *the probability that less than* $\lfloor \frac{m}{2k} \rfloor$ *samples out of* m *independent samples in* $Dom(f^k)$, *are in some fixed set of density* $1/k$.

If there exists an algorithm A that distinguishes between $(\widehat{\overline{f^k}}(\overline{x}, H), \overline{hc^{k-1}}(y, r, x), r, y)$ *and* $(\overline{f^k}(\overline{x}, H), z, r, y)_{z \leftarrow U_{\lceil \frac{m}{4k} \rceil}}$ *with probability* ε_A *and also* $(\varepsilon_A/s - \rho^k + \varepsilon_{Ext^k})/m - 2^{-n} > 2^{n/3}$,

then there exists an algorithm M^A *that runs in time polynomial in* n, $\frac{m}{\varepsilon_A/s - \rho^k + \varepsilon_{Ext^k}}$ *and the running time of* $\widehat{\overline{f^k}}$, $\overline{hc^{k-1}}$ *and A, that distinguishes between* $(\widehat{f^k}(x, \overline{h}), hc(f^{k-1}(x, \overline{h}), r), r)$ *and* $(\widehat{f^k}(x, \overline{h}), z, r)_{z \leftarrow U_s}$ *with probability* $\frac{k-1}{2k} - 2^{-n}$.

At the heart of the proof lies a uniform extraction Lemma. that is presented in Appendix A.

Proof: (of Lemma 54) By a standard hybrid argument there exists an index $i \in [s]$, and an algorithm A' that runs in time polynomial in n, and the running time of $\widehat{\overline{f^k}}$, $\overline{hc^{k-1}}$ and A, and distinguishes between
$(\widehat{\overline{f^k}}(\overline{x}, H), w_1^{k-1}(\overline{x}, H, r, y), \ldots, w_i^{k-1}(\overline{x}, H, r, y), r, y)$ and
$(\widehat{\overline{f^k}}(\overline{x}, H), w_1^{k-1}(\overline{x}, H, r, y), \ldots, w_{i-1}^{k-1}(\overline{x}, H, r, y), z, r, y)_{z \leftarrow U_{\lceil \frac{m}{4k} \rceil}}$ with probability ε_A/s. Recalling the definitions of $\widehat{\overline{f^k}}$ and the w^{k-1}'s, we have that he following pair of distributions can be inverted (using Ext^k) to the above one.
$(\widehat{f^k}(\overline{x}_1, H_1), hc(f^{k-1}(\overline{x}_1, H_1), r)_{1,\ldots,i-1}, \ldots, \widehat{f^k}(\overline{x}_m, H_m), hc(f^{k-1}(\overline{x}_m, H_m), r)_{1,\ldots,i-1},$

$w_i^{k-1}(\overline{x}, H, r, y), r, y)$ and
$(\widehat{f^k}(\overline{x}_1, H_1), hc(f^{k-1}(\overline{x}_1, H_1), r)_{1,\ldots,i-1}, \ldots, \widehat{f^k}(\overline{x}_m, H_m), hc(f^{k-1}(\overline{x}_m, H_m), r)_{1,\ldots,i-1}, z, r, y)_{z \leftarrow U_{\lceil \frac{m}{4k} \rceil}}$.

Hence there exists an algorithm A'', with essentially the same running time of A', and distinguishes between the above pair of distributions with probability ε_A/s. The crux of the proof is that the definition of w_i matches the settings of the uniform extraction Lemma (Lemma A1) (by setting $f(x, \overline{h}) = (\widehat{f^k}(x, \overline{h}), hc(f^{k-1}(x, \overline{h}))_{1,\ldots,i-1})$ and $b(x, \overline{h}) = hc(f^{k-1}(x, \overline{h}))_i$). Therefore, algorithm A'' can be used to construct an algorithm M^A that runs in time polynomial n, $\frac{m}{\varepsilon_A/s - \varepsilon_{Ext^k} - \rho^k}$ and in the running time of $\widehat{\widehat{f^k}}$, $\overline{hc^k}$ and A and distinguishes $(\widehat{f^k}(x, \overline{h}), hc(f^{k-1}(x, \overline{h}))_{1,\ldots,i}, r)$ from $(\widehat{f^k}(x, \overline{h}), hc(f^{k-1}(x, \overline{h}))_{1,\ldots,i-1}, z, r)_{z \leftarrow U_s}$ with probability $\frac{k-1}{2k}$. Since the value of i is also polynomially computable (with success probability $1-2^{-n}$), M^A distinguishes between $(\widehat{f^k}(x, \overline{h}), hc(f^{k-1}(x, \overline{h})), r)$ and $(\widehat{f^k}(x, \overline{h}), z, r)_{z \leftarrow U_s}$ with probability $\frac{k-1}{2k} - 2^{-n}$. ∎

6 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 61 (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 52 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 62 *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 61 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 63 *Let $C > 0$ be a constant, Theorem 62 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 $\mathcal{O}(n^{1+\frac{160}{C}}\log(n)^2)$.*[9]

Proof: (of Corollary 63) The security of the different pseudorandom generators is immediate by Theorem 62. Since $2 > \frac{\phi(n)}{80}\sum_{j=1}^{\ell}\frac{1}{j} > \frac{\phi(n)}{80}log(\ell)$ it follows that $\ell < 2^{\frac{160}{\phi(n)}}$. Hence for all different values of μ_f, it holds that ℓ and therefore G is polynomial. It is left to calculate the different values of $n \cdot \ell \cdot m$ in order to determine the input length of G in the different cases. Note that in all cases the dominating factor in the value of m is $\log(1/\delta)$.

- $\delta = 2^{-\frac{C}{20}n}$ and $\mu_f = 2^{-Cn}$ - ℓ is clearly a constant and $m = \mathcal{O}(n)$, therefore the input length is $\mathcal{O}(n\log(n)^2)$.
- $\delta = 2^{-log^2(n)}$ and $\mu_f = 2^{-Cn}$ - ℓ is, again, a constant and $m = \mathcal{O}(\log(n)^2)$, therefore the input length is $\mathcal{O}(n\log(n)^2)$.
- $\delta = 2^{-log^2(n)}$ and $\mu_f = 2^{-Cn/log(n)}$, $\ell < 2^{\frac{160\log(n)}{C}} = n^{\frac{160}{C}}$ and $m = \mathcal{O}(\log(n)^2)$, therefore the input length is $\mathcal{O}(n^{1+\frac{160}{C}}\log(n)^2)$.

∎

Proof: (of Theorem 62) Since ℓ is polynomial in n then clearly so is m, it follows by the definition of $\overline{hc^k}$ that the running time of G is polynomial as well. We next show that G stretches its input. The input length of G is $m|x| + |H| + |r| + |y|$, while the output length is $sm(\sum_{j=1}^{\ell}\lceil\frac{1}{4j}\rceil) + |H| + |r| + |y|$ which is, by the choice of ℓ, greater than $mn + |H| + |r| + |y|$. Finally, since $|x| = n$, the latter sum equals to the input length of G. Note that since the input length of G is dominated by the description of H it is in $\mathcal{O}(n\ell m)$.

It is left to calculate the security of G. By a standard hybrid argument an algorithm A that runs in time T_A and breaks G with probability ε_A, implies the existence of an algorithm A' that runs in time polynomial in T_A and distinguishes for some (efficiently computable) index $k \in [\ell]$ between
$(\overline{hc^k}(\overline{x}, H, r, y), \ldots, \overline{hc^\ell}(\overline{x}, H, r, y), H, r, y)$ and $(U_{\lceil\frac{m}{4k}\rceil}, \overline{hc^{k+1}}(\overline{x}, H, r, y)\ldots, \overline{hc^\ell}(\overline{x}, H, r, y), H, r, y)$
with probability ε_A/ℓ. Since given $\widehat{f^k}(\overline{x}, H)$, r and y it is easy to compute $\overline{hc^{k+1}}(\overline{x}, H, r, y), \ldots, \overline{hc^\ell}(\overline{x}, H, r, y)$, then there exists an algorithm A'' that also runs in time polynomial in T_A and distinguishes between $(\widehat{f^k}(\overline{x}, H), \overline{hc^{k-1}}(\overline{x}, H, r, y), r, y)$

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

and $(\overline{f^k}(\overline{x},H),U_{\lceil \frac{m}{4k} \rceil},r,y)$ with probability ε_A/ℓ. By our choice of m and Chernoff's bound we have that $\rho^k < \rho^l < \frac{\delta}{\phi(n)n}$. Similarly for the proper choice of extractor [10], we have that $Ext^k < Ext^l < \frac{\delta}{\phi(n)n}$.

Recall that for $\mu_f = 2^{-\phi(n)n}$, it holds that s, the output length of hc, equals $\lceil \phi(n)n/20 \rceil$ and that $\mu_{hc} < 2^{-\frac{\phi(n)n}{20}} < \delta$. Since $2s(Ext^k + \rho^k) < \lceil \phi(n)n/20 \rceil \frac{2}{\phi(n)n}\delta < \delta$, then Lemma 53 yields that $\frac{\varepsilon_{A''}}{T_{A''}} < poly(n)\delta^{\alpha}$ for some constant $\alpha > 0$. Since $\varepsilon_{A''} \geq \varepsilon_A/\ell$ and since ℓ is polynomial in n and the running time of A'' is polynomial in the running time of A, it follows that $\frac{\varepsilon_A}{T_A} < poly'(n)\delta^{\alpha'}$ for some constant $\alpha' > 0$. ∎

References

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

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

[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 Symposium on the Theory of Computing*, pages 25–32, 1989.

[HHR05] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. Electronic Colloquium on Computational Complexity (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.

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

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

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

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

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

[10] For example, for a random $h \in \mathcal{H}$ (where $\mathcal{H}$ is a family of pairwise independent hash functions), $Ext(x,h) = (h(x),h)$ is a suitable strong extractor.

[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.

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

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

A A Uniform Extraction Lemma

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

Lemma A1 (A Uniform Extraction Lemma) *Let $f:\{0,1\}^n \to \{0,1\}^{\ell(n)}$ and $b:\{0,1\}^n \to \{0,1\}$ be a polynomial-time computable functions, let $\delta \in [0,1]$ be noticeable, let $\varepsilon_{Ext}, \varepsilon_A, \delta \in [0,1]$ and let $m,k,r,d \in \mathbb{N}$. Let ρ be the probability that when taking m independent samples in $\{0,1\}^n$ less than k samples are in some fixed set of density δ. Let $Ext:\{0,1\}^m \times \{0,1\}^d \to \{0,1\}^r$ be a (k,ε_{Ext})-strong-extractor and let D and ξ be the following distributions,*
$D = (f(x_1),\ldots,f(x_m),y,Ext(y,b_s(x_1),\ldots,b_s(x_m)))_{(x_1,\ldots,x_m)\leftarrow U_{nm},y\leftarrow U_d}$,
$\xi = (f(x_1),\ldots,f(x_m),y,z)_{,(x_1,\ldots,x_m)\leftarrow U_{nm},y\leftarrow U_d,z\leftarrow U_r}$.
Finally, let A be an algorithm that runs in time T_A and distinguishes between D and ξ with probability ε_A.

Then if $\frac{\varepsilon_A-\varepsilon_{Ext}-\rho}{m} - 2^{-n} > 2^{-n/3}$, then there exists an algorithm M^A that runs in time $poly(n,\frac{m}{\varepsilon_A-\varepsilon_{Ext}-\rho},T_A)$ and distinguishes between $(f(x),b(x))_{x\leftarrow U_n}$ and $(f(x),y)_{x\leftarrow U_n,y\leftarrow U}$ with probability $\frac{1-\delta}{2}$.

Corollary A2 *Let $S_n \subseteq \{0,1\}^n$ be a set of density 2δ, and let μ_b be the security of b w.r.t. f and the uniform distribution over S_n. Let $hc(x_1,\ldots,x_m,y) \stackrel{def}{=} (Ext(y,b_s(x_1),\ldots,b_s(x_m)),y)$ and let $f^k(x_1,\ldots,x_m,y) \stackrel{def}{=} (x_1),\ldots,f(x_m)$.*

Let μ_{hc} be the security of hc w.r.t. f^k. Assuming that $\mu_b > 2(\rho+\varepsilon_{Ext})$, then $\mu_{hc} < poly(n,m)(\frac{\mu_b}{\delta})^\alpha$ for some constant $\alpha > 0$.

Proof: (of Corollary A2) Assume not. Let $p(n, \frac{m}{\varepsilon_A - \varepsilon_{Ext} - \rho}, T_A)$ be the polynomial whose existence is guaranteed by Lemma A1, clearly $p(n, \frac{m}{\varepsilon_A - \varepsilon_{Ext} - \rho}, T_A) < q(m,n)(\frac{T_A}{\varepsilon})^{1/\alpha}$ for some positive polynomial q and constant $\alpha > 0$. By the contradiction assumption there exists an algorithm A with running time T_A that distinguishes between D and χ with probability ε_A such that $\frac{\varepsilon_A}{T_A} \geq 2q(n,m)(\frac{\mu_b}{\delta})^{\alpha}$. Since clearly $\varepsilon_A > \mu_b$, then $\varepsilon_A - \varepsilon_{Ext} - \rho \geq \varepsilon_A/2$ and therefore $p(n, \frac{m}{\varepsilon_A - \varepsilon_{Ext} - \rho}, T_A) < p(n, \frac{2m}{\varepsilon_A}, T_A) < q(n,m)(\frac{T_A}{\varepsilon_A})^{1/\alpha} \leq q(n,m)\frac{1}{2q(n,m)}\frac{\delta}{\mu_b} = \frac{\delta}{2\mu_b}$.

By Lemma A1 there exists an algorithm M^A that runs in time $T_{M^A} < \frac{\delta}{2\mu_b}$ and distinguishes between $P_n = (f(x), b(x))_{x \leftarrow U_n}$ and $Q_n = (f(x), y)_{x \leftarrow U_n, y \leftarrow U}$ with probability $\frac{1}{2} - \delta/2$. Clearly,

$$\Delta_{M^A}(P_n, Q_n) \leq \Pr_{x \leftarrow \{0,1\}^n}[x \notin S_n]/2 + \Delta_{M^A}(P_n, Q_n | x \in S_n)/2 \cdot \Pr_{x \leftarrow \{0,1\}^n}[x \in S_n] \leq (1 - 2\delta)/2 + \Delta_{M^A}(P_n, Q_n | x \in S_n).$$

Therefore, $\varepsilon_{M^A} \stackrel{\text{def}}{=} \Delta_{M^A}(P_n, Q_n | x \in S_n) \geq (2\delta - \delta)/2 = \delta/2$. Hence $\frac{\varepsilon_{M^A}}{T_{M^A}} > \frac{\delta/2}{\delta/2\mu_b} = \mu_b$. Thus, a contradiction to the hardness assumption of b is derived. ∎

Proof: (of Lemma A1) For a given set $S \subseteq \{0,1\}^n$ of density δ, we define the following, not necessarily efficiently computable, predicate $Q : \{0,1\}^n \to \{0,1\}$.

$$Q(x) = \begin{cases} U_1 & x \in S, \\ b(x) & \text{otherwise.} \end{cases} \tag{4}$$

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

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

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

The following algorithm given oracle to the characteristic function χ^S of any set of density δ, distinguishes between $(f(x), b(x))_{x \leftarrow S}$ and $(f(x), y)_{x \leftarrow S, y \leftarrow U}$ with probability $(\varepsilon_A - \rho + \epsilon)/m - 2^{-n}$.

M^A **with oracle** χ^S**:**

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

We conclude by quoting the following lemma due to Holenstein, which guarantees the existence of an algorithm that runs in time polynomial in n, $\frac{m}{(\varepsilon_A - \varepsilon_{Ext} - \rho)}$ and the running time of M^A, and distinguishes between $(f(x), b(x))_{x \leftarrow U_n}$ and $(f(x), y)_{x \leftarrow U_n, y \leftarrow U}$ with probability $\frac{1-\delta}{2}$. ∎

Lemma A3 *[Hol05, Lemma 2.5] Let* $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, $b : \{0,1\}^n \to \{0,1\}$, $\delta : \mathbb{N} \to [0,1]$ *and* $\gamma : \mathbb{N} \to [0,1]$ *be polynomial-time computable functions, where* δ *is noticeable and* $\gamma > 2^{-n/3}$.

Further, assume that there exists an oracle algorithm $A^{()}$ *such that, for infinitely many* n*'s, the following holds: For any set* $S \subseteq \{0,1\}^n$ *with density* δ, A^{χ_S} *outputs a circuit* C *satisfying*

$$Ex[\Delta_C((f(x), b(x))_{x \leftarrow S_n}, (f(x), y)_{x \leftarrow S_n, y \leftarrow U})] \geq \gamma/2$$

where the expectation is over the randomness of A.

Then, there is an algorithm M^A, *which call* A *as a black-box* $poly(n, 1/\gamma)$ *times that distinguishes between* $(f(x), b(x))_{x \leftarrow \{0,1\}^n}$ *and* $(f(x), y)_{x \leftarrow \{0,1\}^n, y \leftarrow U}$ *with probability* $\frac{1-\delta}{2}$.