

# Pseudorandom Generators from One-Way Functions: A Simple Construction for Any Hardness

Thomas Holenstein

ETH Zurich, Department of Computer Science, CH-8092, Zurich  
thomahol@inf.ethz.ch

**Abstract.** In a seminal paper, Håstad, Impagliazzo, Levin, and Luby showed that pseudorandom generators exist if and only if one-way functions exist. The construction they propose to obtain a pseudorandom generator from an  $n$ -bit one-way function uses  $\mathcal{O}(n^8)$  random bits in the input (which is the most important complexity measure of such a construction). In this work we study how much this can be reduced if the one-way function satisfies a stronger security requirement. For example, we show how to obtain a pseudorandom generator which satisfies a standard notion of security using only  $\mathcal{O}(n^4 \log^2(n))$  bits of randomness if a one-way function with exponential security is given, i.e., a one-way function for which no polynomial time algorithm has probability higher than  $2^{-cn}$  in inverting for some constant  $c$ .

Using the uniform variant of Impagliazzo's hard-core lemma given in [7] our constructions and proofs are self-contained within this paper, and as a special case of our main theorem, we give the first explicit description of the most efficient construction from [6].

## 1 Introduction

A pseudorandom generator is a deterministic function which takes a uniform random bit string as input and outputs a longer bit string which cannot be distinguished from a uniform random string by any polynomial time algorithm. This concept, introduced in the fundamental papers of Yao [16] and Blum and Micali [1] has many uses. For example, it immediately gives a semantically secure cryptosystem: the input of the pseudorandom generator is the key of the cryptosystem, and the output is used as a one-time pad. Other uses of pseudorandom generators include the construction of pseudorandom functions [2], pseudorandom permutations [11], statistically binding bit commitment [13], and many more.

Such a pseudorandom generator can be obtained from an arbitrary one-way function, as shown in [6]. The given construction is not efficient enough to be used in practice, as it requires  $\mathcal{O}(n^8)$  bits of input randomness (for example, if one would like to have approximately the security of a one-way function with  $n = 100$  input bits, the resulting pseudorandom generator would need several petabits of input, which is clearly impractical). On the other hand, it is possible

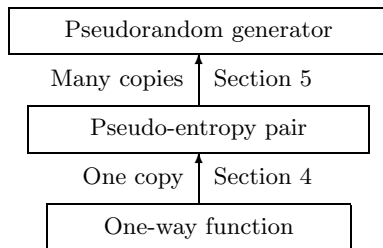
to obtain a pseudorandom generator very efficiently from an arbitrary one-way *permutation* [4] or from an arbitrary regular one-way function [3] (see also [5]), i.e., a one-way function where every image has the same number of preimages. In other words, if we have certain guarantees on the *combinatorial structure* of the one-way function, we can get very efficient reductions.

In this paper we study the question whether a pseudorandom generator can be obtained more efficiently under a stronger assumption on the *computational difficulty* of the one-way function. In particular, assume that the one-way function is harder to invert than usually assumed. In this case, one single invocation of the one-way function could be more useful, and fewer invocations might be needed. We will see that is indeed the case, even if the pseudorandom generator is supposed to inherit a stronger security requirement from the one-way function, and not only if it is supposed to satisfy the standard security notion.

## 2 Overview of the Construction

The construction given in [6] uses several stages: first the one-way function is used to construct a false entropy generator, i.e., a function whose output is computationally indistinguishable from a distribution with more entropy. (This is the technically most difficult part of the construction and the security proof can be significantly simplified by using the uniform hard-core lemma from [7].) Next, the false entropy generator is used to construct a pseudoentropy generator (a function whose output is computationally indistinguishable from a distribution which has more entropy than the input), and finally a pseudorandom generator is built on top of that. If done in this way, their construction is very inefficient (requiring inputs of length  $\mathcal{O}(n^{34})$ ), but it is also sketched in [6] how to “unroll” the construction in order to obtain an  $\mathcal{O}(n^{10})$  construction. Similarly it is mentioned that an  $\mathcal{O}(n^8)$  construction is possible (by being more careful).

In this work we explicitly describe an  $\mathcal{O}(n^8)$  construction (in an unrolled version the construction we describe is the one sketched in [6]). Compared to [6] we choose a different way of presenting this construction; namely we use a two-step approach (see Figure 1). First, (in Section 4) we use the one-way function to construct a pair  $(g, P)$  where  $g$  is an efficiently evaluable function and  $P$  is a predicate. The pair will satisfy that predicting  $P(x)$  from  $g(x)$  is computationally



**Fig. 1.** Overview of our construction

difficult (in particular, more difficult than it would be information theoretically). In [5] the term *pseudo-entropy pair* is coined for such a pair and we will use this term as well. In a second step we use many instances of such a pseudo-entropy pair to construct a pseudorandom generator.

Further, we generalize the construction to the case where stronger security guarantees on the one-way function are given. This enables us to give more efficient reductions under stronger assumptions.

Independently of this work, Haitner, Harnik, and Reingold [5] give a better method to construct a pseudo-entropy pair from a one-way function. Their construction has the advantage that the entropy of  $P(x)$  given  $g(x)$  can be estimated, which makes the construction of the pseudorandom generator from the pseudo-entropy pair more efficient.

### 3 Definitions and Result

#### 3.1 Definitions and Notation

**Definition 1.** *A one-way function with security  $s(n)$  against  $t(n)$ -bounded inverters is an efficiently evaluable family of functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  such that for any algorithm running in time at most  $t(n)$*

$$\Pr_{x \leftarrow_R \{0,1\}^n} [f(A(f(x))) = f(x)] < \frac{1}{s(n)}$$

for all but finitely many  $n$ .

For example the standard notion of a one-way function is a function which is one-way with security  $p(n)$  against  $p(n)$ -bounded inverters for all polynomials  $p(n)$ .

In [15] it is shown that a random permutation is  $2^{n/10}$ -secure against  $2^{n/5}$ -bounded inverters, and also other reasons are given why it is not completely unreasonable to assume the existence of one-way permutations with exponential security. In our main theorem we can use one-way functions with exponential security, a weaker primitive than such permutations.

**Definition 2.** *A pseudorandom-generator with security  $s(\ell)$  against  $t(\ell)$ -bounded distinguishers is an efficiently evaluable family of (expanding) functions  $h : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+1}$  such that for any algorithm running in time at most  $t(\ell)$*

$$\left| \Pr_{x \leftarrow_R \{0,1\}^\ell} [A(h(x)) = 1] - \Pr_{u \leftarrow_R \{0,1\}^{\ell+1}} [A(u) = 1] \right| \leq \frac{1}{s(\ell)},$$

for all but finitely many  $\ell$ .

The standard notion of a pseudorandom generator is a pseudorandom generator with security  $p(\ell)$  against  $p(\ell)$ -bounded distinguishers, for all polynomials  $p(\ell)$ .

As mentioned above, we use pseudo-entropy pairs as a step in our construction. For such a pair of functions we first define the advantage an algorithm  $A$  has in predicting  $P(w)$  from  $g(w)$  (by convention, we use the letter  $w$  to denote the input here).

**Definition 3.** For any algorithm  $A$ , any function  $g : \{0, 1\}^n \rightarrow \{0, 1\}^m$  and any predicate  $P : \{0, 1\}^n \rightarrow \{0, 1\}$ , the advantage of  $A$  in predicting  $P$  given  $g$  is

$$\text{Adv}^A(g, P) := 2 \left( \Pr_{w \leftarrow_R \{0, 1\}^n} [A(g(w)) = P(w)] - \frac{1}{2} \right).$$

The following definition of a pseudo-entropy pair contains (somewhat surprisingly) the conditioned entropy  $H(P(W)|g(W))$ ; we give an explanation below.

**Definition 4.** A pseudo-entropy pair with gap  $\phi(n)$  against  $t(n)$ -bounded predictors is a pair  $(g, P)$  of efficiently evaluable functions  $g : \{0, 1\}^n \rightarrow \{0, 1\}^m$  and  $P : \{0, 1\}^n \rightarrow \{0, 1\}$  such that for any algorithm  $A$  running in time  $t(n)$

$$\text{Adv}^A(g, P) \leq 1 - H(P(W)|g(W)) - \phi,$$

for all but finitely many  $n$  (where  $W$  is uniformly distributed over  $\{0, 1\}^n$ ).

The reader might think that it would be more natural if we used the best advantage for computationally unbounded algorithms (i.e., the information theoretic advantage), instead of  $1 - H(P(W)|g(W))$ . Then  $\phi$  would be the gap which comes from the use of  $t(n)$ -bounded predictors. We quickly explain why we chose the above definition. First, to get an intuition for the expression  $1 - H(P(W)|g(W))$ , assume that the pair  $(g, P)$  has the additional property that for every input  $w$ ,  $g(w)$  either fixes  $P(w)$  completely or does not give any information about it, i.e., for a fixed value  $v$  either  $H(P(W)|g(W) = v) = 1$  or  $H(P(W)|g(W) = v) = 0$  holds. Then, a simple computation shows that  $1 - H(P(W)|g(W))$  is a tight upper bound on the advantage of computationally unbounded algorithms, i.e., in this case our definition coincides with the above “more natural definition”. We mention here that the pairs  $(g, P)$  we construct will be close to pairs which have this property. If there are values  $v$  such that  $0 < H(P(W)|g(W) = v) < 1$ , the expression  $1 - H(P(W)|g(W))$  is *not* an upper bound anymore and in fact one might achieve significantly greater advantage than  $1 - H(P(W)|g(W))$ . Therefore in this case, Definition 4 requires something stronger than the “more natural definition”, and, consequently, constructing a pseudorandom generator from a pseudo-entropy pair becomes easier.<sup>1</sup>

We use  $\parallel$  to denote concatenation of strings,  $ax$  denotes the multiplication of bitstrings  $a$  and  $x$  over  $\text{GF}(2^n)$  (with an arbitrary representation), and  $x|_\lambda$  denotes the first  $\lambda$  bits of the bit string  $x$ . For fixed  $x$  and  $\bar{x}$ ,  $x \neq \bar{x}$ , the probability that  $(ax)|_i$  equals  $(a\bar{x})|_i$  for uniformly chosen  $a$  can be computed as

$$\Pr_{a \leftarrow \{0, 1\}^n} [(ax)|_i = (a\bar{x})|_i] = \Pr_{a \leftarrow \{0, 1\}^n} [(a(x - \bar{x}))|_i = 0^i] = 2^{-i}, \tag{1}$$

an expression we will use later.

---

<sup>1</sup> In fact, we do not know a direct way to construct a pseudorandom generator from a pseudo-entropy pair with the “more natural definition”.

For bitstrings  $x$  and  $r$  of the same length  $n$  we use  $x \odot r := \bigoplus_{i=1}^n x_i r_i$  for the inner product. We use the convention that  $f^{-1}(y) := \{x \in \{0, 1\}^n \mid f(x) = y\}$ , i.e.,  $f^{-1}$  returns a set.

For two distributions  $P_{X_0}$  and  $P_{X_1}$  over  $\mathcal{X}$  the statistical distance is

$$\Delta(X_0, X_1) := \frac{1}{2} \sum_{x \in \mathcal{X}} |P_{X_0}(x) - P_{X_1}(x)|.$$

We also say that a distribution is  $\varepsilon$ -close to another distribution if the statistical distance of the distributions is at most  $\varepsilon$ . For a distribution  $P_X$  over  $\mathcal{X}$  the min-entropy is  $H_\infty(X) := -\log(\max_{x \in \mathcal{X}} P_X(x))$ . For joint distributions  $P_{XY}$  over  $\mathcal{X} \times \mathcal{Y}$  the conditional min-entropy is defined with  $H_\infty(X|Y) := \min_{y \in \mathcal{Y}} H_\infty(X|Y = y)$ .

Finally, we define  $[n] := \{1, \dots, n\}$ .

### 3.2 Result

We give a general construction of a pseudorandom generator from a one-way function. The construction is parametrized by two parameters  $\varepsilon$  and  $\phi$ . The parameter  $\varepsilon$  should be chosen such that it is smaller than the target indistinguishability of the pseudorandom generator: an algorithm which distinguishes the output of the pseudorandom generator with advantage less than  $\varepsilon$  will not help us in inverting  $f$ . The second parameter  $\phi$  should be chosen such that the given one-way function cannot be inverted with probability more than about  $2^{-n\phi}$  (as an example, for standard security notions choosing  $\phi = \frac{1}{n}$  and  $\varepsilon = 2^{-n}$  would be reasonable – these should be considered the canonical choices).

**Theorem 1.** *Let functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ ,  $\phi : \mathbb{N} \rightarrow [0, 1]$ ,  $\varepsilon : \mathbb{N} \rightarrow [0, 1]$  be given, computable in time  $\text{poly}(n)$ , and satisfying  $2^{-n} \leq \varepsilon \leq \frac{1}{n} \leq \phi$ .*

*There exists an efficient to evaluate oracle function  $h_{\varepsilon, \phi}^f$  with the following properties:*

- $h_{\varepsilon, \phi}^f$  is expanding,
- $h_{\varepsilon, \phi}^f$  has input of length  $\mathcal{O}(\frac{n^4}{\phi^4} \log(\frac{1}{\varepsilon}))$ , and
- an algorithm  $A$  which distinguishes the output of  $h_{\varepsilon, \phi}^f$  from a uniform bit string with advantage  $\gamma$  can be used to get an oracle algorithm which inverts  $f$  with probability  $\mathcal{O}(\frac{1}{n^3})2^{-n\phi}$ , using  $\text{poly}(n, \frac{1}{\gamma - \varepsilon})$  calls to  $A$ .

For example, if we set  $\phi := \log(n)/n$  and  $\varepsilon := n^{-\log(n)} = 2^{-\log^2(n)}$  and use a standard one-way function in the place of  $f$ , then  $h_{\varepsilon, \phi}^f$  will be a standard pseudorandom generator, using  $\mathcal{O}(n^8)$  bits<sup>2</sup> of randomness.

**Corollary 1.** *Assume that  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  is a one-way function with security  $p(n)$  against  $p(n)$ -bounded inverters, for all polynomials  $p(n)$ . Then there exists a pseudorandom generator  $h : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+1}$  with security  $p(\ell)$  against  $p(\ell)$ -bounded distinguishers, for all polynomials  $p(\ell)$ . The construction calls the one-way function for one fixed  $n$  dependent of  $\ell$  and satisfies  $\ell \in \mathcal{O}(n^8)$ .*

<sup>2</sup> This could be insignificantly reduced by choosing  $\varepsilon$  slightly bigger.

Alternatively if we have a much stronger one-way function which no polynomial time algorithm can invert with better probability than  $2^{-cn}$  for some constant  $c$ , we can set  $\phi$  to some appropriate small constant and  $\varepsilon := n^{-\log(n)}$ , which gives us a pseudorandom generator using  $\mathcal{O}(n^4 \log^2(n))$  bits of input:

**Corollary 2.** *Assume that  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  is a one-way function with security  $2^{-cn}$  against  $p(n)$ -bounded inverters, for some constant  $c$  and all polynomials  $p(n)$ . Then there exists a pseudorandom generator  $h : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+1}$  with security  $p(\ell)$  against  $p(\ell)$ -bounded distinguishers, for all polynomials  $p(\ell)$ . The construction calls the one-way function for one fixed  $n$  dependent of  $\ell$ , and satisfies  $\ell \in \mathcal{O}(n^4 \log^2(n))$ .*

If we want a pseudorandom generator with stronger security we set  $\varepsilon$  smaller in our construction. For example, if a one-way function  $f$  has security  $2^{cn}$  against  $2^{cn}$  bounded distinguishers, we set  $\phi$  (again) to an appropriate constant and  $\varepsilon := 2^{-n}$ . With these parameters our construction needs  $\mathcal{O}(n^5)$  input bits, and, for an appropriate constant  $d$ , an algorithm with distinguishing advantage  $2^{-dn}$ , and running in time  $2^{dn}$ , can be used to get an inverting algorithm which contradicts the assumption about  $f$ . (A corollary similar to the ones before could be formulated here).

The proof of Theorem 1 is in two steps (see Figure 1). In Section 4 we use the Goldreich-Levin Theorem and two-universal hash-functions to obtain a pseudo-entropy pair. In Section 5 we show how such a pair can be used to obtain a pseudorandom generator.

### 3.3 Extractors

Informally, an extractor is a function which can extract a uniform bit string from a random string with sufficient min-entropy. The following well known left-over hash lemma from [10] shows that multiplication over  $\text{GF}(2^n)$  with a randomly chosen string  $a$  and then cutting off an appropriate number of bits can be used to extract randomness. For completeness we give a proof (adapted from [12]).

**Lemma 1 (Left-over hash lemma).** *Let  $x \in \{0, 1\}^n$  be chosen according to any source with min-entropy  $\lambda$ . Then, for any  $\varepsilon > 0$ , and uniform random  $a$ , the distribution of  $((ax)|_{\lambda - 2\log(\frac{1}{\varepsilon})} \parallel a)$  is  $\frac{\varepsilon}{2}$ -close to a uniform bit string of length  $\lfloor \lambda - 2\log(\frac{1}{\varepsilon}) \rfloor + n$ .*

*Proof.* Let  $m := \lfloor \lambda - 2\log(\frac{1}{\varepsilon}) \rfloor$ , and  $P_{VA}$  be the distribution of  $(ax)|_m \parallel a$ . Further, let  $P_U$  be the uniform distribution over  $\{0, 1\}^{m+n}$ . Using the Cauchy-Schwartz inequality  $(\sum_{i=1}^k a_i)^2 \leq k \sum_{i=1}^k a_i^2$  we obtain for the statistical distance in question

$$\Delta(VA, U) = \frac{1}{2} \sum_{v \in \{0,1\}^m, a \in \{0,1\}^n} \left| P_{VA}(v, a) - \frac{1}{2^n 2^m} \right|$$

$$\begin{aligned}
 &\leq \frac{1}{2} \sqrt{2^n 2^m} \sqrt{\sum_{v,a} P_{VA}^2(v,a) - 2 \sum_{v,a} \frac{P_{VA}(v,a)}{2^n 2^m} + \sum_{v,a} \left(\frac{1}{2^n 2^m}\right)^2} \\
 &= \frac{1}{2} \sqrt{2^n 2^m} \sqrt{\sum_{v,a} P_{VA}^2(v,a) - \frac{1}{2^n 2^m}}. \tag{2}
 \end{aligned}$$

Let now  $X_0$  and  $X_1$  be independently distributed according to  $P_X$  (i.e., the source with min-entropy  $\lambda$ ). Further, let  $A_0$  and  $A_1$  be independent over  $\{0, 1\}^n$ . The collision probability of the output distribution is

$$\Pr\left[\left((X_0 A_0)|_m \parallel A_0\right) = \left((X_1 A_1)|_m \parallel A_1\right)\right] = \sum_{v,a} P_{VA}^2(v,a).$$

Thus we see that equation (2) gives an un upper bound on  $\Delta(VA, U)$  in terms of the collision probability of two independent invocations of the hash-function on two independent samples from the distribution  $P_X$ . We can estimate this collision probability as follows:

$$\begin{aligned}
 &\Pr\left[\left((X_0 A_0)|_m \parallel A_0\right) = \left((X_1 A_1)|_m \parallel A_1\right)\right] \\
 &= \Pr[A_0 = A_1] \Pr\left[\left(X_0 A_0\right)|_m = \left(X_1 A_0\right)|_m\right] \\
 &\leq \Pr[A_0 = A_1] \left(\Pr[X_0 = X_1] + \Pr\left[\left(X_0 A_0\right)|_m = \left(X_1 A_0\right)|_m \mid X_0 \neq X_1\right]\right) \\
 &\leq \frac{1}{2^n} \left(\frac{1}{2^{m+2\log(1/\varepsilon)}} + \frac{1}{2^m}\right) = \frac{1 + \varepsilon^2}{2^n 2^m}, \tag{3}
 \end{aligned}$$

where we used (1) in the last inequality. We now insert (3) into (2) and get  $\Delta(VA, U) \leq \frac{\varepsilon}{2}$ .  $\square$

Using the usual definition of an extractor, the above lemma states that multiplying with a random element of  $\text{GF}(2^n)$  and then cutting off the last bits is a strong extractor. Consequently, we will sometimes use the notation  $\text{Ext}_m(x, a)$  to denote the function  $\text{Ext}_m(x, a) := (ax)|_m \parallel a$ , extracting  $\lfloor m \rfloor$  bits from  $x$ .

Further we use the following proposition on independent repetitions from [8], which is a quantitative version of the statement that for  $k$  independent repetitions of random variables, the min-entropy of the resulting concatenation is roughly  $k$  times the (Shannon-)entropy of a single instance (assuming  $k$  large enough and tolerating a small probability that something improbable occurred). A similar lemma with slightly weaker parameters is given in [10] (the latter would be sufficient for our application, but the expression from [8] is easier to use).

**Proposition 1.** *Let  $(X_1, Y_1), \dots, (X_k, Y_k)$  i.i.d. according to  $P_{XY}$ . For any  $\varepsilon$  there exists a distribution  $P_{\bar{X}\bar{Y}}$  which has statistical distance at most  $\frac{\varepsilon}{2}$  from  $(X_1, \dots, X_k, Y_1, \dots, Y_k)$  and satisfies*

$$H_\infty(\bar{X}|\bar{Y}) \geq kH(X|Y) - 6\sqrt{k \log(1/\varepsilon)} \log(|\mathcal{X}|).$$

We can combine the above propositions as follows:

**Lemma 2.** *Let  $k, \varepsilon$  with  $k > \log(1/\varepsilon)$  be given. Let  $(X_1, Y_1), \dots, (X_k, Y_k)$  i.i.d. according to  $P_{XY}$  over  $\mathcal{X} \times \mathcal{Y}$  with  $\mathcal{X} \subseteq \{0, 1\}^n$ . Let  $A$  be uniform over  $\{0, 1\}^{kn}$ . Then,*

$$\text{Ext}_{kH(X|Y) - 8 \log(|\mathcal{X}|) \sqrt{k \log(1/\varepsilon)}}(X_1 \parallel \dots \parallel X_k, A) \parallel Y_1 \parallel \dots \parallel Y_k$$

*is  $\varepsilon$ -close to  $U \times Y^k$ , where  $U$  is an independent uniform chosen bitstring of length  $\lfloor kH(X|Y) - 8 \log(|\mathcal{X}|) \sqrt{k \log(1/\varepsilon)} \rfloor + kn$ .*

*Proof.* Combine Lemma 1 and Proposition 1. □

## 4 A Pseudo-Entropy Pair from Any One-Way Function

The basic building block we use to get a pseudo-entropy pair is the following theorem by Goldreich and Levin [4] (recall that  $x \odot r = x_1 r_1 \oplus \dots \oplus x_n r_n$  is the inner product of  $x$  and  $r$ ):

**Proposition 2 (Goldreich-Levin).** *There is an oracle algorithm  $B^{(\cdot)}$  such that for any  $x \in \{0, 1\}^n$  and any oracle  $A$  satisfying*

$$\Pr_{r \leftarrow_R \{0, 1\}^n} [A(r) = x \odot r] \geq \frac{1}{2} + \gamma$$

*$B^A$  does  $\mathcal{O}(\frac{n}{\gamma^2})$  queries to  $A$  and then efficiently outputs a list of  $\mathcal{O}(\frac{1}{\gamma^2})$  elements such that  $x$  is in the list with probability  $\frac{1}{2}$ .*

This proposition implies that for any one-way function  $f$ , no efficient algorithm will be able to predict  $x \odot r$  from  $f(x)$  and  $r$  much better than random guessing, as otherwise the one-way function can be broken.

This suggests the following method to get a pseudo-entropy pair: if we define  $g(x, r) := f(x) \parallel r$  and  $P(x, r) := x \odot r$ , then predicting  $P(x, r)$  from  $g(x, r)$  is computationally hard. The problem with this approach is that since  $f(x)$  may have many different preimages,  $H(P(X, R) | g(X, R)) \approx 1$  is possible. In this case,  $P(x, r)$  would not only be *computationally* unpredictable, but also *information theoretically* unpredictable, and thus  $(g, P)$  will not be a pseudo-entropy pair.

The solution of this problem (as given in [6]), is that one additionally extracts some information of the input  $x$  to  $f$ ; the amount of information extracted is also random. The idea is that in case one is lucky and extracts roughly  $\log(|f^{-1}(f(x))|)$  bits, then these extracted bits and  $f(x)$  fix  $x$  in an information theoretic way, but computationally  $x \odot r$  is still hard to predict because of Proposition 2.

Thus, we define functions  $g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^{m+4n}$  and  $P : \{0, 1\}^{4n} \rightarrow \{0, 1\}$  as follows (where  $i \in [n]$  is a number<sup>3</sup>,  $x, a$ , and  $r$  are bitstrings, and we ignore padding which should be used to get  $(ax)_i$  to length  $n$ )

<sup>3</sup> Technically, we should choose  $i$  as a uniform number from  $[n]$ . We can use an  $n$  bit string to choose a uniform number from  $[2^n]$  and from this we can get an “almost” uniform number from  $[n]$  (for example by computing the remainder when dividing by  $n$ ). This only gives an exponentially small error which we ignore from now on.

$$g(x, i, a, r) := f(x) \parallel i \parallel a \parallel (ax)|_i \parallel r \tag{4}$$

$$P(x, i, a, r) := x \odot r. \tag{5}$$

We will alternatively write  $g(w)$  and  $P(w)$ , i.e., we use  $w$  as an abbreviation for  $(x, i, a, r)$ . We will prove that  $(g, P)$  is a pseudo-entropy pair in case  $f$  is a one-way function. Thus we show that no algorithm exceeds advantage  $1 - H(P(W)|g(W)) - \phi$  in predicting  $P(w)$  from  $g(w)$  (the gap  $\phi$  does not appear in the construction, but the pair will have a bigger gap if the one-way function satisfies as stronger security requirement, as we will see).

We first give an estimate on  $H(P(W)|g(W))$ . The idea is that we can distinguish two cases: either  $i \geq \log(|f^{-1}(f(x))|)$ , in which case  $H(P(W)|g(W)) \approx 0$ , since  $(ax)|_i$ ,  $a$ , and  $f(x)$  roughly fix  $x$ , or  $i < \log(|f^{-1}(f(x))|)$ , in which case  $H(P(W)|g(W)) \approx 1$ .

**Lemma 3.** *For the functions  $g$  and  $P$  as defined above*

$$H(P(W)|g(W)) \leq \frac{E_{x \leftarrow_R \{0,1\}^n} [\log(|f^{-1}(f(x))|)] + 2}{n}$$

*Proof.* From (1) and the union bound we see that if  $i > \log(|f^{-1}(y)|)$  the probability that  $x$  is not determined by the output of  $g$  is at most  $2^{-(i - \log(|f^{-1}(y)|))}$ . This implies  $H(P(W)|g(W), f(X) = y, I = i) \leq 2^{-(i - \log(|f^{-1}(y)|))}$ , and thus

$$\begin{aligned} H(P(W)|g(W)) &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} H(P(W)|g(W), f(X) = f(x)) \\ &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \frac{1}{n} \sum_{i=1}^n H(P(W)|g(W), f(X) = f(x), I = i) \\ &\leq \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left( \frac{\log(|f^{-1}(f(x))|)}{n} \right. \\ &\quad \left. + \frac{1}{n} \sum_{i=\lceil \log(|f^{-1}(f(x))|)}^n 2^{-(i - \log(|f^{-1}(f(x))|))} \right) \\ &\leq \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \frac{\log(|f^{-1}(f(x))|) + 2}{n} \\ &= \frac{E_{x \leftarrow_R \{0,1\}^n} [\log(|f^{-1}(f(x))|)] + 2}{n}. \end{aligned}$$

□

We can now show that  $(g, P)$  is a pseudo-entropy pair. For this, we show that any algorithm which predicts  $P$  from  $g$  with sufficient probability can be used to invert  $f$ . Recall that  $\phi$  is usually  $\frac{1}{n}$ .

**Lemma 4.** *Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  and  $\phi : \mathbb{N} \rightarrow [0, 1]$  be computable in time  $\text{poly}(n)$ . Let functions  $g$  and  $P$  be as defined above. There exists an oracle*

algorithm  $B^{(\cdot)}$  such that, for any  $A$  which has advantage  $\text{Adv}^A(g^f, P^f) \geq 1 - H(P^f(W)|g^f(W)) - \phi$  in predicting  $P^f$  from  $g^f$ ,  $B^A$  inverts  $f$  with probability  $\Omega(\frac{1}{n^3})2^{-n\phi}$  and  $\mathcal{O}(n^3)$  calls to  $A$ .

We find it convenient to present our proof using random experiments called “games”, similar to the method presented in [14].

*Proof.* Assume that a given algorithm  $A(y, i, a, z, r)$  has an advantage exceeding the bound in the lemma in predicting  $P$  from  $g$ . To invert a given input  $y = f(x)$ , we will choose  $i, a,$  and  $z$  uniformly at random. Then we run the Goldreich-Levin algorithm using  $A(y, i, a, z, \cdot)$ , i.e., the Goldreich-Levin calls  $A(y, i, a, z, r)$  for many different  $r$ , but always using the same  $y, i, a,$  and  $z$ . This gives us a list  $L$  containing elements from  $\{0, 1\}^n$ . For every  $\bar{x} \in L$  we check whether  $f(\bar{x}) = y$ . If at least one  $\bar{x} \in L$  satisfies this we succeeded in inverting  $f$ .

In order to see whether this approach is successful, we first define  $\alpha$  to be the advantage of  $A$  for a fixed  $y, i, a$  and  $z$  in predicting  $\bar{x} \odot r$  for a preimage  $\bar{x}$  of  $y$ :

$$\alpha(y, i, a, z) := \max_{\bar{x} \in f^{-1}(y)} \left( 2 \Pr_{r \leftarrow \{0,1\}^n} [A(y, i, a, z, r) = \bar{x} \odot r] - 1 \right).$$

We maximize over all possible  $\bar{x} \in f^{-1}(y)$ , since it is sufficient if the above method finds *any* preimage of  $y$ . We will set the parameters of the algorithm such that it succeeds with probability  $\frac{1}{2}$  if  $\alpha(y, i, a, z) > \frac{1}{4n}$  (i.e., with probability  $\frac{1}{2}$  the list returned by the algorithm contains  $\bar{x}$ ). It is thus sufficient to show for uniformly chosen  $x, i, a,$  and  $z$  the inequality  $\alpha(f(x), i, a, z) > \frac{1}{4n}$  is satisfied with probability  $\Omega(\frac{1}{n^3})2^{-n\phi}$ .

Together with Lemma 3, the requirement of this lemma implies that in the following Game 0 the expectation of the output is at least  $1 - H(P^f(W)|g^f(W)) - \phi \geq 1 - \frac{1}{n} \mathbb{E}_x[\log(|f^{-1}(f(x))|)] - \frac{2}{n} - \phi$  (this holds even *without* the maximization in the definition of  $\alpha$  and using  $\bar{x} = x$  instead – clearly, the maximization cannot reduce the expected output of Game 0).

**Game 0:**

$x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$   
 $a \leftarrow_R \{0, 1\}^n, z := (ax)|_i$   
**output**  $\alpha(y, i, a, z)$

Note that even though we can approximate  $\alpha(y, i, a, z)$  we do not know how to compute the exact value in reasonable time. However, we do not worry about finding an efficient implementation of our games.

If  $i$  is much larger than  $\log(|f^{-1}(y)|)$  then predicting  $P(w)$  from  $g(w)$  is not very useful in order to invert  $f$ , since  $(ax)|_i$  gives much information about  $x$  which we do not have if we try to invert  $y$ . Thus, we ignore the cases where  $i$  is much larger than  $\log(|f^{-1}(y)|)$  in Game 1.

**Game 1:**

$x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$   
**if**  $i \leq \log(|f^{-1}(y)|) + n\phi + 3$  **then**  
 $a \leftarrow_R \{0, 1\}^n, z := (ax)|_i$

```

output  $\alpha(y, i, a, z)$ 
fi
output 0
    
```

It is not so hard to see that the probability that the if clause fails is at most  $1 - \frac{1}{n} E_x[\log(|f^{-1}(f(x))|)] - \frac{3}{n} - \phi$ . Thus, in Game 1 the expectation of the output is at least  $\frac{1}{n}$  (because the output only decreases in case the if clause fails, and in this case by at most one).

In Game 2, we only choose the first  $j$  bits of  $z$  as above, where  $j$  is chosen such that these bits will be  $\frac{1}{4n}$ -close to uniform (this will be used later). We fill up the rest of  $z$  with the best possible choice; clearly, this cannot decrease the expectation of the output.

**Game 2:**

```

 $x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$ 
if  $i \leq \log(|f^{-1}(y)|) + n\phi + 3$  then
     $j := \min(\lfloor \log(|f^{-1}(y)|) \rfloor - 2 \log(4n), i)$ 
     $a \leftarrow_R \{0, 1\}^n, z_1 := (ax)|_j$ 
    set  $z_2 \in \{0, 1\}^{j-i}$  such that  $\alpha(y, i, a, z_1 \| z_2)$  is maximal
    output  $\alpha(y, i, a, z_1 \| z_2)$ 
fi
output 0
    
```

We now chose  $z_1$  uniformly at random. Lemma 1 implies that the statistical distance of the previous distribution of  $z_1$  to the uniform distribution (given  $a, i$ , and  $y$  but not  $x$ ) is at most  $\frac{1}{4n}$ . Thus, the expectation of the output is at least  $\frac{1}{2n}$ .

**Game 3:**

```

 $x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$ 
if  $i \leq \log(|f^{-1}(y)|) + n\phi + 3$  then
     $j := \min(\lfloor \log(|f^{-1}(y)|) \rfloor - 2 \log(4n), i)$ 
     $a \leftarrow_R \{0, 1\}^n, z_1 \leftarrow_R \{0, 1\}^j$ 
    set  $z_2 \in \{0, 1\}^{j-i}$  such that  $\alpha(y, i, a, z_1 \| z_2)$  is maximal
    output  $\alpha(y, i, a, z_1 \| z_2)$ 
fi
output 0
    
```

As mentioned above, we will be satisfied if we have values  $y, i, a, (z_1 \| z_2)$  such that  $\alpha(y, i, a, z_1 \| z_2) \geq \frac{1}{4n}$ . In Game 4, we thus do not compute the expectation of  $\alpha$  anymore, but only output success if this is satisfied, and fail otherwise.

**Game 4:**

```

 $x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$ 
if  $i \leq \log(|f^{-1}(y)|) + n\phi + 3$  then
     $j := \min(\lfloor \log(|f^{-1}(y)|) \rfloor - 2 \log(4n), i)$ 
     $a \leftarrow_R \{0, 1\}^n, z_1 \leftarrow_R \{0, 1\}^j$ 
    set  $z_2 \in \{0, 1\}^{j-i}$  such that  $\alpha(y, i, a, z_1 \| z_2)$  is maximal
    if  $\alpha(y, i, a, z_1 \| z_2) > \frac{1}{4n}$ 
        output success
    
```

fi  
fi  
output fail

The usual Markov style argument shows that the probability that the output is success is at least  $\frac{1}{4n}$  (this is easiest seen by assuming otherwise and computing an upper bound on the expectation of the output in Game 3: it would be less than  $\frac{1}{2n}$ ).

In Game 5, we choose all of  $z$  uniformly at random.

**Game 5:**  
 $x \leftarrow_R \{0, 1\}^n, y := f(x), i \leftarrow_R [n]$   
**if**  $i \leq \log(|f^{-1}(y)|) + n\phi + 3$  **then**  
 $a \leftarrow_R \{0, 1\}^n, z \leftarrow_R \{0, 1\}^i$   
**if**  $\alpha(y, i, a, z) > \frac{1}{4n}$   
**output success**  
**fi**  
**fi**  
**output fail**

In Game 5, we can assume that  $z$  is still chosen as  $z_1 || z_2$ . For  $z_1$ , the distribution is the same as in Game 4, for  $z_2$ , we hope that we are lucky and choose it exactly as in Game 4. The length of  $z_2$  is at most  $2 \log(4n) + n\phi + 3$ , and thus this happens with probability at least  $\frac{1}{128n^2} 2^{-n\phi}$ . Thus, in Game 4, with probability at least  $\frac{1}{512n^3} 2^{-n\phi}$  the output is success. As mentioned at the start of the proof, in this case running the Goldreich-Levin algorithm with parameter  $\frac{1}{4n}$  will invert  $f$  with probability  $\frac{1}{2}$ , which means that in total we have probability  $\Omega(\frac{1}{n^3}) 2^{-n\phi}$  in inverting  $f$ . □

## 5 A Pseudorandom Generator from a Pseudo-Entropy Pair

We now show how we can obtain a pseudorandom generator from a pseudo-entropy pair  $(g, P)$  as constructed in the last section. The idea here is that we use many (say  $k$ ) parallel copies of the function  $g$ . We can then extract about  $kH(g(W))$  bits from the concatenated outputs of  $g$ , about  $kH(W|g(W)P(W))$  bits from the concatenated inputs, and about  $k(H(P(W)|g(W)) + \phi)$  bits from the concatenated outputs of  $P$ . Using the identity  $H(g(W)) + H(P(W)|g(W)) + H(W|g(W)P(W)) = H(W)$ , we can see that this will be expanding, and we can say that the  $k\phi$  bits of pseudorandomness from  $P$  are used to get the expanding property of  $h$ .

The key lemma in order to prove the security of the construction is the following variant of Impagliazzo’s hard-core lemma [9] proven in [7]<sup>4</sup>. For a set  $\mathcal{T}$

---

<sup>4</sup> The proposition here is slightly stronger than the corresponding lemma in [7], as we do not require  $\gamma$  to be noticeable. It is easy to see that the proof in [7] works in this case as well.

let  $\chi_{\mathcal{T}}$  be the characteristic function of  $\mathcal{T}$ :

$$\chi_{\mathcal{T}}(x) := \begin{cases} 1 & x \in \mathcal{T} \\ 0 & x \notin \mathcal{T}. \end{cases}$$

**Proposition 3 (Uniform Hard-Core Lemma).** *Assume that the given functions  $g : \{0, 1\}^n \rightarrow \{0, 1\}^m$ ,  $P : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $\delta : \mathbb{N} \rightarrow [0, 1]$  and  $\gamma : \mathbb{N} \rightarrow [0, 1]$  are computable in time  $\text{poly}(n)$ , where  $\delta$  is noticeable and  $\gamma > 2^{-n/3}$ .*

*Further, assume that there exists an oracle algorithm  $A^{(\cdot)}$  such that, for infinitely many  $n$ , the following holds: for any set  $\mathcal{T} \subseteq \{0, 1\}^n$  with  $|\mathcal{T}| \geq \delta 2^n$ ,  $A^{\chi_{\mathcal{T}}}$  outputs a circuit  $C$  satisfying*

$$\mathbb{E} \left[ \Pr_{x \leftarrow_{RT}} [C(g(x)) = P(x)] \right] \geq \frac{1 + \gamma}{2}$$

(where the expectation is over the randomness of  $A$ ).

*Then, there is an algorithm  $B$  which calls  $A$  as a black box  $\text{poly}(\frac{1}{\gamma}, n)$  times, such that*

$$\text{Adv}^B(g, P) \geq 1 - \delta$$

*for infinitely many  $n$ . The runtime of  $B$  is bounded by  $\text{poly}(\frac{1}{\gamma}, n)$  times the runtime of  $A$ .*

The advantage of using Proposition 3 is as follows: in order to get a contradiction, we will use a given algorithm  $A$  as oracle to contradict the hardness of a pseudo-entropy pair, i.e., we will give  $B$  such that  $\text{Adv}^B(g, P) \geq 1 - H(P(W)|g(W)) - \phi$ . Proposition 3 states that for this it is sufficient to show how to get circuits which perform slightly better than random guessing on a fixed set of size  $2^n(H(P(W)|g(W)) + \phi)$ , given access to a description of this set. Often, this is a much simpler task.

In the following construction of a pseudorandom generator from a pseudo-entropy pair we assume that parameters  $\varepsilon$  and  $\phi$  are provided (thus they reappear in Theorem 1). The parameter  $\varepsilon$  describes how much we lose in the indistinguishability (by making our extractors imperfect), while  $\phi$  is the gap of the pseudo-entropy pair.

Further we assume that parameters  $\alpha$  and  $\beta$  are known which give certain information about the combinatorial structure of the given predicate. We will get rid of this assumption later by trying multiple values for  $\alpha$  and  $\beta$  such that one of them must be correct.<sup>5</sup>

**Lemma 5.** *Let  $g$  and  $P$  be efficiently evaluable functions,  $g : \{0, 1\}^n \rightarrow \{0, 1\}^m$ ,  $P : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $\varepsilon : [0, 1] \rightarrow \mathbb{N}$ , and  $\phi : [0, 1] \rightarrow \mathbb{N}$  be computable in*

<sup>5</sup> Haitner, Harnik, and Reingold [5] construct a pseudo-entropy pair for which  $H(P(W)|g(W)) = \frac{1}{2}$  is fixed. Because of this, they are able to save a factor of  $n$  in the seed length under standard assumptions (they do not need to try different values for  $\alpha$ ).

polynomial time,  $\phi > \frac{1}{n}$ . Assume that parameters  $\alpha$  and  $\beta$  are such that

$$\begin{aligned} \alpha &\leq H(P(W)|g(W)) \leq \alpha + \phi/4 \\ \beta &\leq H(g(W)) \leq \beta + \phi/4. \end{aligned}$$

There is an efficient to evaluate oracle function  $h_{\alpha,\beta,\varepsilon,\phi}^g$  with the following properties:

- $h_{\alpha,\beta,\varepsilon,\phi}^g$  is expanding,
- $h_{\alpha,\beta,\varepsilon,\phi}^g$  has inputs of length  $\mathcal{O}(n^3 \frac{1}{\phi^2} \log(\frac{1}{\varepsilon}))$ , and
- any algorithm  $A$  which distinguishes the output of  $h_{\alpha,\beta,\varepsilon,\phi}^g$  from a uniform bit string with advantage  $\gamma$  can be used to get an oracle algorithm  $B^A$  satisfying  $\text{Adv}^B(g, P) \geq 1 - H(P(W)|g(W)) - \phi$  which does  $\text{poly}(\frac{1}{\gamma-\varepsilon}, n)$  calls to  $A$ .

*Proof.* Let  $k := 4096 \cdot (\frac{n}{\phi})^2 \cdot \log(\frac{3}{\varepsilon})$  be the number of repetitions (this is chosen such that

$$\frac{k\phi}{8} = 512 \frac{n^2}{\phi} \log\left(\frac{3}{\varepsilon}\right) = 8n\sqrt{k \log\left(\frac{3}{\varepsilon}\right)}, \tag{6}$$

which we use later). To simplify notation we set  $\lambda := n - \alpha - \beta - \phi/2$ . Using the notation  $w^k := w_1 \parallel \dots \parallel w_k$ ,  $g^{(k)}(w^k) := g(w_1) \parallel \dots \parallel g(w_k)$  and  $P^{(k)}(w^k) := P(w_1) \parallel \dots \parallel P(w_k)$ , the function  $h_{\alpha,\beta,\varepsilon,\phi}$  is defined as

$$\begin{aligned} h_{\alpha,\beta,\varepsilon,\phi}(w^k, s_1, s_2, s_3) &:= \\ &\text{Ext}_{k(\beta-\phi/8)}(g^{(k)}(w^k), s_1) \parallel \text{Ext}_{k(\alpha+7\phi/8)}(P^{(k)}(w^k), s_2) \parallel \text{Ext}_{k(\lambda-\phi/8)}(w^k, s_3). \end{aligned}$$

Clearly, the input length is  $\mathcal{O}(n^3 \frac{1}{\phi^2} \log(\frac{1}{\varepsilon}))$ . We further see by inspection that, excluding the additional randomness  $s_1, s_2$ , and  $s_3$ , the function  $h$  maps  $kn$  bits to at least  $k(\alpha + \beta + \lambda) + 5k\frac{\phi}{8} - 3 = k(n - \frac{\phi}{2}) + k\frac{5\phi}{8} - 3 = k(n + \frac{\phi}{8}) - 3 > kn$  bits. Since the additional randomness is also completely contained in the output,  $h_{\alpha,\beta,\varepsilon,\phi}$  is expanding for almost all  $n$ .

We now show that an algorithm  $A$  which has advantage  $\gamma$  in distinguishing  $h_{\alpha,\beta,\varepsilon,\phi}(w^k, s_1, s_2, s_3)$  from a uniform bit string of the same length can be used to predict  $P(w)$  given  $g(w)$  as claimed above. Per definition the probability that the output is true in the following game is at least  $\frac{1+\gamma}{2}$ .

**Game 0:**

```

(w1, ..., wk) ←R {0, 1}nk
b ←R {0, 1}
if b = 0 then                                     (Run A with the output of h)
    s1 ←R {0, 1}mk, v1 := Extk(β-φ/8)(g(k)(wk), s1)
    s2 ←R {0, 1}k, v2 := Extk(α+7φ/8)(P(k)(wk), s2)
    s3 ←R {0, 1}nk, v3 := Extk(λ-φ/8)(wk, s3)
else                                               (Run A with uniform randomness)
    v1 ←R {0, 1}mk+k(β-φ/8)
    
```

```

v2 ←R {0, 1}k+k(α+7φ/8)
v3 ←R {0, 1}nk+k(λ-φ/8)
fi
output b = A(v1||v2||v3)

```

We now make two transition based on statistical indistinguishability. First, we replace the last part  $v_3$  in the *if*-clause of Game 0 with uniform random bits. Because  $H(W|g(W)P(W)) = H(W) - H(g(W)) - H(P(W)|g(W)) \geq n - \alpha - \beta - \frac{\phi}{2} = \lambda$ , Lemma 2 implies that conditioned on the output of  $g^{(k)}$  and  $P^{(k)}$  (and thus also conditioned on the extracted bits of those outputs)  $\text{Ext}_{k\lambda-k\phi/8}(w^k, s_3) = \text{Ext}_{k\lambda-8n\cdot\sqrt{k\log(\frac{3}{\varepsilon})}}(w^k, s_3)$  is  $\frac{\varepsilon}{3}$ -close to the uniform distribution (here we used (6)). Thus this only loses  $\varepsilon/3$  of the advantage  $\gamma$  in distinguishing.

Second, we replace  $v_1$  in the *else*-clause with  $\text{Ext}_{k(\beta-\phi/8)}(g^{(k)}(w^k), s_1)$ . Since  $H(g(W)) \geq \beta$ , Lemma 2 implies that we only lose  $\varepsilon/3$  in the advantage again. In total, in the following Game 1 we have advantage at least  $\gamma - 2\varepsilon/3$  over random guessing.

**Game 1:**

```

(w1, ..., wk) ←R {0, 1}nk
b ←R {0, 1}
if b = 0 then
    s1 ←R {0, 1}mk, v1 := Extk(β-φ/8)(g(k)(wk), s1)
    s2 ←R {0, 1}k, v2 := Extk(α+7φ/8)(P(k)(wk), s2)
    v3 ←R {0, 1}nk+k(λ-φ/8)
else
    s1 ←R {0, 1}mk, v1 := Extk(β-φ/8)(g(k)(wk), s1)
    v2 ←R {0, 1}k+k(α+7φ/8)
    v3 ←R {0, 1}nk+k(λ-φ/8)
fi
output b = A(v1||v2||v3)

```

We would like to ignore the parts which are the same in case  $b = 0$  and  $b = 1$ . It is easy to see that  $A'$  in Game 2 can be designed such that it calls  $A$  with the same distribution as in Game 1.

**Game 2:**

```

(w1, ..., wk) ←R {0, 1}nk
b ←R {0, 1}
if b = 0 then
    s ←R {0, 1}k, v := Extk(α+7φ/8)(P(k)(wk), s)
else
    v ←R {0, 1}k+k(α+7φ/8)
fi
output b = A'(g(k)(wk)||v)

```

Later we want to use Proposition 3. Thus we will have an oracle  $\chi_{\mathcal{T}}$  which implements the characteristic function of a set  $\mathcal{T}$  of size at least  $(\alpha + \phi)2^n$ . From now on we will use the oracle implicitly in the games by testing whether  $w \in \mathcal{T}$ .

In Game 3 it is easy to check that in case  $b = 0$  the distribution with which  $A'$  is called does not change from Game 2. On the other hand, if  $b = 1$ , then (since  $|\mathcal{T}| \geq 2^n(\alpha + \phi)$ ) the  $p_i$  contain independent random variables with entropy at least  $\alpha + \phi$  (where the entropy is conditioned on  $g(w_i)$ ). Using Lemma 2 we see that in this case  $v$  is  $\frac{\varepsilon}{3}$ -close to uniform, implying that in Game 3 the advantage of  $A'$  in predicting  $b$  is still  $\gamma - \varepsilon$ .

**Game 3:**

```

 $(w_1, \dots, w_k) \leftarrow_R \{0, 1\}^{nk}$ 
 $b \leftarrow_R \{0, 1\}$ 
for  $i \in [n]$  do
  if  $w_i \in \mathcal{T} \wedge b = 1$  then
     $p_i \leftarrow_R \{0, 1\}$ 
  else
     $p_i := P(w_i)$ 
  fi
od
 $s \leftarrow_R \{0, 1\}^k, v := \text{Ext}_{k(\alpha+7\phi/8)}(p^k, s)$ 
output  $b = A'(g^{(k)}(w^{(k)})||v)$ 

```

From Game 3, we will now apply a standard hybrid argument to get a predictor for a single position. For this, consider Game 4.

**Game 4:**

```

 $(w_1, \dots, w_k) \leftarrow_R \{0, 1\}^{nk}$ 
 $j \leftarrow_R [n]$ 
for  $i \in \{1, \dots, j-1\}$  do
  if  $w_i \in \mathcal{T}$  then  $p_i \leftarrow_R \{0, 1\}$  else  $p_i := P(w_i)$  fi
od
for  $i \in \{j+1, \dots, n\}$  do  $p_i := P(w_i)$  od
 $b \leftarrow_R \{0, 1\}$ 
if  $w_j \in \mathcal{T} \wedge b = 1$  then  $p_j \leftarrow_R \{0, 1\}$  else  $p_j := P(w_j)$  fi
 $s \leftarrow_R \{0, 1\}^k, v := \text{Ext}_{k(\alpha+7\phi/8)}(p^k, s)$ 
output  $b = A'(g^{(k)}(w^{(k)})||v)$ 

```

The distribution  $A'$  is called in Game 4 in case  $b = 0$  and  $j = 1$  is the same as in Game 3 in case  $b = 0$ ; the distribution used in Game 4 in case  $b = 1$  and  $j = n$  is the same as in Game 3, in case  $b = 1$ . Further, the distribution in Game 4 does not change if  $b$  is set from 1 to 0 and  $j$  is increased by one. This implies that the advantage of  $A'$  in predicting  $b$  is  $(\gamma - \varepsilon)/k$ .

In Game 5, we replace  $A'$  with  $A''$  which does all the operations common in case  $b = 0$  and  $b = 1$  (the  $w$  chosen in Game 5 corresponds to  $w_j$  in Game 4, and  $A''$  chooses the value of  $j$ , and all other  $w_i$  before calling  $A'$ ).

**Game 5:**

```

 $w \leftarrow_R \{0, 1\}^n$ 
 $b \leftarrow_R \{0, 1\}$ 
if  $w \in \mathcal{T} \wedge b = 1$  then
     $p \leftarrow_R \{0, 1\}$ 
    output  $A''(g(w)||p) = b$ 
else
    output  $A''(g(w)||P(w)) = b$ 
fi
    
```

An easy calculation now yields that for  $w \leftarrow_R \mathcal{T}$  and  $p \leftarrow_R \{0, 1\}$  the probability that

$$1 \oplus p \oplus A''(g(w)||p) = P(w)$$

is at least  $\frac{1}{2} + \frac{\gamma - \varepsilon}{k}$ . Since this works for any  $\mathcal{T}$  with  $|\mathcal{T}| \geq (\alpha + \phi)2^n$ , and thus for every  $\mathcal{T}$  with  $|\mathcal{T}| \geq (H(P(W)|g(W)) + \phi)2^n$ , we can apply Proposition 3 and get the lemma. □

With this lemma, we can now prove Theorem 1.

*Proof (of Theorem 1).* Given  $\varepsilon$  and  $\phi$ , we use the construction of Lemma 4 to get a predicate which we use in the construction of Lemma 5 for  $\frac{16n}{\phi^2}$  different values of  $\alpha$  and  $\beta$  (note that  $0 \leq H(g(W)) \leq n$ ), such that for at least one of those choices the requirements of Lemma 5 hold. Further, in those applications we use  $\varepsilon' := \Omega(\frac{\phi^4}{n^5})$  in place of  $\varepsilon$ . Since  $\varepsilon' = \Omega(\varepsilon^{10})$ , this satisfies  $\mathcal{O}(\log(\frac{1}{\varepsilon})) = \mathcal{O}(\log(\frac{1}{\varepsilon'}))$ .

For every choice of  $\alpha$  and  $\beta$  we concatenate  $h_{\alpha, \beta, \varepsilon', \phi} : \{0, 1\}^\ell \rightarrow \{0, 1\}^{\ell+1}$  with itself, in order to obtain a function  $h'_{\alpha, \beta, \varepsilon', \phi} : \{0, 1\}^\ell \rightarrow \{0, 1\}^{16n\phi^{-2}\ell+1}$ , i.e., the first part of the output of  $h_{\alpha, \beta, \varepsilon', \phi}$  is used to call  $h_{\alpha, \beta, \varepsilon', \phi}$  again, and this process is repeated  $16n\phi^{-2}\ell \in \mathcal{O}(n^5 \frac{1}{\phi^4})$  times, and every time we get one more bit of the final output.

The function  $h_{\varepsilon, \phi} : \{0, 1\}^{16n\phi^{-2}\ell} \rightarrow \{0, 1\}^{16n\phi^{-2}\ell+1}$  divides its input into  $\frac{16n}{\phi^2}$  blocks of length  $\ell$ , calls the functions  $h'_{\alpha, \beta, \varepsilon', \phi}$  with separate blocks, and XORs the outputs.

Assume now that an algorithm  $A$  can distinguish the output of  $h_{\varepsilon, \phi}$  from a uniform random string with advantage  $\gamma$ . For every choice of  $\alpha$  and  $\beta$  (and in particular the choice which satisfies the requirements of Lemma 5) we try the following to invert  $f$ . First, since we can simulate the other instances, we see that we have advantage  $\gamma$  in distinguishing the output of  $h'_{\alpha, \beta, \varepsilon', \phi}$  from a random string. We can use the hybrid argument to get an algorithm which has advantage  $\gamma' := \Omega(\gamma\phi^4 n^{-5})$  in distinguishing the output of  $h_{\alpha, \beta, \varepsilon', \phi}$  from a random string. From Lemma 5 we get an algorithm which predicts  $P$  from  $g$  with advantage at least  $1 - H(P(W)|g(W)) - \phi$ , and the number of calls is bounded by  $\text{poly}(\frac{1}{\gamma' - \varepsilon'}, n) = \text{poly}(\frac{1}{\gamma - \varepsilon}, n)$ . Finally, Lemma 4 implies that we can get an inverter with the claimed complexity and success probability. □

## Acknowledgments

I would like to thank Ueli Maurer, Krzysztof Pietrzak, Dominik Raub, Renato Renner, Johan Sjödin, and Stefano Tessaro for helpful comments and discussions. Also, I would like to thank the anonymous referees who provided helpful criticism about the presentation of this paper. I was supported by the Swiss National Science Foundation, project no. 200020-103847/1.

## References

1. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *Siam Journal on Computation*, 13(4):850–864, 1984.
2. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.
3. Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudo-random generators. *Siam Journal on Computation*, 22(6):1163–1175, 1993.
4. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.
5. Iftach Haitner, Danny Harnik, and Omer Reingold. On the power of the randomized iterate. Technical Report TR05-135, Electronic Colloquium on Computational Complexity (ECCC), 2005.
6. Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *Siam Journal on Computation*, 28(4):1364–1396, 1999.
7. Thomas Holenstein. Key agreement from weak bit agreement. In *Proceedings of the Thirty Seventh Annual ACM Symposium on Theory of Computing*, pages 664–673, 2005.
8. Thomas Holenstein and Renato Renner. On the smooth Rényi entropy of independently repeated random experiments. In preparation, 2005.
9. Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *The 36th Annual Symposium on Foundations of Computer Science*, pages 538–545, 1995.
10. Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstract). In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 12–24, 1989.
11. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *Siam Journal on Computation*, 17(2):373–386, 1988.
12. Michael Luby and Avi Wigderson. Pairwise independence and derandomization. Technical Report ICSI TR-95-035, International Computer Science Institute, Berkeley, CA, 1995.
13. Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
14. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Technical Report 332, <http://eprint.iacr.org/2004/332>, 2004.

15. Hoeteck Wee. On obfuscating point functions. In *Proceedings of the Thirty Seventh Annual ACM Symposium on Theory of Computing*, pages 523–532, 2005.
16. Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *The 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, 1982.