J. Cryptology (2004) 17: 209–229
DOI: 10.1007/s00145-003-0141-4

Journal of
CRYPTOLOGY

Nearly One-Sided Tests and the Goldreich–Levin Predicate

Gustav Hast
Department of Numerical Analysis and Computer Science,
Royal Institute of Technology,
100 44 Stockholm, Sweden
ghast@nada.kth.se

Communicated by Oded Goldreich

Received 29 November 2001 and revised 23 December 2002
Online publication 10 July 2003

Abstract. We study statistical tests with binary output that rarely outputs one, which we call nearly one-sided statistical tests. We provide an efficient reduction establishing improved security for the Goldreich–Levin hard-core bit against nearly one-sided tests. The analysis is extended to prove the security of the Blum–Micali pseudo-random generator combined with the Goldreich–Levin bit. Finally, some applications where nearly one-sided tests occur naturally are discussed.

Key words. Nearly one-sided statistical test, Goldreich–Levin predicate, Pseudo-random generator, Provable security, List decoding of Hadamard code.

1. Introduction

Many algorithms are probabilistic and therefore require a source of randomness to be implemented correctly. This is true in particular for most cryptographic algorithms. Obtaining random material is often a hard and time consuming process and therefore it is convenient to use a pseudo-random generator to generate much random looking material from a short truly random seed. One would of course like to have a guarantee that by exchanging random material for the output of a generator, the performance of the algorithms are not changed in a harmful way.

The pioneering works of Blum and Micali [5] and Yao [15] laid the foundation of the theory of pseudo-randomness. Blum and Micali showed how to construct a pseudo-random bit generator (PRBG) whose security is based on the hardness of solving the discrete logarithm problem. More specifically they proved and used the fact that the most significant bit is a hard-core predicate for exponentiation. A predicate b is a hard-core predicate for a function g if it is not feasible to determine the boolean value of $b(x)$ efficiently given the value of $g(x)$. In [10] Goldreich and Levin showed how to construct a hard-core predicate for any one-way function. This construction can be applied on

the above-mentioned PRBG so that the security can be based on the one-wayness of an arbitrary permutation f. The proof of security in [5] was a polynomial reduction from solving the discrete logarithm problem (or if we use the result from [10] inverting f) to breaching the security of the bit generator.

In this work we analyze the security of the well-known pseudo-random generator obtained by combining the works [5] and [10]. We refer to this generator as *BMGL*. As noted in previous works (e.g., [9], [10] and [13]) the exact efficiency of a reduction between two different cryptographic primitives is of vital interest when determining the practical security consequences of the reduction. Examples of more recent works that deal with the issue to bridge the gap between theoretical complexity-based cryptography and practical cryptography by improved reductions and analysis are [3] and [6]. For a more extensive list see [2]. In the case of *BMGL*, the reduction relates the one-wayness of a permutation to the pseudo-randomness of the output from *BMGL*. A distribution is considered to be pseudo-random if there is no statistical test, from a specific set of admissible tests, that more than negligibly can distinguish between elements from that distribution and from the uniform distribution. (A distribution is in fact not considered to be pseudo-random, but instead an ensemble of distributions. For simplicity reasons we do not make this distinction throughout the introduction.) Normally the set of admissible tests is specified by a maximum running time. Improvements to the security reduction of *BMGL* and its analysis have been made earlier by Rackoff (explained in [7]), Levin [14] and Håstad and Näslund [12].

Earlier analyses of reductions have characterized the efficiency of a statistical test D, distinguishing the distributions X and Y, by using a measure δ such that

$$\left| \Pr_{x \in X}[D(x) = 1] - \Pr_{y \in Y}[D(y) = 1] \right| \geq \delta. \tag{1}$$

In this paper we consider nearly one-sided statistical tests which are tests that, on truly random input, almost always output zero and rarely output one. The measure in (1) does not capture whether or not a test is nearly one-sided and therefore we introduce the notion of a parameterized distinguisher, which for a test and a certain pair of input distributions imposes two thresholds, separating the corresponding output distributions of the test. We say that a test D (δ_1, δ_2)-distinguishes X and Y if

$$\Pr_{x \in X}[D(x) = 1] \leq \delta_1 < \delta_2 \leq \Pr_{y \in Y}[D(y) = 1].$$

Thus, if δ_1 is small and X is the uniform distribution, the test is considered to be nearly one-sided. (We do not formally define "small" but instead use the parameterized distinguisher to express formal results in this paper.) The use of this characterization of a distinguisher enables a more careful analysis of the reduction from inverting a permutation to distinguishing between the output of *BMGL* and the uniform distribution. In particular, if only nearly one-sided tests are considered to be admissible the new analysis obtains a substantial increase of the efficiency of the reduction. This limitation turns out to be quite natural in many applications. The notion of nearly one-sided statistical tests has previously been investigated in [4].

The outline of this paper is as follows: In Section 3 the Goldreich–Levin hard-core bit is explained and a motivating discussion is held on why low rate predictors are more

powerful than ordinary predictors when list decoding the Hadamard code. In the next section we prove a theorem about list decoding Hadamard codes with both erasures and errors and in Section 5 this theorem is used to establish the reduction from inverting a function to predicting the Goldreich–Levin bit. Section 6 discusses nearly one-sided statistical tests and their connection with low rate predictors and in Section 7 the security of the *BMGL* is shown. Applications are considered in Section 8, in particular we study the security consequences of the use of a PRBG in signature and encryption schemes. The paper is concluded with some open questions.

2. Notation

In this work we use the following notation:

1. By $[m]$ we mean the set $\{1, \ldots, m\}$ and $2^{[m]}$ is the set of all subsets of $[m]$.
2. The xor operation is denoted by $\oplus$.
3. The function $b(r, x)$ is the inner product of r and x modulo 2.
4. The ith unit vector e^i is a bit string containing only zeros, except for the ith bit (which is one). The dimension of e^i is implicitly given by its use.
5. We let $\langle J, L \rangle$, where J and L are sets, denote the size of $J \cap L$ modulo 2.
6. If S is a set we denote the size of the set by $|S|$. If x is a bit string, the length of x is denoted by $|x|$.
7. When the logarithmic function log is used without the base having been specified, it is implicit base 2.
8. The uniform distribution of bit strings of length n is denoted by U_n.

3. The Goldreich–Levin Bit and List Decoding of Hadamard Code

Goldreich and Levin showed in [10] how to modify an arbitrary one-way function to make it have a hard-core predicate: if f is a one-way function, then $b(r, x)$ (the inner product of r and x modulo 2) is a hard-core predicate for the one-way function $f'(r, x) = (r, f(x))$. This means that there is no efficient algorithm that given $(r, f(x))$ as input (where r and x are drawn from the uniform distribution) can guess the value of $b(r, x)$ significantly better than a random guess.

The above result is shown using a reduction from inverting f to predicting the value of $b(r, x)$. The efficiency of the reduction depends on how well the bit $b(r, x)$ is guessed, which usually is measured by the advantage $\varepsilon(n)$ of the guessing algorithm P, often called the predictor:

$$\varepsilon(n) = \Pr_{r,x \in U_n}[P(r, f(x)) = b(r, x)] - \tfrac{1}{2}.$$

The main part of the reduction consists of a list-decoding algorithm for the Hadamard code. The ith bit of the Hadamard code of x is exactly $b(i, x)$, where i is interpreted in the natural way as a bit string of the same length as x. The task for a list-decoding algorithm is to produce a list of possible x, having oracle access to a Hadamard code with a certain fraction of errors. The algorithm should come with a lower bound on the probability that x appears in the list output and an upper bound on the number of oracle

queries made. Given the value of $f(x)$, the predictor P corresponds in a natural way to the oracle of the Hadamard code, and the advantage of P (over a fix x) is closely related to the number of errors of the oracle.

Now suppose that we have a predictor P that on some input answers with high confidence and on other inputs just flips a coin. The informative answers from P (when it does not just flip a coin) would then be somewhat clouded by the random noise provided by the other answers. We therefore give the predictor more freedom by also letting it output $\perp$ in those cases when the confidence in the prediction is low. Instead of characterizing this type of predictor with only its advantage in the traditional sense, we also use its rate, which is how often it outputs a prediction. The advantage is generalized in a natural way for this different type of predictor.

Definition 1. A predictor $P\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1,\perp\}$ is said to have rate $\delta(n)$ and advantage $\varepsilon(n)$ in predicting $b(x,r)$ from $f(x)$ and r, where

$$\delta(n) = \Pr_{x,r\in U_n}[P(f(x),r) \neq \perp]$$

and

$$\varepsilon(n) = \Pr_{x,r\in U_n}[P(f(x),r) = b(x,r)] - \tfrac{1}{2}\Pr_{x,r\in U_n}[P(f(x),r) \neq \perp].$$

Note that the advantage of P is not the advantage conditioned on that P always answers, $\Pr_{x,r\in U_n}[P(f(x),r) = b(x,r) \mid P(f(x),r) \neq \perp] - \frac{1}{2}$, which could be natural to assume. Let us compare a predictor P with advantage ε and rate δ and a classical predictor P' with the same advantage (and with rate 1). If we apply P and P' on N samples of the form $(f(x),r)$, P' would answer correctly on $(\frac{1}{2}+\varepsilon)N$ samples (on average) and erroneously on $(\frac{1}{2}-\varepsilon)N$. On the other hand P would answer correctly on $(\delta/2+\varepsilon)N$ samples and erroneously on $(\delta/2-\varepsilon)\delta N$. The absolute difference between the number of correct and erroneous answers is the same for P and P', but P only answers on a δ-fraction of the samples while P' answers on all of them. This means that when P answers, its confidence is higher than the confidence of the answers of P'.

Going back to the list-decoding algorithm for the Hadamard code, this new type of predictor corresponds to a Hadamard code oracle with both errors and erasures, where the fraction of erasures is $1-\delta$ (where δ is the rate of the predictor). The heart of the improved reduction is in the analysis of the list-decoding algorithm with an oracle that has a relatively large part of erasures (or in other words a predictor with low rate). We briefly discuss why a predictor with low rate is more powerful than one with a higher rate and the same advantage. By more powerful we mean here that the predictor does not have to be called as many times in the list-decoding algorithm. Later we show how a nearly one-sided test for the Goldreich–Levin bit can be easily turned into a predictor with low rate.

Assume that P is a predictor with rate δ and advantage ε. Earlier analyses, that only made use of the advantage, have shown (see Section 2.5.2 of [7]) that the number of needed calls to P should be at least proportional to ε^{-2} for the list-decoding algorithm to succeed with probability one-half. The probability that P makes a correct prediction is $\frac{1}{2}\delta+\varepsilon$. We now ignore all the calls that received $\perp$-answers from P. The probability that

P, on each of the remaining calls, guesses correctly is then $(\frac{1}{2}\delta + \varepsilon)\delta^{-1} = \frac{1}{2} + \delta^{-1}\varepsilon$. In some sense this gives us a not fully working predictor with advantage $\delta^{-1}\varepsilon$, the problem being that it does not make predictions for all inputs and that it on average only makes one prediction per δ^{-1} calls. If the first problem mentioned can be dealt with in the list-decoding algorithm we can expect that the number of calls to P is proportional to the new advantage inverted and squared (according to the old analysis) times the extra factor of δ^{-1}: $O((\delta^{-1}\varepsilon)^{-2}\delta^{-1}) = O(\delta\varepsilon^{-2})$. Note that if the advantage of P is at least a constant fraction of the rate, we have that $\delta = O(\varepsilon)$ and thus the number of calls needed would only be $O(\varepsilon^{-1})$ compared with $O(\varepsilon^{-2})$ before. In the proofs of Theorems 1 and 3 we show that this intuition really has merit.

4. List Decoding of Hadamard Code with Errors and Erasures

The main part of the proof that the Goldreich–Levin bit is a hard-core predicate, consists of a list-decoding algorithm of a binary Hadamard code with errors. To ensure that the power of the low rate of the predictor does not vanish, we repeat the analysis of the list-decoding algorithm (not the original one in [10], but one due to Rackoff explained in [7]) while letting the Hadamard code also contain erasures. As far as the author is aware of, no previous work has been done on list decoding the Hadamard code in the presence of errors and erasures.

Theorem 1. *There is an algorithm* **LD** *that, on input l and n and with oracle access to a binary Hadamard code of x (where $|x| = n$) with an e-fraction of errors and an s-fraction of erasures, can output a list of 2^l elements in time $O(nl2^l)$ asking $n2^l$ oracle queries such that the probability that x is contained in the list is at least one-half if $l \geq \log_2(8n(e+c)/(c-e)^2 + 1)$, where $c \stackrel{\text{def}}{=} 1 - s - e$ (the fraction of correct answers from the oracle).*

Proof. Let C be an oracle that represents a Hadamard code of a fixed x with an e-fraction of errors and an s-fraction of erasures. In Table 1 the list-decoding algorithm $\mathbf{LD}^C$ is defined. First we prove its correctness with respect to the claim made in Theorem 1 that it outputs x with at least probability one-half. We then analyze its time complexity.

Correctness of $\mathbf{LD}^C$: We start by proving the following claim about the value of $C'(e^i \oplus s^J)$ which is a principal component in the calculations made in step 3 of our list-decoder $\mathbf{LD}^C$.

Claim 2. *Let s^J and C' be defined as in the description of $\mathbf{LD}^C$. Then for a nonempty $J \subseteq [l]$ and $L = \{j \mid j \in [l],\ b(s^j, x) = 1\}$ the following equalities hold*:

$$\begin{aligned}
\Pr[(-1)^{\langle J,L\rangle} C'(e^i \oplus s^J) = 0] &= s,\\
\Pr[(-1)^{\langle J,L\rangle} C'(e^i \oplus s^J) = (-1)^{b(e^i,x)}] &= c,\\
\Pr[(-1)^{\langle J,L\rangle} C'(e^i \oplus s^J) = -(-1)^{b(e^i,x)}] &= e,
\end{aligned}$$

where the probabilities are taken over the choices of s^j in step 1 of $\mathbf{LD}^C$.

Table 1. The list-decoder $\mathbf{LD}^C$.

Implementation of list-decoder $\mathbf{LD}^C$: Let $\mathbf{LD}$ have oracle access to C: $\{0,1\}^n \to \{0,1,\perp\}$. On input l and n, $\mathbf{LD}^C$ proceeds as follows:

1. Choose $s^1, \ldots, s^l$ independently from U_n.
2. Define a predictor C' that uses C so that
$$C'(r) = \begin{cases} 1 & \text{if} \quad C(r) = 0, \\ -1 & \text{if} \quad C(r) = 1, \\ 0 & \text{if} \quad C(r) = \perp. \end{cases}$$
3. Calculate
$$d_L^i = \sum_{J \subseteq [l], J \neq \emptyset} (-1)^{\langle J, L \rangle} C'(e^i \oplus s^J)$$
for all $L \subseteq [l]$ and $i \in [n]$, where $s^J \stackrel{\text{def}}{=} \oplus_{j \in J} s^j$ for all $J \subseteq [l]$.
4. Output the list $\{z^L\}_{L \subseteq [l]}$ where the ith bit of z^L is defined as $(1 - \operatorname{sgn}(d_L^i))/2$.

Proof. We observe that

$$b(s^J, x) = \bigoplus_{j \in J} b(s^j, x) = \bigoplus_{j \in J \cap L} b(s^j, x) = \langle J, L \rangle$$

and

$$C'(e^i \oplus s^J) = \begin{cases} (-1)^{b(e^i,x) \oplus b(s^J,x)} & \text{if } C \text{ answers correctly,} \\ -(-1)^{b(e^i,x) \oplus b(s^J,x)} & \text{if } C \text{ answers incorrectly,} \\ 0 & \text{if } C \text{ answers } \perp. \end{cases}$$

As J is nonempty, the value of $e^i \oplus s^J$ will be uniformly distributed and thus the probability that C answers "$\perp$" is s, incorrectly is e and correctly is c. As $b(s^J, x) = \langle J, L \rangle$ the claim follows from

$$(-1)^{b(s^J,x)} C'(e^i \oplus s^J) = \begin{cases} (-1)^{b(e^i,x)} & \text{if } C \text{ answers correctly,} \\ -(-1)^{b(e^i,x)} & \text{if } C \text{ answers incorrectly,} \\ 0 & \text{if } C \text{ answers } \perp. \end{cases}$$ □

As a consequence of their construction, the values of s^J, for nonempty $J \subseteq [l]$, are pairwise independent and uniformly distributed. Let L be defined as $\{j \mid j \in [l],\ b(s^j, x) = 1\}$ and study for a fixed i the value of d_L^i calculated in step 3 of $\mathbf{LD}^C$. It is a sum of expressions of the form that is analyzed in Claim 2. The probability of different results (expressed in terms of the ith bit of x) of each term in this sum is specified in the claim. Using the value (sign) of the sum we can thereby guess the ith bit of x and by knowing the different probabilities we can calculate an upper bound for the probability that the guess is incorrect.

For our guess to be correct we would like to have more terms that equal $(-1)^{b(e^i,x)}$ than $-(-1)^{b(e^i,x)}$. As we know the probability for each outcome we can, by using Chebyschev's inequality (defined below), give an upper bound on the probability that

the guess is incorrect. For every nonempty $J \subseteq [l]$, define ζ_c^J to be the indicator variable for the event that $(-1)^{\langle J,L\rangle} C'(e^i \oplus s^J) = (-1)^{b(e^i,x)}$ and let ζ_e^J be the indicator variable for the event that $(-1)^{\langle J,L\rangle} C'(e^i \oplus s^J) = -(-1)^{b(e^i,x)}$. We would like to be able to state that $\sum_J \zeta_c^J > \sum_J \zeta_e^J$ with high probability.

Chebyschev's Inequality (from [1]). *For any positive t*

$$\Pr[|Y - \mu| \geq t\sigma] \leq t^{-2},$$

where σ is the standard deviation and μ is the expectation of the random variable Y.

This inequality is to be applied on the number of incorrect answers $Y = \sum_J \zeta_e^J$ which has the expected value of $\mu = Ne$ (where $N \stackrel{\text{def}}{=} 2^l - 1$ is the number of terms in the sum) and the standard deviation σ is less than $\sqrt{Ne}$ (using the fact of pairwise independency). We set $t = \sqrt{N}(c-e)/2\sqrt{e}$ which gives us

$$\Pr\left[\left|\sum_J \zeta_e^J - Ne\right| > \frac{N(c-e)}{2}\right] \leq \frac{4e}{N(c-e)^2}.$$

Applying the same inequality on the number of correct answers $Y = \sum_J \zeta_c^J$ with $\mu = Nc$, $\sigma < \sqrt{Nc}$ and $t = \sqrt{N}(c-e)/2\sqrt{c}$ gives

$$\Pr\left[\left|\sum_J \zeta_c^J - Nc\right| > \frac{N(c-e)}{2}\right] \leq \frac{4c}{N(c-e)^2}.$$

If none of the sums $\sum_J \zeta_c^J$ and $\sum_J \zeta_e^J$ deviates more than $N(c-e)/2$ from their expected values we can conclude that the number of correct answers outnumbers the number of incorrect answers. Thus, the probability that this is not the case and thereby we are not able to make a correct prediction is at most $4(e+c)/N(c-e)^2$.

For the algorithm to succeed (in the supposed fashion) each of the n different bits of x has to be predicted correctly. In other words d_L^i has to have the correct sign for each $i \in [n]$. An upper bound for this not occurring is $4n(e+c)/N(c-e)^2$ which is the sum of the upper bounds for each bit prediction failure. If $N \geq 8n(e+c)/(c-e)^2$, then this bound is less than one-half. As N equals $2^l - 1$ we conclude that if the input l satisfies

$$l \geq \log\left(\frac{8n(e+c)}{(c-e)^2} + 1\right),$$

then the probability that x appears in the output list is at least one-half.

Efficiency of $\mathbf{LD}^C$: The first step of $\mathbf{LD}^C$ takes time $O(nl)$ and the last step takes time $O(n2^l)$. The time-consuming step of the algorithm is the calculation of the different values of d_L^i. The naive way to do this would be by calculating each d_L^i value independently for each L. This would make the algorithm work in time $O(n^2 2^{2l})$ making $O(n2^l)$ calls to C, as there are $n2^l$ different d_L^i values and each value is a sum of $2^l - 1$ terms and each term can be calculated in time $O(n)$.

We now show a better way to calculate d_L^i for all $L \subseteq [l]$. We define $a(J) \stackrel{\text{def}}{=} C'(e^i \oplus s^J)$ and $\hat{a}_L \stackrel{\text{def}}{=} \sum_J (-1)^{\langle J,L\rangle} a(J)$. As $d_L^i = \hat{a}_L - a(\emptyset)$ it is enough to calculate $\hat{a}_L$. We define

$$\hat{a}_{L'}^{k,J'} \stackrel{\text{def}}{=} \sum_{J \subseteq [k]} (-1)^{\langle J,L'\rangle} a(J \cup J'),$$

where $L' \subseteq [k]$ and $J' \subseteq [l]\backslash[k]$. For each k there are 2^l different possible combinations of L' and J' giving a total of $l2^l$ different variables on the form $\hat{a}_{L'}^{k,J'}$ with $k \in [l]$. For $k = 0$ we have that $\hat{a}_{\emptyset}^{k,J'} = a(J')$ and by using the recursive formula

$$\hat{a}_{L'}^{k,J'} = \begin{cases} \hat{a}_{L'-\{k\}}^{k-1,J'} - \hat{a}_{L'-\{k\}}^{k-1,J'\cup\{k\}} & \text{if } k \in L', \\ \hat{a}_{L'}^{k-1,J'} + \hat{a}_{L'}^{k-1,J'\cup\{k\}} & \text{if } k \notin L', \end{cases}$$

we can calculate all variables on the form $\hat{a}_{L'}^{k+1,J'}$ given that we know the values of $\hat{a}_{L'}^{k,J'}$. By iteratively increasing k we can calculate the values of $\hat{a}_{L'}^{l,\emptyset}$ (which equals $\hat{a}_{L'}$) for all $L' \subseteq [l]$.

To calculate $\hat{a}_{L'}^{k,J'}$ for $k = 0$ we have to make an oracle query for each value of J', giving a total of 2^l oracle queries. If $k > 0$ we instead perform an addition or subtraction (demanding constant time) for each $\hat{a}_{L'}^{k,J'}$ we calculate, giving a total work time of $O(l2^l)$ to calculate $\hat{a}_{L'}$ and thus also $d_{L'}^i$ for all $L' \subseteq [l]$.

By identifying $\hat{a}_{L'}$ as the discrete Fourier transform of the function $a(J)$: $2^{[l]} \to \mathbb{R}$ (neglecting a constant factor) we see that the above algorithm is essentially the fast Fourier transform algorithm. □

5. Goldreich–Levin Hard-Core Bit

In this section we make an efficient reduction from inverting a function f to predicting the Goldreich–Levin bit of f. The list-decoding algorithm from the previous section is the main component of the algorithm that performs the reduction.

Theorem 3. *Let P be a probabilistic algorithm with running time t_P: $\mathbb{N} \to \mathbb{N}$, and rate δ_P: $\mathbb{N} \to [0, 1]$ and advantage ε_P: $\mathbb{N} \to [0, \frac{1}{2}]$ in predicting $b(x, r)$ from $f(x)$ and r. Define $h(n)$ to be $\log_2(\delta_P(n)/\varepsilon_P(n)^2)$. Then there exists an algorithm* **Inv** *that runs in expected time $(t_P(n) + h(n)\log_2(n)) \cdot h(n) \cdot O(n^2)$ and satisfies*

$$\Pr_{x \in U_n}[f(\mathbf{Inv}(f(x), n)) = f(x)] = \Omega\left(\frac{\varepsilon_P(n)^2}{\delta_P(n)}\right).$$

The theorem states that if there is an algorithm P that predicts the Goldreich–Levin bit of f, then there exists an algorithm **Inv** that inverts f. If P, for all possible values of x, would have approximately the same advantage (and rate) in predicting $b(r, x)$ from $f(x)$ and r, this can be shown by directly applying the list-decoding algorithm **LD** with the appropriate value of l. However, as this is not true in general we need to make an averaging argument. This is done by calling **LD** with small values on l with high probability (to cope with values of x giving P a high advantage), and calling **LD** with

Table 2. The inverter **Inv**.

Description of inverter Inv: P is a predictor with rate $\delta_P(n)$ and advantage $\varepsilon_P(n)$. On input $y = f(x)$ and n **Inv** proceeds as follows:

1. Select j from $\{-1, \ldots, h-2\}$, where $h = \lfloor h(n) \rfloor = \lfloor \log(\delta_P(n)/\varepsilon_P(n)^2) \rfloor$, with probability 2^{j-h} and set $l = \lceil \log(n\delta_P(n)/\varepsilon_P(n)^2) \rceil - j + 2$. If no j is chosen, stop and output $\perp$.
2. Call the list-decoder $\mathbf{LD}^{P_y}$ with input l and n, where $P_y(\cdot) \stackrel{\text{def}}{=} P(y, \cdot)$.
3. Apply f on each element x' of the output from the list-decoder. If $f(x') = y$, then output x' and stop.
4. Output $\perp$.

big values on l with low probability (to cope with values of x giving P an intermediate advantage).

Proof. The inverting algorithm **Inv** is depicted in Table 2. For readability reasons we fix the value of n and let $\varepsilon \stackrel{\text{def}}{=} \varepsilon_P(n)$ and $\delta \stackrel{\text{def}}{=} \delta_P(n)$. Before proceeding to the analysis of the success probability of **Inv** we need some more specific definitions about the predicting capabilities of the predictor P. By ε_x and δ_x we denote the rate and the advantage of the predictor P when the value of x is fixed,

$$\delta_x = \Pr_{r \in U_n} [P(f(x), r) \neq \perp]$$

and

$$\varepsilon_x = \Pr_{r \in U_n} [P(f(x), r) = b(x, r)] - \tfrac{1}{2} \Pr_{r \in U_n} [P(f(x), r) \neq \perp].$$

Furthermore, we define S_j to be the set of all $x \in \{0, 1\}^n$ with $\varepsilon_x > 0$ such that

$$2^j \frac{\varepsilon^2}{\delta} \leq \frac{\varepsilon_x^2}{\delta_x} < 2^{j+1} \frac{\varepsilon^2}{\delta},$$

and let q_j be the fraction of $x \in S_j$, that is to say $q_j \stackrel{\text{def}}{=} 2^{-n}|S_j|$. (If $\delta_x = 0$ we know that $\varepsilon_x = 0$ and we define ε_x^2/δ_x as equal to zero, and thus those x are not contained in any set S_j.)

The success probability of **Inv** depends upon the success probability of the list-decoder which is stipulated by Theorem 1. We view $P_{f(x)}(\cdot) \stackrel{\text{def}}{=} P(f(x), \cdot)$ as an oracle providing the Hadamard code of x with a $(1 - \delta_x)$-fraction of erasures, a $(\delta_x/2 - \varepsilon_x)$-fraction of errors and a $(\delta_x/2 + \varepsilon_x)$-fraction of correct answers. Theorem 1 stipulates that if the input

$$l \geq \log\left(\frac{8n(e+c)}{(c-e)^2} + 1\right) = \log\left(\frac{8n\delta_x}{(2\varepsilon_x)^2} + 1\right) = \log\left(\frac{2n\delta_x}{\varepsilon_x^2} + 1\right),$$

then the success probability of the list-decoder $\mathbf{LD}^{P_{f(x)}}$ is at least one-half. If the list-decoder is successful, then x has to appear in its output which will cause **Inv** to output x (or some x' such that $f(x') = f(x)$) successfully.

We now proceed to analyze the success probability of **Inv**. We show that if $x \in S_j$ and j is selected, then the condition on l is satisfied. Remember that the condition is $l \geq \log(2n\delta_x/\varepsilon_x^2+1)$. Suppose $x \in S_j$, then $2n\delta_x/\varepsilon_x^2+1 \leq 2^{-j}(2n\delta/\varepsilon^2)+1 \leq 2^{2-j}(n\delta/\varepsilon^2)$ (where the second inequality follows from the fact that $j \leq h-2$) implying that if $l \geq \log(2^{2-j}(n\delta/\varepsilon^2))$, then the list-decoder will succeed with probability at least one-half. If j is selected we set $l = \lceil \log(n\delta/\varepsilon^2) \rceil + 2 - j$ and thereby the condition on l is fulfilled and x will show up in the output from the list-decoder with at least probability one-half, as long as $x \in S_j$.

A lower bound for the probability that the inverting algorithm succeeds can now be calculated:

$$\begin{aligned}
&\sum_{j=-1}^{h-2} \Pr[j \text{ is selected}]\Pr[x \in S_j]\Pr[\mathbf{LD} \text{ succeeds} \mid j \text{ selected and } x \in S_j] \\
&\qquad \geq \sum_{j=-1}^{h-2} 2^{j-h} q_j \tfrac{1}{2} \\
&\qquad = 2^{-(h+1)} \sum_{j=-1}^{h-2} q_j 2^j.
\end{aligned}$$

We now show that $\sum_{j=-1}^{h-2} q_j 2^j > \frac{1}{4}$ enabling us to conclude that the success probability of **Inv** is $\Omega(2^{-h}) = \Omega(\varepsilon^2/\delta)$.

We note that $S_j = \emptyset$ for $j > \log(\delta/\varepsilon^2) - 2$ because

$$\begin{aligned}
\frac{\varepsilon_x^2}{\delta_x} &= \frac{(\Pr_{r\in U_n}[P_{f(x)}(r) = b(x,r)] - \frac{1}{2}\Pr_{r\in U_n}[P_{f(x)}(r) \neq \perp])^2}{\Pr_{r\in U_n}[P_{f(x)}(r) \neq \perp]} \\
&\leq \frac{(\frac{1}{2}\Pr_{r\in U_n}[P_{f(x)}(r) \neq \perp])^2}{\Pr_{r\in U_n}[P_{f(x)}(r) \neq \perp]} \\
&\leq \tfrac{1}{4}.
\end{aligned}$$

From the definition of S_j we know that for $x \in S_j$ we have that $\varepsilon_x^2/\delta_x < 2^{j+1}(\varepsilon^2/\delta)$ which implies that

$$\sum_{x\in\{0,1\}^n:\varepsilon_x>0} \frac{\varepsilon_x^2}{\delta_x} < \sum_{j=-\infty}^{\infty} |S_j| 2^{j+1} \frac{\varepsilon^2}{\delta}.$$

Note that $\varepsilon_x > 0$ implies that $\delta_x > 0$ so we do not have to worry about division by zero. Dividing both sides by $2^{n+1}(\varepsilon^2/\delta)$ and using that $S_j = \emptyset$ for $j > h-2$ gives

$$\frac{\delta}{2\varepsilon^2} \cdot 2^{-n} \sum_{x\in\{0,1\}^n:\varepsilon_x>0} \frac{\varepsilon_x^2}{\delta_x} < \sum_{j=-\infty}^{h-2} q_j 2^j. \tag{2}$$

The following claim helps us analyze the left-hand side of the inequality.

Claim 4. *Let ε, δ, ε_x and δ_x be defined as above, then*

$$2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \frac{\varepsilon_x^2}{\delta_x} \geq \frac{\varepsilon^2}{\delta}.$$

Proof. We regard the values of ε_x as fixed for all x and all the δ_x as non-negative variables under the constraint that $2^{-n} \sum_x \delta_x = \delta$, and try to minimize the sum appearing in the claim. We define f to be the sum with variables δ_x,

$$f(\{\delta_x\}) = 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \frac{\varepsilon_x^2}{\delta_x}.$$

We can assume that all ε_x are non-negative because substituting negative values of ε_x with zero would only increase the right-hand side of the equation and keep the left-hand side fixed. (We can even decrease the left-hand side by setting $\delta_x = 0$ and increase $\delta_{x'}$ for some x' with positive $\varepsilon_{x'}$.)

We note that by setting $\delta_x^* = \varepsilon_x \delta / \varepsilon$ we have $2^{-n} \sum_x \delta_x^* = \delta 2^{-n} \sum_x \varepsilon_x / \varepsilon = \delta$ and get equality in the equation in the claim as

$$f(\{\delta_x^*\}) = 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \frac{\varepsilon_x^2}{\varepsilon_x \delta / \varepsilon} = \frac{\varepsilon}{\delta} 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \varepsilon_x = \frac{\varepsilon^2}{\delta}.$$

Still to be shown is that setting $\delta_x = \delta_x^*$ is in fact optimal if we want to minimize f under the constraint that $2^{-n} \sum_x \delta_x = \delta$ and $\delta_x \geq 0$. Therefore assume that this is not the case and that there is an assignment $\{\delta_x'\}$ such that $f(\{\delta_x'\}) < f(\{\delta_x^*\})$ while fulfilling the constraints $2^{-n} \sum_x \delta_x' = \delta$ and $\delta_x' \geq 0$.

Each term in f depends only on a single variable. Consider a term in f, $\varepsilon_x^2 / \delta_x$. The term is convex (for positive δ_x) and the derivative is $-\varepsilon_x^2 / \delta_x^2$ and thereby we conclude that

$$\frac{\varepsilon_x^2}{\delta_x'} \geq \frac{\varepsilon_x^2}{\delta_x^*} + (\delta_x' - \delta_x^*) \left(-\frac{\varepsilon_x^2}{\delta_x^{*2}} \right)$$

for all admissible (positive) values on δ_x'. We use this expression and the fact that $-\varepsilon_x^2 / \delta_x^{*2}$ equals $-\varepsilon^2 / \delta^2$ for all δ_x^* to bound $f(\{\delta_x'\})$ from below:

$$\begin{aligned} f(\{\delta_x'\}) &= 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \frac{\varepsilon_x^2}{\delta_x'} \\ &\geq 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \left(\frac{\varepsilon_x^2}{\delta_x^*} + (\delta_x' - \delta_x^*) \left(-\frac{\varepsilon^2}{\delta^2} \right) \right) \\ &= 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \frac{\varepsilon_x^2}{\delta_x^*} + 2^{-n} \sum_{x \in \{0,1\}^n : \varepsilon_x > 0} \left((\delta_x' - \delta_x^*) \left(-\frac{\varepsilon^2}{\delta^2} \right) \right) \end{aligned}$$

$$
\begin{aligned}
&= f(\{\delta_x^*\}) + 2^{-n}\left(-\frac{\varepsilon^2}{\delta^2}\right)\sum_{x\in\{0,1\}^n:\varepsilon_x>0}\left(\delta_x' - \delta_x^*\right)\\
&= f(\{\delta_x^*\}) + 2^{-n}\left(-\frac{\varepsilon^2}{\delta^2}\right)(\delta-\delta)\\
&= f(\{\delta_x^*\}).
\end{aligned}
$$

However, this contradicts our assumption and thus the claim is correct. □

Now we continue the proof of Theorem 3. We know that $\sum_{j=-\infty}^{-2} q_j 2^j \leq \frac{1}{4}$ (because $\sum_{j=-\infty}^{-2} q_j \leq 1$) and can therefore conclude, using Claim 4 on the left-hand side of (2), that

$$\sum_{j=-1}^{h-2} q_j 2^j > \tfrac{1}{4}.$$

As noted earlier this enables us to draw the conclusion that **Inv** succeeds in inverting a randomly chosen x with probability $\Omega(\varepsilon^2/\delta)$ as stated in Theorem 3.

Still to be shown is the expected running time of **Inv**. The running time of **Inv** depends on the value of l that is set in the first step of the algorithm. In the second step the list-decoding algorithm takes time $O(nl2^l)$ and also $n2^l$ queries to P have to be processed. In total the step takes time $O(nl2^l) + t_P(n)\cdot n2^l$ which makes the running times of the other parts of the algorithm negligible. As $l = \lceil \log(n\delta/\varepsilon^2)\rceil - j + 2$ we conclude that the running time of **Inv** is $O(n^2 2^{-j}(\delta/\varepsilon^2))\cdot(t_P(n) + h(n)\log(n))$ and thereby the expected running time is

$$
\begin{aligned}
&\sum_{j=-1}^{h-2} \Pr[j \text{ is selected}]\cdot(t_P(n)+h(n)\log(n))\,O\left(n^2 2^{-j}\frac{\delta}{\varepsilon^2}\right)\\
&\qquad = \sum_{j=-1}^{h-2} 2^{j-h}(t_P(n)+h(n)\log(n))\,O\left(n^2 2^{-j}\frac{\delta}{\varepsilon^2}\right)\\
&\qquad = (t_P(n)+h(n)\log(n))\sum_{j=-1}^{h-2} O(n^2)\\
&\qquad = (t_P(n)+h(n)\log(n))\cdot h(n)\cdot O(n^2),
\end{aligned}
$$

which establishes Theorem 3. □

We note that it seems unavoidable that the running time of **Inv** is expressed in expected time instead of strict time if we do not want to worsen the time-success ratio. This is realized by considering the case when P answers and answers correctly with the same probability for all values of x ($\delta_x = \delta_P(n)$ and $\varepsilon_x = \varepsilon_P(n)$ for all x of length n). When **Inv** calls **LD**, P is invoked 2^l times where l has to be at least $\log(n\delta_P(n)/\varepsilon_P(n)^2)$ for **LD** to work in the supposed fashion. Thus, every time **Inv** succeeds in inverting f it has used considerably more time than it uses on average.

Another observation on Theorem 3 is that knowledge about the value of $\varepsilon_P(n)^2/\delta_P(n)$ is required if we would like to implement **Inv**. Therefore we define a new algorithm **Inv**$'$, with oracle access to a predictor P, that takes an additional input h and behaves exactly as **Inv** but uses h instead of $\log(\delta_P(n)/\varepsilon_P(n)^2)$ in the first step of the algorithm. From the proof of Theorem 3 we can conclude that as long as $h \geq \log(\delta_P(n)/\varepsilon_P(n)^2)$ the probability of success will not decrease and the running time and the number of queries to P will be the same as in Theorem 3 except that we substitute $\log(\delta_P(n)/\varepsilon_P(n)^2)$ with h. We thus have the following corollary which will be useful in the proof of Theorem 6.

Corollary 5. *Let P be an arbitrary algorithm predicting $b(x, r)$ from $f(x)$ and r. There exists an algorithm* **Inv**$'$ *with oracle access to P such that on input y, n and h it runs in expected time $h^2 \log_2(n) \cdot O(n^2)$ and makes an $h \cdot O(n^2)$ number of expected calls and satisfies*

$$\Pr_{x \in U_n}[f(\mathbf{Inv}'^{P}(f(x), n, h)) = f(x)] = \Omega(2^{-h})$$

if $h \geq \log_2(\delta_P(n)/\varepsilon_P(n)^2)$, where P has rate $\delta_P\colon \mathbb{N} \to [0, 1]$ and advantage $\varepsilon_P\colon \mathbb{N} \to [0, \frac{1}{2}]$ in predicting $b(x, r)$ from $f(x)$ and r.

6. Nearly One-Sided Statistical Tests

A statistical test is a probabilistic algorithm that takes an input and outputs a bit. The purpose of a statistical test is to distinguish between different distributions. This is done by having different output distributions when the input is drawn from different distributions. The output distribution is characterized by the probability that the output is equal to one, respectively zero. If the test rarely outputs 1, on uniform input, we consider the test to be nearly one-sided. The classical way to measure how well a statistical test distinguishes between two distributions (or in fact two ensembles of distributions) is through the following definition.

Definition 2. An algorithm $D\colon \{0, 1\}^* \to \{0, 1\}$ $\delta(n)$-distinguishes the ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$ if for infinitely many values of n,

$$\left| \Pr_{x \in X_n}[D(x) = 1] - \Pr_{y \in Y_n}[D(y) = 1] \right| \geq \delta(n).$$

The characterization of a statistical test in terms of this definition is rather coarse. For example, it cannot be used to show if a test is nearly one-sided. This is remedied by the definition of a parameterized distinguisher which for a test and two input distributions separates the corresponding output distributions by imposing two specific thresholds.

Definition 3. An algorithm $D\colon \{0, 1\}^* \to \{0, 1\}$ $(\delta_1(n), \delta_2(n))$-distinguishes the ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$ if for infinitely many values of n,

$$\Pr_{x \in X_n}[D(x) = 1] \leq \delta_1(n) < \delta_2(n) \leq \Pr_{y \in Y_n}[D(y) = 1].$$

In addition, D is said to be a $(\delta_1(n), \delta_2(n))$-distinguisher for the ensembles X and Y if D $(\delta_1(n), \delta_2(n))$-distinguishes X and Y.

Let D be a $(\delta_1(n), \delta_2(n))$-distinguisher for the ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$. If $\delta_1(n)$ is "small," then D is said to be a nearly one-sided statistical test with respect to X. For the special case when X is the ensemble of the uniform distributions (i.e., $X_i = U_i$ for all $i \in \mathbb{N}$), then D is merely said to be a nearly one-sided statistical test. As previously mentioned, we do not properly define the notion of a "nearly one-sided statistical test" as we do not define "small," but instead we use the more general notion of a parameterized distinguisher to express the formal result (i.e., Theorem 6).

We now discuss the connection between statistical tests and predictors, and in particular how a nearly one-sided statistical test for the Goldreich–Levin bit can be easily turned into a low rate predictor that predicts the Goldreich–Levin bit.

Assume that a distinguisher D satisfies

$$p_1 = \Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 1]$$

and

$$p_2 = \Pr_{r,x\in U_n}[D(f(x), r, b(r, x)) = 1].$$

It is easy to transform this distinguisher into a predictor P guessing $b(r, x)$ with advantage $p_2 - p_1$. On input $(f(x), r)$ the predictor P samples a uniform bit σ, queries for $D(f(x), r, \sigma)$ and outputs σ iff $D(f(x), r, \sigma) = 1$ and otherwise outputs $1 - \sigma$. However, if the probability p_1 is very small, then the truly informative answers from D are when it returns 1. In those cases the probability that the prediction is correct is

$$\frac{\Pr_{\sigma\in U_1}[\sigma = b(r, x)] \cdot \Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 1 \mid \sigma = b(r, x)]}{\Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 1]} = \frac{p_2}{2p_1},$$

giving an advantage of

$$\frac{p_2}{2p_1} - \frac{1}{2} = \frac{p_2 - p_1}{2p_1}.$$

This should be compared with the success probability when D outputs 0,

$$\begin{aligned}
&\frac{\Pr_{\sigma\in U_1}[1 - \sigma = b(r, x)] \cdot \Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 0 \mid 1 - \sigma = b(r, x)]}{\Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 0]} \\
&\quad = \frac{\frac{1}{2} \cdot (1 - \Pr_{r,x\in U_n,\sigma\in U_1}[D(f(x), r, \sigma) = 1 \mid 1 - \sigma = b(r, x)])}{1 - p_1} \\
&\quad = \frac{1 - (2p_1 - p_2)}{2(1 - p_1)} \\
&\quad = \frac{1 + p_2 - 2p_1}{2(1 - p_1)},
\end{aligned}$$

giving an advantage of

$$\frac{1 + p_2 - 2p_1}{2(1 - p_1)} - \frac{1}{2} = \frac{1 + p_2 - 2p_1 - (1 - p_1)}{2(1 - p_1)} = \frac{p_2 - p_1}{2(1 - p_1)}.$$

(Note that we have implicitly extended the notion of advantage to apply when also conditioned on the output of D.)

Assume that the test is nearly one-sided (with respect to the input distribution $\{(f(x), r, \sigma) \mid r, x \in U_n, \sigma \in U_1\}$) and thereby the value of p_1 is close to zero. In this case the advantage of the prediction when D outputs 1 is a factor p_1^{-1} better than if it outputs 0. This is why it is better to change P slightly so it still returns σ iff $D(f(x), r, \sigma) = 1$ but otherwise outputs $\perp$. This new predictor has rate p_1 and advantage $(p_2 - p_1)/2$. Thus we have transformed a nearly one-sided statistical test for the Goldreich–Levin bit into a low rate predictor that predicts the Goldreich–Levin bit.

7. Blum–Micali Pseudo-Random Generator

In [5] Blum and Micali constructed a PRBG based on a one-way permutation and a hard-core predicate associated with the permutation. As an example they used exponentiation modulo a prime as a one-way permutation and the most significant bit as its hard-core predicate. By using an arbitrary one-way permutation and the hard-core predicate of Goldreich–Levin [10] the following construction, referred to as *BMGL*, is obtained:

Construction 1. Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time-computable length-preserving permutation and let $l\colon \mathbb{N} \to \mathbb{N}$ where $n < l(n)$. Given the input x_0 and r such that $|x_0| = |r| = n$, define

$$G^{f,l}(x_0, r) = b(r, x_1)b(r, x_2)\cdots b(r, x_{l(n)}),$$

where $x_i = f(x_{i-1})$ for $i = 1, \ldots, l(n)$ and $b(r, x_i)$ is the inner product of r and x_i modulo 2. With $G_n^{f,l}$ we denote the output distribution of $G^{f,l}(x_0, r)$, where r and x_0 are chosen uniformly and independently from $\{0,1\}^n$.

The security of the *BMGL* is described by our next theorem. For nearly one-sided tests the reduction is an improvement over the previous most efficient reduction.

Theorem 6. *Let D be a $(p_1(n), p_2(n))$-distinguisher for $\{U_{l(n)}\}$ and $\{G_n^{f,l}\}$ with running time $t_D\colon \mathbb{N} \to \mathbb{N}$. Then there exists an algorithm* **Inv**$''$ *that runs in expected time $[(t_f(n)\cdot l(n) + t_D(n)) + m(n)\log_2(n)]\cdot m(n)\cdot O(n^2)$, where $t_f(n)$ is the time to calculate the function f on an n-bit input and*

$$m(n) \stackrel{\text{def}}{=} \log_2 \frac{p_2(n) l(n)^2}{(p_2(n) - p_1(n))^2},$$

and satisfies

$$\Pr_{x \in U_n}[\mathbf{Inv}''(f(x)) = x] = \Omega(2^{-m(n)})$$

for infinitely many values of n.

Table 3. The predictor P^i.

Implementation of predictor P^i: The predictor has access to a distinguisher D: $\{0,1\}^{l(n)} \to \{0,1\}$. On input y and r, P^i proceeds as follows:

1. Set $x_{i+1} = y$ and calculate x_j for $j \in \{i+2, \ldots, l(n)\}$ where $x_j = f(x_{j-1})$ and $n = |y|$.
2. Toss i unbiased coins $c_1, c_2, \ldots, c_{i-1}$ and σ.
3. Create the bit string $z = c_1 \cdots c_{i-1}\sigma b(x_{i+1}, r) \cdots b(x_{l(n)}, r)$. If $D(z) = 1$, then return σ, otherwise return $\perp$.

Note that the running time of **Inv**$''$ only depends upon $p_1(n)$ and $p_2(n)$ logarithmically. Assuming $t_f(n) \cdot l(n) \leq t_D(n)$ and ignoring logarithmic factors the time–success ratio is

$$\frac{p_2(n) l(n)^2 t_D(n)}{(p_2(n) - p_1(n))^2} \cdot O(n^2),$$

ameliorating the ratio given by the best previous analysis with a factor of $p_2(n)$. Furthermore, if we assume that $p_1(n) \leq c \cdot p_2(n)$ for some constant $c < 1$ the time–success ratio is

$$\frac{l(n)^2 t_D(n)}{p_2(n)} \cdot O(n^2),$$

still ignoring logarithmic factors.

Proof. The proof mainly consists of defining a number of predictors and showing that combined, these have good predicting qualities. This is done by a hybrid argument. The proof is concluded by applying Corollary 5 with these predictors.

The predictor P^i is depicted in Table 3. The bit string z that D is applied on is created according to the ith hybrid distribution:

$$H_n^i = \{r_1 \cdots r_i b_{i+1} \cdots b_{l(n)} \mid r_1 \cdots r_i \in U_i, b_1 \cdots b_{l(n)} \in G_n^{f,l}\}.$$

The fact that f is a permutation implies that the generator output has the same prefix and suffix distribution. By this we mean that the distribution $\{b_1 \cdots b_{l(n)-i} \mid b_1 \cdots b_{l(n)} \in G_n^{f,l}\}$ equals $\{b_{i+1} \cdots b_{l(n)} \mid b_1 \cdots b_{l(n)} \in G_n^{f,l}\}$. This motivates the claim that the bit string z in P^i is constructed according to the distribution H_n^i.

The extreme hybrids, H_n^0 and $H_n^{l(n)}$, are easily identified as the generator output distribution $G_n^{f,l}$, respectively the uniform distribution $U_{l(n)}$. According to the assumption in the theorem, the algorithm D $(p_1(n), p_2(n))$-distinguishes between these two distributions. Let

$$\delta_i \stackrel{\text{def}}{=} \Pr_{z \in H_n^i}[D(z) = 1],$$

giving $\delta_0 \geq p_2(n)$ and $\delta_{l(n)} \leq p_1(n)$ for infinitely many values of n. (For readability reasons, the dependency on n is implicit in δ_i.)

Table 4. The inverter $\mathbf{Inv}''$.

Implementation of inverter $\mathbf{Inv}''$: The inverter has access to a distinguisher D: $\{0,1\}^{l(n)} \to \{0,1\}$. On input y, $\mathbf{Inv}''$ proceeds as follows:

1. Choose $i \in [l(n)]$ uniformly, where $n = |y|$.
2. Choose j from $\{-2, -1, \ldots, \lfloor \log l(n) \rfloor\}$ with probability $2^{-(3+j)}$.
3. Call $\mathbf{Inv}'$ from Corollary 5 with input y, n and

$$h = \left\lceil \log \frac{p_2(n) l(n)^2}{2^{2j}(p_2(n) - p_1(n))^2} \right\rceil$$

providing P^i as the predictor. Return the output from $\mathbf{Inv}'^{P_i}$.

The predictor P^i constructs z according to H_n^i and makes a prediction if $D(z) = 1$. Thus the rate of the predictor (the probability that the output is not $\perp$) is δ_i. For P^i to be successful in its prediction it has to choose $\sigma = b(f^{-1}(y), r)$ in step 2. Now condition on that event when constructing z in P^i. In this case z is constructed according to H_n^{i-1}, as in P^{i-1}, and the rate of P^i is thus δ_{i-1}. The probability that σ is chosen equal to $b(f^{-1}(y), r)$ is one-half implying that the success probability of P^i is $\frac{1}{2}\delta_{i-1}$. The advantage ε_i of P^i is thus $\frac{1}{2}(\delta_{i-1} - \delta_i)$.

Table 4 presents our inverter $\mathbf{Inv}''$. It uniformly selects a predictor and guesses a value h and invokes the inverting algorithm $\mathbf{Inv}'$ from Corollary 5. If the guessed value is large enough compared with the predicting quality of the predictor, the corollary can be used to obtain a lower bound on the probability that $\mathbf{Inv}'$ succeeds in inverting. This enables us to derive a lower bound on the probability that $\mathbf{Inv}''$ succeeds in inverting its input.

We start the analysis by looking at the predicting capabilities of the predictors P^i for $i \in [l(n)]$. We can assume that $\delta_i \in [\delta_{l(n)}, \delta_0]$ for all $i \in [l(n)]$ because given the rates $\{\delta_i\}_{i=0}^{l(n)}$ we can construct the rates

$$\delta_i' = \begin{cases} \delta_0 & \text{if} \quad \delta_i > \delta_0, \\ \delta_{l(n)} & \text{if} \quad \delta_i < \delta_{l(n)}, \\ \delta_i & \text{otherwise} \end{cases}$$

that only weaken the predicting capabilities of all the predictors that have a positive advantage, which are the only ones that add to the success probability of $\mathbf{Inv}''$. By a similar argument we can assume that $\delta_i \leq \delta_{i-1}$.

Define S_j to be the set containing all $i \in [l(n)]$ such that

$$2^j \frac{p_2(n) - p_1(n)}{l(n)} \leq \varepsilon_i < 2^{j+1} \frac{p_2(n) - p_1(n)}{l(n)},$$

and let q_j be the fraction of $i \in S_j$, $q_j \stackrel{\text{def}}{=} |S_j|/l(n)$. We note that $\sum_{i=1}^{l(n)} \varepsilon_i = \sum_{i=1}^{l(n)} \frac{1}{2}(\delta_{i-1} - \delta_i) = \frac{1}{2}(\delta_0 - \delta_{l(n)}) \geq \frac{1}{2}(p_2(n) - p_1(n))$ for infinitely many values of n, and can thus

conclude, by adding the upper bounds, that for these values of n,

$$\tfrac{1}{2}(p_2(n) - p_1(n)) < \sum_{j=-\infty}^{\infty} |S_j| 2^{j+1} \frac{p_2(n) - p_1(n)}{l(n)}.$$

Multiplying the above equation by $2/(p_2(n) - p_1(n))$ and substituting $|S_j|/l(n)$ with q_j we get

$$4 \sum_{j=-\infty}^{\infty} q_j 2^j > 1.$$

The set S_j has to be empty if $2^j > l(n)$ and $\sum_{j=-\infty}^{-3} q_j 2^j \leq \frac{1}{8}$ (because $\sum_{j=-\infty}^{-3} q_j \leq 1$) which gives us

$$\sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} q_j 2^j > \tfrac{1}{8}.$$

If the parameters i and h are set in **Inv**$''$ such that $h \geq \log(\delta_i/\varepsilon_i^2)$, then the success probability for **Inv**$''$ is $\Omega(2^{-h})$ according to Corollary 5. This occurs if i and j are chosen so that $i \in S_j$ because then $\delta_i \leq p_2(n)$ (for infinitely many values of n) and $2^j(p_2(n) - p_1(n))/l(n) \leq \varepsilon_i$ according to the definition of S_j and thus we have

$$\log \frac{\delta_i}{\varepsilon_i^2} \leq \log \frac{p_2(n)}{(2^j(p_2(n) - p_1(n))/l(n))^2} \leq \left\lceil \log \frac{p_2(n) l(n)^2}{2^{2j}(p_2(n) - p_1(n))^2} \right\rceil = h.$$

A lower bound on the total success probability of **Inv**$''$ which is valid for infinitely many values of n can now be expressed:

$$\begin{aligned}
&\sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} \Pr_{i \in [l(n)]}[i \in S_j] \Pr[j \text{ is selected}] \Pr[\textbf{Inv}' \text{ succeeds}] \\
&\quad = \sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} q_j 2^{-(j+3)} \Omega(2^{-\lceil \log(p_2(n) l(n)^2 / 2^{2j}(p_2(n) - p_1(n))^2) \rceil}) \\
&\quad = \sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} q_j 2^j \Omega\left(\frac{(p_2(n) - p_1(n))^2}{p_2(n) l(n)^2} \right) \\
&\quad = \Omega\left(\frac{(p_2(n) - p_1(n))^2}{p_2(n) l(n)^2} \right).
\end{aligned}$$

The running time of the predictor P^i, called t_{P^i}, is $O(t_f(n) \cdot l(n)) + t_D(n)$. The first two steps of **Inv**$''$ can be done in time $O(\log(l(n)))$ which is negligible compared with the running time of the last step. Corollary 5 states that the expected running time for **Inv**$'$ is $(t_{P^i} + h(j)\log(n)) \cdot h(j) \cdot O(n^2)$ where

$$h(j) = \left\lceil \log \frac{p_2(n) l(n)^2}{2^{2j}(p_2(n) - p_1(n))^2} \right\rceil = \left\lceil \log \frac{p_2(n) l(n)^2}{(p_2(n) - p_1(n))^2} \right\rceil - 2j.$$

The expected running time of **Inv**$''$ is thus

$$\begin{aligned}
&\sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} \Pr[j \text{ is selected}] \cdot [(t_{P^i} + h(j)\log(n)) \cdot h(j) \cdot O(n^2)] \\
&\qquad = \sum_{j=-2}^{\lfloor \log(l(n)) \rfloor} 2^{-(j+3)} \cdot ((O(t_f(n) \cdot l(n)) + t_D(n)) + h(j)\log(n)) \cdot h(j) \cdot O(n^2) \\
&\qquad = ((O(t_f(n) \cdot l(n)) + t_D(n)) + h(-2)\log(n)) \cdot h(-2) \cdot O(n^2) \\
&\qquad = ((t_f(n) \cdot l(n) + t_D(n)) + m(n)\log(n)) \cdot m(n) \cdot O(n^2),
\end{aligned}$$

where $m(n) \stackrel{\text{def}}{=} \log(p_2(n)l(n)^2/(p_2(n) - p_1(n))^2)$. The second equality is justified because the term with $j = k$ is always at least twice as big as the term with $j = k + 1$. As the first term in the sum is with $j = -2$ we can conclude that the whole sum is at most that term doubled. This concludes the proof of Theorem 6. □

8. Applications

Using pseudo-random material instead of real random material in probabilistic algorithms may be convenient for many reasons. A system can have problems obtaining enough random material or the results perhaps need to be reproducible without storing all the random material used.

An important use of pseudo-random material can be found in many implementations of cryptographic primitives. The security definitions of these primitives often express either the need for authentication or that of confidentiality. In the case of authentication there is often a natural nearly one-sided test corresponding to an attacker of the protocol, but in the case of confidentiality this is not always so.

Here we consider the security consequences of using pseudo-random material from a PRBG instead of true random bits in a cryptographic system. We assume that the system that uses true randomness is secure and want to establish that the corresponding system is also secure. The standard method for showing this is done by assuming that there is a successful attacker, when using pseudo-random material in the system, and transforming this attacker into a statistical test distinguishing between output from the PRBG and the uniform distribution. This statistical test will become nearly one-sided if the successful attacker produces a certain type of breaking which we call verifiable breakings. If the PRBG is *BMGL*, then the proof of security can prosper by our more efficient reduction established in Theorem 6. With a verifiable breaking of a cryptographic system we mean that a successful breaking can be verified, possibly with the help of secrets lying in the system such as secret keys.

8.1. *Signature Schemes*

As an example of how natural nearly one-sided tests come up in the case of authentication, we take a closer look at signature schemes. Using the standard definition of security in an adaptive chosen message attack environment [11], an attacker may query an oracle

for signatures of any messages of its choosing. The attacker is considered successful if it, on a verification key as input, can output a valid signature on any message. Furthermore, the signature should of course not be identical to a signature returned by the oracle. If there is an attacker (from a certain group of attackers) that has more than a negligible probability of breaking the signature scheme, then the signature scheme is considered to be insecure against that group of attackers. Now assume that S is a secure signature scheme but S^G is not, where the only difference between S^G and S is that S^G uses bits from a pseudo-random generator G when S uses true random bits. Then there is an adversary A that can break the signature scheme S^G, but not S, with non-negligible probability. This adversary can be easily transformed into a nearly one-sided statistical test D, distinguishing between the output of G and the uniform distribution, by first simulating S, using its input as the source of randomness and then attacking S using A.

8.2. *Encryption Schemes*

A natural way to define the security of an encryption system is to say that given a ciphertext it should be infeasible to produce the corresponding plaintext. If we adopted this as our security definition, an attacker A could be easily turned into a natural nearly one-sided test that would distinguish between encryptions made by a secure encryption scheme and encryptions made by an encryption scheme that A attacks successfully. In particular, this means that if we exchange the true random material that is used in a secure encryption scheme against bits produced by a generator, then a successful attacker on the resulting scheme can be turned into a nearly one-sided test distinguishing between the uniform distribution and the generator output.

Unfortunately, this natural way of defining security is not strict enough for all situations. In general we do not permit an attacker to be able to conclude anything at all about the plaintext by looking at the ciphertext. This notion was captured by the definition of semantic security in [11]. Using this definition there is no natural way to transform an attacker into a nearly one-sided test. For example, if an adversary could predict, by looking at a ciphertext, the ith bit of the plaintext (that was drawn from the uniform distribution) with probability $\frac{1}{2} + \varepsilon$, where ε is non-negligible, this would render the system insecure. However, this adversary corresponds with a predictor with rate one, and thus not to a nearly one-sided test.

8.3. *Other Applications*

In many algorithms pseudo-randomness is provided by a pseudo-random function. A well-known construction of pseudo-random functions is the GGM construction [8] which uses a pseudo-random generator as a building block. We note that a nearly one-sided statistical test, distinguishing between the use of real random functions and a family of GGM functions, can be transformed into a nearly one-sided statistical test distinguishing the uniform distribution and the underlying generator output. Briefly, this is so because the hybrid arguments used in the security proof of GGM will uphold the one-sidedness in distinguishing between real random functions and GGM functions.

For some applications in simulations, nearly one-sided tests will occur naturally. Consider a probabilistic algorithm for a decision problem that errs with very small probability. Assume that this algorithm uses the output from a PRBG as its source of

randomness and that this causes the algorithm to err with substantially higher probability. Then this algorithm can be used to produce a nearly one-sided test distinguishing the PRBG output and the uniform distribution.

9. Open Problems

In this work we have studied nearly one-sided tests for the Goldreich–Levin hard-core bit. A natural extension would be to consider what impact nearly one-sided tests have on other hard-core predicates. Additional applications where nearly one-sided tests occur could also be investigated.

Acknowledgments

I am very grateful to Johan Håstad for providing me with good ideas and continuous support. I am also grateful to the referee of this journal for valuable advice about the presentation as well as for suggesting interesting extensions. Finally, I thank Mikael Goldmann, Oded Goldreich, Jonas Holmerin, Åsa Karsberg, and Mats Näslund for much appreciated help and comments.

References

[1] N. Alon and J. H. Spencer: *The Probabilistic Method*, 2nd edn. Wiley Interscience, New York, 2000.
[2] M. Bellare: Practice-Oriented Provable-Security. *Proceedings*, *ISW* 97, LNCS 1396, pp. 221–231. Springer-Verlag, Berlin, 1998.
[3] M. Bellare and P. Rogaway: The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. *Proceedings*, *Eurocrypt* 1996, LNCS 1070, pp. 399–416. Springer-Verlag, Berlin, 1996.
[4] M. Blum and O. Goldreich: Towards a Computational Theory of Statistical Tests. *Proceedings*, 33*rd IEEE FOCS*, pp. 406–416, 1992.
[5] M. Blum and S. Micali: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, **13**(4) (1984), 850–864.
[6] R. Fischlin and C. P. Schnorr: Stronger Security Proofs for RSA and Rabin Bits. *Journal of Cryptology*, **13**(2) (2000), 221–244.
[7] O. Goldreich: *Foundations of Cryptography*: *Basic Tools*. Cambridge University Press, Cambridge, 2001.
[8] O. Goldreich, S. Goldwasser and S. Micali: How to Construct Random Functions. *Journal of the ACM*, **33**(4) (1986), 792–807.
[9] O. Goldreich, R. Impagliazzo, L. A. Levin, R. Venkatesan and D. Zuckerman: Security Preserving Amplification of Hardness. *Proceedings*, 31*st IEEE FOCS*, pp. 318–326, 1990.
[10] O. Goldreich and L. A. Levin: A Hard Core Predicate for any One Way Function. *Proceedings*, 21*st ACM STOC*, pp. 25–32, 1989.
[11] S. Goldwasser, S. Micali and R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, **17**(2) (1988), 281–308.
[12] J. Håstad and M. Näslund: Practical Construction and Analysis of Pseudo-Randomness Primitives. *Proceedings*, *Asiacrypt* 2001, LNCS 2248, pp. 442–459. Springer-Verlag, Berlin, 2001.
[13] A. Herzberg and M. Luby: Public Randomness in Cryptography. *Proceedings*, *Crypto* 1992, LNCS 740, pp. 421–432. Springer-Verlag, Berlin, 1993.
[14] L. A. Levin: Randomness and Non-Determinism. *Journal of Symbolic Logic*, **58**(3) (1993), 1102–1103.
[15] A. C. Yao: Theory and Application of Trapdoor Functions. *Proceedings*, 23*rd IEEE FOCS*, pp. 80–91, 1982.