J. Cryptol. (2008) 21: 1–26
DOI: 10.1007/s00145-007-9009-3

Journal of
CRYPTOLOGY

Practical Construction and Analysis of Pseudo-Randomness Primitives

Johan Håstad
Royal Inst. of Technology, 10044 Stockholm, Sweden
johanh@kth.se

Mats Näslund
Communications Security Lab, Ericsson Research, 16480 Stockholm, Sweden
mats.naslund@ericsson.com

Communicated by Moti Yung

Received 18 June 2003 and revised 25 February 2007
Online publication 15 September 2007

Abstract. We give a careful, fixed-size parameter analysis of a standard (Blum and Micali in SIAM J. Comput. 13(4):850–864, 1984; Goldreich and Levin in Proceedings of 21st ACM Symposium on Theory of Computing, pp. 25–32, 1989) way to form a pseudo-random generator from a one-way function and then pseudo-random functions from said generator (Goldreich et al. in J. Assoc. Comput. Mach. 33(4):792–807, 1986) While the analysis is done in the model of exact security, we improve known bounds also asymptotically when many bits are output each round and we find all auxiliary parameters efficiently, giving a uniform result. These optimizations makes the analysis effective even for security parameters/key-sizes supported by typical block ciphers and hash functions. This enables us to construct very practical pseudo-random generators with strong properties based on plausible assumptions.

Key words. Hard core function, One-way function, Pseudo random generator, Exact security.

1. Introduction

One of the most fundamental cryptographic primitives is the *Pseudo-Random Generator* or more succinctly a PRG, a deterministic algorithm that expands a few truly random bits to long "random looking" strings. Having such generators are basic for many applications in cryptography and computer science in general.

Despite the importance, many protocols used in practice often leave unspecified what PRG to use. While there is usually no interoperability problems in doing so, this unfortunately leaves room for "ad-hoc", possibly insecure, solutions.

From a theoretical point of view, a sound theory of pseudo-randomness did not emerge until the seminal works of Blum and Micali [5], and Yao [26] in the early 80's. The theoretical area was in a sense closed when, in [12], it was shown that necessary and sufficient conditions for the existence of a PRG is the existence of another fundamental primitive: the *one-way function*; a function easy to compute, but hard to invert. We do not know if such functions exist, but many strong candidates exist, such as a good block cipher. Still, the construction in [12] is complex, requiring key-sizes of millions of bits, and hence is not useful in practice.

On the subject of one-way functions and good block ciphers let us point to a sometimes overlooked fact. The first mapping that comes to mind in connection with a good block cipher is the mapping from clear-text to cipher-text which also is a permutation. This mapping, however, can be inverted by anybody that can compute it and hence it does not give a one-way function. Instead the natural one-way function takes as input the key and outputs the encryption of a fixed clear-text.

The reason for the ineffectiveness of the theoretical constructions is that one-wayness is in itself not a strong property. A function may be hard to invert but still have very undesirable properties. For instance, even if f is one-way, most of x may still be easily deduced from $f(x)$ and basing pseudo-randomness on one-wayness alone appears to require elaborate constructions. However, if one assumes only a little more than one-wayness, e.g. that the function f is also a permutation, the situation becomes much more favorable and reasonably practical constructions can be found from the work of Blum and Micali [5] mentioned above, and later work by Goldreich and Levin [10]. In [5] it is shown that if f is a permutation and has at least a single bit of information, $b(x)$, that does not leak via $f(x)$, then a PRG can be built. In [10], then, it is shown that every one-way function, in particular ones being permutations, have such a hard bit $b(x)$. For specific, assumed one-way functions, slightly simpler constructions are known. For instance, the papers [1,5] provide PRGs based on exponentiation mod p and the RSA function. Common to all these results, however, is that *the security is only proven to hold asymptotically*. Unfortunately, this has sometimes (e.g. [7,24]) been misunderstood (or neglected), implying that some deployed PRGs may actually be practically breakable, even though they are "based" on provably secure constructions.

In this paper we make a careful analysis of the general transformation from a one-way function to a PRG, see Sect. 3. The analysis is done in the model of exact security, but we add new elements of the general analysis when we output many bits for each iteration of f, improving the dependence on this parameter also asymptotically. First, we non-uniformly reduce inversion of f to distinguishing the generator from randomness, given some auxiliary parameters. We then give efficient sampling procedures to determine the values of these parameters, giving a uniform inversion algorithm. Values of the parameters that give almost as strong results as the existential bounds can, for many parameter values, be found in time less than the time needed for successive inversions.

A related primitive is that of pseudo-random functions; functions that can not be distinguished from random functions on the same domain/range. Goldreich, Goldwasser, and Micali [9] showed how such an object could be built from a PRG. In Sect. 3.3, we show how to use this construction to further enhance our generator.

Our explicit theorems allow us, in Sect. 5 to construct a generator that is efficient in practice based on the assumption that e.g. AES (or preferably the 256-bit block version of Rijndael), mapping keys to cipher-texts, fixing a plaintext, remains hard to invert even when iterated.

1.1. *History of this Paper and Related Works*

Manuscripts containing some of the results of this paper have existed in many forms. In particular this paper could be considered to be final version of [13]. It is also very related to the stream cipher proposal submitted to the NESSIE-initiative and appearing in [14,15].

Hast [11] has analyzed what happens if PRGs are used in some types of cryptographic constructions. He assumes that a certain cryptographic construction is secure when true random bits are used and insecure when pseudo-random are bits used. For some situations this yields a very one-sided statistical tests for randomness and exploiting the one-sidedness he is able to get a tighter connection between the one-wayness of the function on which the PRG is used and the usefulness of the generators. His results partly build on the results of this paper.

2. Preliminaries

2.1. *Notation*

The length of binary string x is denoted $|x|$, and by $\{0,1\}^n$ we denote the set of x such that $|x| = n$. We write $\mathcal{U}_n$ for the uniform distribution on $\{0,1\}^n$. We use $\|S\|$ to denote the cardinality of a set S. We use $\log x$ to refer to the logarithm in base 2 of a real number x. We sometimes treat real numbers as integers and we assume that some rounding is used. Sometimes, when it is important that we round a certain way we explicitly use $\lceil x \rceil$ to denote the smallest integer larger than x.

We let

$$\langle r, x\rangle_2 = r_1 \cdot x_1 + r_2 \cdot x_2 + \cdots + r_n \cdot x_n \bmod 2$$

denote the inner product mod 2. We use bit strings of length t to represent integers i in the range $0 \leq i \leq 2^t - 1$, elements $\alpha \in \mathbb{F}_{2^t}$ and subsets of $[t]$, the integers $\{0, 1, \ldots, t-1\}$. Sometimes when we want to emphasize that we want to discuss the binary representation, we use $\text{bin}(i)$ and $\text{bin}(\alpha)$ to highlight this fact. We use $\oplus$ to denote bitwise exclusive-or of vectors.

We assume that we are given a representation of $\mathbb{F}_{2^t}$, i.e. an irreducible polynomial of degree t over $\mathbb{F}_2$. Finding such a polynomial is computationally much easier than most tasks considered in this paper.

Let $G : \{0,1\}^n \to \{0,1\}^{L(n)}$ and let A be an algorithm with binary output. We say that A is a $(L(n), T(n), \delta(n))$-*distinguisher* for G, if A runs in time $T(n)$ and $|\Pr_{x \in \mathcal{U}_n}[A(G(x)) = 1] - \Pr_{y \in \mathcal{U}_{L(n)}}[A(y) = 1]| \geq \delta(n)$. We call $\delta(n)$ the *advantage* of A. If no such A exists, G is called $(L(n), T(n), \delta(n))$-*secure*. Recall that a function $\delta(n)$ is *negligible* if for all $c > 0$, $\delta(n) = o(n^{-c})$.

A function f is *one-way* if, when y is computed as $y = f(x)$ for a uniformly random x, any probabilistic polynomial algorithm can find a x', which may or may not equal x, such that $f(x') = y$ only with negligible probability.

Our model of computation is slightly generous but realistic. We assume that simple operations like arithmetical operations and exclusive-ors on small size integers can be done in unit time. To be more precise we need words of size n where n is size of the input on which we apply our one-way function, e.g. $n = 128$ or 256 for a typical block cipher. We also, in many sampling procedures, ignore the cost of updating trivial counters.

2.2. *Probabilistic Facts*

For our analysis, we need to consider some Bernoulli random variables, $\{Y_i\}$, obtained as outputs of probabilistic algorithms. Specifically, we study the "centered" distributions $X_i = Y_i - E[Y_i]$, i.e. $E[X_i] = 0$ and $\Pr[X_i = 1 - p_i] = p_i$ and $\Pr[X_i = -p_i] = 1 - p_i$ for some $0 \leq p_i \leq 1$. Let $X = \sum_{i=1}^{s} X_i$ and suppose $p = \frac{1}{s}\sum_{i=1}^{s} p_i$. Let us first assume that the variables X_i are independent, then the following variant of Chernoff-bound is stated in [2] as Theorem A.1.4.

Lemma 2.1. $\Pr[X \geq a] \leq e^{-\frac{2a^2}{s}}$.

In some situations we have very biased random variables and in these cases the following bounds, found as Theorems A.1.11 and A.1.13 of [2], respectively, are useful.

Lemma 2.2. *Assume* $\beta > 1$, *then* $\Pr[X \geq (\beta - 1)ps] \leq (e^{\beta-1}\beta^{-\beta})^{ps}$.

Lemma 2.3. $\Pr[X \leq -a] \leq e^{-\frac{a^2}{2ps}}$.

When we do not have full independence we also use Chebychev's inequality which in the current framework, using $\sigma^2(X_i) \leq \frac{1}{4}$, has the following consequence.

Lemma 2.4. *If the variables are pairwise independent we have* $\Pr[X \geq a] \leq \frac{s}{4a^2}$.

Typically we use the above lemmas to prove that sampling gives an answer that is not too far away from the correct values. In those cases Y_i is a 0–1 variable that is adjusted to get expectation 0 and then a measures the distance of the sample from the expectation. In some cases we only have an upper estimate of $E[Y_i]$ and we want to prove that we do not get many more variables that take the value 1 than expected in the case when all unknown expectations take their maximal value. In those cases it is intuitively clear that analyzing the case when each Y_i indeed has the maximal expectation gives an upper bound for the general case. Formally this can be argued by introducing a coupled Boolean variable $\tilde{Y}_i$ for which $\tilde{Y}_i \geq Y_i$ is always true and $\tilde{Y}_i$ has the maximal expectation allowed for Y_i.

2.3. *Pseudo-Random Generators from One-Way Permutations*

Suppose we have a one-way function, f, that in addition is a permutation. Furthermore, suppose that we have a family of efficiently computable 0/1-functions, $B = \{b_r\}$, $b_r(x) \in \{0, 1\}$, such that given $f(x)$ and a randomly chosen r, $b_r(x)$ is computationally indistinguishable from a random 0/1 coin toss. We then say that B is a (family of) *hard-core predicates* for f. The following construction, due to Blum and Micali [5], shows how to construct a PRG: choose x_0, r (the seed), let $x_{i+1} = f(x_i)$, then output $g(x_0, r) = b_r(x_1), b_r(x_2), \ldots$ as the generator output.

Theorem 2.5 (Blum–Micali, 1984). *Suppose f is a permutation and there is a polynomial time algorithm D that distinguishes* (*with non-negligible advantage*) $g(x, r)$ *from* *a completely random string. Then, there is a polynomial time algorithm P that given r, $f(x)$ predicts $b_r(x)$ with non-negligible advantage.*

Due to the iterative construction, f must not lose one-wayness under iteration. This can be guaranteed if f is a permutation, or, heuristically if f is randomly chosen, as quantified in Theorem 2.8 below. Assumptions along these lines have been proposed by Levin [17] and were the first conditions to be proved to be necessary and sufficient for the existence of PRGs.

This leaves us with one question: which one-way functions (if any) have hard-cores, and if so, what do these hard-cores look like?

2.4. *A Hard-Core for any One-Way Function*

In 1989, Goldreich and Levin [10], proved that *any* one-way function has a hard-core. Perhaps surprisingly, the hard-core they found is also extremely simple to describe. If r, x are binary strings of length n, let r_i (and x_i) denote the ith bit of r (and x). Let $B \triangleq \{b_r(x) \mid r \in \{0,1\}^n\}$ where $b_r(x) \triangleq \langle r, x\rangle_2$ is the inner product mod 2.

Theorem 2.6 (Goldreich–Levin, 1989). *Suppose there is a polynomial time algorithm A, that given r, $f(x)$ for randomly chosen r, x, distinguishes $b_r(x)$ from a completely random bit, with non-negligible advantage. Then there exists a polynomial time algorithm B, that inverts $f(x)$ on a random x with non-negligible probability.*

If f is a one-way function, existence of such A would thus be contradictory.

We again stress that this does not automatically imply that a PRG can be built from any one-way function, as the construction by Blum and Micali only works for one-way permutations.

As established already in [10], a way to improve efficiency would be to extract more than one bit per iteration of f. In theory it is possible to output as many as $m \in O(\log n)$ (where $n = |x|$) bits, by multiplying the binary vector x by a random $m \times n$ binary matrix, R. Denote the set of all such matrices $\mathfrak{M}_m$, and the corresponding functions $\{B_R^m(x) \mid R \in \mathfrak{M}_m\}$. That is, $B_R^m(x) \triangleq Rx \bmod 2$.

2.5. *Iteration of One-Way Functions*

We first define a general experiment for inversion of functions.

Definition 2.7. For a function $f : \{0,1\}^n \to \{0,1\}^n$, let $f^{(i)}(x)$ denote f iterated i times, $f^{(i)}(x) \triangleq f(f^{(i-1)}(x))$, $f^{(0)}(x) \triangleq x$.

Let A be a probabilistic algorithm which takes an input from $\{0,1\}^n$ and has output in the same range. We then say that A is a (T, δ, i)-inverter for f if when given $y = f^{(i)}(x)$ for an x chosen uniformly at random, in time T with probability δ it produces z such that it has verified that $f(z) = y$.

Note that the number z might be on the form $f^{(i-1)}(x')$ but this is not required. Allowing A to output z without having computed $f(z)$ is not natural as f is assumed to be easy to compute and one more function evaluation verifies or rejects a candidate for z.

It is interesting to investigate what happens for a random function.

Theorem 2.8. *Let A be an algorithm that tries to invert a black box function $f : \{0,1\}^n \to \{0,1\}^n$, and makes T calls to the oracle for f. If A is given $y = f^{(i)}(x)$ for a random x, then the probability (over the choice of f and x) that A finds a z such that $f(z) = y$ is bounded by $2T(i+1)/(2^n - T) + i(i+1)2^{-(n+1)}$. On the other hand, there is an algorithm that, using at most T oracle calls, outputs a correct z except with probability $(1-(i+1)2^{-n})^{T-i}$.*

Proof. For the lower bound on the required number of oracle calls, consider the process of computing $f^{(i)}(x)$ and let W be the values of f computed in this process, $W = \{f^{(j)}(x) \mid j = 0, \ldots, i\}$. We will only consider the case when all points of W are distinct. The probability that we see the same value twice under the evaluation of $f^{(i)}(x)$ is bounded by $i(i+1)2^{-(n+1)}$ and in the case we assume that the inverter is successful.

We relax the inverter's task and consider it successful if it ever queries for a $w \neq y$ with $w \in W$, or, if it queries on any w with $f(w) \in W$. As $y \in W$ this gives an overestimate of the success probability.

Suppose that up to the k'th query the inverter has not already succeeded. In such a case the set W is completely independent of A and hence the probability that the next question is for an input $w \neq y$, $w \in W$ is bounded by $\|W\|/(2^n - k)$ and conditioned on the input not being in W, the probability of the obtained value $f(w)$ being in W is $\|W\|2^{-n}$. We conclude that the probability that the inverter is successful within its T queries is bounded by $2T\|W\|/(2^n - T)$.

For the upper bound consider the following algorithm.

Start by setting $x_0 = y$ and then $x_j = f(x_{j-1})$ for $j = 1, 2, \ldots$. We continue this process until either $x_j = y$, and we are done, or x_j is a value we have seen previously. In the latter case we change x_j to an arbitrary value we have not seen previously and continue.

Let us analyze this algorithm. If y occurs twice in W (W as above), then the answer is guaranteed to be found within i steps and hence we can assume that this is not the case and that the $i+1$ values in W are distinct. Until we have obtained a $x_j \in W$ each value x_j for $j \geq 1$ is a uniformly random value and as soon as we obtain an $x_j \in W$ we find y within at most i additional evaluations of f. This implies that the probability of not finding a preimage is bounded by $(1-(i+1)2^{-n})^{T-i}$. □

Note that, disregarding minor details, both the upper and lower bounds are of the form $\Theta(Ti2^{-n})$ for the probability of finding a preimage. Thus our characterization is rather tight.

Consider for instance the block cipher Rijndael [6] as a one-way function. As discussed in the introduction the one-way function is the mapping from the key to the encryption of a fixed plaintext. It is not unreasonable to expect this function to be almost as hard to invert as a random function, so that the best achievable time over success ratio to invert it after being iterated i times is, by the above theorem, about $2^n/i$. The security is defined as follows.

Definition 2.9. A *σ-secure one-way function* is an efficiently computable function f that maps $\{0,1\}^n \to \{0,1\}^n$, such that the average time over success ratio for inverting the ith iterate is at most $\sigma 2^n/i$. That is, f cannot be (T,δ,i)-inverted for any values of T, δ and i such that $T/\delta < \sigma 2^n/i$.

A block cipher, $f(p,k)$, $|p|=|k|=n$, is called σ-secure if the function $f_p(k)$, for fixed, known plaintext p, is a σ-secure one-way function of the key k.

Note that if f is a permutation, only the case $i=1$ is of interest and we have a standard notion of security.

2.6. *Generating Pairwise Independent Matrices*

Our proof is an extension of the proof by Rackoff (see [8]) for the original result of Goldreich and Levin. For the extension we need a way to generate pairwise independent matrices.

Lemma 2.10. *Fix any $x \in \{0,1\}^n$. For $m \le t$, from $m+t$ randomly chosen $a_0,\dots,a_{m-1}$ and $b_0,\dots,b_{t-1} \in \{0,1\}^n$, it is possible in time $2m2^t + t^2 + m + 4t$ to generate a set of 2^t uniformly distributed, pairwise independent matrices $R^1,\dots,R^{2^t} \in \mathfrak{M}_m$. Furthermore, there is a collection of $m\times(m+t)$ matrices $\{M^j\}_{j=1}^{2^t}$ and a vector $z \in \{0,1\}^{m+t}$ such $B^m_{R^j}(x) = M^j z$ for all j.*

Proof. Remember the convention that t-bit strings can be considered both as integers, elements of $\mathbb{F}_{2^t}$ and subsets of $[t]$. Let $\{\alpha_j\}_{j=1}^{2^t}$ be an enumeration of distinct elements of $\mathbb{F}_{2^t}$. Choose randomly and independently m strings, $a_0,\dots,a_{m-1}$ and t strings $b_0,\dots,b_{t-1}$, each of length n. The jth matrix, R^j is now defined by letting its ith row R^j_i be

$$R^j_i \triangleq a_i \oplus \left(\bigoplus_{l\in \mathrm{bin}(\alpha_j \cdot t^i)} b_l \right),$$

where the multiplication, $\alpha_j \cdot t^i$, is carried out in $\mathbb{F}_{2^t}$, and $\oplus$ is bitwise addition mod 2.

Clearly the matrices are uniformly distributed, since the a_i are chosen at random. To show pairwise independence it suffices to show that an exclusive-or of any subset of elements from any two matrices is unbiased. Since the columns are independent, it is enough to show that the exclusive-or of any non-empty set of rows from two distinct matrices R^{j_1} and R^{j_2} is unbiased. Consider any such set of rows, $S_1 \subset R^{j_1}$, and $S_2 \subset R^{j_2}$. If $S_1 \neq S_2$ then the addition of the vectors a_i ensures that the exclusive-or of the rows in unbiased and hence we need only consider the case $S_1 = S_2 = S$. In this case the exclusive-or can be written as

$$\bigoplus_{i\in S} \bigoplus_{l\in \mathrm{bin}((\alpha_{j_1}+\alpha_{j_2})\cdot t^i)} b_l,$$

but this is the same as

$$\bigoplus_{l\in \mathrm{bin}((\alpha_{j_1}+\alpha_{j_2})\cdot(\sum_{i\in S} t^i))} b_l,$$

which is unbiased if, and only if, $\mathrm{bin}((\alpha_{j_1}+\alpha_{j_2})\cdot(\sum_{i\in S} t^i)) \neq 0$. However, $\sum_{i\in S} t^i \neq 0$, and as $\alpha_{j_1} \neq \alpha_{j_2}$, we have two nonzero elements and hence their product is nonzero.

Notice that if we know $\langle a_i, x\rangle_2$ and $\langle b_l, x\rangle_2$ for all a_i, b_l (a total of $m+t$ bits), then by the linearity of the above construction, we also know the matrix-vector products $R^j x$ for all j.

To calculate all the matrices we first compute the reduction of t^i for all $i = t + 1, \dots, 2t$ in $\mathbb{F}_{2^t}$. Using an iterative procedure this can be done with $3t$ operations on t bit words and since we only care about $t \leq n$ these can be done in unit time. We generate the vectors a and b in time $m+t$ operations and compute $\bigoplus_{l\in \mathrm{bin}(t^i)} b_l$ for each $i = 0, \dots, 2t$ using t^2 operations. By using a gray-code construction each row of a matrix can now be generated with two operations and thus the total number of operations is $2m2^t + t^2 + m + 4t$. □

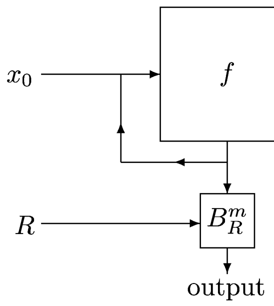
3. The Construction and Its Security

3.1. *The Basic PRG*

We start by defining our generator.

Definition 3.1. Let n, and m, L, λ be integers such that $L = \lambda m$ and let $f : \{0, 1\}^n \to \{0, 1\}^n$. The generator $BMGL^f_{n,m,L}(x, R)$ stretches $n + nm$ bits to L bits as follows. The input is interpreted as $x_0 = x$ and $R \in \mathfrak{M}_m$. Let $x_i = f(x_{i-1})$, $i = 1, 2, \dots, \lambda$ and let the output be $\{B^m_R(x_i)\}^{\lambda}_{i=1}$.

See also the figure below. We could output also the matrix R but we do not. In fact it might be better to think of it as a parameter of the generator and this point of view is useful for us in Sect. 3.3.



A proof of security for a concrete f and fixed n, m, requires an exact analysis, and that analysis is the bulk of this paper. As the parameters f, n, m and L remain fixed throughout the paper we sometimes feel free to suppress them.

3.1.1. *Security of the Generator*

We start by stating our main theorem.

Theorem 3.2. *Suppose that $G = BMGL^f_{n,m,L}$ is based on an n-bit function f, computable by E operations, and that G produces L bits in time S. Suppose that this generator can be (L, T, δ)-distinguished. Then, setting $\delta' = \frac{\delta m}{L}$, there exists integers $i \leq L/m \triangleq \lambda$, $0 \leq j \leq 2\log\delta'^{-1}$, such that for $t = \max(m, 2 + \lceil\log((2n+1)\delta'^{-2}) - j\rceil)$, f can be $(T', 2^{-(7/2+j)}, i)$-inverted, where T' equals*

$$(1 + o(1))2^{m+t}(2m + t + 1 + T + S + E)(n + 1).$$

Values of i and j such that f can be $(8T', ((j+1)(j+2)2^{(j+10)/2})^{-1}, i)$-inverted can, with probability at least $1/4$, be found in time $O(j^4 2^{-j/2}\delta'^{-2}(T + S))$.

The obtained dependence on the parameter j is somewhat arbitrary. We have concentrated on getting a good time/success tradeoff in the non-uniform case and not too high preprocessing time in the uniform case. Small values of j give similar results in both cases. As will be seen in the proof, the bounds in the uniform case for $j = 0$ are slightly better than stated.

A similar result could be obtained from the original works by Blum–Micali and Goldreich–Levin, but we are interested in a tight result and hence we have to be more careful than in [10] where, basically, any polynomial time reduction from inverting f to distinguishing the generator would be enough. Optimizations of the original proof also appeared in [18], but are not stated explicitly. We start with the following lemma.

Lemma 3.3. *Let $L = \lambda m$. Suppose that $BMGL^f_{n,m,L}$ runs in time $S(L)$. If this generator is not $(L, T(L), \delta)$-secure, then there is an algorithm $P^{(i)}$, $1 \leq i \leq L/m$ that, using $T(L) + S(L)$ operations, given $f^{(i)}(x)$, R, for random $x \in \mathcal{U}_n$, $R \in \mathfrak{M}_m$, distinguishes $B^m_R(f^{(i-1)}(x))$ from $\mathcal{U}_m$ with advantage $\delta' = \frac{\delta m}{L}$.*

$P^{(i)}$ depends on an integer i, and using $c_1(1 + o(1))\delta'^{-2}(T(L) + S(L))$ operations, where c_1 is the constant given by (11), a value of i achieving advantage $\delta_i \geq \delta'/2$ can be found with probability at least $1/2$.

We conjecture that the time needed to find i is optimal up to the value of the constant c_1. Even if a good value i was given at no cost, the straightforward way, by sampling, to verify that it actually is as good as claimed would take time $\Omega(\delta'^{-2}(T(L) + S(L)))$. It is not difficult to see that the below proof can be modified to find an i with δ_i arbitrarily close to δ' and such that the probability of success can be arbitrarily close to 1. This improvement comes at the expense of an increase in the constant c_1. The constants we have used should be considered only as an example rather than as a choice of any special significance.

The proof of Lemma 3.3 is a standard hybrid argument and the bulk of the work is to establish the uniform part i.e., how to efficiently find the parameter i. The reader may choose to skip the rather long proof on first reading and return here later.

Proof of Lemma 3.3. Define distributions H^i, $i = 0, 1, \ldots, \lambda$, on $\{0, 1\}^L$ such that H^i first produces an L-bit output of $BMGL$ but then replaces the first im bits by random bits. Clearly H^0 equals the distribution of outputs of $BMGL$ and H^λ is the uniform distribution on $\{0, 1\}^L$. Let D be the algorithm that distinguishes the output from *BMGL* from random bits and let ϵ_i be the probability that D outputs 1 on the distribution H^i. We let $\delta_i = \epsilon_i - \epsilon_{i-1}$ be the advantage of D in distinguishing H^i from H^{i-1}. From the triangle inequality follows the existence of an i, with $\delta_i \geq \delta'$. With this i known, one easily constructs the algorithm $P^{(i)}$.

For the uniform part of the lemma we need to find such an i efficiently and the problem is that even though $\mathrm{E}_i[\delta_i] = \delta'$, there is a large number of possibilities for the individual δ_i. Basically, these possibilities all lie between the two extreme cases: (1) There are a few large δ_i, while most are close to 0. (2) All δ_i are about the same, but none is very large. Suppose we try random i's. In the first case, we may need to try many i, but it can be done with a rather low sampling accuracy. In the second case, we expect to find a fairly good i rather quickly, but we need a higher precision in the sampling. The idea is therefore to divide the sampling into a number stages, each with different sampling accuracy. Stage j chooses some random i-values and samples D on H^i and H^{i-1}. As soon as a sufficiently good i is detected, the procedure terminates. As the number of samples needed to determine the advantage to accuracy ϵ is about ϵ^{-2} we can try many values of i when trying to find a very large δ_i. Formally we proceed as follows.

Define $b_0 = 32$, $b_1 = \lceil 2^{11/2}(1 - e^{-1})^{-1} \rceil$ and

$$b_j = \lceil 2^{11/2}(1 - e^{-1})^{-1}(2^{(j-1)/2} - 2^{(j-2)/2}) \rceil$$

for $j > 1$. Furthermore let $T_0 = 350$ and $T_j = 2^{2-j}(6 + j)/2$ for $j \geq 1$.

In stage j, for $j = -2\log\delta', -2\log\delta' - 1, \ldots, 0$, choose b_j different random values of i and sample H^i and H^{i-1} each $t_j = \lceil T_j \delta'^{-2} \rceil$ times and run D on each of the samples. Let c_i^j be the difference in the number of times one is output in the two distribution. Halt and output i if either

1. $j \geq 1$ and $c_i^j \geq 2^{(1+j)/2} t_j \delta' - \sqrt{t_j}$.
2. $j = 0$ and $c_i^0 \geq 5 t_0 \delta'/8$.

If no i is ever chosen halt with failure.

We need to analyze the procedure. We have

$$\sum_{i=0}^{\lambda-1} \delta_i = \lambda\delta'$$

and thus either

$$\sum_{i \mid \delta_i \geq 2\delta'} \delta_i \geq \lambda\delta'/8 \tag{1}$$

or

$$\sum_{i \mid \delta_i < 2\delta'} \delta_i \geq 7\lambda\delta'/8. \tag{2}$$

We prove that in the former case we are likely we find a good output in a stage with $j \geq 1$ and in the latter case we are likely to succeed for $j = 0$. We start with the first case and let us assume that $j \geq 1$.

Suppose that at stage j an i is picked such that $\delta_i \geq 2^{(1+j)/2}\delta'$. Then, by Lemma 2.1, i is output with probability at least $(1 - \frac{1}{e})$. Let a_j be the fraction of i such that $\delta_i \geq 2^{(1+j)/2}\delta'$. We claim that

$$a_1 + \sum_{j=2}^{-2\log\delta'} a_j (2^{(j-1)/2} - 2^{(j-2)/2}) \geq 2^{-9/2}. \tag{3}$$

To see this, for i such that $\delta_i \geq 2\delta'$, let $\tilde{\delta}_i$ be the largest value of form $2^{(1+j)/2}\delta'$ which is smaller than or equal to δ_i, and for all other i, let $\tilde{\delta}_i = 0$. Then, by (1), we have

$$\sum_i \tilde{\delta}_i \geq \lambda\delta' 2^{-7/2}. \tag{4}$$

Now notice that

$$\begin{aligned} \frac{1}{\lambda}\sum_i \tilde{\delta}_i &= 2a_1\delta' + \sum_{j=2}^{-2\log\delta'} a_j (2^{(1+j)/2} - 2^{j/2})\delta' \\ &= 2\delta' \left(a_1 + \sum_{j=2}^{-2\log\delta'} a_j (2^{(j-1)/2} - 2^{(j-2)/2}) \right) \end{aligned} \tag{5}$$

and clearly, (4) and (5) implies (3).

By the above reasoning the probability that the algorithm terminates in an individual iteration during stage j is at least $a_j(1 - e^{-1})$. The probability that the algorithm will fail to output an i in a stage with $j \geq 1$ is thus bounded by

$$\prod_{j=1}^{-2\log\delta'} (1 - a_j(1 - e^{-1}))^{b_j} \leq e^{-(1-e^{-1})\sum_j a_j b_j} \leq e^{-2}$$

where the last inequality follows from (3) and the definition of b_j. This takes care of the case when (1) holds. Let us now establish that when (2) holds we are likely to find a good i in stage $j = 0$.

We must have at least $\lambda/16$ different i such that $\delta_i \geq 3\delta'/4$ since otherwise

$$\sum_{i \mid \delta_i < 2\delta'} \delta_i < 2\delta'\lambda/16 + \lambda 3\delta'/4 = 7\lambda\delta'/8. \tag{6}$$

This implies that in our 32 attempts the probability that we do not try an i with $\delta_i \geq 3\delta'/4$ is bounded by

$$(15/16)^{32} \leq e^{-2}.$$

Given that we sample using such an i the probability that we will not output it is, by Lemma 2.1 with $a = \delta' t_0/8$ and $s = 2t_0$ bounded by

$$e^{-\frac{t_0\delta'^2}{64}} \leq e^{-T_0/64}. \tag{7}$$

We must also bound the probability that algorithm terminates with an i such that $\delta_i \leq \delta'/2$ before finding a good value of i. Let us analyze the probability that such a bad i would be output during an individual run of stage j provided that it is chosen as a candidate. Let us first consider $j \geq 1$.

To halt we need an observed difference of at least

$$2^{(1+j)/2} t_j \delta' - \sqrt{t_j}$$

while the expected difference is at most $t_j\delta'/2$ and thus we are at least

$$\left(2^{(1+j)/2} - \frac{1}{2}\right) t_j \delta' - \sqrt{t_j} \geq 2^{j/2} t_j \delta' - \sqrt{t_j} \tag{8}$$

away from the mean. By Lemma 2.1 this implies that the probability of halting in a particular run of stage j is bounded by

$$\exp\left(-\frac{2(2^{j/2} t_j \delta' - \sqrt{t_j})^2}{2t_j}\right) = e^{-(2^{j/2}\sqrt{t}_j\delta' - 1)^2}.$$

By the definition of t_j we have

$$2^{j/2}\sqrt{t}_j\delta' - 1 \geq 2\sqrt{(6+j)/2} - 1$$

and thus the probability to end with an output with $\delta_i \leq \delta'/2$ in any stage $j \geq 1$ is bounded by

$$\sum_{j=1}^{-2\log\delta'} e^{-(2\sqrt{(6+j)/2}-1)^2} b_j \leq \sum_{j=1}^{-2\log\delta'} e^{-(2\sqrt{(6+j)/2}-1)^2} 2^{(10+j)/2} (1 - e^{-1})^{-1} \leq e^{-2} \tag{9}$$

where the first inequality follows from the definition of b_j and the second inequality is due to a computer calculation.

For $j = 0$ and any picked i the probability of outputting the given i if $\delta_i \leq \delta'/2$ is also bounded by (7) as this time to get this result the outcome of the sampling has to be $t_0\delta'/8$ on the other side of the mean. Now observe that

$$e^{-T_0/64} = e^{-350/64} \leq e^{-2}/32.$$

Let us sum up. If (1) holds then the probability of finding a good i in a stage with $j \geq 1$ is $(1 - e^{-2})$. The probability of halting with an output i for which $\delta_i \leq \delta'/2$ is bounded by e^{-2}. We conclude that we halt with an i with probability at least $1 - 2e^{-2} \geq \frac{1}{2}$ in this case.

If (2) holds then, by (9) the probability of ending with a i with $\delta_i \le \delta'/2$ in a stage $j > 0$ is bounded by e^{-2}. In stage 0 let us say that sampling is successful for a given i if $\delta_i \le \delta'/2$ and we do not halt or $\delta_i \ge 3\delta'/4$ and we do halt. If δ_i is in the intermediate range, sampling is always considered successful. By the above analysis sampling is successful with probability at least $(1 - e^{-T_0/32})^{32} \ge (1 - e^{-2})$. The probability of never choosing an i with $\delta_i \ge 3\delta'/4$ is bounded by e^{-2}. We conclude that we halt with an i with $\delta_i \ge \delta'/2$ with probability at least

$$(1 - e^{-2})^3 \ge \frac{1}{2}.$$

Finally, apart from trivial arithmetic operations the running time of the procedure is given by the number of samples produced which is bounded by

$$\sum_{j=0}^{\infty} 2b_j t_j, \tag{10}$$

and this, up to low order terms, equals

$$\delta'^{-2} \sum_{j=0}^{\infty} 2b_j T_j \triangleq c_1 \delta'^{-2}. \tag{11}$$

As $b_j \in O(2^{j/2})$ and $T_j \in O(j 2^{-j})$ the sum converges and can numerically be checked to bounded by 24460. The dominating term is the first term $2b_0 T_0 = 22400$. This completes the proof. □

The value of the constant c_1 can be improved significantly. The bulk of the work is done for small values of j and here we can save time by more quickly abandoning a choice of i that does not look promising.

We now give the theorem of Goldreich and Levin [10] trying to be careful with our estimates and construction. Apart from the value of the constants we have an improvement over previous results in the dependence on the parameters m and ϵ. Previous constructions, reducing the general case of $m > 1$ to that of $m = 1$ would reduce the distinguishing advantage from ϵ to $\epsilon/2^m$ and thus, by standard analysis, would require a sampling complexity on the order $2^{2m}\epsilon^{-2}$. By avoiding this reduction and using a more direct analysis, we are able to reduce this dependence to roughly $2^m \cdot \max(2^m, \epsilon^{-2})$, gaining either a factor2^m, or ϵ^{-2}.

Theorem 3.4. *Fix x. Suppose there is an algorithm, P, using T operations, when given random R distinguishes $B_R^m(x)$ from random strings of length m with advantage at least ϵ where ϵ is given. Then, for $t \triangleq \max(m, \lceil \log(\epsilon^{-2}(2n+1)) \rceil)$, we can in time*

$$(1 + o(1))2^{m+t}(2m + t + 1 + T)(n + 1)$$

produce a list of $2^{t+m}(n + 1)$ values such that the probability that x appears in this list is at least $1/2$.

As we understand, a statement similar (up to a constant) to Theorem 3.4, for the special case of $m = 1$, can be derived from [18]. We prove now that we can compute useful information about x.

Lemma 3.5. *Let P, T, x and ϵ be as in Theorem 3.4. Then for any set of N vectors $\{v_i\}_{i=1}^N \in \{0,1\}^n$ and any $t \geq m$ we can in time $(1+o(1))2^{m+t}(2m+t+T+1)(N+1)$ produce a set of lists $\{b_i^{(k)}\}_{i=1}^N$, $k = 1, 2, \ldots, 2^{t+m}(N+1)$ such that with probability $1/2$ we have for at least one k, $\langle x, v_i\rangle_2 = b_i^{(k)}$, except for at most $\frac{N}{\epsilon^2 2^{t-1}}$ of the N possible values of i.*

Before we establish Lemma 3.5 let us use it to prove Theorem 3.4.

Proof of Theorem 3.4. Set $t = \max(m, \lceil \log(\epsilon^{-2}(2n+1))\rceil)$. We apply Lemma 3.5 with $N = n$, and let $\{v_i\}_{i=1}^n$ be the unit vectors so that $\langle v_i, x\rangle_2$ gives the ith bit of x. With probability $1/2$ one list gives all inner-products correctly and hence determine x. $\square$

We continue to establish Lemma 3.5.

Proof of Lemma 3.5. Start by randomly generating the 2^t pairs of matrices $\{R^j, M^j\}$ as described in Lemma 2.10. Now repeat the process below for each $i = 1, \ldots, N$.

Select 2^t (pairwise) independent random strings $s^j \in \{0,1\}^m$, and let S_i^j be the $m \times n$ matrix defined by $S_i^j \triangleq s^j \otimes v_i$, the outer product, i.e., $(S_i^j)_{t,l} = (s^j)_t \cdot (v_i)_l$. By linearity

$$(R^j + S_i^j)x = R^j x + s^j \langle v_i, x\rangle_2, \tag{12}$$

which is $B_{R^j}^m(x)$ if $\langle v_i, x\rangle_2 = 0$, and a random string otherwise.

We want to compute the following statistic:

$$c_l^i = 2^{-t} \sum_{j=0}^{2^t-1} P(R^j + S_i^j, M^j z_l) \tag{13}$$

for all 2^{t+m} values of z_l. Naively this seems to require 2^{2t+m} operations for each i but as described in Sect. 3.2 below it is possible to compute these values for fixed i with $(2m+t+T)2^{m+t}$ operations.

Focus on the choice for l that gives $M^j z_j = B_{R^j}^M(x)$. If $\langle v_i, x\rangle_2 = 0$, then c_l^i is the outcome of a uniform random, pairwise independent sample values of the form $\{P(R, B_R^m(x))\}$. On the other hand, if $\langle v_i, x\rangle_2 = 1$, we claim that it is a sample of $\{P(R, u)\}$ over random u. Furthermore the values of R and u are pairwise independent.

To see that u is uniformly distributed note that for $R = (R^j + S_i^j)$, the distinguisher P is called on $\{P(R, B_R^m(x) + s^j)$. Fixing the value R, all possible values of R^j are equally likely and hence each possible S_i^j is equally likely giving a uniform distribution of s^j. Pairwise independence follows from pairwise independence of R^j and S^j.

Suppose p_R is the probability that $P(R, B_R^m(x)) = 1$ for random R and x and let p_U be the same probability that $P(R, u) = 1$ for random R and u. Note by assumption we have $p_R = p_U + \epsilon$. Let $p \triangleq (p_R + p_U)/2$. In reality we do not know the value of p but for the moment let us assume we do.

We guess that $\langle v_i, x\rangle_2 = 0$ if $c_l^i \geq p$ and $\langle v_i, x\rangle_2 = 1$ otherwise. The choice is correct unless the average of 2^t pairwise independent Boolean variables is at least $\epsilon/2$ away from its mean. By Lemma 2.4 this happens with probability at most $2^{-t}\epsilon^{-2}$.

This implies that for the correct values of l and p, the expected number of errors is $2^{-t}\epsilon^{-2}N$, and by Markov's inequality, with probability at least at $1/2$ it is below $2^{1-t}\epsilon^{-2}N$. There are 2^{t+m} possible values of l and once l is fixed the only information on p needed is for which $i \in [1..N]$ we have $c_l^i \geq p$, i.e. how the c_l^i values are partitioned around p. There are only $N+1$ possibilities for this, and we can thus try them all.

The time needed to construct the matrices is negligible, computing the values c_l^i can be done it time $2^{t+m}(2m+t+T)N$, and at most time $2^{t+m}(N+1)$ is needed to output the final lists. □

We could have reduced the number of lists by assuming that we know at least an approximate value of p. This does not, however, affect the overall complexity in any significant way and introduces additional non-uniformity in the non-uniform model and additional possibilities for sampling errors in the uniform model and hence we have chosen not to use this possibility.

We can now use Theorem 3.4 to establish Theorem 3.2. Also this proof is quite long in the uniform case, so the reader may choose to skip this part of the proof on first reading.

Proof of Theorem 3.2. First we apply Lemma 3.3 to see that there is an i for which we have an algorithm $P^{(i)}$ that when given $f^{(i)}(x)$ runs in time $S(L)+T(L)$ and distinguishes $B_R^m(f^{(i-1)}(x))$ from random bits with advantage at least δ'', where δ'' is $\delta'/2$ or δ' depending on whether we want to find i efficiently, or only show existence (i.e. uniform/non-uniform algorithm). Since δ'' is an average over all x we need to do some work before we can apply Theorem 3.4.

For each x we have an advantage δ_x. For $j \geq 0$, let a_j be the fraction of x with $\delta_x \geq 2^{(j-2)/2}\delta''$. As the expectation of δ_x is δ'' the same reasoning that lead to (3) gives

$$a_0/2 + \sum_{j=1}^{\infty} a_j\left(2^{(j-2)/2} - 2^{(j-3)/2}\right) \geq 2^{-3/2}. \tag{14}$$

Let us first describe a non-uniform algorithm. Let j_0 be the value of j that maximizes $a_j 2^j$ and apply Theorem 3.4 with $\epsilon = 2^{(j_0-2)/2}\delta'' = 2^{(j_0-2)/2}\delta'$. Let us analyze this algorithm.

The running time is immediate from Theorem 3.4 and the probability of success is at least $a_{j_0}/2$ which we claim is at least $2^{-(7/2+j_0)}$. To see this note that if this is not the case then, by the choice of j_0,

$$a_j \leq 2^{-(5/2+j)}$$

holds for all j and hence

$$\sum_{j=0}^{\infty} a_j 2^{(j-2)/2} \le \sum_{j=0}^{\infty} 2^{-(j+7)/2} \le 2^{-7/2}\frac{1}{1-2^{-1/2}} < 2^{-3/2}$$

contradicting (14). This concludes the proof in the non-uniform case and let us address the question how to efficiently find a good value of j. We proceed by a similar procedure as used in the proof of Lemma 3.3.

Let d be a constant to be determined and define

$$b_j = \lceil (j+1)(j+2)^2 2^{(8+j)/2} \rceil$$

for $j \ge 2$, $T_j = d(j+1)2^{-j}$, and set $t_j = \lceil T_j \delta''^{-2} \rceil$.

In stage j, $j = -2\log\delta'', -2\log\delta'' - 1, \ldots, 2$ choose b_j different independent random values of x and run $P^{(i)}$ t_j times in each of the cases when the auxiliary bits are given by $B_R^m(f^{(i-1)}(x))$ or as uniformly random bits. Let c^j be the number of chosen x for which the difference in the number of times one is output on the two distributions is at least $2^{(j-2)/2}\delta'' t_j - \sqrt{t_j}$. If $c^j \ge 4(j+2)$ apply Theorem 3.4 with $\epsilon = 2^{(j-3)/2}\delta''$ and halt. If no j is ever chosen apply Theorem 3.4 with $\epsilon = \delta''/2$.

Let us analyze this procedure. First we claim that there is a $j \ge 2$ with

$$a_j \ge ((j+1)(j+2)2^{(1+j)/2})^{-1} \tag{15}$$

or that $a_0 \ge \frac{1}{4}$. Indeed suppose this is not the case then

$$\begin{aligned} &a_0/2 + \sum_{j=1}^{\infty} a_j (2^{(j-2)/2} - 2^{(j-3)/2}) \\ &\quad < a_0 2^{-1/2} + \sum_{j=2}^{\infty} ((j+1)(j+2)2^{(1+j)/2})^{-1} 2^{(j-2)/2} \\ &\quad < 2^{-5/2} + 2^{-3/2} \sum_{j=2}^{\infty} \frac{1}{(j+1)(j+2)} = 2^{-3/2} \end{aligned}$$

contradicting (14). We want to prove that whenever we encounter a $j \ge 2$ that satisfies (15) the probability of halting is significant.

For each picked x during stage j the probability that $\delta_x \ge 2^{(j-2)/2}\delta''$ and the sampling gives a difference of at least $2^{(j-2)/2}\delta'' t_j - \sqrt{t_j}$ is at least $a_j(1-e^{-1})$. Thus the expected value of c^j is at least

$$b_j a_j (1-e^{-1}) \ge 2^{7/2}(1-e^{-1})(j+2) \ge 7(j+2).$$

By Lemma 2.3 the probability that we do not halt with this value of j is bounded by

$$\exp\left(-\frac{(3(j+2))^2}{14(j+2)}\right) = \exp\left(-\frac{9(j+2)}{14}\right) \le e^{-2}.$$

We need also analyze the probability that we halt with a bad j before we find the good j as described above. We prove that it is unlikely to end in a stage j for which

$$a_{j-1} \leq ((j+1)(j+2)2^{(8+j)/2})^{-1}. \tag{16}$$

Consider any x chosen in stage j. The probability that $\delta_x \geq 2^{(j-3)/2}$ is a_{j-1}. If $\delta_x \leq 2^{(j-3)/2}$ the probability that the difference in the sampling observed is above $2^{(j-2)/2}\delta'' t_j - \sqrt{t_j}$ is bounded, by Lemma 2.1, by

$$\exp\left(-\frac{2((2^{(j-2)/2} - 2^{(j-3)/2})\delta'' t_j - \sqrt{t_j})^2}{2t_j}\right) \leq e^{-((2^{(j-2)/2} - 2^{(j-3)/2})\delta'' \sqrt{t_j} - 1)^2}.$$

Now, as

$$t_j \geq T_j \delta''^{-2} = d(j+1)2^{-j}\delta''^{-2},$$

it follows that for sufficiently large value of d that the probability that an x with $\delta_x \leq 2^{(j-3)/2}\delta''$ gives the required difference is at most

$$((j+1)(j+2)2^{(j+8)})^{-1}$$

and given (16) it follows that the expected value of c^j is at most $2(j+1)$. By Lemma 2.2 with $\beta = 2$ it follows that the probability that j is output is bounded by

$$\left(\frac{e}{4}\right)^{-2(j+2)}.$$

Thus the probability of outputting any j such that (16) is true is bounded by

$$\sum_{j=2}^{\infty} \left(\frac{e}{4}\right)^{-2(j+2)} \leq e^{-2}.$$

Summing up, with probability at least $(1 - e^{-2})^2$ we either halt with a $j \geq 2$ such that (16) is false or by calling the algorithm of Theorem 3.4 with $\epsilon = \delta''/2$ and having $a_0 \geq \frac{1}{4}$.

In the first case as the algorithm of Theorem 3.4 is called with $\epsilon = 2^{(j-3)/2}\delta'' = 2^{(j-5)/2}\delta'$ the running time is 8 times that of the non-uniform case. The pre-processing time to find j is proportional to

$$\sum_{k \geq j} b_k T_k$$

and as the success probability is at least $a_{j-1}/2$ the conclusion of Theorem 3.2 with this value of j holds.

In the case when the algorithm of Theorem 3.4 is called with $\epsilon = \delta''/2 = \delta'/4$ the statement of Theorem 3.2 with $j = 0$ is true with some margin, as success probability is at least 1/8 and running time is at most $4T'$. □

Instead of applying Lemma 3.5 with the unit vectors we can, as suggested in [8], use it with $\{v_i\}$ describing the words of an error correcting code, e.g. a suitable Goppa-code [19]. (Similar ideas appears in [17].) If we have code words of length N, containing n information bits, and we are able to efficiently correct e errors we get:

Theorem 3.6. *Fix x. Suppose there is an algorithm, P, that using T operations given R distinguishes $B_R^m(x)$ from random strings of length m with advantage ϵ where ϵ is given. Suppose further we have a linear error correcting code, with n information bits, N message bits that is able to correct e errors in time T_C. Then setting $t = \max(m, \lceil \log(\epsilon^{-2}(2N+1)/e)\rceil)$ we can in time*

$$(1+o(1))2^{m+t}(2m+t+1+T+T_C)(N+1)$$

produce a list of $2^{t+m}(N+1)$ numbers such that the probability that x appears in this list is at least $1/2$.

Proof. We apply Lemma 3.5 with the given value of t and $\{v_i\}_{i=1}^N$ given by the row vectors of the generator matrix of the error correcting code. Running the decoding algorithm on each obtained "codeword" gives a list as claimed. □

Similar to Theorem 3.2, this translates to the quality of the inverter. We only state the resulting algorithm in existential form using O-notation.

Theorem 3.7. *Suppose we have a linear error correcting code with n information bits, $O(n)$ message bits that is able to correct $\Omega(n)$ errors in time T_C and that $G = BMGL_{n,m,L}^f$ is based on an n-bit function f, computable by E operations, and that G produces L bits in time S. If G can be (L, T, δ)-distinguished then, with $\delta' = \frac{\delta m}{L}$, there is an $i \leq L/m \triangleq \lambda$ and $0 \leq j \leq 2\log\delta'^{-1}$ such that for $t = \max(m, O(1) + 2\log\delta'^{-1} - j)$ such that f can be $(T', \Omega(2^{-j}), i)$-inverted where T' equals*

$$O(2^{t+m}(t+m+S+T+E+T_C)n).$$

In particular this implies that the asymptotic time-success ratio decreases by a factor n for the parameters discussed after Theorem 3.2.

3.2. *Computing the Numbers in* (13)

We want to prove the following lemma.

Lemma 3.8. *For each i it is possible to compute the 2^{m+t} numbers*

$$c_l^i = 2^{-t}\sum_{j=0}^{2^t-1} P(R^j + S_i^j, M^j z_l) \tag{17}$$

in time $2^{m+t}(2m+t+T)$.

Proof. We make use of the fast Fourier transform and let us briefly recall some facts. Let $\{\chi_u(z) = (-1)^{\langle u,z\rangle_2}\}$ be the characters. Then any function $g : \{0, 1\}^s \to \{-1, 1\}$ can be expressed as

$$g(z) = \sum_{u=0}^{2^s-1} (-1)^{\langle z,u\rangle_2} \hat{g}_u,$$

where $\hat{g}_u \triangleq E_v[\chi_u(v)g(v)] = 2^{-s} \sum_v (-1)^{\langle u,v\rangle_2} g(v)$. This is the discrete Fourier series expansion of g and, by a standard algorithm, these numbers can be computed with $s2^s$ operations.

First run P on all the 2^{m+t} possible inputs of form $(R^j + S_i^j, r)$ and record the answers: $\{P(R^j + S_i^j, r)\}$. Now

$$2^{-t} \sum_{j=0}^{2^t-1} P(R^j + S_i^j, M^j z_l) = 2^{-t} \sum_{j=0}^{2^t-1} \sum_{r=0}^{2^m-1} P(R^j + S_i^j, r)\Delta(r, M^j z_l), \tag{18}$$

where $\Delta(r, r') = 1$ if $r = r'$ and $\Delta(r, r') = 0$ otherwise. We have that

$$\Delta(r, r') = 2^{-m} \sum_{\alpha \subseteq [0..m-1]} (-1)^{\langle r \oplus r', \alpha\rangle_2}.$$

This implies that the sum (18) equals

$$\begin{aligned} c_l^i &= 2^{-(m+t)} \sum_{j,r,\alpha} P(R^j + S_i^j, r)(-1)^{\langle r \oplus M^j z_l, \alpha\rangle_2} \\ &= 2^{-(m+t)} \sum_{j,\alpha} (-1)^{\langle M^j z_l, \alpha\rangle_2} \sum_r P(R^j + S_i^j, r)(-1)^{\langle r,\alpha\rangle_2}. \end{aligned}$$

Let $Q(j, \alpha)$ be the inner sum and fix a value of j. The inner sums exactly give the Fourier transform of the numbers $P(R^r + S_i^j, r)$ and hence the 2^m different numbers $Q(j, \alpha)$ can be calculated in time $m2^m$ for this fixed j and hence all the numbers $Q(j, \alpha)$ can be computed in time $m2^{t+m}$. Finally we have

$$c_l^i = 2^{-(m+t)} \sum_{j,\alpha} (-1)^{\langle M^j z_l, \alpha\rangle_2} Q(j, \alpha) = 2^{-(m+t)} \sum_{j,\alpha} (-1)^{\langle z_l, M^{j,T} \alpha\rangle_2} Q(j, \alpha),$$

where $M^{j,T}$ is the transpose of M^j. But this is just a rearrangement (induced by $M^{j,T}$) of the standard Fourier-transform of size 2^{t+m} and can be computed with $(t + m)2^{t+m}$ operations. The lemma follows. □

3.3. *Applying the GGM Construction*

As shown, the *BMGL* generator can produce any number of output bits. We here investigate an alternative way, inspired by a construction of pseudo-random functions due to Goldreich, Goldwasser and Micali [9]. It has the advantage that we iterate f fewer times and hence the assumption needed for security is weaker.

The construction can be based on any PRG, $G : \{0,1\}^n \to \{0,1\}^{2n}$, but we use $G = G(x, R) = BMGL^f_{n,m,2n}(x, R)$ for some f. This generator is applied to many seeds and here it is important that we treat R as a public parameter rather than as part of the seed. Thus all executions of G use the same random R and we denote this generator by $G^{f,R}$ and we let $G^{f,R}_0(x)$ $(G^{f,R}_1(x))$ be the first (last) n bits of $G^{f,R}(x)$.

First, let us assume that we know in advance how may output bits that are desired. We apply [9] to obtain $2^d n$ output bits (where d is given) from $n(m+1)$-bits.

Definition 3.9. Fix $n, d \in \mathbb{N}$. For $x \in \{0,1\}^n$, $s \in \{0,1\}^d$ put

$$g_x(s) \triangleq G^{f,R}_{s_d}(G^{f,R}_{s_{d-1}}(\cdots G^{f,R}_{s_2}(G^{f,R}_{s_1}(x))\cdots)),$$

and define $GGM^{f,R}_{d,n} : \{0,1\}^n \to \{0,1\}^{2^d n}$ by

$$GGM^{f,R}_{d,n}(x) \triangleq g_x(00\ldots0), g_x(00\ldots1), \ldots, g_x(11\ldots1)$$

(the concatenation of g_x applied to all d-bit inputs).

The construction can be pictured as a full binary tree $T = (V, E)$ of depth d. Associate $v \in V$ with its breath-first order number; the root is 1 and the children of v are $2v, 2v+1$. Given x, the root is first labeled by $\mathcal{L}(1) = x$. For a non-leaf v labeled $\mathcal{L}(v) = y \in \{0,1\}^n$, label its children by $\mathcal{L}(2v) = G_0(y)$, $\mathcal{L}(2v+1) = G_1(y)$, respectively. The output of $GGM^{f,R}_{d,n}$ is the concatenation of all the leaf-labels of the tree.

Notice an advantage of the above method in the case that $G = BMGL^f_{n,m,2n}$. To produce $L = 2^d n$ bits, each application of G iterates f $2n/m$ times instead of $2^d n/m$, which, in light of Theorem 2.8, retains more of the one-wayness of f.

Lemma 3.10. *Suppose $m|2n$ and $L = 2^d n = \lambda m$ and that $GGM^f_{d,n}$ runs in time $S(L)$. If this generator is not $(L, T(L), \delta)$-secure, then there is an algorithm $P^{(i,j)}$, $1 \le j \le 2^d - 1$, $1 \le i \le 2n/m$ that, using $T(L) + S(L)$ operations, that given $f^{(i)}(x)$, R, for random $x \in \mathcal{U}_n$, $R \in \mathfrak{M}_m$, distinguishes $B^m_R(f^{(i-1)}(x))$ from $\mathcal{U}_m$ with advantage $\delta' = \frac{\delta m}{2L}$.*

$P^{(i,j)}$ depends on the integers i and j, and using $c_1(1 + o(1))\delta'^{-2}(T(L) + S(L))$ operations, where c_1 is the constant given by (11), values of i and j achieving advantage $\delta_i \ge \delta'/2$ can be found with probability at least $1/2$.

Note that the value of δ' is half the size compared to that of Lemma 3.3 but on the other hand we have a much smaller value on the number of iterations, i.

The proof is basically the same as in [9], and we only give a sketch for the purpose of self-containment. We use the integer j to pinpoint a location in the construction where we change the output of G and i controls how many of bits of the interesting output of G that we change to random bits.

Proof sketch. Consider the binary tree, T, describing a computation of $GGM^{f,R}_{d,n}$ as above. The tree has depth d, $2^d - 1$ internal vertices and 2^d leaves. We construct hybrid distributions $H^0, \ldots, H^{2^d-1}$ on the vertex-labels of such trees. Again, associate each $v \in V$ by its breadth-first order number. Then, H^j is defined by a simulation algorithm, $GGM^j(x)$, which on input x, assigns labels as follows. Assign the root, $v = 1$, the label x. For $v \in V$, $v = 1, 2, \ldots, j$, label v's children by letting $\mathcal{L}(2v), \mathcal{L}(2v + 1)$ be independent, random n-bit strings. Then, for $v = j + 1, \ldots, 2^d - 1$: $\mathcal{L}(2v) = G_0(\mathcal{L}(v))$, $\mathcal{L}(2v + 1) = G_1(\mathcal{L}(v))$. Finally return labels of the leaves in T.

Observe that H^{2^d-1} gives the uniform distribution over the labels (in particular the leaves) and H^0 labels the vertices exactly as $GGM^{f,R}_{n,d}$ does on a random seed x. Since D_1 distinguishes $GGM^{f,R}_{d,n}(x)$ from random 2^d n-bit strings with advantage δ, for some $j \leq 2^d$, it must be the case that D_1 distinguishes H^j, H^{j+1} with advantage at least $2^{-d}\delta$. This distinguishes the output of G from random bits and using the proof of Lemma 3.3 we find good values of i and j. □

3.3.1. *Unknown Output Length*

If the length of the "stream" is unknown beforehand, we let the basic generator G expand n bits to $3n$ bits. Apply the tree-construction as above, labeling left/right children by the first, respectively second n-bit substring of G's output. The remaining n bits are used to produce an output at each vertex as we traverse the tree breadth-first. The analysis is analogous and is omitted. To save memory, the traversal can be implemented in iterative depth-first fashion.

3.4. *Concrete Examples*

What does all this say? Suppose that we base the construction on Rijndael$(x) \triangleq$ Rijndael$_x(p)$ (for a fixed plaintext p) and that we want to generate $L = 2^{30}$ bits, applying our construction with $m = 32$. One choice of parameters gives the following corollary. We state the corollary in the non-uniform case and in such an important application, (almost) optimal parameters would be found once and for all.

Corollary 1. *Consider $G = BMGL^{\text{Rijndael}}_{256,32,2^{30}}$ (using key/block length 256) and where Rijndael is computable by E operations, and assume that G runs in time S. If G can be $(2^{30}, T, 2^{-32})$-distinguished, then there is $i < 2^{25}$, and $0 \leq j \leq 114$ such that setting $t = \max(32, 123 - j)$, Rijndael can be $(T', 2^{-(7/2+j)}, i)$-inverted for $T' = 2^{41+t}(65 + t + T + S + E)$.*

Similarly, using $GGM^{\text{Rijndael}}_{22,256}$ (to generate the same length outputs), the result holds with t replaced by $t + 2$ and some $i < 16$.

This is simply substituting the parameters and noting that the $o(1)$ in Theorem 3.2 comes from disregarding the time to construct the matrices described in Lemma 2.10 and for the current choice of parameters using $(1 + o(1))(n + 1) \leq 2^9$ is an overestimate.

Assuming we have a simple statistical test such as Diehard tests [20], or those by Knuth [16], it is reasonable to assume[1] that $65 + t + T + E \leq S$. From the first part of the corollary, then, the essential part of computing the generator comes from the 2^{25} computations of Rijndael and we end up with a time for the inverter equivalent to at most 2^{67+t} Rijndael computations. Then, since $t = 123 - j$ this is 2^{190-j} Rijndael computations and we have time-success ratio at most 2^{194} computations of Rijndael and since $i \leq 2^{25}$, Rijndael would not be 2^{-37}-secure.

Alternatively, bootstrapping the *BMGL* construction by the GGM method, we conclude from the second part of the corollary that such a test would mean that Rijndael cannot be even 2^{-56}-secure. Thus, though somewhat more cumbersome to implement, the GGM method is more security preserving.

4. Re-keying

The classical definition of a PRG produces a single, "unbounded" key stream. However, in many application, it is desirable to generate several "independent" random streams from the same seed, re-keying each by some other information. For instance, in an application as a key-derivation function, it might be desirable to derive both encryption and authentication keys from one and the same master seed. Another example is when the PRG is used as a stream cipher: data is often packed into chunks, and these chunks may be lost or reordered during transmission. Hence it is desirable to very efficiently seek back and forth in the decryption key stream. Not all stream ciphers allow this. The alleged RC4 [24] does not, whereas SEAL [23] does.

For this application we assume that our generator is based on a block cipher, $f_p(x) = E(x, p)$. To generate t segments of random bits, $s_1, \ldots, s_t$, we assume a random n-bit key and a matrix $R \in \mathfrak{M}_m$ have been chosen. Let p_j be an n-bit string that uniquely, within the life-time of an n-bit seed x, identifies one of these segments. For instance, p_j may contain information such as the "name" of a key, a packet/message sequence number, identity of the sender/receiver, etc. To generate s_j, iterate the function $f(x) \triangleq f_{p_j}(x)$ as described above, until enough keys stream bits have been generated (corresponding to the length of the current segment). Thus, p_j has the place of an initialization vector, IV, and is used on a per-segment basis. Formally we have

Definition 4.1. Let n, and m, L_j, λ_j, $0 \leq j < t$ be integers such that $L_j = \lambda_j m$, $L \triangleq \sum_{j=0}^{t-1} L_j$ and let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. Finally let $\{p_j\}$ be distinct n-bit strings. The generator $BMGL_{n,m,t,\{L_j\},\{p_j\}}(f)$ stretches $n(1+m)$ bits to L bits (made up of t segments of length $L_0, \ldots, L_{t-1}$), as follows. The input is interpreted as $x_0 \in \{0,1\}^n$, and $R \in \mathfrak{M}_m$. Let $x_{i,j} = f(p_j, x_{i-1,j})$, $j = 0, \ldots, t-1$ and $i = 1, \ldots, \lambda_j$ where $x_{0,j} = x_0$ for all j, and let the output be $\{B_R^m(x_{i,j})\}_{i,j}$ for $j = 0, 1, \ldots, t-1$ and $i = 1, 2, \ldots, \lambda_j$.

Observe that for security, only x_0 must be random and secret and the $\{p_j\}$ and $\{L_j\}$ values may be determined "on the fly" rather than as fixed parameters. It remains to analyze the impact of this generalization on the provable security properties.

[1] Common "practical" tests are almost always much faster than the generator tested.

The attack scenario is now a bit different. We need to model the fact that an adversary now obtains several short stream segments generated by iterating f with fixed x and varying plaintext p_j. Intuitively though, a good block cipher should not lose security just because an attacker sees several plain-text/cipher-text pairs. We therefore define the following notion of security.

Definition 4.2. Let $f = f(p, x)$ be a function $\{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}^n$, and define $f_p(x) \triangleq f(p, x)$. Let t be an integer and A be a probabilistic algorithm which takes an input from $\{0, 1\}^{n\times(1+t)}$ and has output in $\{0, 1\}^n$. We then say that A is a (S, δ, t, i)-inverter for f if when given $y = f_p^{(i)}(x)$ and $\{y_j = f_{p_j}(x) \mid j = 0, 1, \ldots, t - 1\}$ for an x chosen uniformly at random, in time S with probability δ it produces z such that $f_p(z) = y$.

The difference now is that the algorithm, besides the iterated $y = f_p^{(i)}(x)$ it tries to invert, also gets some "help" in the form of other, non-iterated functions, $\{f_{p_j}(x)\}$. In the case that f_p and the f_{p_j}s are randomly chosen, black-box functions we would naturally assume that the information about the other functions, $\{f_{p_j}(x)\}$ would be of no use in inverting $f_p^{(i)}(x)$. Likewise, for a secure a block cipher, getting $f(p_j, x)$ for different plaintexts p_j does not seem to help in inverting $f(p, x)$, at least not if brute force search is the only method of attack available.

Modeling the $f(p_j, \cdot)$ as independent random functions for different p_j the following theorem is relevant.

Theorem 4.3. *Let A be an algorithm that tries to invert a black box function f_p : $\{0, 1\}^n \to \{0, 1\}^n$. If A is given $y = f_p^{(i)}(x)$ and some other, independently chosen functions evaluated at x; $\{f_{p_j}(x)|j = 0, 1, \ldots, t - 1\}$, for a random x, then the probability (over the choice of fs and x) that A finds a z such that $f_p(z) = y$ using at most T oracle-evaluations of f_p and S oracle-evaluations of $\{f_{p_j}\}$, is bounded by $(S + 2T(i + 1))/(2^n - (T + S)) + i(i + 1)2^{-(n+1)}$.*

This is a very slight extension of Theorem 2.8. Evaluations of f_p are treated exactly as in the proof of Theorem 2.8. Evaluations of the other functions is harmless unless they happen to take place at x but the probability of this happening is at most $(2^n - (T + S))^{-1}$ as x is not different from any other point on which none of the functions have been evaluated.

This leads to the following definition:

Definition 4.4. A *σ-secure one-way function family* is an efficiently computable set of functions, $F_n = \{f_p(x) \mid p \in \{0, 1\}^n\}$ such that each f_p maps $\{0, 1\}^n \to \{0, 1\}^n$, and so that the functions in F_n cannot be (S, δ, t, i)-inverted for any $S/\delta < \sigma 2^n/i$.

We naturally assume that a primitive like Rijndael/AES with varying plaintexts constitutes a σ-secure one-way function family for some reasonably large σ. Varying assumptions on σ will as previously lead directly to different quantitative security levels of the derived generator.

We now need to establish the following generalization of Lemma 3.3:

Lemma 4.5. *Let* $t, \{L_j\}, \{\lambda_j\}, \{p_j\}$ *be as in Definition* 4.1 *and set* $\lambda \triangleq \sum_j \lambda_j$. *Suppose that* $BMGL_{n,m,t,\{L_j\},\{p_j\}}(f)$ *runs in time* $S(L)$. *If this generator is not* $(L, T(L), \delta)$*-secure, then there is an algorithm* $P^{(i,j)}$, $j \in [0..t-1]$, $i \in [1..\lambda_j]$ *that, using* $T(L) + S(L)$ *operations, given* $f_{p_j}^{(i)}(x)$, R, *and* $\{f_{p_t}(x) \mid t > j\}$ *for random* $x \in \mathcal{U}_n$ *and* $R \in \mathfrak{M}_m$, *distinguishes* $B_R^m(f_{p_j}^{(i-1)}(x))$ *from* $\mathcal{U}_m$ *with advantage* $\delta' \geq \frac{\delta m}{L}$.

The algorithm $P^{(i,j)}$ *depends on integers* i, j, *and using* $O(\delta'^{-2}(T(L) + S(L)))$ *operations, values of* i, j *achieving advantage* $\delta_{i,j} \geq \delta'/2$ *can be found with probability at least* $1/2$.

It is not difficult to see that from the above lemma, whose proof is completely analogous to that of Lemma 3.3, follows the security of the generalized generator.

5. Practical Considerations

5.1. *Number of Bits Output per Application of the Core*

We would for efficiency reasons like to output as many bits as possible per application of Rijndael. On the other hand, we cannot output too many, if we are to relate the security of Rijndael to the proof of security for the generator. The effect of varying m is clearly visible in the above theorems. With a 256-bit key-size, m in the range 30–50 seems acceptable. Tests made at $m = 80$ gives almost no speed-up as the output generation itself then becomes too expensive.

One problem with a large value of m is that size of the seed grows with m. With $n = 256$, you need $256(m + 1)$ bits of seeding material: 256 for the initial x_0 (the secret key), and $256m$ to specify the matrix R. This is 1312 bytes when $m = 40$. This is quite large, and below we discuss how to decrease this number.

5.2. *Decreasing Seed Size*

The impact on security of varying m is clearly visible in the above theorems. Though increasing speed, a practical problem with a large m is the seed size; nm bits for R. First note though, that the security does not depend on the fact that R is secret; only that it is random.

It is possible to decrease the number of bits to only n by instead of binary matrix multiplication, performing a multiplication by a random element in the finite field $\mathbb{F}_{2^n}$, and selecting any fixed set of m bits of this, see [21]. A drawback of this construction is that instead of the direct reduction from a distinguisher for $B_R^m(x)$ to a predictor for $\langle v_i, x\rangle_2$ (Lemma 3.5), the restricted sample-space of elements makes us need to use the so called *Computational XOR-Lemma* [25]. Unfortunately, this reduces the initial δ-advantage of the distinguisher to a $2^{-m}\delta$-advantage for the predictor for $\langle v_i, x\rangle_2$. Thus, you might need to reduce m somewhat. First, we lose a factor 2^m for the above reason. Secondly, a smaller m implies a proportional factor increase in the number of iterations of f, which in the worst case implies a cubic additional loss in this factor. For instance, reducing m from 40 (in the random matrix case) to about 18 (in the $\mathbb{F}_{2^n}$ setting) gives roughly the same security, but a seed size decrease by a factor of 20.

While this might seem to give some slow-down, in practice it would typically not be too noticeable, as also each output is now generated more quickly.

An alternative, suffering the same security drawback, is to pick R as a random Toeplitz matrix, specified by $n+m-1$ bits [10].

6. Other Attacks

So far in the paper, we have established certain "security guarantees" for our generator: if the one-way function f is secure, then a lower bound on the security of the derived generator can be found, but it could of course be that the generator is even more secure, only our proof techniques fail to produce a better bound. In fairness, one should also ask if there are also upper bounds on the security, i.e. attacks faster than brute force search for the seed. It is quite easy to see that this is indeed the case. Let m, n as previously be the parameters defining the generator.

We first note that the $m \times n$ output generating matrix R either has full rank, or it does not, the latter happening with probability $\approx 2^{-(n-m)}$. In the first case, it is obvious that the output leaks m bits of information on each x_i. In the second case, there is a trivial distinguishing attack that compares exclusive-ors of bits generated by linearly dependent rows of R. In either case, we have an attack with a roughly 2^{n-m} time/success ratio.

Secondly, BMGL is of course not immune against generic attacks, e.g. time-memory trade-offs. For instance, if implementing f by an n-bit block cipher with the plaintext p kept fixed, e.g. $p = 00\ldots0$, a pre-computation attack such as [4] could, from roughly 2^t bits of output and using 2^t memory, retrieve the internal state (at some point) using 2^{n-t} workload, making the PRG forwards predictable from there on.

Finally, we note that versions of this construction has been subject to extensive practical statistical testing [22].

7. Summary and Conclusions

We have given a careful security analysis of a very natural PRG. Apart from optimizing known constructions and analysis we have introduced a new analysis method when several bits are output for each iteration of the one-way function.

Another common method to derive PRGs from a block cipher is to run it in *counter mode*. Though simpler, the security proof of such constructions, see [3], relies on the assumption that the core, f, is a pseudo-random function. (The same applies to the OFB mode of operation.) The strictly weaker type of security assumption we have proposed (a function being one-way on its iterates), although it has been proposed before by Levin, is for the first time made in a quantitative sense and we believe that this concept will be useful for future study of one-way functions.

Acknowledgements

We thank Bernd Meyer, Gustav Hast, and anonymous reviewers of earlier versions of this paper for helpful comments. In particular the idea to use the construction in Sect. 3.3 is due to an anonymous referee.

References

[1] W. Alexi, B. Chor, O. Goldreich, C.P. Schnorr, RSA and Rabin functions: Certain parts are as hard as the whole, *SIAM J. Comput.* **17**(2), 194–209 (1988)

[2] N. Alon, J. Spencer, *The Probabilistic Method*, 2nd edn. (Wiley, New York, 2000)

[3] M. Bellare, A. Desai, E. Jokipii, P. Rogaway, A concrete security treatment of symmetric encryption: analysis of the DES modes of operation, in *Proceedings of the 38th IEEE Conference on Foundations of Computer Science*, 1997, pp. 394–403

[4] A. Biryukov, A. Shamir, Cryptanalytic time/memory/data tradeoffs for stream ciphers, in *Advances in Cryptology-ASIACRYPT 2000*. Lecture Notes in Computer Science, vol. 1976 (Springer, Berlin, 2000), pp. 1–13

[5] M. Blum, S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* **13**(4), 850–864 (1984)

[6] J. Daemen, V. Rijmen, AES proposal: Rijndael, www.nist.gov/aes/

[7] D. Eastlake, S. Crocker, J. Schiller, Randomness recommendations for security, RFC 1750, IETF, 1994

[8] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudo-Randomness* (Springer, Berlin, 1999)

[9] O. Goldreich, S. Goldwasser, S. Micali, How to construct random functions, *J. Assoc. Comput. Mach.* **33**(4), 792–807 (1986)

[10] O. Goldreich, L.A. Levin, A hard core predicate for any one way function, in *Proceedings of 21st ACM Symposium on Theory of Computing*, 1989, pp. 25–32

[11] G. Hast, Nearly one-sided tests and the Goldreich–Levin predicate, *SIAM J. Comput.* **28**, 1364–1396 (1999)

[12] J. Håstad, R. Impagliazzo, L.A. Levin, M. Luby, Pseudo-random number generators from any one-way function, *SIAM J. Comput.* **28**, 1364–1396 (1999)

[13] J. Håstad, M. Näslund, Practical construction and analysis of pseudo-randomness primitives, in *Advances in Cryptology—Asiacrypt 2001*, ed. by C. Boyd. Lecture Notes in Computer Science, vol. 2248 (Springer, Berlin, 2001), pp. 442–459

[14] J. Håstad, M. Näslund, *BMGL*: Synchronous key-stream generator with provable security, in *Proceedings of the 1st Open NESSIE Workshop*, 13–14 November 2000

[15] J. Håstad, M. Näslund, Improved analysis of the *BMGL* key-stream generator, in *Proceedings of the 2nd Open NESSIE Workshop*, 12–13 September 2001

[16] D. Knuth, *Seminumerical Algorithms*, 2nd edn. The Art of Computer Programming, vol. 2 (Addison-Wesley, Reading, 1982)

[17] L. Levin, One-way functions and pseudo-random generators, *Combinatorica* **7**, 357–363 (1987)

[18] L. Levin, Randomness and non-determinism, *J. Symb. Log.* **58**(3), 1102–1103 (1993)

[19] F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error Correcting Codes* (North-Holland, Amsterdam, 1977)

[20] G. Marsaglia, The Diehard statistical tests, http://stat.fsu.edu/~geo/diehard.html

[21] M. Näslund, Universal hash functions & hard-core bits, in *Proceedings of Advances in Cryptology—Eurocrypt 1995*. Lecture Notes in Computer Science, vol. 921 (Springer, Berlin, 1995), pp. 356–366

[22] S. Pyka, The statistical evaluation of the NESSIE submission *BMGL*, NESSIE Public report NES/DOC/SAG/WP3/039/1, 2001

[23] P. Rogaway, D. Coppersmith, A software-optimized encryption algorithm, *J. Cryptol.* **11**(4), 273–287 (1988)

[24] B. Schneier, *Applied Cryptography* (Wiley, New York, 1995)

[25] U.V. Vazirani, V.V. Vazirani, Efficient and secure pseudo-random number generation, in *Proceedings of 25th IEEE Symposium on Foundations of Computer Science*, 1984, pp. 458–463

[26] A.C. Yao, Theory and applications of trapdoor functions, in *Proceedings of 23rd IEEE Symposium on Foundations of Computer Science*, 1982, pp. 80–91