Practical Construction and Analysis of Pseudo-randomness Primitives

Johan Håstad* and Mats Näslund

[1] NADA
Royal Institute of Technology
SE-10044 Stockholm, Sweden
johanh@nada.kth.se
[2] Communications Security Lab
Ericsson Research
SE-16480 Stockholm, Sweden
mats.naslund@era.ericsson.se

Abstract. We give a careful, fixed-size parameter analysis of a standard [1, 4] way to form a pseudorandom generator by iterating a one-way function and then pseudo-random functions from said generator, [3]. We improve known bounds also asymptotically when many bits are output each iteration and we find all auxiliary parameters efficiently. The analysis is effective even for security parameters of sizes supported by typical block ciphers and hash functions. This enables us to construct very practical pseudorandom generators with strong properties based on plausible assumptions.

1 Introduction

One of the most fundamental cryptographic primitives is the *pseudo random generator*, a deterministic algorithm that expands a few truly random bits to long "random looking" strings. Having such implies (among other things) *semantically secure* crypto systems, [5], secure key-generation for asymmetric cryptography etc.

Due to the fact that a sound theory of pseudo randomness did not emerge until the seminal works of Blum and Micali, [1], and Yao, [15], in the early 80's, early constructions were "ad-hoc", and many of them later turned out to be completely insecure. In a theoretical sense the area was closed when, in [6], it was shown that necessary and sufficient conditions for the existence of a pseudo-random generator 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 (mapping keys to cipher-texts, keeping the plaintext fixed), hash functions, etc. Still, the construction in [6] is complex, requiring key-sizes

* Work partially supported by the Göran Gustafsson foundation and NSF grant CCR-9987077.

of millions of bits, and an "ad-hoc" approach is still therefore often used in practice. Thus, a construction with provable properties, useful in practice is highly desirable.

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 mentioned above, and later work by Goldreich and Levin [4]. In [1] 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 pseudo-random generator can be built. In [4], then, it is shown that every one-way function, in particular ones being permutations, have such a hard bit $b(x)$. In this paper we make a careful analysis of this transformation from a one-way function to a pseudorandom generator, see Sect. 3. We add new elements of the analysis when we output many bits for each iteration of f, improving the dependence on this parameter. 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, see Sect. 3.1. Values of the parameters that give almost as strong results as the existential bounds can, for most parameter values, be found in time less than the time needed for successive inversions.

A related primitive are the pseudo-random functions; functions that can not be distinguished from random functions on the same domain/range. Goldreich, Goldwasser, and Micali, [3], showed how such could be built from a pseudo random generator. In Sect. 3.2, we apply the same kind of fixed parameter analysis to their construction and apply it to further enhance our generator.

Our explicit theorems allow us to construct a generator that is efficient in practice based on the assumption that e.g. Rijndael (mapping keys to ciphertexts, fixing a plaintext) remains hard to invert even when iterated, see Sect. 4.

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 $\{0,1\}^n$. Except otherwise noted, log refers to logarithm in base 2.

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*. Finally, recall that a function $\nu(n)$ is *negligible* if for all c, $\nu(n) \in o(n^{-c})$.

Our model of computation is slightly generous but realistic. We assume that simple operations like arithmetical operations and exclusive-ors on small[1] size integers can be done in unit time.

2.2 Pseudo-random Generators from One-way Permutations

Suppose we have a one-way function, that in addition is a permutation. Furthermore, suppose that we have a family of 0/1-functions, $B = \{b_r\}$, $b_r(x) \in \{0,1\}$, such that given $f(x)$ and a randomly chosen b_r, $b_r(x)$ is computationally indistinguishable from a random 0/1 coin toss. (Note that one-wayness of f is necessary.) We then say that B is a (family of) *hard-core functions* for f. The following construction, due to Blum and Micali [1], now shows how to construct a pseudo-random generator (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 (Blum-Micali, '84). *Suppose there is an efficient algorithm D that distinguishes (with non-negligible advantage) $g(x, r)$ from a completely random string. Then, there is an efficient algorithm P that given $r, f(x)$ predicts $b_r(x)$ with non-negligible advantage.*

Due to the iterative construction, f must not loose one-wayness under iteration. This can be guaranteed if f is a permutation, or, heuristically if f is randomly chosen, see Theorem 1. Assumptions along these lines have been proposed by Levin in [8] and were in fact the first conditions to be proved to be both necessary and sufficient for the existence of pseudorandom generators.

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?

2.3 A Hard-core for any One-way Function

In 1989, Goldreich and Levin [4], proved that *any* one-way function (not only permutations) have hard-cores[2]. Perhaps surprisingly, the hard-cores they found are 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), fixing an order left-to-right, or right-to-left. Let $B \triangleq \{b_r(x) \mid r \in \{0,1\}^n\}$ where

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

that is, the inner product mod 2.

[1] 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.

[2] 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. Without this assumption, the construction no longer becomes practical, [6].

Theorem (Goldreich-Levin, '89). *Suppose there is an efficient algorithm A, that given $r, f(x)$ for randomly chosen r, x, distinguishes (with non-negligible advantage) $b_r(x)$ from a completely random bit. Then there exists an efficient algorithm B, that inverts $f(x)$ on random x with non-negligible probability.*

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

As established already in [4], a way to improve efficiency would be to extract more than one bit per iteration of f. 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 $\mathcal{M}_m$, and the corresponding functions $\{B_R^m(x) \mid R \in \mathcal{M}_m\}$. That is, $B_R^m(x) \triangleq R \cdot x \bmod 2$. The above thus leads to a general construction, given any one-way function.

3 The Construction and its Security

3.1 The Basic PRG

Definition 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 $BMGL_{n,m,L}^f(x, R)$ stretches $n + nm$ bits to L bits as follows. The input is interpreted as $x_0 = x$ and $R \in \mathcal{M}_m$. Let $x_i = f(x_{i-1})$, $i = 1, 2, \ldots, \lambda$ and let the output be $\{B_R^m(x_i)\}_{i=1}^{\lambda}$.*

A proof of the practical security for a concrete f and fixed n, m, requires a very exact analysis, and that analysis is the bulk of this paper. To begin with, we would like to relate the difficulty of inverting an iterated function f to that of distinguishing outputs of $BMGL_{n,m,L}^f$ from random bits. This is is made difficult by the fact that we no longer require f to be a permutation. However, under one additional and natural assumption on the "behavior" of f, we can bring the analysis one step further, relating the security of $BMGL_{n,m,L}^f$ more directly to the difficulty of inverting f itself. Our measure of success is as follows.

Definition 2. *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 $f(z) = y$.

Note that the number z might be on the form $f^{(i-1)}(x')$ but this is not required. It is interesting to investigate what happens for a random function.

Theorem 1. *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 $T(i+1)2^{-n}$. On the other hand, there is an algorithm that using at most T oracle calls outputs a correct z except with probability $(1 - (i+1)2^{-n})^{T-i} + i^2 2^{-(n+1)}$.*

Proof (sketch). 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 at which f is computed in this process. If an inverter does not compute f at any $w \in W$, there is no correlation between the inverter and and the evaluation process. If the inverter makes T calls to $f(w)$, the probability of doing this for a $w \in W$ is at most $(i+1)T2^{-n}$ and this can be formalized.

To construct an inverter, first assume that the $i+1$ values seen under the evaluation of $f^{(i)}(x)$ are distinct. This happens except with probability (over random f) $\binom{i}{2}2^{-n} \leq i^2 2^{-(n+1)}$ and if it does not happen we simply give up. Now consider the following inverter. It is given $y = f^{(i)}(x)$. Start by setting $x_0 = 0^n$ and $x_j = f(x_{j-1})$ for $j = 1, 2, \ldots$. Continue this process until either $x_j = y$ (and it is done) or x_j is a value it has seen previously. In the latter case it changes x_j to a random value it has not seen previously and continues. Each value it sees is a random value and if it ever gets one of the $i+1$ values in W, it finds the y within at most i additional evaluations of f. The probability of not finding such a good value in the $T-i$ first steps is at most $(1-(i+1)2^{-n})^{T-i}$. □

Consider for instance the block cipher Rijndael [13] as a one-way function (fixing a message, mapping keys to cipher-texts). It is reasonable to expect that Rijndael is 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 not too much smaller than $2^n/i$. The security is now defined as follows.

Definition 3. *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 $T/\delta < \sigma 2^n/i$.*

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

Hence, for our "practical" choice, f = Rijndael, we expect it to be about 1-secure in the above terminology. Note also that if f is a permutation, only the case $i = 1$ is of interest and we have a standard notion of security.

Security of the Generator. Our objective is to show that if $BMGL^f_{n,m,L}$ is not (L, T, δ)-secure for "practical" values of L, T, δ, then there is also a practical attack on the underlying one-way function f. In particular, we show the following theorem:

Theorem 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 $k = \max(m, 1 + \log((2n+1)\delta'^{-2}) - j)$, f can be $(T', d_j/2, i)$-inverted, where d_j is given by (7) and (8), page 17, and T' equals*

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

Values of i and j such that f can be $((8+o(1))T', d_j/16, i)$-inverted can, with probability at least $1/4$, be found in time $O(\delta'^{-2}(T+S))$.

The time-success ratio for most ranges of δ and T is worst when the value of j is small. For $j \in O(1)$ and $m, k, E \leq S \leq O(T)$ the ratio is $O(n^2L^2\delta^{-2}2^mT)$. The preprocessing time (to find i, j) is small compared to the running time except in the cases when j is large. In those cases the time to find j is still smaller than the running time of the inverter while the running time to find i might be larger for some choices of the parameters.

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 [4] were, basically, any polynomial time reduction from inverting f to distinguishing the generator would be enough. Optimizations of the original proof also appeared in [9], but are not stated explicitly.

The proof of Theorem 2 has two main milestones. We first show (Lemma 1 below) that a distinguisher for BMGL can be turned into a distinguisher for $B_R^m(f^{(i-1)}(x))$, given $R, f^{(i)}(x)$, for some i. Then we show (Theorem 3) how this latter distinguisher is converted to an inverter for $f^{(i)}$.

We thus start with the following lemma.

Lemma 1. *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 $x \in \mathcal{U}_n$, $R \in \mathcal{M}_m$, distinguishes $B_R^m(f^{(i-1)}(x))$ from $\mathcal{U}_m$ with advantage $\delta' = \frac{\delta m}{L}$.*

$P^{(i)}$ depends on an integer i, and using $c_1\delta'^{-2}(T(L)+S(L))$ operations, where c_1 is the constant given by (5), page 16, 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 found 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'. The cost is simply an increase in the constant c_1.

Assuming for the moment the following Lemma (a proof is found in the Appendix), we can use it to show Lemma 1.

Lemma 2. *Let F be a function $F : \{0,1\}^n \times \mathcal{M}_m \to (\{0,1\}^m)^\lambda$, computable in time $\leq S$. Let H^i be the distribution on $(\{0,1\}^m)^\lambda$ induced by replacing the first im bits of $F(x,R)$ by random bits.*

Suppose that H^0 $(= F(x,R))$ and H^λ $(= (\mathcal{U}_m)^\lambda)$ are distinguishable with advantage δ, by an algorithm D running in time T. Then, a value of $i < \lambda$ for which H^i, H^{i+1} can be distinguished with advantage $\delta' = \frac{\delta}{2\lambda}$, can with probability at least $\frac{1}{4}$, be found in time $c_1\delta'^{-2}(T+S)$ where c_1 is an absolute constant.

For the moment, just note that the existence of such an i (and even slightly better δ') follows directly from the triangle inequality.

Proof (of Lemma 1, sketch). The proof uses an optimized version of the so called *universality of the next-bit-test*, by Yao [15], see also [1].

We assume we know the good value of i as in Lemma 2. Let $F(x,R) = BMGL^{f}_{n,m,\lambda m}(x,R)$. On input $f^{(i)}(x), R, \gamma$, where γ is either random, or, equal to $B^m_R(f^{(i-1)}(x))$ we do as follows. We easily generate an element according to distribution H^{i+1} as in Lemma 2, with the exception that the $i+1$st m-bit block is assigned the value γ. We feed this value to D and answers as it does. We see that precisely depending on whether γ is random or not, we run D on an input from H^i, or, from H^{i+1} and the lemma follows. □

We now give the theorem of Goldreich and Levin [4] 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 parameter m. While previous constructions would yield a factor proportional to 2^{2m} we decrease this to 2^m. The improvement is due to the fact that we treat the case of general m directly rather than reducing it to the case $m = 1$ (see later discussion).

The second main step towards Theorem 2 is:

Theorem 3. *Fix x. Suppose there is an algorithm, P, using T operations, when given random R distinguishes $B^m_R(x)$ from random strings of length m with advantage at least ϵ where ϵ is given. Then, for $k \triangleq \max(m, \log(\epsilon^{-2}(2n+1)))$, we can in time*

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

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

As we understand, a statement similar (upto a constant), for the special case of $m = 1$, can be derived from [9]. In most application one has $m \leq \log(\epsilon^{-2}(2n+1))$ and thus the latter value of k should be considered standard.

We now collect the last pieces for the proof of Theorem 2 by proving the above Theorem 3 which, in turn, relies on the following prelimnaries.

Lemma 3. *Fix any $x \in \{0,1\}^n$. For $m < k$, from $m+k$ randomly chosen $a_0, \ldots, a_{m-1}$ and $b_0, \ldots, b_{k-1} \in \{0,1\}^n$, it is possible in time $2m2^k + k^2 + m + 4k$ to generate a set of 2^k uniformly distributed, pairwise independent matrices $R^1, \ldots, R^s \in \mathcal{M}_m$. Furthermore, there is a collection of $m \times (m+k)$ matrices $\{M_j\}_{j=1}^{2^k}$ and a vector $z \in \{0,1\}^{m+k}$ such $B^m_{R^j}(x) = M_j z$ for all j.*

The proof is given in the Appendix. The construction generalizes that of Rackoff for the case $m = 1$, see [2]. If $k < m$, we use $k' = m$ above and then simply only take the first 2^k matrices.

Lemma 4. *Let P be an algorithm, mapping pairs $\mathcal{M}_m \times \{0,1\}^m \to \{0,1\}$, whose running time is T, let R^j, M_j be the matrices generated as described in Lemma 3 and let $S = \{S_j\}_{j=1}^{2^k}$ be an arbitrary matrix set in $\mathcal{M}_m$.*

In time $2^{m+k}(2m+k+T)$ it is possible to compute 2^{m+k} values, $c_1, \ldots, c_{2^{m+k}}$ such that for at least one l we have $c_l = E_j[P(R^j + S_j, B^m_{R^j}(x))]$. The value of l is independent of S.

The role of the set S is explained shortly.

Proof. First run P on all the 2^{m+k} possible inputs of form (R^j+S_j, r) and record the answers: $\{P(R^j+S_j,r)\}$. A fixed value of l above corresponds to a value of the $m+k$ bits z_l in Lemma 3. Let us assume that z_l is the correct choice, i.e. $B^m_{R^j}(x) = M_j z_l$. We define

$$c_l \triangleq 2^{-k}\sum_{j=0}^{2^k-1} P(R^j+S_j, M_j z_l) = 2^{-k}\sum_{j=0}^{2^k-1}\sum_{r=0}^{2^m-1} P(R^j+S_j,r)\Delta(r, M_j z_l), \quad (1)$$

where $\Delta(r,r') = 1$ if $r = r'$ and 0 otherwise. The naive way to calculate this number would require time 2^{2k+m} but we can do better using the Fast Fourier transform. First note 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 (1) equals

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

Let $Q(j,\alpha)$ be the inner sum and fix a value of j. Notice that each α-value then correspond to a Fourier transform 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 $m2^{k+m}$. Finally we have

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

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

We prove now that we can compute useful information about x.

Lemma 5. *Let P, T, x and ϵ be as in Theorem 3. Then for any set of N vectors $\{v_i\}_{i=1}^N \subset \{0,1\}^n$ and any $k \geq m$ we can in time $(1+o(1))2^{m+k}(2m+k+T+1)(N+1)$ produce a set of lists $\{b_i^{(j)}\}_{i=1}^N$, $j = 1,2,\ldots,2^{k+m}(N+1)$ such that with probability $1/2$ we have for at least one j, $\langle x, v_i\rangle_2 = b_i^{(j)}$, except for at most $\frac{N}{\epsilon^2 2^{k-1}}$ of the N possible values of i.*

Proof. Start by randomly generating the 2^k matrices $\{R^j\}$ as shown in Lemma 3. Now repeat the process below for each $i = 1,\ldots,N$. Select 2^k (pairwise) independent random strings $s_j^i \in \{0,1\}^m$, and let S_j^i be the $m \times n$ matrix defined by $S_j^i \triangleq s_j^i \otimes v_i$ (the outer product, i.e. $(S_j^i)_{k,l} = (s_j^i)_k \cdot (v_i)_l$). Notice that by linearity

$$(R^j + S_j^i)x = R^j x + s_j^i\langle v_i, x\rangle_2, \quad (2)$$

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

As described in Lemma 4, we now compute the values $\{c^i_l\}$.

$$c^i_l = 2^{-k} \sum_{j=0}^{2^k-1} P(R^j + S^i_j, M_j z_l).$$

Focus on the correct choice for l. If $\langle v_i, x\rangle_2 = 0$, then c^i_l is the average of a uniformly random, pairwise independent sample of the distinguisher P on inputs of the form $\{P(R, B^m_R(x))\}$. On the other hand, if $\langle v_i, x\rangle_2 = 1$, it is a sample of $\{P(R, u)\}$ over random u.

Suppose p_R is the probability that P outputs 1 when the m bits are picked as $B^m_R(x)$ and let p_U be the same probability when the m bits are picked randomly. Let $p \triangleq (p_R + p_U)/2$. Note that we do not know the value of p. We deal with this problem later, so for the moment suppose we do.

We guess that $\langle v_i, x\rangle_2 = 0$ if $c^i_l \geq p$ and $\langle v_i, x\rangle_2 = 1$ otherwise. The choice is correct unless the average of 2^k pairwise independent Boolean variables is at least $\epsilon/2$ away from its mean. By Chebychev's inequality the probability that this happens is bounded by $2^{-k}\epsilon^{-2}$.

This implies that for the correct values of l and p, the expected number of errors is $2^{-k}\epsilon^{-2}N$, and by Markov's inequality, with probability at least at $1/2$ it is below $2^{1-k}\epsilon^{-2}N$. There are 2^{k+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 $c^i_l \geq p$ (if any). Thus, there are only $N + 1$ such choices.

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

We finally establish Theorem 3.

Proof (of Theorem 3). Set $k = \max(m, \log(\epsilon^{-2}(2n + 1)))$. We apply Lemma 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. □

We can now use Theorem 3 and Lemma 1 to establish Theorem 2, see the Appendix.

Instead of applying Lemma 5 with the unit vectors we can, as suggested in [2], use it with $\{v_i\}$ describing the words of an error correcting code, e.g. a suitable Goppa-code, [10]. (Similar ideas appears in [8].) If we have code words of length N, containing n information bits, and we are able to efficiently correct e errors we get the following variant of Theorem 3:

Theorem 4. *Fix x. Suppose there is an algorithm, P, that using T operations given R distinguishes $B^m_R(x)$ 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 $k = \max(m, \log(\epsilon^{-2}(2N + 1)/e))$ we can in time*

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

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

Proof. We apply Lemma 5 with the given value of k 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 2, this translates to the quality of the inverter. We only state the resulting algorithm in existential form using O-notation.

Theorem 5. *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* $G = BMGL^f_{n,m,L}$ *is based on an* n*-bit function* f*, computable by* E *operations, and that* G *produces* L *bits in time* S*. 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* $k = \max(m, O(1) + 2\log\delta'^{-1} - j)$ *such that* f *can be* $(T', \Omega(2^{-j/2}(j+1)^{-2}), i)$*-inverted where* T' *equals*

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

In particular, this implies that the asymptotic time-success ratio decreases by a factor n for the parameters discussed after Theorem 2.

3.2 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, [3]. 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, $G : \{0,1\}^n \to \{0,1\}^{2n}$, though we for concreteness think of $G = G(x, R) = BMGL^f_{n,m,2n}(x, R)$ for some f. For simplicity of notation, we shall exclude R from it, keeping in mind that probabilities should be taken also over the choice of R. First, let us assume that we know in advance how may output bits that are desired. We apply [3] to obtain $2^d n$ output bits (where d is given) from $n(m+1)$-bits.

Definition 4. *Fix* $n, d \in \mathbb{N}$*. Let* $G(x)$ *be a generator, stretching* n *bits to* $2n$ *bits, and let* $G_0(x)$ *(*$G_1(x)$*) be the first (last)* n *bits of* $G(x)$*. For* $x \in \{0,1\}^n$*,* $s \in \{0,1\}^d$ *put* $g_x(s) \triangleq G_{s_d}(G_{s_{d-1}}(\cdots G_{s_2}(G_{s_1}(x))\cdots))$*, and define* $GGM^G_{d,n} : \{0,1\}^n \to \{0,1\}^{2^d n}$ *by*

$$GGM^G_{d,n}(x) \triangleq g_x(00\ldots0), g_x(00\ldots1), \cdots, g_x(11\ldots1)$$

(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 n. Associate $v \in V$ with its breadth-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 $GGM^G_{d,n}$ is simply the concatenation of all the "leaves" 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 1, retains more of the one-wayness of f.

Lemma 6. *Suppose that D_1 is a $(2^d n, T, \delta)$-distinguisher for $GGM^G_{d,n}(x)$ where G can be computed in time S. Then, there is an integer $i \leq 2^d$ and algorithm D^i that is an $(2n, T + 2^d S, 2^{-d}\delta)$-distinguisher for G.*

D^i depends on i, and a value of i achieving advantage $\delta_i \geq 2^{-(d+1)}\delta$ can be found with probability at least $1/2$ in time $c_1 2^{2d}\delta^{-2}(T + 2^d S)$ where c_1 is the constant given by (5), page 16.

Proof (sketch). Consider the binary tree T, describing a computation of $GGM^G_{d,n}$ 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^i is defined by a simulation algorithm, $GGM^i(x)$, which on input x, assigns labels as follows. Assign the root, $v = 1$, the label x. For $v \in V$, $v = 1, 2, \ldots, i$, label v's children by letting $\mathcal{L}(2v), \mathcal{L}(2v+1)$ be independent, random n-bit strings. Then, for $v = i+1, \ldots, 2^d - 1$: $\mathcal{L}(2v) = G_0(\mathcal{L}(v))$, $\mathcal{L}(2v+1) = G_1(\mathcal{L}(v))$. Finally return the labels of the leaves in T.

Observe that H^{2^d-1} gives the uniform distribution over the node labels (in particular, over the leaves) and H^0 labels the vertices exactly as $GGM^G_{n,d}$ does on a random seed x. Since D_1 distinguishes $GGM^G_{d,n}(x)$ from random $2^d n$-bit strings with advantage δ, for some $i \leq 2^d$, it must be the case that D_1 distinguishes H^i, H^{i+1} with advantage at least $2^{-d}\delta$.

Finding i is now done in complete analogy with Lemma 2, letting the function F there correspond to the node labeling.

We now construct D^i: when D^i gets input $\gamma \in \{0,1\}^{2n}$, it selects random x and feeds D_1 a value y, computed as $GGM^{i+1}(x)$ with the following exception: $i+1$ is not assigned any label[3], and the children of $i+1$ are assigned the left/right n-bit half of γ respectively. It is not too hard to see that if γ is random, we give D_1 a value according to exactly the same distribution as H^{i+1}, whereas if $\gamma = G(x')$, D_1 is given a value from the same distribution as $GGM^i(x)$, i.e. H^i. Thus, by returning D_1's answer to y, D^i's advantage equals that of D_1. □

Unknown Output Length. If the the length of the "stream" is unknown beforehand, we let the basic generator G expand n bits to $3n$ bits. Apply the

[3] As the labels of non-leaves are never exposed, one can conceptually think of the process as labeling $i+1$ afterwards.

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. To save memory, the traversal can be implemented in iterative depth-first fashion.

3.3 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 $m = 32$ (32 bits per iteration). One choice of parameters gives the following corollary.

Corollary 1. *Consider $G = BMGL^{Rijndael}_{256,32,2^{30}}$ (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 $k = \max(32, 123 - j)$, Rijndael can be (T', d_j, i)-inverted (d_j given by (7) and (8)) for $T' = 2^{41+k}(65 + k + T + S + E)$.*

Similarly, setting $G' = BMGL^{Rijndael}_{256,32,512}$ and then using $GGM^{G'}_{22,256}$ (to generate the same length outputs), the result holds for some $i < 16$.

This is simply substituting the parameters and noting that the $o(1)$ in Theorem 2 comes from disregarding the time to construct the matrices described in Lemma 3 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, [11], or those by Knuth, [7], it is reasonable to assume[4] that $65 + k + T + E \leq S$. 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+k} Rijndael computations. The maximum of $2^k(d_j/2)^{-1}$ is obtained for $j = 5$ in which case it equals $2^{124} \cdot 7.5 \leq 2^{127}$. We conclude that in this case we get a time-success ratio that is equivalent to 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^{-57}-secure. Thus, though somewhat more cumbersome to implement, the GGM method is more security preserving.

If we want to find the values of i and j efficiently the ratio increases by a factor 2^6. Note that for the case with small j the time needed to find i and j is much smaller than the running time of the inverter.

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

4 Discussion

4.1 Choice of f

To implement the generator in practice, we suggest to base the one-way function on Rijndael. First of all it is widely believed to be secure and has shown to be very efficient. (A trial implementation of BMGL gives speeds in the range $2 - 10$Mb/s on a standard PC, depending on choice of m.) Secondly, as our construction requires that the block size of the cipher is equal to the key size, the fact that Rijndael supports both 128 and 256-bit block size is advantageous, as it makes it possible to vary the security parameter (key size).

Again note that the one-way function we suggest to use is to fix a message, p, let the input be the encryption key, x, and the output the cipher-text. To obtain a permutation and at the same time increased speed, it might appear to be better to have the mapping from clear-text to crypto-text and iterate $f_x(p)$ rather than $f_p(x)$. The problem is that this is by definition *not* a one-way function: anybody that can compute it can invert it. A possibility is also to use an efficient cryptographic hash function as f.

4.2 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 [12]. 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 5), the restricted sample-space of elements makes us need to use the so called *Computational XOR-Lemma*, [14]. Unfortunately, this reduces the initial δ-advantage of the distinguisher to a $2^{-m}\delta$-advantage for the predictor for $\langle v_i, x\rangle_2$.

An alternative, suffering the same security drawback, is to pick R as a random Toeplitz matrix, specified by $n + m - 1$ bits, [4].

5 Summary and Conclusions

We have given a careful security analysis of a very natural pseudorandom generator. 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 addmitedly simpler, the proof of such constructions relies on the assumption that the core, f, is a pseudo-random function. 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.

Acknowledgment. We thank Bernd Meyer, Gustav Hast, and anonymous reviewers of different versions of this paper for helpful comments.

References

1. M. Blum and S. Micali: *How to Generate Cryptographically Strong Sequences of Pseudo-random Bits.* SIAM Journal on Computing, **13**(4), 850–864, 1984.
2. O. Goldreich: *Modern Cryptography, Probabilistic Proofs and Pseudo-randomness.* Springer-Verlag, 1999.
3. O. Goldreich, S. Goldwasser and S. Micali: *How to Construct Random Functions.* J. ACM, **33**(4), 792–807, 1986.
4. O. Goldreich and L. A. Levin: *A Hard Core Predicate for any One Way Function.* Proceedings, 21st ACM STOC, 1989, pp. 25–32.
5. S. Goldwasser and S. Micali: *Probabilistic encryption.* J. Comput. Syst. Sci., **28**(2), 270–299, 1984.
6. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby: *Pseudo Random Number Generators from any One-way Function.* SIAM Journal on Computing, **28**, 1364–1396, 1999.
7. D. Knuth: *Seminumerical algorithms,* (2 ed.), Volume 2 of *The art of computer programming,* Addison-Wesley, 1982.
8. L. Levin: *One-way Functions and Pseudorandom Generators.* Combinatorica **7**, 357–363, 1987.
9. L. Levin: *Randomness and Non-determinism.* J. Symb. Logic, **58**(3), 1102–1103, 1993.
10. F. J. MacWilliams and N. J. A. Sloane: *The Theory of Error Correcting Codes.* North-Holland, 1977.
11. G. Marsaglia: *The Diehard statistical Tests.* http://stat.fsu.edu/~geo/diehard.html
12. M. Näslund: *Universal Hash Functions & Hard-Core Bits.* Proceedings, Eurocrypt '95, LNCS 921, pp. 356–366, Springer Verlag.
13. J. Daemen and V. Rijmen: *AES Proposal: Rijndael.* www.nist.gov/aes/
14. U. V. Vazirani and V. V. Vazirani: *Efficient and Secure Pseudo-Random Number Generation.* Proceedings, 25th IEEE FOCS, 1984, pp. 458–463.
15. A. C. Yao: *Theory and Applications of Trapdoor Functions.* Proceedings, 23rd IEEE FOCS, 1982, pp. 80–91.

A Additional Proofs

Proof (of Lemma 2). Let δ_i be D's advantage on H^i, H^{i+1}. 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, $\{S(j)\}_{j\geq 0}$, each with different sampling accuracy. Stage $S(j)$ chooses some random i-values and samples D on inputs generated from H^i, H^{i+1}. As soon as a sufficiently "good" i is detected, the procedure terminates. Below we quantify the needed accuracy and the criterion for selecting the good i.

For $j \in \{0, 1, \ldots, -2\log\delta'\}$ let a_j be the fraction of i such that $\delta_i \geq 2^{(j-1)/2}\delta'$. By the assumption of the lemma we have

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

Define b_0 to be $\lceil 4(1-2^{-1/2})^{-1}\rceil$ and

$$b_j = \lceil 4(1-2^{-1/2})^{-1}(2^{(j-1)/2} - 2^{(j-2)/2})\rceil = \lceil 2^{(j+3)/2}\rceil,$$

for $j > 0$. The b_j-values, together with a parameter T_j now define the sampling accuracy. Given these values, we determine i as follows.

In stage $S(j)$, $j = -2\log\delta', -2\log\delta' - 1, \ldots, 0$ choose b_j different random values of i and sample H^i and H^{i+1} each $T_j\delta'^{-2}$ times and run D on each of the samples. If the difference in the number of 1-outputs is at least $(2^{(j-1)/2}T_j - \sqrt{T_j/2})\delta'^{-1}$ choose this i and halt. If no i is ever chosen halt with failure. We need to analyze the procedure and determine T_j.

Suppose that at stage j an i is picked such that $\delta_i \geq 2^{(j-1)/2}\delta'$. We claim that the algorithm halts with this i as output with probability at least $1/2$. To establish this first consider the following fact, the proof of which we leave to the reader.

Fact. *Let X be a random variable with mean μ and standard deviation σ. Then we have*

$$\Pr[X \leq \mu - \sigma] \leq 1/2.$$

From this, the above claim now follows since the expected difference in the number of 1-outputs when $\delta_i \geq 2^{(j-1)/2}\delta'$ is at least $2^{(j-1)/2}T_j\delta'^{-1}$ and the standard deviation (being the sum of $T_j\delta'^{-2}$ variables each being the difference of two 0/1-valued variables) is at most $\delta'^{-1}\sqrt{T_j/2}$. This implies that the probability that the algorithm halts for an individual iteration during stage j is at least $a_j/2$. The probability that algorithm will fail to output any number is thus bounded by

$$\prod_j (1 - a_j/2)^{b_j} \leq e^{-\sum_j a_j b_j/2} \leq e^{-2},$$

where the last inequality follows from (3) and the definition of b_j.

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

The expected difference of the number of 1-outputs in the two experiments is at most $T_j\delta'^{-1}/2$ and we have to estimate the probability that it is at least $(T_j 2^{(j-1)/2} - \sqrt{T_j/2})\delta'^{-1}$. This is, provided

$$T_j(2^{(j-1)/2} - 1/2) - \sqrt{T_j/2} \geq 0, \tag{4}$$

by a simple invocation of Chernoff bounds, at most

$$e^{-\frac{(T_j(2^{(j-1)/2}-1/2)-\sqrt{T_j/2})^2}{2T_j}}.$$

Let us call this probability p_j. The overall probability of ever outputting an i with $\delta_i \leq \delta'/2$ is bounded by

$$\sum_j b_j p_j.$$

We now define T_j to be the smallest number satisfying (4) such that $p_j < 2^{-(j+3)}b_j^{-1}$ and such that $T_j\delta'^{-2}$ is an integer. We get that with this choice the probability of outputting an i with $\delta_i \leq \delta'/2$ is at most $1/4$ and hence the probability that we do get a good output is at least $(1-e^{-2})\frac{3}{4} \geq .64$. The total number of samples of the algorithm is bounded by $c_1\delta'^{-2}$, where

$$c_1 \triangleq 2\sum_j b_j T_j. \tag{5}$$

Note that this sum converges since $T_j \in O(j2^{-j})$ and $b_j \in O(2^{j/2})$. In fact, it can numerically be calculated to be bounded by 5300. Moreover, the sum is completely dominated by the first term which is over 4600, and the sum of all but the first three terms is bounded by 250. Thus, a more careful analysis what to do for small j could lead to considerable improvements in this constant. □

Before we continue let us make some needed definitions. Let $\mathrm{bin}(i)$ be the map that sends the integer i, $0 \leq i < 2^m$ to its binary representation as an m-bit string. In the sequel, we perform some computations in $\mathbb{F}_{2^k}$, the finite field of 2^k elements, represented as $\mathbb{Z}_2[t]/(q(t))$ where $q(t)$ is a polynomial of degree k, irreducible over $\mathbb{Z}_2$. We assume that such q is available to us. If not, it can be found in expected time at most k^4 which is negligible compared to our other running times considered. Viewing $\mathbb{F}_{2^k}$ as a vector space over $\mathbb{F}_2$, for any $\gamma = \sum_{i=0}^{k-1}\gamma_i t^i \in \mathbb{F}_{2^k}$, we let in the natural way $\mathrm{bin}(\gamma)$ denote the vector $(\gamma_0, \ldots, \gamma_{k-1})$ corresponding to γ's representation over the standard polynomial basis. Note also that $\mathrm{bin}(\gamma)$ can be interpreted as a subset of $[0..k-1]$ in the obvious way.

Proof (of Lemma 3). First choose randomly and independently m n-bit strings, $a_0, \ldots, a_{m-1}$ and k strings $b_0, \ldots, b_{k-1}$, each also of length n. The jth matrix, R^j is now defined by $\{a_i\}$, $\{b_l\}$, and an element $\alpha_j \in \mathbb{F}_{2^k}$ as follows. Its ith row, R_i^j, $0 \leq i < m$, is defined by

$$R_i^j \triangleq a_i \oplus \left(\oplus_{l \in \mathrm{bin}(\alpha_j \cdot t^i)} b_l\right),$$

where α_j is the lexicographically jth element of $\mathbb{F}_{2^k}$ (i.e. the lexicographically jth binary string), and the multiplication, $\alpha_j \cdot t^i$, is carried out in $\mathbb{F}_{2^k}$, 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. Take such a set of rows, $S_1 \subset R^{j_1}$, and $S_2 \subset R^{j_2}$. We may actually assume that $S_1 = S_2 = S$, say, since otherwise, the a-vectors makes the result uniformly distributed. In this case the xor can be written as

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

but this is the same as

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

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 $\alpha_{j_1} \neq \alpha_{j_2}$, $\alpha_{j_1} + \alpha_{j_2} \neq 0$ too, so we have two nonzero elements and hence their product is nonzero.

Notice that if we know $\sum_i a_{li} x_i$ and $\sum_i b_{li} x_i \bmod 2$ for all a_l, b_l (a total of $m + k$ 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 $i = k+1, \ldots, 2k$ in $GF[2^k]$. Using an iterative procedure this can be done with $3k$ operations on k bit words and since we only care about $k \leq n$ these can be done in unit time. Now generate the vectors a and b in time $m + k$ operations. Then we compute $\oplus_{l \in bin(t^i)} b_l$ for each $i = 0, \ldots, 2k$ using k^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 $2m2^k + k^2 + m + 4k$. □

Proof (of Theorem 2). First we apply Lemma 1 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.

For each x we have an advantage δ_x. Let a_j be the fraction of x with $\delta_j \geq 2^{(j-1)/2}\delta''$. Since the expected value of δ_x is δ'' we have

$$a_0 + \sum_{j=1}^{\infty} a_j (2^{(j-1)/2} - 2^{(j-2)/2}) \geq 1 - 2^{-1/2}. \tag{6}$$

Now define

$$d_0 \triangleq \frac{1}{2}(1 - 2^{-1/2}) \tag{7}$$

and

$$d_j \triangleq (2j(j+1)2^{(j-1)/2})^{-1} \tag{8}$$

for $j \geq 1$. Since

$$d_0 + \sum_{j=1}^{\infty} d_j(2^{(j-1)/2} - 2^{(j-2)/2}) = 1 - 2^{-1/2}, \tag{9}$$

we must have $a_j \geq d_j$ for some j and this is our choice for j in the existential part. We now apply Theorem 3 with $\epsilon = 2^{(j-1)/2}\delta'$. To eliminate the list we apply f to each element in it to see if it is a correct pre-image in which case it is output. Since whenever $\delta_x \geq \epsilon$ we have a probability $1/2$ of having $f^{(i-1)}(x)$ in the list and hence the probability of being successful for a random x is at least $d_j/2$.

To get a uniform algorithm, we need to sample to find a suitable value of j. Consider the following procedure for parameters d and T_j to be determined.

For $j = -2\log\delta'', -2\log\delta'' - 1, \ldots, 0$ choose $d(j+3)d_j^{-1}$ different random values of x and run $P^{(i)}$, for each x, $T_j\delta''^{-2}$ each on the two distributions given by choosing the m extra bits as $B_R^m(f^{(i-1)}(x))$ or as random bits. If the difference in the number of 1-outputs for the two distributions is at least $(2^{(j-1)/2}T_j - \sqrt{T_j/2})\delta''^{-1}$ for at least $d(j+3)/4$ different values, choose this j and apply the algorithm of Theorem 3 with $\epsilon = 2^{(j-2)/2}\delta'' = 2^{(j-4)/2}\delta'$.

First we analyze the probability that the algorithm outputs j if it ever gets to a stage where $a_j \geq d_j$. For each x chosen, the probability that it will satisfy $\delta_x \geq 2^{(j-1)/2}\delta''$ and yield the desired difference is by the choice of j and Fact A, at least $a_j/2 \geq d_j/2$. Thus, for sufficiently large d, with probability at least $1 - 2^{-(j+3)}$, this desirable distance will be detected $d(j+3)/4$ times and j will be output. Hence, except with this probability the algorithm will produce some output and we have to analyze the probability that a worse j is output at an earlier stage.

We claim that unless $a_{j-1} \geq d_j/8$, the probability of j being output is $2^{-(j+3)}$. Suppose that $a_{j-1} < d_j/8$ and consider an individual execution in stage j. For a suitable choice of T_j we will prove that the probability that we observe a difference greater than $(2^{(j-1)/2}T_j - \sqrt{T_j/2})\delta''^{-1}$ is bounded by $d_j/6$. This is sufficient, for large enough d, to establish the claim.

By assumption $\delta_x \leq 2^{(j-2)/2}\delta''$ except with probability $d_j/8$ and thus we need to prove that given that this inequality is true, the probability to get the desired difference is at most $d_j/24$. By assumption the expected value of the observed difference is $2^{(j-2)/2}T_j\delta''^{-1}$, and by applying Chernoff bounds it is hence sufficient to choose T_j large enough so that

$$e^{-\frac{(T_j(2^{(j-1)/2} - 2^{(j-2)/2}) - \sqrt{T_j/2})^2}{2T_j}} \leq \frac{d_j}{24}.$$

This can be done with $T_j = O((j+3)2^{-j})$. The expected number of samples computed, given that j_0 is the largest value such that $a_{j_0} \geq d_{j_0}$, is at most

$$\sum_{j=j_0}^{\infty} d(j+3)d_j^{-1}T_j\delta''^{-2} + 2^{-(j_0+3)}\sum_{j=0}^{j_0-1} d(j+3)d_j^{-1}T_j\delta''^{-2},$$

which is $O(j_0^4 2^{-j_0/2}\delta'^{-2})$.

In the case where we efficiently find i and j, the final value of ϵ for which we call upon Theorem 3 is a factor $2^{-3/2}$ smaller than in the existential case, and hence the increase in the running time is increased by a factor $8+o(1)$, where the $o(1)$ comes from the increase in the additive term k. By the above argument the guarantee for the fraction of the inputs for which the procedure has probability at least $1/2$ of finding the inverse image, is at least $1/8$ of that in the existential case. □