Key Feedback Mode: a Keystream Generator with Provable Security

Johan Håstad*
NADA, Royal Inst. of Technology
SE-10044 Stockholm, Sweden

Mats Näslund†
Communications Security Lab
Ericsson Research
SE-16480 Stockholm, Sweden

October 11, 2000

Abstract

We propose a key feedback mode of operation for the AES algorithm Rijndael (or any other block cipher), giving efficient synchronous keystream generators. We show that if the block cipher possesses simple properties, normally accepted to exist in any secure block cipher, then also the generator is secure.

1 Introduction

For confidentiality, the strongest notion of security that we can hope to achieve in practice is so called *semantic security*: whatever computational information that can be non-trivially derived from an encryption, $E(m)$, should also be possible to obtain even without $E(m)$. This notion was put forward in the seminal work by Goldwasser and Micali [9]. In the asymmetric case, we know that semantic security is impossible by deterministic systems, but [9] showed how to achieve it by probabilistic means. For (symmetric) block ciphers, we can easily see that the most obvious way of using it, Electronic Codebook Mode (ECB), is not semantically secure either. For instance, it is possible to determine relations between certain parts of a message, such as whether one part is a permutation of the another or not.

For this and other reasons, alternative *modes* of operation to ECB have been suggested, the most common being Cipher Block Chaining (CBC), Output

*johanh@nada.kth.se
†mats.naslund@era-t.ericsson.se

Feedback Mode (OFB), and Cipher Feedback Mode (CFB). The purpose of CBC is to remove the obvious weaknesses of ECB, but it is perhaps not immediately clear that it removes *all* weaknesses. However, in [1], Bellare et al. show that if the block cipher is in a sense "ideal", then the concrete security of CBC is as good as might hope.

The two remaining modes, OFB and CFB, are interesting as they provide means of using the block cipher as a key stream generator or stream cipher. A stream cipher is from a theoretical point easy to analyze as the only property needed from the key stream to give a semantic security is *pseudo randomness*: it should be computationally infeasible to distinguish, with non trivial success rate, the key-stream from a completely random stream of the same length. Therefore, it is natural to ask: how can a block cipher be deployed to produce pseudo random key streams? Does OFB/CFB provide security in this sense?

To begin with, it is natural to assume that the underlying block cipher in question is, by itself, secure. That is, under a reasonable attack model such as known/chosen plaintexts, a successful attack retrieving the key requires about 2^n work, where n is the keysize. If so, the block cipher is a *one-way function*: given the key, it is easy to compute the cipher text block, but given only cipher text blocks, it is hard to go backwards and find key. In the remainder of the paper we shall therefore assume that the block cipher we use (e.g. the AES algorithms Rijndael, [17, 3]) is one-way in this sense—any candidate found not to be one-way will certainly not be selected.

Constructing provably secure cryptosystems and pseudo random generators based on the presumed one-way properties of number-theoretic problems has achieved a lot of attention, [2, 9] being among the first in a long line of works from the 1980's. In the more classical, symmetric cryptography case, not much appears to have been done. Although it can be shown (in a theoretical sense) that any one-way function can be used to build a pseudo random generator, [10], such constructions are far too complex to have any practical implications. Moreover, it is easy to give examples of functions that although they might be one-way, they fail miserably in giving a pseudo random generators when used in CFB or OFB modes. In this paper we propose a new mode of encryption, *Key Feedback Mode*. It is slightly more complex than OFB/CFB, but on the other hand, we can actually prove that if the block cipher used is secure in a reasonable sense, then the mode indeed produces pseudo-random bit streams. Specifically, we give a quantitative relation between the workload needed to distinguish the keystream from true randomness, to the workload needed to retrieve the secret key.

The basic ingredients in our construction are complexity-theoretical results that have been known for little over a decade and are mainly due to works of Blum and Micali [2], Levin [12], and Goldreich and Levin [8]. The contribution of this paper is a highly optimized version of these results, which enables us to derive results for practical values of the parameters (key sizes) involved. In fact, whereas previous results were only applicable for key sizes of a few thousand bits or more, our method starts to be effective somewhere in the range 100–150 bits, and at 256 bits (or more), the results are quite strong. As a concrete example,

we can show that if we base our construction on a 256-bit block cipher, f, and generate 1Gbit of keystream, then an attack that correctly distinguishes this keystream from a truly random 1Gbit string with success rate 2^{-32}, gives an attack on the block cipher that is at least 2^{25} times more successful then one would expect for a "perfect" cipher, see §2.4.3.

Except for the aforementioned work in [1], there does not appear to be much previous theoretical work relating security of modes of block ciphers to the security of the cipher itself. Besides CBC, [1] also studies some different counter modes (see discussion in §3.3).

The paper is organized as follows. We first give some preliminary background in §2, briefly describing the theoretical results upon which the actual construction (presented in §2.4) is built. We then give a formal proof of security, §2.4.2. Some estimates of efficiency in practical implementations appear in §3.1.3, and a few "non-mathematical" attacks are discussed in §3.2.

2 Description

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 probability distribution on $\{0,1\}^n$. Except otherwise noted, log refers to logarithm in base 2, and ln is the natural logarithm.

We remind the reader of the two most fundamental concepts in cryptography and the theory of pseudo-randomness.

A *one-way function* is a function f so that f is polynomial time computable, but for any probabilistic, polynomial time algorithm, I, any $c > 0$ and sufficiently large n,

$$\Pr[f(I(f(x))) = f(x)] \leq \frac{1}{n^c},$$

probability over $x \in \mathcal{U}_n$ and I's random choices.

Secondly, we say that two probability distributions D_1, D_2, on strings of length n are (computationally) $\delta(n)$-*distinguishable*, if there is an efficient probabilistic algorithm, A, such that

$$|\Pr_{y \in D_1}[A(y) = 1] - \Pr_{y \in D_2}[A(y) = 1]| \geq \delta(n).$$

Here, $\delta(n)$ is called the *advantage* of A. If no such A exist, D_1, D_2 are called $\delta(n)$-*indistinguishable*. The reader familiar with statistical tests may think of A as such, trying to decide which distribution it sees as input. Note though that what A actually does is completely unspecified—any feasible test is permitted. Finally, a sequence of distributions distributions $\{D_n\}_{n \geq 1}$, where D_n has support on $\{0,1\}^n$, is said to be *pseudo-random* if for any $c > 0$ and sufficiently large n, D_n is n^{-c}-indistinguishable from $\mathcal{U}_n$.

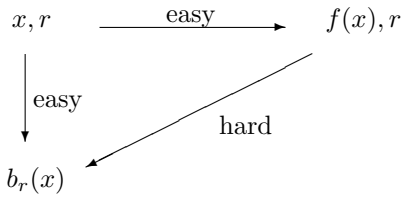
2.2 Pseudo Random Generators from One-way Function

As discussed, a block cipher running in a mode producing a secure key stream generator is really a pseudo random generator (henceforth abbreviated PRG). We therefore need to construct such a device. Suppose we have a one-way function, that in addition is a permutation, i.e. for each n, f is a one-to-one correspondence $\{0,1\}^n \to \{0,1\}^n$. Furthermore, suppose that we have a family of 0/1-functions, $B = \{b_r\}$, $b_r(x) \in \{0,1\}$, with the property that given $f(x)$ and a randomly chosen b_r, $b_r(x)$ is computationally indistinguishable from a random 0/1 coin toss. (I.e. predicting $b_r(x)$, non-negligibly exceeding the trivial guessing strategy's success rate of 1/2 is infeasible.) Then, the following construction, due to Blum and Micali [2], shows how to construct a PRG.

Choose x_0, r (the seed), let $x_{i+1} = f(x_i)$, output $g(x_0, r) = b_r(x_1), b_r(x_2), \ldots$ as the generator output.

Theorem 1 (Blum-Micali, '86). *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.*

We shortly give proof of this theorem for the specific case of the generator constructed in this paper, see §2.4.2. A set of functions, B as above, is called a (family of) *hard-core functions* for f. This can be depicted as in the figure below.



Notice that for the theorem to say anything, it is really necessary that f is a one-way function. If not, we *can* predict $b_r(x)$ from $r, f(x)$ by first inverting f and we would not get any contradiction.

For the above theorem it is important that f is a permutation. The problem if it is not, is that x_i might then after a few iterations of f become restricted to a small subset of the domain of f. Possibly, f could be easy to invert on that small subset. If f is not a permutation, but f behaves like a random function, however, then it can be shown (see Theorem 3) that after a moderate number of iterations, f is still one-way even when restricted to this set of inputs. Assumptions along these lines have been proposed by Levin [12] and are in fact necessary and sufficient of the existence of pseudorandom generators.

In summary, from a theoretical standpoint, this leaves us with the question: which one-way functions (if any) have hard-cores, and if so, what do these hard-cores look like?

2.3 The Goldreich-Levin Theorem

In 1989, Goldreich and Levin [8], proved that *any* one-way function (not only permutations) have hard-cores[1]. 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. The set $B = \{b_r\}$ consists of 2^n functions, each indexed by such a string r and $b_r(x)$ is defined as

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

that is, the inner product mod 2.

Theorem 2 (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.*

(A proof is given in §2.4.2.) So, if f is believed to be a one-way function, the existence of such A would be a contradiction. Note that the model is that adversary, trying to predict $b_r(x)$, gets both $f(x)$ and r for random r. Thus, a proof of security implies that in a practical application, r should be random, but need not be kept secret.

So far, the construction above gives one pseudo-random output bit, $b_r(x)$, per application of the function f. Even if f is fairly easy to compute, it is obvious that computing b_r will in practice be negligible compared to computing f. As established already in [8], one way to improve efficiency would therefore be to extract more than one bit at a time.

Indeed, it turns out that instead of outputting a single inner product modulo 2, it is possible to output as many as $m \in O(\log n)$ (where $n = |x|$) b_rs, corresponding to multiplying the binary vector x by a random $m \times n$ binary matrix, R. We denote the set of all such matrices $\mathfrak{M}_m$, and the corresponding functions $\{B_R^m \mid R \in \mathfrak{M}_m\}$. That is,

$$B_R^m(x) = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \bmod 2.$$

It is clear that from a theoretical point of view, outputting $\sim \log n$ bits is, in general, the best we can hope for. To give a security proof in practice with a real function f we need an exact analysis keeping track of all constants and that careful analysis is the bulk of this paper.

[1] 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. It is true that PRGs *can* be built from any one-way function, but without the permutation assumptions, the construction no longer becomes practical, [10].

2.4 The Construction

As described above we have a general construction given any one-way function.

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

Our goal is to see how secure this generator is given an assumption about the difficulty of inverting f. We first define security in terms of a distinguisher.

Definition 2. *Let D be a probabilistic algorithm which takes an input from $\{0,1\}^L$ and has binary output. Let p_r be the probability that D outputs 1 on a random input and p_G the probability that it outputs 1 on the output from a generator G. We then say that D is a (L, T, δ)-distinguisher for G if D runs in time T and $p_G \geq p_r + \delta$. If there is no (L, T, δ)-distinguisher for G then it is said to be* (L, T, δ)-secure.

Our definition of a distinguisher is only stated for the case of distinguishing random bits from the output of a generator. We note that there are more general situations where the distinguisher is trying to distinguish two distributions A and B and also that it can be given an element of a third distribution C, which might be coupled with A and/or B, as an aid. We do not give the formal definition of this concept at this moment.

Though we will not here make any particular assumption on the running time of the distinguisher, it is interesting to note that "practical" distinguishers are almost always faster than the generator undergoing the test (excluding poor generators such as linear congruential ones). This can be verified by anyone having experience with tests similar to Diehard, [15], or those proposed by Knuth [11].

The next section exactly relates the difficulty of inverting an iterated function f to the possibility of distinguishing the output of $BMGL_{n,m,L}(f)$ from random bits. When f is a cryptographic function such as a block-cipher or hash-function, then under the commonly used additional assumption of "random behavior" of f, we can bring things one step further, relating the security of $BMGL_{n,m,L}(f)$ more directly to the difficulty of inverting f itself. We first define our measure of success.

Definition 3. *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 the the value 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 3. *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 $(1-(i+1)2^{-n})^{T-i}$.*

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 i 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.

For the upper bound 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 $(1-(i+1)2^{-n})^{T-i}$. □

We would therefore expect that the best achievable time over success ratio to invert a random, iterated function is about $2^n/i$. Another argument that this is the "correct" complexity can be seen from [5]. Our concern the size of the image, $\mathrm{Im}(f^{(i)}(x))$. In [5] it is shown that if f is a randomly chosen function, then the expected size of $\mathrm{Im}(f^{(i)}(x))$ is $(1-\tau_i)2^n$, where $\tau_0 = 0$, $\tau_i = e^{-1+\tau_{i-1}}$. A Taylor expansion shows that $1-\tau_i$ is $\Theta(1/i)$.

The one-way functions, f, we will here consider are symmetric block-encryption functions viewed as encryption of a fixed message. Thus, the input to f is the key and the output is the cipher-text. (Alternatively, a cryptographic hash-function can be used as f, in which case input-output are taken in the obvious way.)

Definition 4. *A* σ-secure one-way function *is an efficiently computable function, $f(x)$, that maps n-bit strings to n-bit strings, such that the average time over success ratio for inverting the ith iterate is $\sigma 2^n/i$. That is, f cannot be (T, δ, i)-inverted for any $T/\delta < \sigma 2^n/i$.*

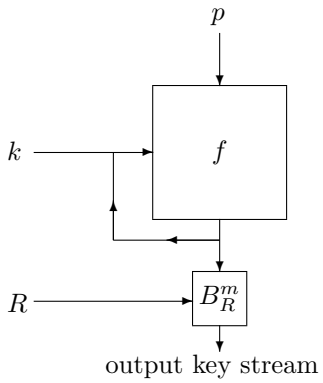
A block cipher is σ-secure, if the mapping $f_p(k)$, for known plaintext p, is a σ-secure one-way function of the key to the ciphertext.

(If a hash-function is used, σ-security is defined in a completely analogous way.) In the sequel we can now for concreteness think of f as Rijndael, [3]. Preferably, if there was an attack on $BMGL_{n,m,L}(f)$, we would like do draw direct conclusions from this as to the security of f. However, since the construction *iterates* f, and the fs we will use are not necessarily permutations, we cannot do this directly. Nevertheless, since one normally assumes that block-ciphers and hash-functions behave like random functions, we would naturally

also assume that Theorem 3 remains true for such f. A parameter of central interest is the success over time ratio which for a random function (and presumably also for our f) by Theorem 3 is about $2^n/i$ after the function has been iterated i times. Hence, for our "practical" choices of f, we expect them to be 1-secure in the above terminology.

2.4.1 The New Mode of Operation

Using a block cipher $f = f_p(k)$ as above in the $BMGL_{n,m,L}(f)$ construction, we now view as our new mode of operation for that cipher. As can be seen, it is in some sense a "dual" and more elaborate version of OFB. By duality we refer to the fact that the feedback is to the key, not to the plaintext block. The output generation is therefore by necessity a bit more complex as it does not (and in fact cannot) use the ciphertext output block directly. The mode, which we assign the working title KFB (Key Feedback mode) can thus be viewed as follows



The role of the initialization vector in OFB (the "IV") is taken by the output generation matrix R (the plaintext p may be chosen as a constant), and the initial value of k is the key to our cryptosystem. As for OFB, the IV need not be kept secret, but must be random. The number of bits needed for our IV may at this point seem very large. We shall later discuss ways of significantly reducing the size, while still keeping a provable security. To start with though, we assume it is of "full" size.

Also note that in practice, the speed is a bit lower than for OFB, the extra overhead being dominated by the need to perform the key schedule for each "block" of output keystream. We return to efficiency/implementation issues later.

We next formally prove the security of the mode.

2.4.2 Security of the generator

Our objective is to show that if $BMGL_{n,m,L}(f)$ 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 (block cipher) f. In particular, we show the following theorem:

Theorem 4. *Suppose 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. Suppose this generator can be (L, T, δ)-distinguished. Then, setting $\delta' = \frac{\delta m}{2L}$, there is an integer $i \leq L/m \triangleq \lambda$ such that f can be $(T', \delta'/4, i)$-inverted, where T' equals*

$$\delta'^{-1}(2n+1)2^{m+2}(n[2m+T+1+\log(2n+1)+2\log\delta'^{-1}]+E).$$

In particular, the time over success ratio is about $n^2 m^{-1} 2^m L^2 \delta^{-2} T$. For any $\mu \in (0,1)$, the value of i can, with probability at least $(1-\mu)^{\log\lambda}$, be found in time $\frac{3}{2}\lambda^2\delta^{-2}(T+S)\log\lambda\ln\mu^{-1}$.

This result can in principle be obtained directly from the original works by Blum-Micali and Goldreich-Levin, but here we are interested in a tight result and hence we have to be more careful than in [8] were, basically, any polynomial time reduction from the inverting f to distinguishing the generator would be enough. Optimizations of the original proof also appeared in [13]. In the remainder of the paper, we shall treat operations on n-bit strings (such as taking their bitwise XOR), as elementary, taking constant time. The first step is the lemma below.

Lemma 5. *Let $L = \lambda m$. Suppose that $BMGL_{n,m,L}(f)$ 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 $f^{(i)}(x), R$, distinguishes $B_R^m(f^{(i-1)}(x))$ from $\mathcal{U}_m$ with advantage $\delta' \geq \frac{\delta m}{2L}$.*

$P^{(i)}$ depends on an integer i, and for any $0 < \mu < 1$, using $\frac{3}{2}\lambda^2\delta^{-2}\log\lambda(T(L)+S(L))\ln\mu^{-1}$ operations, i can be found with probability at least $(1-\mu)^{\log\lambda}$.

The proof uses an optimized version of the so called *universality of the next-bit-test*, by Yao [20], see also [2].

Proof. For simplicity let us refer to our generator simply as G. Let D be an algorithm that uses $S(L)$ operations and distinguishes the output of G from a random L-bit string with advantage δ. Define the hybrid distributions H^i, $i = 0, 1, \ldots, \lambda$, on $\{0,1\}^L$ as follows. First, x_0, R are randomly chosen in $\{0,1\}^n \times \mathfrak{M}_m$. Next, the first im bits of H^i are obtained as outcomes of random coin-flips. Then, to generate bits $im+1, im+2, \ldots, L$, we apply the generator by setting $x_j = f^{(j)}(x_0)$, $j \geq i$, and let the jth m-bit block of H^i be $B_R^m(x_j)$.

Observe that H^0 equals the distribution of outputs of G, and that H^λ is the uniform distribution on $\{0,1\}^L$. By assumption, D distinguishes H^0 and H^λ with advantage δ, so from the triangle follows the existence of an i, $0 \leq i < \lambda$, for which D distinguishes H^i from H^{i+1} with advantage at least δ/λ. We postpone the discussion how to find such an i for the moment.

The algorithm $P^{(i)}$ (depending on i) now works as follows. On input $(r, y = f^{(i)}(x), R) \in \{0,1\}^m \times \{0,1\}^n \times \mathfrak{M}_m$ it produces an element by first flipping im coins, appending r, and then iteratively applying f, B_R^m $\lambda - i$ times as in the generator construction. Call the generated element z and observe that if r is random, then z is an element from H^{i+1}, whereas if $r = B_R^m(f^{(i-1)}(x))$, z is

from H^i. $P^{(i)}$ then feeds z to D and answers the same as D does. Clearly, $P^{(i)}$ distinguishes with the same advantage as D distinguishes H^i and H^{i+1}.

To find the right i, we perform a binary search and evaluate the alternatives by sampling. Due to sampling errors the quality of the found i can not be guaranteed to be exactly as good as the i in the existence proof above. Assume for notational simplicity that λ is a power of 2.

It must either be the case that D distinguishes between $H^0, H^{\lambda/2}$, or, between $H^{\lambda/2}, H^{\lambda}$ with advantage at least $\delta/2$. In general, for $j = 1, \ldots, \log\lambda$, there musty be an $l_j \in [0..2^j - 1]$ such that D distinguishes $H^{l_j 2^{-j}\lambda}, H^{(l_j+1)2^{-j}\lambda}$ with advantage at least $\delta_j \triangleq 2^{-j}\delta$. We determine an l_j that does almost this well. The problem of finding the best l_j is that we have sampling errors. Let t be a parameter.

Set $l_0 \triangleq 0$. Repeat for $j = 1, \ldots, \log\lambda$. We run D t times on $H^{(2l_{j-1}+1)2^{-j}}$, $H^{(2l_{j-1})2^{-j}}$, $H^{(2l_{j-1}+2)2^{-j}}$ and record the number of 1-answers as c, c_l and c_r, respectively. If $|c_r - c| \geq |c - c_l|$ we set $l_j = 2l_{j-1} + 1$ and otherwise $l_j = 2l_{j-1}$.

We say that the j iteration above is successful if for the l_j found we have that D distinguishes $H^{l_j 2^{-j}\lambda}$ and $H^{(l_j+1)2^{-j}\lambda}$, with advantage at least $(1 - 2^{j-(\log\lambda+1)})\delta_j$. If all iterations are successful, this gives us $\delta_{\log\lambda}$ as claimed. Note, that by assumption we start in a successful situation. We now estimate the probability that we are successful in iteration j.

Given success up to iteration $j-1$, we know that for the true probabilities p_l and p_r of D of outputting 1 on $H^{l_{j-1}2^{-(j-1)}\lambda}$ and $H^{(l_{j-1}+1)2^{-(j-1)}\lambda}$ we have $|p_l - p_r| \geq (1 - 2^{(j-1)-(\log\lambda+1)})\delta_{j-1}$. Now assume that one of of the two choices for l_j is bad. Let p be the probability that D outputs 1 on $H^{(2l_{j-1}+1)2^{-j}\lambda}$. Clearly it is enough to estimate the probability that $l_j = 2l_{j-1}$ in the case when $p_l - p_r \geq (1 - 2^{(j-1)-(\log\lambda+1)})\delta_{j-1}$, and $p_l - p < (1 - 2^{j-(\log\lambda+1)})\delta_j$. We have to estimate the probability that $S = c_l + c_r - 2c$ is positive. Clearly the expected value of this is $t(p_l + p_r - 2p)$ which by the above assumption equals

$$t(p_r - p_l) + 2t(p_l - p) > t2^{j-1-(\log\lambda+1)}\delta_{j-1} = \frac{t\delta}{2\lambda}.$$

Setting $t = \frac{1}{2}\lambda^2\delta^{-2}\ln\mu$ and applying a Chernoff bound, the error is bounded by μ. To analyze the running time is straightforward since we have t calls to G for each step in binary search. □

We now reprove the theorem of Goldreich and Levin trying to be careful with our estimates and construction.

Theorem 6. *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 at least ϵ where ϵ is given. Then we can in time*

$$(2n+1)\epsilon^{-2}2^m(2m + \log((2n+1)\epsilon^{-2}) + 1 + T)n$$

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

Before we give the proof, some additional preliminaries. 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 $\mathbb{Z}_2$. (Computationally, for the practical values of k that will be of interest to us, we may assume that such q is available by table look-up and if really needed, we can easily find a q with an expected number of k^4 operations.) 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.

Lemma 7. *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 $mk^3 2^k + m + k$ to generate a set of $s = 2^k$ uniformly distributed, pairwise independent matrices $R^1, \ldots, R^s \in \mathfrak{M}_m$. 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 $B_{R^j}^m(x) = M_j z$ for all j.*

The construction is similar to that of Rackoff in his proof of the Goldreich-Levin theorem, see [6, 7].

Proof. Choose randomly and independently m strings, $a_0, a_1, \ldots, a_{m-1}$ and k strings $b_0, \ldots, b_{k-1}$, each 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 $\mathbb{F}_{2^k}$ (this is simply 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 and b_l are chosen at random. To show pairwise independence it suffices to show that an XOR of any subset of elements from any two matrices is unbiased. Since the columns are independent, it is enough to show that the XOR 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 XOR uniformly distributed. Thus, the XOR can be written

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

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. It is easy to check that they can be calculated as described. If z_l denotes the lth choice for the $m+k$ bits, then one z_l will as claimed determine all the matrix products by setting $M_j = [I_m|B^j]$, where I_m is the $m \times m$ identity matrix and B^j is the $m \times k$ matrix whose ith row is $\mathrm{bin}(\alpha_j \cdot t^i)$. □

The requirement $m < k$ is really not essential for the applications we have in mind for the lemma. We will only use it to guarantee the existence of *at least* 2^k matrices with the above properties. Technically, if $k < m$, we can carry out the same details, replacing k by $m+1$ in the proof and then simply not use the $2^{m+1} - 2^k$ "extra" matrices. However, as will be seen, the interesting uses of the lemma is when $k > m$.

Before we go on, recall that any function $g : \{0,1\}^m \to \{-1,1\}$ can be expressed as a linear combination over the orthonormal set of functions $\{\chi_u(z) = (-1)^{\langle u,z\rangle_2}\}$, where $\langle u,z\rangle_2 = \sum_{i=0}^{m-1} u_i x_i \bmod 2$. More precisely,

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

where $\hat{g}_u \triangleq \mathrm{E}_v[\chi_u(v)g(v)] = 2^{-m}\sum_v (-1)^{\langle u,v\rangle_2} g(v)$. In other words, this is the discrete Fourier series expansion of g.

Recall that the Fourier transform of 2^t elements can be computed in time $t2^t$ by the following observations:

$$\hat{g}_u = \sum_{v=0}^{2^{t-1}-1} (-1)^{\langle u,v\rangle_2} g(v) + \sum_{v=2^{t-1}}^{2^t-1} (-1)^{\langle u,v\rangle_2} g(v), \tag{2}$$

and if $u' = u + 2^{t-1}$ then similarly:

$$\hat{g}_{u'} = \sum_{v=0}^{2^{t-1}-1} (-1)^{\langle u,v\rangle_2} g(v) - \sum_{v=2^{t-1}}^{2^t-1} (-1)^{\langle u,v\rangle_2} g(v), \tag{3}$$

neglecting the constant factor 2^{-t}. We thus compute two sub-transforms of half the size by conditioning on the most significant bit of v, and from the sign, determined by the msb of u, we can combine the solution for the two subproblems. For convenience, we shall therefore sometimes change the range of function from $\{0,1\}$ to $\{-1,1\}$ in the natural way.

Lemma 8. *Let P be an algorithm, mapping pairs $\mathfrak{M}_m \times \{0,1\}^m \to \{-1,1\}$, whose running time is T, let R^j, M_j be the matrices generated as described in Lemma 7 and let S be an arbitrary matrix in $\mathfrak{M}_m$.*

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,

$$c_l = \mathrm{E}_j[P(R^j + S, B^m_{R^j}(x))].$$

The value of l is independent of S.

(The role of the matrix S may seem unclear at this point, but will be explained shortly.)

Proof. First run P on all the 2^{m+k} possible inputs of form (R^j+S, r) and record the answers: $\{P(R^j+S, r)\}$. A fixed value of l above corresponds to a value of the $m+k$ bits z_l in Lemma 7. Let us assume that z_l is the correct choice, i.e. $B^m_{R^j}(x) = M_j z_l$. Then, by construction,

$$c_l \triangleq \sum_{j=0}^{2^k-1} P(R^j+S, M_j z_l) = \sum_{j=0}^{2^k-1}\sum_{r=0}^{2^m-1} P(R^j+S, r)\Delta(r, M_j z_l), \qquad (4)$$

where $\Delta(r, r') = 1$ if $r = r'$ and 0 otherwise. The naive way to calculate this number would require time 2^{2k+m} and we want to 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 (4) equals

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

Let $Q(j, \alpha)$ be the inner sum. Fix a value of j. The different α-values then correspond to a Fourier transform and hence the 2^m different $Q(j, \alpha)$ can be calculated in time $m2^m$. Thus, all the numbers $Q(j, \alpha)$ can be computed in time $m2^{k+m}$. Finally we have

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

where M_j^T is the transpose. Comparing this to (1), (2), and (3) above, this is just a rearrangement of a standard Fourier-transform of size 2^{k+m} and can be computed with $(k+m)2^{k+m}$ operations. The lemma follows. □

Now we prove that we can compute useful information about x.

Lemma 9. *Suppose*[2] *that $N2^m > mk^3$. Let P, T, x be as in Theorem 6 and let t be a parameter such that $2^k = t\epsilon^{-2}$ for some $k \geq m$. Then for any set of N vectors $\{v_i\}_{i=1}^N \subset \{0,1\}^n$ we can in time $2^{m+k}(2m+k+T+1)N$ produce a set of lists $\{b_i^{(j)}\}_{i=1}^N$, $j = 1, 2, \ldots, 2^{k+m}$ 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 $\frac{N}{2t}$ of the N possible values of i.*

[2] This is reasonable for the values we will consider.

Proof. Start by randomly generating the 2^k matrices $\{R^j\}$ as described in Lemma 7. Now repeat the process below for each $i = 1, \ldots, N$.

Select a random string $s_i \in \{0,1\}^m$, and let S_i be the $m \times n$ matrix defined by $S_i \triangleq s_i \otimes v_i$ (the outer product[3]). Notice that by linearity

$$(R^j + S_i)x = R^j x + s_i \langle v_i, x \rangle_2, \tag{5}$$

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

As described in Lemma 8, we now compute the values $\{c^i_l\}$.

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

Focus on the correct choice for l. If $\langle v_i, x \rangle_2 = 0$, then c^i_l is the outcome of a uniform 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.

Let p_1 be the probability that P outputs 1 on $(R, B^m_R(x))$ and p_2 the probability that it outputs 1 on (R,u). We can without loss of generality[4] assume that $p_2 = \frac{1}{2}$ so that $p_1 \geq \frac{1}{2} + \epsilon$. Let $p \triangleq (1+\epsilon)/2$. Then we would simply 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 this happens is bounded by $2^{-k}\epsilon^{-2} = t^{-1}$.

This implies that for the correct value of l the expected number of errors is N/t, and by Markov's inequality, with probability at least at $1/2$ it is below $2N/t$. Trying all these alternatives for l gives the lists described in the lemma.

The total running time is now $2^k[2^m(2m+k+T)N + mk^3] + m + k$, which by the assumption on m, k, N is at most $2^{k+m}(2m+k+T+1)N$. □

Let us next establish Theorem 6.

Proof of Theorem 6. We apply Lemma 9 with $t = 2n+1$, $N = n$, and set the vectors $\{v_i\}_{i=1}^n$ to be the unit vectors so that $\langle v_i, x \rangle_2$ gives x_i, the ith bit of x. With probability $1/2$ one of the lists give all the inner-products correctly and hence determine x. □

We use this to establish Theorem 4.

Proof of Theorem 4. First we apply Lemma 5 to get an i for which we have an algorithm that when given $f^{(i)}(x)$ runs in time $S(L) + T(L)$ and distinguishes $B^m_R(f^{(i-1)}(x))$ from random bits with advantage at least $\delta' \triangleq \frac{\delta m}{2L}$. Since δ' is an average over all x we need to do some work before we can apply Theorem 6.

[3] $(S_i)_{k,l} = (s_i)_k \cdot (v_i)_l$

[4] If not, modify P to $P'(R,r) = P(R,r)$ with prob. $1/2$ and $P'(R,r) = 1 - P(R,r')$, for $r' \in \mathcal{U}_m$, otherwise. This will reduce the distinguishing advantage, but only by a factor 2.

For each x we have an advantage δ_x and we know that the expected value of δ_x is at least δ'. Let $\mathcal{G} \triangleq \{x \mid \delta_x \geq \delta'/2\}$, the set of *good* x. We proceed as follows.

Choose a random value of $j \geq 1$ where a specific value j_0 is chosen with probability 2^{-j_0}. If $2^{-j_0} \geq \delta'/2$ then we apply Theorem 6 with $\epsilon = 2^{-j_0}$ while if $2^{-j_0} < \delta'/2$ we give up. Let $p_x = 1$ if we by this retrieve x, 0 otherwise.

To analyze this procedure, first observe that for $x \in \mathcal{G}$, the preconditions for Theorem 6 are satisfied if $\delta_x \geq 2^{-j_0} \geq \delta'/2$, and in this case the algorithm finds x with probability at least $1/2$. The above condition is true with probability (over j_0) at least $\delta_x - \delta'/2$. For $x \notin \mathcal{G}$ we accept the possibility of failure, but still $\mathrm{E}[p_x] \geq 0$. This implies that the overall probability of success is

$$\mathrm{E}_x[p_x] \geq \frac{1}{2}\,\mathrm{E}_{x\in\mathcal{G}}[\delta_x - \delta'/2]\,\Pr_x[x \in \mathcal{G}] \geq \frac{1}{2}\,\mathrm{E}_x[\delta_x - \delta'/2] \geq \delta'/4.$$

To evaluate the expected running time, we have

Claim 10. *The expected running time of the algorithm in Theorem 6, when run with parameter $\epsilon = 2^{-j}$ with probability 2^{-j} provided $2^{-j} \geq \delta'/2$, is bounded by*

$$\delta'^{-1}(2n+1)n2^{m+2}(2m + T + 1 + \log(2n+1) + 2\log\delta'^{-1})$$

in our previous notation.

Proof. We need to bound the expected value, when $\epsilon = 2^{-j}$ with probability 2^{-j}, of $t\epsilon^{-2}2^m(2m + \log(t\epsilon^{-2}) + T + 1)N$ (where $N = n$, $t = 2n+1$) truncating at $j_0 - 1$, where j_0 is the smallest integer satisfying $2^{-j_0} < \delta'/2$. Rewriting this, collecting powers of ϵ^{-1} and $\log\epsilon^{-1}$ we get

$$tN2^m([2m + T + 1 + \log t]\epsilon^{-2} + \epsilon^{-2}\log\epsilon^{-2}).$$

First note that

$$\mathrm{E}_\epsilon[\epsilon^{-2}] = \sum_{j=1}^{j_0-1} 2^j < 2^{j_0} \leq 4\delta'^{-1}.$$

Then

$$\mathrm{E}_\epsilon[\epsilon^{-2}\log\epsilon^{-2}] \leq 4\delta'^{-1}\log(2\delta'^{-1}) \leq 6\delta'^{-1}\log(\delta'^{-1})$$

for δ' "of interest" (say $\delta' \leq 1/4$). Thus the expected running time is as claimed. □

We then finally need to try all candidate x in the list, applying f to each. The expected length of the list is

$$(2n+1)2^m\,\mathrm{E}_\epsilon[\epsilon^{-2}] = 4(2n+1)2^m\delta'^{-1}.$$

Adding this gives the running total time as stated. □

Some values of j above will turn out to be more profitable than others and the above algorithm can be optimized by a preprocessing stage that finds the best j; one for which $\Pr_x[\delta_x \in [2^{-j}, 2^{-j+1})]$ is fairly large. Since we are only analyzing the time for a single inversion we omit the details.

Instead of applying Lemma 9 with the unit vectors we can, as suggested in [7], use it with $\{v_i\}$ describing the words of an error correcting code. (Similar ideas appears in [12].) If we have code words of length N, containing n information bits, and we are able to efficiently correct e errors we get:

Theorem 11. *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, C, with n information bits, N message bits that is able to correct e errors in time T_C. Then we can in time*

$$\frac{N}{e}\epsilon^{-2}2^{m+1}([2m + \log(N\epsilon^{-2}/e) + T + 2]N + T_C)$$

produce a list of $\frac{2^{m+1}N}{e}\epsilon^{-2}$ numbers such that the probability that x appears in this list is at least $1/2$.

Proof. We apply Lemma 9 with $t = 2N/e$ and $\{v_i\}_{i=1}^N$ as the row vectors of the generator matrix for C. This produces $\frac{2^{m+1}N}{e}\epsilon^{-2}$ vectors $\{c_j\} \subset \{0,1\}^N$, such that with probability at least $1/2$, one c_j is at Hamming distance at most e from the correct codeword corresponding to x. Running the decoding algorithm on each c_j then produces a list as claimed. □

In much the same way as the proof of Theorem 4, this translates to the quality of the inverter.

Theorem 12. *Suppose 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 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 $\delta' = \frac{\delta m}{2L}$, there is an $i \le L/m \triangleq \lambda$ such that f can be $(T', \delta'/4, i)$-inverted where T' equals*

$$\delta'^{-1}\frac{N}{e}2^{m+3}\left([2m + T + \log(N/e) + 2\log\delta'^{-1} + 2]N + E + T_C\right).$$

In particular, the time over success ratio is about $\frac{N^2}{e}m^{-1}2^m L^2\delta^{-2}(T + T_C/N)$. For any $\mu \in (0,1)$, the value of i can, with probability at least $(1-\mu)^{\log\lambda}$, be found in time $\frac{3}{2}\lambda^2\delta^{-2}\log\lambda(T+S)\ln\mu^{-1}$.

2.4.3 A Concrete Example

What does all this say? Suppose that we want to generate $L = 2^{30}$ bits (1Gbit) of key stream, using a good block-cipher f, applying our construction with $m = 40$.

Corollary 13. *Consider $G = BMGL_{256,40,2^{30}}(f)$ where f is a block-cipher or hash-function (with key/block length 256) and where f is computable by E operations, and assume that G runs in time $S \sim EL/m$. If G can be $(2^{30}, T, 2^{-32})$-distinguished, then there is an $i < 2^{30}/40$ so that f can be (T', δ', i)-inverted where $\delta' \approx 2^{-59}$, and the running time T' is composed of approximately 2^{116} applications of the distinguisher, about 2^{109} applications of f and about 2^{124} additional operations. The value of i can be found (with probability at least 0.65) using about 2^{121} applications of G and the distinguisher.*

Making the additional assumption that $T \leq S$, the pre-processing to find i is about 2^{152}, and once i has been found, the time over success ratio is about 2^{208}. In light of Definition 4, f can not be 2^{-25}-secure.

Recall that we would normally expect f to be 1-secure.

Proof. Apply Theorem 4, setting $\mu = \frac{\ln(4/3)}{\ln \lambda}$. Since G iterates f $i = L/m$ times, Theorem 3 suggests that the best possible time over success ratio would be about 2^{231}, which contradicts any belief that f is 2^{-25}-secure. □

Comment: The assumption $S = EL/m$ is natural as the output generation should not be significantly more expensive than computing f. The assumption $T \leq S$ is, again, motivated by considering "practical" tests a la Diehard or those by Knuth.

If more than a single inversion is to be performed, it is possible to improve the above Corollary by using a good error correcting code and applying Theorem 12 instead. In particular, using a certain *Goppa-code*, see [14], for the same n, L, m and assumed distinguisher performance as above, once the value of i is found, we can reduce the workload per inversion by approximately a factor of 8. Still, the pre-processing time needed to find i is not affected by the use of the code, and the achieved time-over-success ratio for a single inversion is more or less the same and we here omit further details.

A smaller m gives higher security. Let us see how security varies with m, δ:

Corollary 14. *Consider $G = BMGL_{256,4,2^{30}}(f)$ (generating 1Gbit, extracting 4 bits per iteration of f) where f is computable by E operations. If G can be $(2^{30}, T, 2^{-40})$-distinguished, then there is an $i < 2^{28}$ so that f can be (T', δ', i)-inverted where $\delta' > 2^{-71}$, and the running time T' is composed of approximately 2^{92} applications of the distinguisher, about 2^{84} applications of f and about 2^{99} additional operations. The value of i can be found (with probability at least 0.65) using about 2^{143} applications of G and the distinguisher.*

Making the additional assumption that $T \leq S$, the time to find i is about 2^{179}, and once i is found, the time over success ratio is about 2^{197}. In light of Definition 4, f can not be 2^{-32}-secure.

The proof is as above. We give another example in the next section where we discuss methods of reducing the IV-size and how they affect security.

3 Discussion

3.1 Issues

3.1.1 Choice of f

The two obvious alternatives are to use conjectured very strong one-way functions which are either given by encryption functions or one-way hash-functions. (Due to the overhead of the key-schedule, the latter will in practice be a bit more efficient.) It would be advantageous to use one which supports a combination of block- and key-size of 256 bits (as Rijndael can do).

Note that the one-way function we are suggesting to use is to have a fixed message, p, and let the input be the encryption key, k, and the output the cipher-text. In some sense it would be better to have the mapping from clear-text to crypto-text as our f since it is a permutation, i.e. iterate $f_k(p)$ rather than $f_p(k)$. This would also eliminate the need to perform the key-scheduling on each iteration. The problem is that this is by definition not a one-way function since anybody that can compute it can also invert it. We are unable to get any provable properties when f is used in this way.

3.1.2 Number of bits output per application of f

We would for efficiency reasons like to output as many bits as possible per application of f. On the other hand, we cannot output too many, if we are to relate the security of f to the proof of security for the generator. The effect of varying m is clearly visible in the above theorems.

One problem with a large value of m is that size of the seed grows with m, we discuss this issue below.

3.1.3 Efficiency

The output generation, consisting of computing the m inner products mod 2 should not be critical issue for the performance. It is true that most micro processors do not have an instruction to compute inner products between registers, but it is quite easy to implement this operation fairly efficient. One possible way could be to do the following.

We assume a 32-bit architecture. Precompute a 256-byte table, `tab`, where `tab[i]` holds the remainder modulo 2, of the number of 1s in the binary representation of `i`. Let $\wedge, \oplus$, and $\gg$ denote bitwise AND, XOR, and right shift, respectively. The inner product of two 8-word (i.e. 256 bit) vectors `r[]`,`x[]` is done by

```
c = 0;
for i = 1 to 8 do
    c = c ⊕ (r[i] ∧ x[i]);
c = c ⊕ (c ≫ 16);
c = c ⊕ (c ≫ 8);
return tab[c ∧ 255];
```

That is, about 40 instructions suffices. Alternatively, the table look-up in the last line can be replaced by a "table" of size 16, stored in an integer:

```
c = c ⊕ (c ≫ 4);
return (6996hex ≫ (c ∧ 15)) ∧ 1;
```

Using a block cipher, each iteration then requires one key set-up and one encryption. Looking at the performance evaluations for Rijndael, see e.g. [4], speeds of several Mb/s seem well within range.

For some quick tests, we used a rather naive, completely non-optimized C-implementation of RC6, [18], and extracted $m = 32$ bits per iteration. We achieved speeds somewhat below 1Mbit/s on a standard PC.

3.1.4 Size of Initialization Data

To determine an $m \times n$ matrix to be used as part of the seed, we need nm bits. We note that it is possible to reduce this to only n bits by choosing matrices R as follows. Consider the finite field of 2^n elements. Binary strings of length n are interpreted as elements of this field in the natural way. The function $h_A(x)$, mapping this field to itself, defined by $x \mapsto Ax$, can be represented as a matrix multiplication by a boolean $n \times n$ matrix, A', where A' is uniquely determined by n bits (and the irreducible polynomial representing the field). Now define a function $r_A(x)$ by selecting any fixed set of m bits of Ax. This was in [16] shown to give hard core functions. That is, independently of m, n bits are always enough to specify the m-bit output function (as long as $m \leq n$).

There is however a drawback with this construction. In our current reduction we can show that a distinguisher for $B_R^m(x)$ with advantage δ, essentially enables us to predict $\langle v_i, x\rangle_2$ with the same advantage. This is then amplified using Lemma 7. Since the set of possible A suggested above is more restrictive, we are not able to prove the equivalent of Lemma 7 with this space of matrices. Still, it is possible to get the same kind of reduction, but the only way (we know) of doing this is via the so called *Computational XOR-Lemma*, [19]. Unfortunately, involving this lemma reduces the δ-advantage of the distinguisher to a $2^{-m}\delta$-advantage for the predictor for $\langle v_i, x\rangle_2$. Nevertheless, for small m, when small seeds are of interest, this could be a possible trade-off one is willing to make.

Moreover, for a fixed desired security level and seed-size, we can to some extent now choose m to achieve this. Specifically, doing the same kind of argument as in §2.4.3 we can show that

Corollary 15. *Consider $BMGL_{256,8,2^{30}}(f)$ where we choose the matrix R as above (i.e. as 8 rows out of an 256×256 matrix determined by an element of $\mathbb{F}_{2^{256}}$), rather than as random 8×256 matrices. If G can be $(2^{30}, T, 2^{-32})$-distinguished, then there is an $i < 2^{27}$ so that f can be (T', δ', i)-inverted where $\delta' \approx 2^{-70}$, and the running time T' is composed of approximately 2^{101} applications of the distinguisher, about 2^{93} applications of f and about 2^{108} additional operations. The value of i can be found (with probability at least 0.65) using about 2^{142} applications of G and the distinguisher.*

Making the additional assumption that $T \leq S$, the pre-processing to find i is about 2^{176}, and once i has been found, the time over success ratio is about 2^{205}. In light of Definition 4, f can not be 2^{-25}-secure.

Comparing this to $m = 40$ and the general way of seeding, we obtain the same level of security but reduce the seed size from 1312 to 64 bytes. The price we pay is the need to reduce m to 8, leading to a slow-down by an estimated factor of 2 to 3 times (we need 5 times more applications of f to generate the same length of the stream, but on the other hand, the output generation itself is now 5 times as fast).

An alternative, suggested in [8], is to pick R as a random Toeplitz matrix in which case $n + m - 1$ bits are sufficient. Also in this case we only know how to prove security through the computational XOR-lemma and hence also in this case we lose a factor 2^m in security for a fixed m.

3.2 Specific Attacks

Though we have proven security in a very general model, there are a few specific attacks whose effectiveness could be interesting to study in detail.

Exhaustive Key- and Internal State Search. Any key stream generator is of course vulnerable to this attack. Our proof above states that to some extent, such attacks are the only ones. A key length of 256 (or even 128) will thwart such attempts for the foreseeable future.

Chosen IVs. We again refer to our formal analysis in the proofs above where we allow "experiments" with the generator in this fashion. Our proof model is that the R-matrix is chosen at random, but in practice, the only "bad" IV would one where R has a row consisting of all zeros as plaintext bits would then leak. Of course, if the same key is used twice, at least the part of the IV corresponding to the plaintext block should be replaced as otherwise, the key stream will be identical.

Cycle Shortening. A worry might be that the generator produces short cycles. However, if we have a cycle of length L, then we clearly also have an $(L, O(L), 1/2)$-distinguisher and thus also an attack of f. Moreover, if we believe f to behave randomly, the expected cycle length is on the order $2^{n/2}$.

3.3 Comparison to Counter Mode

One method to make a pseudorandom generator from a one-way function is to use it in counter mode where the keystream is computed as $f(c), f(c + 1), \ldots$ where c is the secret key. Bellare et al., [1], show that if f is a pseudo random permutation (PRP), then running f in counter mode gives a provably secure cipher. Thus, based on this a assumption they get a very simple and efficient generator.

Our assumpiton, that f's behavior with respect to inversion is similar to that of a random function, is much weaker and hence it is not surprising that our construction is more complex and gives a less efficient generator.

One can argue that in the above mentioned result about countermode you essentially assume that your primitive has the required property (being random) while we have to produce randomness from the weaker property of being hard to invert.

Since our assumptions are strictly weaker we would expect that a cautious user who does not have extreme demands on the speed of the generator would find it a very useful alternative.

3.4 Other Properties

Error robustness. In many applications, in particular over unreliable transmission media such as wireless, certain modes of operation (e.g. CBC) might be disqualified due to error propagation: a single bit-error introduced by the transmission medium destroys one entire block or more of the decrypted plaintext. KFB, like OFB, does not have this property and is error robust: errors are confined to the bits in which they occur.

Synchronization. Like OFB, the mode requires sender/receiver synchronization of the key stream.

Key stream advance/rewind. Some applications such as encryption in packet switched networks, where reordering of packets may be introduced by the transmission mechanism are facilitated if a "random access" feature is present. That is, if it possible to directly (in constant time) jump to any given location within the key-stream. Typically, so called *counter mode* has this property, but unfortunately, KFB does not. It seems difficult to modify the mode in a way that enables such jumps in the stream, without sacrificing the provable properties.

4 Intellectual Property Issues

As far as we have been able to tell, neither the Blum-Micali, nor the Goldreich-Levin construction are covered by patents. Of course, since the block-cipher, f, can be more or less freely chosen, one should be aware that many block ciphers are patented. In the specific case of the AES algorithm, this should not not be a problem.

The authors of this paper have no patent or IPR claims related to the construction proposed here.

5 Summary and Conclusions

We have given a suggestion of a new mode of operation for the AES algorithm Rijndael (or any other block cipher), giving a key stream generator with provable

security properties, that we believe also in practice will give an excellent trade-off between strong cryptographic security and efficiency. We acknowledge the fact that the construction is less efficient than, say OFB, and may not be applicable to high speed, real time applications. However, in many situations where a high confidence in security is desired, the efficiency of the new mode will be fully acceptable, as practical tests have shown.

References

[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway: *A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation.* Proceedings of the 38th IEEE FOCS, 1997.

[2] M. Blum and S. Micali: *How to Generate Cryptographically Strong Sequences of Pseudo-random Bits.* SIAM Journal on Computing, **13**, no 4, 1986, 850–864.

[3] J. Daemen and V. Rijmen: *AES Proposal: Rijndael.* Available at [17].

[4] B. Gladman: *Implementation Experiences with AES Candidate Algorithms.* Proceedings, 2nd Advanced Encryption Standard Candidate Conference, 1999, pp. 7–14.

[5] P. Flajolet and A. Odlyzko: *Random Mapping Statistics.* Proceedings, Eurocrypt '89, LNCS 434, pp. 329–354, Springer-Verlag.

[6] O. Goldreich: *Foundations of Cryptography (Fragments of a Book).* Available on-line at `http://philby.ucsd.edu/cryptolib.html` (Theory of Cryptography Library)

[7] O. Goldreich: *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag 1999.

[8] O. Goldreich and L. A. Levin: *A Hard Core Predicate for any One Way Function.* Proceedings, 21st ACM STOC, 1989, pp. 25–32.

[9] S. Goldwasser and S. Micali: *Probabilistic Encryption.* Journal of Computer and System Sciences **28** (1984), 270–299.

[10] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby: *Pseudo Random Number Generators from any One-way Function.* SIAM Journal on Computing, **28** (1999), 1364–1396.

[11] D. Knuth, *Seminumerical algorithms,* (2 ed.), Volume 2 of *The art of computer programming,* Addison-Wesley, 1982.

[12] L. Levin: *One-way Functions and Pseudorandom Generators.* Combinatorica 7 (1987), 357-363.

[13] L. Levin: *Randomness and Non-determinism.* J. Symb. Logic, **58**(3), 1102–1103, 1993.

[14] F. J. MacWilliams and N. J. A. Sloane: *The Theory of Error Correcting Codes.* North-Holland, 1977.

[15] G. Marsaglia: *The Diehard statistical Tests.* http://stat.fsu.edu/~geo/diehard.html

[16] M. Näslund: *Universal Hash Functions & Hard-Core Bits.* Proceedings, Eurocrypt '95, LNCS 921, pp. 356–366, Springer Verlag.

[17] National Institute of Standards and Technology: *The Advanced Encryption Standard (AES) Homepage.* http://csrc.nist.gov/encryption/aes/aes_home.htm

[18] R. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin: *The RC6 Block Cipher.* Available at [17].

[19] U. V. Vazirani and V. V. Vazirani: *Efficient and Secure Pseudo-Random Number Generation.* Proceedings, 25th IEEE FOCS, 1984, pp. 458–463.

[20] A. C. Yao: *Theory and Applications of Trapdoor Functions.* Proceedings, 23rd IEEE FOCS, 1982, pp. 80–91.