One Way Functions and Pseudorandom Generators[*†]

Leonid A. Levin[‡]

Received 1 September 1985
Revised 31 August 1986

Abstract

Pseudorandom generators transform in polynomial time a short random "seed" into a long "pseudorandom" string. This string cannot be random in the classical sense of [Kolmogorov 65], but testing that requires an unrealistic amount of time (say, exhaustive search for the seed). Such pseudorandom generators were first discovered in [Blum, Micali 82] assuming that the function (a^x mod b) is one-way, i.e., easy to compute, but hard to invert on a noticeable fraction of instances. In [Yao 82] this assumption was generalized to the existence of any one-way permutation. The permutation requirement is sufficient but still very strong. It is unlikely to be proven necessary, unless something crucial, like P=NP, is discovered. Below, among other observations, a weaker assumption about one-way functions is proposed, which is not only sufficient, but also necessary for the existence of pseudorandom generators.

Notations

Let S be the set of finite strings and Ω the set of infinite strings in the alphabet $\{\pm 1\}$; let $\|x\|$ be the length of x and $x \circ y$ be the concatenation of x and y. We identify S (in lexicographical order) with $\mathbb{N} = \{0, 1, ...\}$ and denote the empty string by $\emptyset$.

Let us fix an arbitrary function $r : \mathbb{N} \to \mathbb{N}$ with monotone unbounded $\log_n r(n) < n$ and, for every $\varepsilon > 0$, agree to consider *infeasible* the functions $> r(n^\varepsilon)$ and *negligible* the ones $< 1/r(n^\varepsilon)$ (see 5.6).

Let $\mathrm{U}(x)$ be the Universal Algorithm which divides x into the "program prefix" and the "input" after the first occurrence of a distinguished string. Arbitrary functions below can often be replaced, without loss of generality, by the following modifications of U. Let $U(x) = \mathrm{U}(x)$, if $\|\mathrm{U}(x)\| = \|x\|$; otherwise U diverges. Let $u(x) = U(x)$, if $U(x)$ halts in time $\|x\|^2$ (see 5.1); otherwise $u(x) = x$. Let $\overline{\mathbf{u}}(x) = u(x)$, if $u(x)$ preserves the "program prefix" and $\overline{\mathbf{u}}(x) = x$ otherwise.

1 One-way functions

Let $f, h : S \to S$ be length preserving. A probabilistic algorithm A inverts f on h for an instance x and random $\omega \in \Omega$ if for $y = h(x)$ either $y \notin f(S)$ and $A(\omega, y)$ diverges or $f(A(\omega, y)) = y$. A polynomial time computable f is one-way on h (denoted $\mathrm{OW}(f/h)$) if any A inverting it on h converges in an infeasible time on at least a polynomial fraction of instances of each length (or "on average": see 5.3). So, h just modifies the frequencies of instances.

Let us consider several conjectures about the existence of one-way functions.

(A) $\exists f, h$ $\mathrm{OW}(f/h)$. This is equivalent to $\mathrm{OW}(\overline{\mathbf{u}}/U)$ and, when *infeasible* means non-polynomial, to P$\neq$NP. A generic example is $f(G, H) = (G, 1...1)$ if H is a Hamiltonian cycle of a graph G; otherwise $f(G, H) = 1...1$. And h produces those graphs whose Hamiltonian cycles are the hardest to find (see 5.2).

(B) $\exists f, h$ $\mathrm{OW}(f/h)$ and h also runs in polynomial time. This is equivalent to $\mathrm{OW}(\overline{\mathbf{u}}/u)$. If it is false, the P$\neq$NP question remains open but loses its practical value. The inverse $f^{-1}(h(x))$ always exists, if $h(x)$ has a form $f(h(x))$. An interesting variation is, when h is f or $f^k, k(\|x\|) > 0$. This implies (B) even for non-computable $k(n) \leq n^{O(1)}$, since $h(k \circ x) = k \circ f^k(x)$ can be used. Using $f'(x_1 \circ ... \circ x_i) = f(x_1) \circ ... \circ f(x_i)$, the above variation of (B) can be shown equivalent to B_k, which postulates that the inverting time for $\overline{\mathbf{u}}$ on $\overline{\mathbf{u}}^k$

[*]Supported by NSF grants #DCR-8304498, DCR-8607492

[†]*Combinatorica* 7(4):357–363, 1987. A version of this paper appeared in the *Proceedings of the 17-th Symposium on Theory of Computing,* 1985.

[‡]Boston University, MIT

is *infeasible* on a *constant* fraction of instances. The criterion in section 4 for the existence of pseudorandom generators is B_*, i.e., B_k for all $k(n) < n \log r(n)$ (see 5.6).

(C) $\exists f, h$ $\mathrm{OW}(f/h)$ and h is a permutation. (C) is much stronger than (B). It is equivalent to $\mathrm{OW}(\overline{\mathbf{u}}/i)$ where $i(x) = x$, since h preserves the uniform distribution of instances. (C) means that NP problems (like Tiling: see [Levin 86]) are hard even on generic instances. If $\|f(x)\| = \|x\|$ is not required, then (C) is equivalent to a modification of (B) where the inverting algorithm has access not only to the instance $h(x)$, but to the x itself, explaining the origin of the instance. Just replace f, h by f_h, i, where $f_h(x,z)$ is x if $h(x) = f(z)$ and 1...1 otherwise. The possibility that (C) is false may explain a common experience (so strongly emphasized by D. Hilbert) that most natural mathematical problems are solved within a few centuries, which is a "very polynomial" time.

(D) $\exists f$ $\mathrm{OW}(f/f^k)$ and f is a permutation. These conjectures, equivalent for all k, are proven in [Yao 82] sufficient for the existence of pseudorandom generators.

2 Pseudorandomness

A *test* is a probabilistic algorithm $t : \Omega^2 \times \mathbb{N} \to [-1, 1]$. Positive $t(\omega, \alpha, n)$ indicates a "bet" that the tested function α is random rather than generated as $\alpha(i) = G_s(i)$ by an algorithm G from an n-bit random seed s. And ω is the source of the test's own coin flips. Modifying slightly [Yao 82], we restrict t's running time by $|t(\omega, \alpha, n)|^{-O(1)}$ rather than by $n^{O(1)}$. So, t can run very slow, at the expense of diminishing value of its output. Let $t_G(n)$ be the expected value of $t(\omega, G_s, n) - t(\omega, \alpha, n)$ for uniformly distributed $\alpha, \omega \in \Omega,\ s \in \{\pm 1\}^n$.

Now, G is called a *pseudorandom* generator if (1) $G_s(i)$ is computable within time polynomial in $\|s\| i$, and (2) G_s is "indistinguishable" from random strings, i.e., $|t_G|$ is *negligible*, for any test t. The much more powerful *parallel* generators run in time polynomial in $\|s\| \log i$ and withstand tests which can query entries of α in any order rather than sequentially scan them. Such a goal was mentioned in [Blum, Blum, Shub 83] and reduced in [Goldreich, Goldwasser, Micali 84] to constructing sequential generators.

A *predicting* test first chooses a *focal* string $x = x(\omega)$. Then it queries $\alpha(y)$ for several $y \neq x$, generates a prediction $p \in [-1, 1]$ for $\alpha(x)$ and outputs $\alpha(x)p$. Obviously, the test's expectation, i.e., the covariation of p with $\alpha(x)$ is 0, when α is truly random. The following proposition is a modification of a theorem in [Yao, 82] and its generalization in [Goldreich, Goldwasser, Micali 84].

Proposition 1 *A polynomial time algorithm G is pseudorandom, unless it fails some predicting test.*

Proof. Let t'' be a test with a non-negligible t''_G which can be estimated statistically (see 5.5). We truncate t'' to output 0 when $|t''| < |t''_G/2|$ and normalize it to $t = t''_G(n)^{O(1)} t''$ so that t can be computed in time $|t|^{-1/2}$. Let ω_1 consist of even and ω_2 of odd digits of ω. Let test $t.j(\omega, \alpha, n)$ act like $t(\omega_1, \alpha, n)$, except that it substitutes coin flips $c_i(\omega_2)$ for the answers $\alpha(x_i)$ to the first j different queries x_i, $i \leq j$ ($t.j = 0$ if $< j$ queries are made). Obviously, $h_j = t.j_G < 1/j^2$ and $h_0 = h$. The predicting test t' chooses $j(\omega_2)$ at random with probability $l_j = O(1)/j\|j\|^2$. Then t' runs $t.j$, sets $x(\omega) := x_j$, and produces $p := t.j(\omega, \alpha, n) c_j(\omega_2)/l_j$. Let e_j be the expected value of $t.j \cdot c_j \cdot G_s(x_j)$. Let $\overline{h}_j$ be a modification of h_j in which $-\alpha(x_{j+1})$ is used by $t.j$ instead of $\alpha(x_{j+1})$. Obviously, $h_j = (h_{j-1} + \overline{h}_{j-1})/2$ and $t'_G = \sum_j e_j$. Considering cases $c_j = \pm G(x_j)$, one can see that $e_j = (h_{j-1} - \overline{h}_{j-1})/2 = h_{j-1} - h_j$ and $t'_G = t_G$. ■ (See also 5.4).

3 Multiplicative Isolation Lemma

One of the important ideas of [Yao 82] is that the methods of [Wyner 75] can be applied for computational as well as for purely probabilistic unpredictability. The following is a simple implementation of this idea adjusted to the needs of section 4. Weaker versions (with polynomials instead of $(p_l)^n$) of the Corollary below were given in [Goldwasser 84], [Rackoff 85]. The question, whether an exponential bound can be achieved, was raised by S. Micali.

Let $f : S \to S$; $A : S \to [-1, 1]$; $b : S \to [-1, 1]$; $p : \mathbb{N} \to [0, 1]$; $\varepsilon : \mathbb{N} \to [0, 1]$; $l : \mathbb{N} \to \mathbb{N}$ be all computable in time $T(\|x\|) > \|x\|^2$ and $h = 2|\log \varepsilon|/\varepsilon$. Let the covariation $\mathbf{cov}_x(A(x), b(x))$ denote the average of $A(x)b(x)$ over all x of a given length. Then b is called *(p,T)-isolated* from f if no probabilistic algorithm $A(f(x))$ running in time $\leq T(\|x\|)$ has $\mathbf{cov}_x(A(f(x)), b(x)) \geq p(\|x\|)$.

The range of A can be always narrowed to ± 1 without affecting the covariation. We just replace the real output with the ± 1 result of a biased coin flip with the same expected value.

Lemma 1 *Let $f(x \circ y) = f_0(x) \circ f_1(y)$ and $b(x \circ y) = b_0(x)b_1(y)$, for $\|x\| = l(\|x \circ y\|)$. Then b is (p,T)-isolated from f, unless for some p_0, p_1, T' and both $\theta \in \{0,1\}$, $T' < Th^2 p_\theta$, b_θ is not (p_θ, T')-isolated from f_θ and $p_0 p_1 > p - \varepsilon$, for corresponding lengths.*

Proof. Let an algorithm $A(f(z))$ predict $b(z)$, in time T with covariation p. We employ three algorithms $Q_0(n), L_0(X), R_0(Y)$ dependent on f, b, p, ε, and A. Let $\mathbf{q}_0(X) = \mathbf{cov}_y(b_1(y), A(X \circ f_1(y)))$. Let q be "near maximum" of $\mathbf{q}_0(f_0(x))$ i.e. $\max\{0, \mathbf{q}_0(f_0(x)) - q\}$ averages to $\varepsilon/4$ and $q^\pm$ be $q(1 \pm \varepsilon/4p)$. First Q_0 finds an x with $\mathbf{q}_0(f_0(x)) \geq q^-$, with $o(\varepsilon)$ probability of error.

$Q_0(n)$ starts with $q^- := 0$. For $i = h, h-1, \ldots$ it generates a string x_i of length $l(n)$ at random and computes a lower statistical estimate q_i of $\mathbf{q}_0(f_0(x_i))$ with accuracy i/h and probability of error $o(\varepsilon/h)$ (see 5.5). If $q_i > q^-$, Q sets $q^- := q_i$, $x := x_i$,. It stops when $q^- \varepsilon/8p > i/h$.

Now, R_0 reads the input $Y = f_1(y)$ and outputs $A(f_0(x) \circ Y)$. Obviously, the covariation $p_0 \geq q^- - o(\varepsilon)$ will be achieved. And, $L_0(X) = \mathbf{q}_0(X)/q^+$ (or $\mathrm{sign}(\mathbf{q}_0(X))$, if $|\mathbf{q}_0(X)| > q^+$) achieves the needed covariation $p_1 = (p - \varepsilon/4)/q^+$. A rough statistical average $\overline{\mathbf{q}}_0$ over hp_1^2/ε random samples is used instead of $\mathbf{q}_0$. The sample size does not affect its covariation with b_0, but assures $\overline{\mathbf{q}}_0(X) \leq q^+$ with reliability $1 - o(\varepsilon)$, whenever $\mathbf{q}_0(X) \leq q$.

The above algorithms run in time $Th^2 p_1$. If $p_0 < p_1$, we use Q_1, L_1, R_1, defined correspondingly, instead of Q_0, L_0, R_0. If the problem remains, we use L_0, L_1 with $q = \sqrt{p}$. ■

Corollary 1 *Let $x = x_0 \circ \ldots \circ x_{l(\|x\|)}$; $\|x_i\| = \|f(x_i)\| = \|x_1\|$ except for $\|x_0\| \leq \|x_1\|$; $T_l(k) = T(\lfloor k/l(k) \rfloor)$; $p_l(k) = p(\lfloor k/l(k) \rfloor)$; $f_l(x) = f(x_0) \circ \ldots \circ f(x_{l(\|x\|)})$ and $b_l(x) = \prod b(x_i)$. Then, b_l is $((p_l)^n, h^2 T_l)$-isolated from f_l for any $n(\|x\|) < l(\|x\|)$, if b is (p,T)-isolated from f. Here $h = n|\log p_l|/p_l^n(1-p_l)^{O(1)}$. Thus, isolation is about $(T^{-1/3}, T^{1/3})$ when l is $\log T_l/3|\log p_l|$.*

We just use L to predict b_k from f_k with covariation about $(p_k)^k$ for as small k as possible. Then we use R to predict b from f. This works, since otherwise the above k could be decreased.

4 The criterion

Recall the assumption B_* that inverting $\overline{\mathbf{u}}$ takes an infeasible time on a constant fraction of outputs of $\overline{\mathbf{u}}^k(x)$ for every $0 < k < \|x\| \log r(\|x\|)$.

Theorem 1 *B_* is necessary and sufficient for the existence of pseudorandom generators.*

Necessity. Let G be a pseudorandom generator and let $f(s \circ x) = G_s(1) \circ \ldots \circ G_s(2\|s\|)$, for $\|x\| = \|s\|$. If B_* fails, some probabilistic algorithm A inverts f on $f^k(z)$ for some $k(\|z\|)$ in time $T(\omega, f^k(z))$, and the average of $1/Tk$ over $\omega, z, \|z\| = n$ is not negligible. Let h_j be the average over α, ω of $t.j(\omega, \alpha, n) = 1/T(\omega, f^j(z))$ for $z = \alpha(1) \circ \ldots \circ \alpha(2n)$. Let the test t choose $j(n, \omega)$ at random with probability $l_j = O(1)/j\|j\|^2$ and then run $t.j$. Then $t_G = -h_0 + \sum_j h_j(l_j - l_{j+1})$. But f has an inverse on $\leq 2^{-n}$ of z's. So, $h_0 \leq 2^{-n}$ is negligible while $h_{k(\|z\|)}$ and, thus, t_G are not.

Construction. We combine the techniques from [Blum Micali 82], [Yao 82] and [Goldreich Goldwasser Micali 84]. Let $C(x)$ be an error correcting code (see [Justesen 72]) from which x can be restored in polynomial time even if any 5% of digits of $C(x)$ are altered. Let $b(s)$ be the i-th digit of $C(s')$ where $s = s' \circ i \in S$; $\|i\| = \lceil \log \|C(s)\| \rceil$ and $f(s) = \overline{\mathbf{u}}(s') \circ i$. Any feasible time probabilistic algorithm A guessing $b(f^k(s))$ as, $A(\omega, f^{k+1}(s))$ must be wrong for at least 5% of i-s on a constant fraction of s', ω. Otherwise $\overline{\mathbf{u}}$ could be inverted on $\overline{\mathbf{u}}^{k+1}(x)$ in a feasible time, which contradicts (B_*). So b is $(1-\varepsilon, T)$ -isolated from f for some constant $\varepsilon > 0$ and any feasible $T(n)$.

For $l(n) = \lfloor \log r(\sqrt{n}) \rfloor$ and $k(\|s\|) \leq 2\|s\|$, the Corollary in section 3 implies that $v_k(s) = b_l(f_l^k(s))$ is *independent* from f_l^{k+1}, i.e., (p, T)-*isolated* for any feasible T/p. Let $g_{x \circ y}(s) = g_y(g_x(s)) \in \{\pm 1\}^{\|s\|}$ and $g_{-1}(s) \circ g_1(s) = v_1(s) \circ ... \circ v_{2\|s\|}(s)$. Let $\overline{x}$ be a prefixless code of x, say, every digit of x is doubled and the last digit of the result altered. The parallel pseudo-random generator $G_s(x)$ outputs the last bit of $g_{\overline{x}}(s)$. (See also 5.4)

Unpredictability. Let S_x consist of all prefixes of $\overline{x} \in S$ with the last digits altered and $\overline{x}$. Any $\overline{y}$ extends an element of S_x. For $\alpha : S \to \{\pm 1\}^n$, let $\alpha_x^*(y)$ be the last digit of $g_{y_2}(\alpha(y_1))$, where $y_1 \circ y_2 = \overline{y}$, $y_1 \in S_x$. Clearly, $\alpha_x^* = G_s$ for any x, if for all y $\alpha(y) = g_y(s)$. Let t be a predicting test with non-negligible t_G and $x = x(\omega)$ be its focal string. Let $t^*(\omega, \alpha, n)$ be $t(\omega, \alpha_x^*, n)$ (or 0 if $t < t_G/2$). For a truly random α no correlated predictions are possible and t^* averages to 0, while $t_G^* \geq t_G/2$. Note that t^* queries α only on S_x.

Let us consider a hybrid $\alpha.j$ of g with a truly random α. It copies α except that the k-th bit $\alpha.j_k(y)$ of $\alpha.j(y \circ -1) \circ \alpha.j(y \circ 1)$ is $v_k(\alpha.j(y))$ when $k + 2n\|y\| > j$. Having $f_l^{k+1}(\alpha(y))$ and values of α on all $z \neq y$, one can easily compute $\alpha.j(z)$ for any $z \neq y$ and $j \geq k + 2n\|y\|$. Note that $\alpha.j(y) = \alpha(y)$ for $j > 2n\|y\|$ and $\alpha.0(y) = g_y(\alpha(\emptyset))$. Now we proceed as in section 2: compute $x = x(\omega)$, choose at random its prefix y and $k \leq 2n$ and put $j := k + 2n\|y\|$. Then from $f_l^{k+1}(\alpha(y))$ (and irrelevant values of $\alpha(z)$ for $z \neq y$) we compute a prediction $p := t^*(\omega, \alpha.j, n)\alpha.j_k(y)\|x\|$ for $v_k(\alpha(y))$. Their covariation is $t_G^*/2n$, since each j contributes $(t_{\alpha.j-1}^* - t_{\alpha.j}^*)/2n$. This contradicts the *independence* of v_k from f_l^{k+1}, noted above. ■

5 Notes

5.1 Turing machines

Any simple time bound $> \|x\| \log \|x\|$ in the definition of u works as well. One can even take $O(\|x\|)$ if the Turing machine model is modified slightly. Standard Turing machines create difficulties in accurate complexity considerations. For example, the halting problem with the time bound, say $\|x\|^3$, takes more time than $O(\|x\|^3)$. These problems disappear if Turing machines have unlimited capacity for creating and destroying additional heads in the leftmost cell of the tape. These heads work independently: they can communicate only in neighboring cells and do not collide. The total running time of all heads taken as complexity makes these machines close to standard ones in terms of efficiency.

5.2 Optimal inverting algorithm

For any algorithm f there is an inverse A with optimal (up to a constant factor) running time for $f(A(y))$. It is not known, of course, whether the running time of A is polynomial or exponential compared to the running time of f. It is only observed that there is no speed up for inverting problems [Levin 73/84]. Thus, we can treat the time for inverting f as a definite function.

5.3 Averaging

The above conjectures (A-D) remain equivalent (due to a "padding argument") if OW uses time, infeasible "on average", rather than on a polynomial fraction. The averaging should be done in the following machine independent way. A function $t(x)$ is called *constant on average* for probability distribution μ, if its expectation $\sum t(x)\mu(x) < \infty$. An example is $t(x) = \mathrm{m}(x)/\mu(x)$, where $\sum \mathrm{m}(x) < \infty$. Any $l(x) \leq \|x\|t(x)$ is called *linear on average*. The "average" growth rate of f is then $f_l(n) = \max\{f(x) : l(x) \leq n\}$, in contrast to the worst case growth rate: $\max\{f(x) : \|x\| \leq n\}$. A function f is infeasible on average if f_l is infeasible for all such l. The *norm* is the largest (up to a constant factor) linear on average l with enumerable subgraph $\{(x, q) : q < l(x)\}$. It can be obtained when m is the distribution of outputs of $\mathrm{U}(\alpha)$ on random $\alpha \in \Omega$. The rare instances have large norms and affect little the average growth rate.

Combining this concept with the optimal inverting algorithm we get a uniquely (within a constant factor) defined average inverting time $(\!\!\mathrm{I}f/h)$ of algorithm f on the probability distribution generated by h.

5.4 Improvements

1. The program of t'' in the proof in section 2 need not be known. It can be chosen at random! Moreover, a slightly more complicate construction makes the inputs G, n unnecessary as well.

2. The generator G in section 4 may be sped up in case of $\|x\| >> \|s\|$. Just preprocess x into $x' = (x \bmod s')$ where s' is a new random prime seed independent of s. This transformation is secure in case of truly random functions. Thus, it must be secure in the pseudorandom case as well.

5.5 Statistical Estimates

The following simplification of the Central Limit Theorem, pointed out by A.S. Nemirovsky suffice for our needs: Let $f_{i,j},\ i \le a,\ j \le b^2$ be independent random variables with the same expectation M and standard deviations ≤ 1. The standard deviation of $F_i = \sum_j f_{i,j}/b^2$ is $\le 1/b$ and the probability that $|F_i - M| \ge 2/b$ is $\le 1/4$. So, the probability that most of F_i (and thus their **median** m) are to the right (or left) of $[M \pm 2/b]$, is exponentially small in a.

5.6 Better precision

1. In case $\log\log r(n) = o(\log n)$ all results of the paper remain true if *infeasible* are considered functions $> r^{\varepsilon}(n/\log^{1/\varepsilon} r(n))$. In any case B_* remains equivalent if $n \log r(n)$ is increased to $r(n^{o(1)})$ (the same proof of necessity in section 4 applies).

2. Let f be invertible in a feasible time only on a negligible fraction of instances. Suppose for some coding $y = C_{\varepsilon}(x)$, a list of strings including x can be computed in polynomial (in $\|x\|/\varepsilon$) time from any y' with $1/2 + \varepsilon$ fraction of digits common with y. Then for a negligible $\varepsilon(\|x\|)$, b' will be *independent* from f. So, one can take $l = 1$ and MIL from section 3 need not be used at all; $n \log r(n)$ can be replaced by $O(n)$ in the definition of B_* and a more accurate threshold of $r^{\varepsilon}(n)$ can be used instead of $r(n^{\varepsilon})$ in the definition of infeasibility. P. Elias has shown that Random Linear Codes have all needed properties, if they are decodable in polynomial time. In this case $b(s,i)$, equal to the inner product of s, i as vectors in Z_2^n, is independent from $f'(s,i) = (f(s), i)$ and $b(x, f^k(y))$ is a pseudorandom generator.

Acknowledgments. I am most grateful to P. Gacs, O. Goldreich, S. Goldwasser, P. Elias and S. Micali for a great help.

References

[1] L. Blum, M. Blum, M. Shub. A Simple Secure Pseudo-Random Number Generator. *Advances in Cryptology,* ed. D. Chaum, R.L. Rivest and A.T. Sherman, Plenum Press, 1983, pp 61-78.

[2] M. Blum, S. Micali. How to generate Cryptographically Strong Sequences of Pseudo Random Bits. *FOCS*-1982; *SIAM J. Comput.* 13:850-864, 1984.

[3] O. Goldreich, S. Goldwasser, S. Micali. How to Construct Random Functions. *Proc. 25th Symp. on Foundations of Computer Science,* 1984; *J. ACM*, 33(4):792-807, 1986.

[4] S.Goldwasser. *Probabilistic Encryption: Theory and Applications*, Ph.D. Dissert., University of California at Berkeley (1984), Section 4.2.3.

[5] J.Justesen. A class of constructive, asymptotically-good, algebraic codes. *IEEE Trans. Inform. Theory*, **IT-18**, 5, (1972), 652-656.

[6] A.N. Kolmogorov. Three Approaches to the Concept of the Amount of Information. *Probl. Inf. Transm.,* 1(1), 1965.

[7] L. Levin. Average Case Complete Problems. *SIAM J. Comput.* 1:285-286, 1986.

[8] L. Levin. Randomness Conservation Inequalities. *Information and Control* 61, 1984, section 1.3; In less detail in Theorem 2 of: Universal Sequential Search Problems, *Probl. Inf. Transm.* 9(3), 1973.

[9] C. Rackoff. Personal communication. 1985.

[10] A. Shamir. On the Generation of Cryptographically Strong Pseudo-Random Sequences. *ACM Trans. on Comp. Syst.* 1:38-44, 1983.

[11] A. D. Wyner. The wire-tap channel. *Bell System Technical Journal* 54:1355-1387, 1975.

[12] A. C. Yao. Theory and Applications of Trapdoor Functions. *Proc. 23rd IEEE Symp. on Foundations of Computer Science* pp. 80-91, 1982.