Electronic Colloquium on Computational Complexity, Report No. 90 (2000)

Candidate One-Way Functions Based on Expander Graphs

Oded Goldreich[*]
Department of Computer Science
Weizmann Institute of Science
Rehovot, ISRAEL.
oded@wisdom.weizmann.ac.il

December 3, 2000

Abstract

We suggest a candidate one-way function using combinatorial constructs such as expander graphs. These graphs are used to determine a sequence of *small* overlapping subsets of input bits, to which a hard-wired random predicate is applied. Thus, the function is extremely easy to evaluate: all that is needed is to take multiple projections of the input bits, and to use these as entries to a look-up table. It is feasible for the adversary to scan the look-up table, but we believe it would be infeasible to find an input that fits a given sequence of values obtained for these *overlapping* projections.

The conjectured difficulty of inverting the suggested function does not seem to follow from any well-known assumption. Instead, we propose the study of the complexity of inverting this function as an interesting open problem, with the hope that further research will provide evidence to our belief that the inversion task is intractable.

[*]Supported by MINERVA Foundation, Germany.

ISSN 1433-8092

1 Introduction

In contrary to the present attempts to suggest a practical private-key encryption scheme in place of the DES, we believe that attempts should focus on suggesting practical one-way functions and pseudorandom functions. Being a simpler object, one-way functions should be easier to construct, and such constructions may later yield directly or indirectly a variety of other applications (including private-key encryption schemes).

The current attempts to suggest a practical private-key encryption scheme in place of the DES seem quite ad-hoc: not only that they cannot be reduced to any well-known problem, but (typically) they do not relate to a computational problem of natural appeal. Thus, the study of these suggestions is of limited appeal (especially from a conceptual point of view).

In this manuscript, we propose a general scheme for constructing one-way functions. We do not believe that the complexity of inverting the resulting function follows from some well-known intractability assumptions. We believe that the complexity of inverting this function is a new interesting open problem, and hope that other researcher will be able to obtain better understanding of this problem.

In addition to the abstract presentation, we propose several concrete instantiations of our proposal. It seems to us that a reasonable level of "security" (i.e., hardness to invert) may be achieved at very modest input lengths. Specifically, on input length at the order of a couple of hundreds of bits, inverting the function may require complexity (e.g., time) beyond 2^{100}.

Style and Organization: This write-up is intended to two different types of readers: researchers in the the area of computational complexity as well as researchers interested in the practice of cryptography. Consequently, we provide an asymptotic presentation coupled with suggestions for concrete parameters. The basic suggestion is presented in Sections 2 and 3. Concrete instantiations of this suggestion are proposed in Section 4. Concluding comments appear in Section 5.

2 The Basic Suggestion

We construct a (uniform) collection of functions $\{f_n : \{0,1\}^n \to \{0,1\}^n\}_{n \in \mathbb{N}}$. Our construction utilizes a collection of $\ell(n)$-subsets, $S_1, ..., S_n \subset [n] \stackrel{\text{def}}{=} \{1, ..., n\}$, and a predicate $P : \{0,1\}^{\ell(n)} \to \{0,1\}$. Jumping ahead, we hint that:

1. The function ℓ is relatively small: Theoretically speaking, $\ell = O(\log n)$ or even $\ell = O(1)$. In practice ℓ should be in the range $\{7, ..., 16\}$, whereas n should range between a couple of hundreds and a couple of thousands.

2. We prefer to have $P : \{0,1\}^\ell \to \{0,1\}$ be a random predicate. That is, it will be randomly selected, fixed, and "hard-wired" into the function. For sure, P should *not* be linear, nor depend on few of its bit locations.

3. The collection $S_1, ..., S_n$ should be *expanding*: specifically, for *some* k, every k subsets should cover at least $k + \Omega(n)$ elements of $\{1, ..., n\}$. The complexity of the inversion problem (for f_n constructed based on such a collection) seems to be exponential in the "net expansion" of the collection (i.e., the cardinality of the union minus the number of subsets).

For $x = x_1 \cdots x_n \in \{0,1\}^n$ and $S \subset [n]$, where $S = \{i_1, i_2, ..., i_t\}$ and $i_j < i_{j+1}$, we denote by x_S the projection of x on S; that is, $x_S = x_{i_1} x_{i_2} \cdots x_{i_t}$. Fixing P and $S_1, ..., S_n$ as above, we define

$$f_n(x) \stackrel{\text{def}}{=} P(x_{S_1}) P(x_{S_2}) \cdots P(x_{S_n}) \tag{1}$$

Note that we think of ℓ as being relatively small (i.e., $\ell = O(\log n)$), and aim at having f_n be univertible within time $2^{n/O(1)}$. Thus, the hardness of inverting f_n cannot be due to the hardness of inverting P. Instead, the hardness of inverting f_n is supposed to come from the combinatorial properties of the collection of sets $C = \{S_1, ..., S_n\}$ (as well as from the combinatorial properties of predicate P).

2.1 The preferred implementation

Our preference is to have P be a fixed randomly chosen predicate, which is hard-wired into the algorithm for evaluating f_n. Actually, one better avoid some choices; see next section. (In case $\ell = \Theta(\log n)$ bad choices are rare enough.) In practice, we think of ℓ in the range $\{7, ..., 16\}$, and so hard-wiring a (random) predicate defined on $\{0,1\}^\ell$ is quite feasible. The ℓ-subsets will be determined by combinatorial constructions called expander graphs. At this point the reader may think of them too as being hard-wired into the algorithm. On input $x \in \{0,1\}^n$, the algorithm for computing f_n proceeds as follows:

1. For $i = 1, .., n$, projects x on S_i, forming the ℓ-bit long string $x^{(i)}$.
2. For $i = 1, .., n$, by accessing a look-up table for P, determines the bit $y_i = P(x^{(i)})$.

The output is the n-bit long string $y_1 y_2 \cdots y_n$.

(Note that the n actions, in each of the above two steps, can be performed in parallel.)

2.2 An alternative implementation

An alternative to having P "hard-wired" to the algorithm (as above) is to have it appear as part of the input (and output). That is, letting $\langle P \rangle$ denote the 2^ℓ-bit string that fully specifies P, we have

$$f'_n(\langle P \rangle, x) \stackrel{\text{def}}{=} (\langle P \rangle, P(x_{S_1}) P(x_{S_2}) \cdots P(x_{S_n})) \tag{2}$$

Thus, P is essentially random since the inversion problem is considered with respect to a uniformly chosen input. This implementation is more appealing from a theoretical point of view, and in such a case one better let $\ell = \log_2 n$ (rather than $\ell = O(1)$).

2.3 Determining suitable collections

As hinted above, the collection of ℓ-subsets, $C = \{S_1, ..., S_n\}$, is to be determined by a suitable combinatorial construction known as expander graphs. The reason for this choice will become more clear from the analysis of one obvious attack (presented in Section 3.2). The specific correspondence (between expanders and subsets) depends on whether one uses the bipartite or non-bipartite formulation of expander graphs:

Bipartite formulation: In this case one considers a bipartite graph $B = ((U, V), E)$, where (U, V) is a partition of the vertex set, with $|U| = |V|$, and $E \subset U \times V$ is typically sparse. The expanding property of the graph provides, for every $U' \subset U$ (of size at most $|U|/2$), a lower bound on $|\Gamma(U')| - |U'|$ (in terms of $|U'|$), where $\Gamma(U') = \{v : \exists u \in U' \text{ s.t. } (u, v) \in E\}$.

Our collection of subsets will be defined as $C = \{S_u\}_{u \in U}$, where $S_u = \{v : (u, v) \in E\}$.

Non-bipartite formulation: In this case one considers a graph $G = (V, E)$, so that for every $V' \subset V$ (of size at most $|V|/2$), a suitable lower bound on $|\Gamma(V') \setminus V'|$ holds, where $\Gamma(V') = \{v : \exists v' \in V' \text{ s.t. } (v', v) \in E\}$.

Our collection of subsets is defined as $C = \{S_v\}_{v \in V}$, where $S_v = \{w : (v, w) \in E\} \cup \{v\}$.

In both cases, the lower bound provided by the expansion property on the size of the neighbor set is linear in the size of the vertex set; e.g., for the non-bipartite formulation it holds that $|\Gamma(V') \setminus V'| \geq c \cdot |V'|$ for some constant $c > 0$ and every admissible V'.

3 Avoiding obvious weaknesses

Considering a few obvious attacks, we rule out some obviously bad choices of the predicate P and the collection C.

3.1 The choice of the predicate

We start by discussing two bad choices (for the predicate P), which should be avoided.

Linear predicates. It is certainly bad to use a linear predicate P (i.e., $P(\sigma_1 \cdots \sigma_\ell) = p_0 + \sum_{i=1}^{\ell} p_i \sigma_i$, for some $p_0, p_1, ..., p_\ell$). Under a linear P, the question of inverting f_n, regardless of what collection of subsets C is used, boils down to solving a linear system (of n equations in n variables), which is easy. Having a predicate P that is close to a linear predicate is dangerous too.

Horn predicates. Likewise, one should avoid having any predicate that will make the system of equations (or conditions) solvable in polynomial-time. The only other type of easily solvable equations are these arising from Horn formulae (e.g., an OR of all variables).

Degenerate predicates. The rest of our analysis refers to the collection of sets that determine the inputs to which the predicate P is applied. For this analysis to be meaningful, the predicate should actually depend on all bits in its input (i.e., be non-degenerated).

Good predicates. We believe that most predicates are good for our purpose. In particular, we suggest to use a uniformly chosen predicate.

3.2 The choice of the collection

Since the inverting algorithm can afford to consider all preimages of the predicate P, it is important that the inversion of f_n cannot be performed by interactively inverting P. To demonstrate this point, consider the case $\ell = 1$ and the collection $\{S_1, ..., S_n\}$ such that $S_i = \{i\}$. In this case the S_i's are disjoint and we can recover the preimage by inverting P on each of the bits of the image, separately from the others. For a less trivial example, consider the case where the collection C consists of $n/2\ell$ sub-collections, each having 2ℓ subsets of some distinct set of 2ℓ elements. In this case, inversion can be performed in time $O(n \cdot 2^{2\ell})$ by considering each of these disjoint sets (of 2ℓ elements) separately. Recall that we wish the complexity of inversion to be exponential in n (and not in ℓ, which may be a constant).

In general, a natural inverting algorithm that should be defeated is the following: On input $y = f_n(x)$, the algorithm proceeds in n steps, maintaining a list of *partially specified* preimages of

y under f_n. Initially, the list consists of the unique fully-undetermined string $*^n$. In the first step, depending on the first bit of $y = y_1 \cdots y_n$, we form the list L_1 of strings over $\{*, 0, 1\}$ so that for every $z \in L_1$ it holds that $P(z_{S_1}) = y_1$ and $z_{[n]\setminus S_1} = *^{n-\ell}$, where $[m] \stackrel{\text{def}}{=} \{1, ..., m\}$. In the $i+1^{\text{st}}$ step, we extend L_i to L_{i+1} in the natural manner:

- Let $U' = \cup_{j=1}^{i} S_j$ and $U = \cup_{j=1}^{i+1} S_j$.
- For every $z' \in L_i$, we consider all $2^{|U \setminus U'|}$ strings $z \in \{*, 0, 1\}^n$ satisfying
  1. $z_{U'} = z'_{U'}$,
  2. $z_{U\setminus U'} \in \{0,1\}^{|U\setminus U'|}$, and
  3. $z_{[n]\setminus U} = *^{n-|U|}$.

  The string z is added to L_{i+1} if and only if $P(z_{S_{i+1}}) = y_{i+1}$.

Thus, for every i,

$$L_i = \left\{ z \in \{*,0,1\}^n : \begin{array}{l} z_k = * \text{ if and only if } k \in [n] \setminus \cup_{j=1}^{i} S_j \\ \text{and} \\ P(z_{S_j}) = y_j \text{ for } j = 1, ..., i \end{array} \right\}.$$

The average running-time of this algorithm is determined by the expected size of the list at step i, for the worst possible i. Letting $U = \cup_{j=1}^{i} S_j$,

$$A_{\sigma_1 \cdots \sigma_i} \stackrel{\text{def}}{=} \left\{ z \in \{*,0,1\}^n : \begin{array}{l} z_k = * \text{ if and only if } k \in [n] \setminus U \\ \text{and} \\ P(z_{S_j}) = \sigma_j \text{ for } j = 1, ..., i \end{array} \right\},$$

and X be uniformly distributed over $\{0,1\}^n$, the expected size of L_i equals

$$\begin{aligned} \sum_{\alpha \in \{0,1\}^i} \mathbf{Pr}[f(X)_{[i]} = \alpha] \cdot |A_\alpha| &= \sum_{\alpha \in \{0,1\}^i} \mathbf{Pr}[\exists z \in A_\alpha \text{ s.t. } X_U = z_U] \cdot |A_\alpha| \\ &= \sum_{\alpha \in \{0,1\}^i} \frac{|A_\alpha|}{2^{|U|}} \cdot |A_\alpha| \;=\; 2^{-|U|} \cdot \sum_{\alpha \in \{0,1\}^i} |A_\alpha|^2 \\ &\geq 2^{-|U|} \cdot \frac{\left(2^{|U|}\right)^2}{2^i} \;=\; 2^{|U|-i} \end{aligned}$$

where the inequality is due to the fact that the minimum value of $\sum_i z_i^2$, taken over M $(= 2^i)$ non-negative z_i's summing to N $(= 2^{|U|})$, is obtained when the z_i's are equal, and the value itself is $M \cdot (N/M)^2 = N^2/M$.

Note that the algorithm needs not proceed by the above given order of sets. In general, for every 1-1 permutation π over $[n]$, we may proceed by considering in the ith step the set $S_{\pi(i)}$. Still, the complexity of this (generalized) algorithm is at least exponential in

$$\min_{\pi} \left\{ \max_{i} \left\{ \left| \cup_{j=1}^{i} S_{\pi(j)} \right| - i \right\} \right\} \tag{3}$$

We should thus use a collection such that Eq. (3) is big (i.e., bounded below by $\Omega(n)$).

Bad collections. It is a bad idea to have $S_j = \{j+1, ..., j+\ell\}$, since in this case we have $|\cup_{j=1}^{i} S_j| - i \leq \ell - 1$ for every i. It also follows that we cannot use $\ell \leq 2$, since in this case one can always find an order π so that Eq. (3) is bounded above by $\ell - 1$.

Good collections. An obvious lower bound on Eq. (3) is obtained by the expansion property of the collection $C = \{S_j\}$, where the expansion of C is defined as

$$\max_k \min_{I:\,|I|=k} \{|\cup_{j\in I} S_j| - k|\} \tag{4}$$

A natural suggestion is to determine the collection C according to the neighborhood sets of an *expander graph*. Loosely speaking, known constructions of expander graphs allow to let ℓ be a small constant (in the range $\{7, ..., 16\}$), while guaranteeing that Eq. (4) is a constant fraction of n.

4 Concrete parameters for practical use

If we go for random predicates, then we should keep ℓ relatively small (say, $\ell \leq 16$), since our implementation of the function must contain a 2^ℓ-size table look-up for P. (Indeed, $\ell = 8$ poses no difficulty, and $\ell = 16$ requires a table of 64K bits which seems reasonable.) For concrete security we will be satisfied with time complexities such as 2^{80} or so. Our aim is to have n as small as possible (e.g., a couple of hundreds).

The issue addressed below is which expander to use. It is somewhat "disappointing" that for some specific parameters we aim for, we cannot use the "best" known explicit constructions.

Below we use the bipartite formulation of expanders. By *expansion* we mean a lower bound established on the quantity in Eq. (4). Recall that the time complexity is exponential in this bound.

Random construction. This yields the best results, but the "cost" is that with small probability we may select a bad construction. (The fact that we need to hard-wire the construction into the function description is of little practical concern, since we are merely talking of hard-wiring $n \cdot \ell \cdot \log_2 n$ bits, which for the biggest n and ℓ considered below merely means hard-wiring 20K bits.) Alternatively, one may incorporate the specification of the construction in the input of the one-way function, at the cost of augmenting the input by $n \cdot \ell \cdot \log_2 n$ (where the original input is n-bit long). Specific values that may be used are tabulated below.[1]

degree (i.e., ℓ)	#vertices (i.e., n)	expansion	error prob.
10	256	77	2^{-81}
12	256	90	2^{-101}
14	256	103	2^{-104}
16	256	105	2^{-152}
8	384	93	2^{-83}
10	384	116	2^{-121}
12	384	139	2^{-141}
8	512	130	2^{-101}
10	512	159	2^{-151}
12	512	180	2^{-202}

[1]The expansion was computed in a straightforward manner; the key component is to provide for any fixed k and h an upper bound on the probability that a specific set of k vertices has less than h neighbors.

The last column (i.e., *error prob.*) states the probability that a random construction (with given n and ℓ) does not achieve the stated expansion. Actually, we only provide upper bounds on these probabilities.

Alon's Geometric Expanders [2]. These constructions do not allow $\ell = O(\log n)$, but rather ℓ is polynomially related to n. Still for our small numbers we get meaningful results, when using $\ell = q+1$ and $n = q^2+q+1$, where q is a prime power. Specific values that may be used are tabulated below.[2]

degree (i.e., ℓ)	#vertices (i.e., n)	expansion	comment
10	91	49	expansion too low
12	133	76	quite good
14	183	109	very good

Note that these are all the suitable values for Alon's construction (with $\ell \leq 16$); in particular, ℓ uniquely determines n and $\ell - 1$ must be a prime power.

The Ramanujan Expanders of Lubotzky, Phillips, and Sarnak [6]. Until one plays with the parameters governing this construction, one may not realize how annoying these may be with respect to an actual use: The difficulty is that there are severe restrictions regarding the degree and the number of vertices,[3] making $n \approx 2000$ the smallest suitable choice. Admissible values are tabulated below.[4]

Parameters			Results		
p	q	bipartite?	ℓ	n	expansion (+ comment)
13	5	NO	15	120	20 (unacceptable)
5	13	NO	7	2184	160 (better than needed)
13	17	YES	14	2448	392 (better than needed)

Note that $p = 5$ and $p = 13$ are the only admissible choices for $\ell \leq 16$. Larger values of q may be used, but this will only yield larger value of n.

Using the simple expander of Gaber–Galil [4]. Another nasty surprise is that the easy to handle expander of Gaber–Galil performs very poorly on our range of parameters. This expander has degree 7 (i.e., $\ell = 7$), and can be constructed for any $n = m^2$, where m is an integer. But its expansion is $(c/2) \cdot n$, where $c = 1 - \sqrt{3/4} \approx 0.1339746$, and so to achieve expansion above 80 we need to use $n = 1225$. See table below:

[2]The expansion is computed from the eigenvalues, as in [3]. Actually, we use the stronger bound provided by [2, Thm. 2.3] rather than the simpler (and better known) bound. Specifically, the lower bounds in [2, Thm. 2.3] are on the size of the neighborhood of x-subsets, and so we should subtract x from them, and maximize over all possible x's. (We use the stronger lower bound of $n - \frac{(n-x)(\ell n+1)}{\ell n+1+(n-\ell-2)x}$, rather than the simpler bound of $n - \frac{n^{3/2}}{x}$, both provided in [2, Thm. 2.3].)

[3]Specifically, $\ell = p+1$ and $n = (q^3-q)/2$, where p and q are different primes, both congruent to 1 mod 4, and p is a square mod q. For the non-bipartite case, p is a non-square mod q, and $n = q^3 - q$. Recall that for non-bipartite graphs ℓ equals the degree plus 1 (rather than the degree).

[4]Again, the expansion is computed from the eigenvalues, as in [3].

degree (i.e., ℓ)	#vertices (i.e., n)	expansion	comment
7	400	27	expansion way too low
7	1225	83	good
7	1600	108	very good
7	2500	168	beyond our requirements

A second thought. In some applications having n on the magnitude of a couple of thousands may be acceptable. In such a case, the explicit constructions of Lubotzky, Phillips, and Sarnak [6] and of Gaber and Galil [4] become relevant. In view of the lower degree and greater flexibility, we would prefer the construction of Gaber–Galil.

5 Concluding remarks

5.1 Variations

One variation is to use either a specific predicate or predicates selected at random from a small domain, rather than using a truly random predicate (as in the presentation above). The advantage of these suggestions is that the description of the predicate is shorter, and so one may use larger values of ℓ. Two specific suggestions follow:

1. Use the predicate that partitions its input into two equal length strings and takes their inner product modulo 2. That is, $P(z_1, ..., z_{2t}) = \sum_{i=1}^{t} z_i z_{t+i} \bmod 2$.

   In this case, the predicate is described without reference to ℓ, and so any value of ℓ can be used (in practice). This suggestion is due to Adi Shamir.

2. Use a random low-degree ℓ-variant polynomial as a predicate. Specifically, we think of a random ℓ-variant polynomial of degree $d \in \{2,3\}$ over the finite field of two elements, and such a polynomial can be described by $\binom{\ell}{d}$ bits.

   In practice, even for $d = 3$, we may use $\ell = 32$ (since the description length in this case is less than 6K bits).

On the other extreme, for sake of simplifying the analysis, one may use different predicates in each application (rather than using the same predicate in all applications).

5.2 Directions for investigation

1. *The combinatorial properties of the function f_n.* Here we refer to issues such as under what conditions is f_n 1-to-1 or merely "looses little information"; that is, how is $f_n(X_n)$ distributed, when X_n is uniformly selected in $\{0,1\}^n$. One can show that if the collection $(S_1, ..., S_n)$ is sufficiently expending (as defined above) then the former distribution has min-entropy $\Omega(n)$; i.e., $\mathbf{Pr}[f_n(X_n) = \alpha] < 2^{-\Omega(n)}$, for every $\alpha \in \{0,1\}^n$. We seek min-entropy bounds of the form $n - O(\log n)$.

2. *What happens when f_n is iterated?* Assuming that f_n "looses little information", iterating it may make the inverting task even harder, as well as serves as a starting point for the next item.[5]

[5]An additional motivation for iterating f_n is to increase the dependence of each output bit on the input bits. A dependency of each output bit on all output bits is considered by some researchers to be a requirement from a one-way function; we beg to differ.

3. Modifying the construction to obtained a "keyed"-function with the hope that the result is a pseudorandom function (cf. [5]). The idea is to let the key specify the (random) predicate P. We stress that this modification is applied to the iterated function, not to the basic one.[6] We suggest using $\Theta(\log n)$ iteration; in practice 3–5 iterations should suffice.

Our construction is similar to a construction that was developed by Alekhnovich et. al. [1] in the context of proof complexity. Their results may be applicable to show that certain search method that are related to resolution will require exponential-time to invert our function [Avi Wigderson, private communication]. This direction requires further investogation.

5.3 Inspiration

Our construction was inspired by the construction of Nisan and Wigderson [7]; however, we deviate from the latter in two important aspects:

1. Nisan and Wigderson reduce the security of their construction to the hardness of the predicate in use. In our construction, the predicate is not complex at all (and our hope that the function is hard to invert can not arise from the complexity of the predicate). That is, we hope that the function is harder to invert than the predicate is to compute.[7]

2. The set system used by Nisan and Wigderson has different combinatorial properties than the systems used by us. Specifically, Nisan and Wigderson ask for small intersections of each pair of sets, whereas we seek expansion properties (of a type that cannot be satisfied by pairs of sets).

Our construction is also reminiscent of a sub-structure of of the DES; that is, the mapping from 32-bit long strings to 32-bit long strings induced by the eight S-boxes. However, the connection within input bits and output bits is far more complex in our case. Specifically, in the DES, each of the 8 (4-bit) output strings is a function (computed by an S-box) of 6 (out of the 32) input bits. The corresponding 8 subsets have a very simple structure; the i^{th} subset holds bit locations $\{4(i-1)+j : j = 0,...,5\}$, where $i = 1,...,8$ and 32 is identified with 0. Indeed, inverting the mapping induced on 32-bit strings is very easy.[8] In contrast, the complex relation between the input bits corresponding to certain output bits in our case, defeat such a simple inversion attack. We stress that this complex (or rather expanding) property of the sets of input bits is the heart of our suggestion.

Acknowledgments

We are grateful to Noga Alon, Adi Shamir, Luca Trevisan and Avi Wigderson for useful discussions.

References

[1] M. Alekhnovich, E. Ben-Sasson, A. Razborov, and A. Wigderson. Pseudorandom Generators in Propositional Proof Complexity. In *41st FOCS*, pages 43–53, 2000.

[6] We note that applying this idea to the original function will definitely fail. In that case, by using 2^ℓ queries (and inspecting only one bit of the answers) we can easily retrieve the key P.

[7] We comment that it is not clear whether the Nisan and Wigderson construction can be broken within time comparable to that of computing the predicate; their paper only shows that it cannot be broken substantially faster.

[8] In an asymptotic generalization of the scheme, inversion takes time linear in the number of bits.

[2] N. Alon. Eigenvalues, Geometric Expanders, Sorting in Rounds, and Ramsey Theory. *Combinatorica*, Vol. 6, pages 207–219, 1986.

[3] N. Alon and V.D. Milman. λ_1, Isoperimetric Inequalities for Graphs and Superconcentrators, *J. Combinatorial Theory, Ser. B*, Vol. 38, pages 73–88, 1985.

[4] O. Gaber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *JCSS*, Vol. 22, pages 407–420, 1981.

[5] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *JACM*, Vol. 33, No. 4, pages 792–807, 1986.

[6] A. Lubotzky, R. Phillips, P. Sarnak. Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[7] N. Nisan and A. Wigderson. Hardness vs Randomness. *JCSS*, Vol. 49, No. 2, pages 149–167, 1994.