ON THE CRYPTOGRAPHIC APPLICATIONS OF RANDOM FUNCTIONS

(EXTENDED ABSTRACT)

Oded Goldreich, Shafi Goldwasser, Silvio Micali

Laboratory for Computer Science
M.I.T.
Cambridge, MA 02139

ABSTRACT

Now that "random functions" can be efficiently constructed([GGM]), we discuss some of their possible applications to cryptography:

1) Distributing unforgable ID numbers which can be locally verified by stations which contain only a small amount of storage.

2) Dynamic Hashing: even if the adversary can change the key-distribution depending on the values the hashing function has assigned to the previous keys, still he can not force collisions.

3) Constructing deterministic, memoryless authentication schemes which are provably secure against chosen message attack.

4) Construction Identity Friend or Foe systems.

The first author was supported in part by a Weizmann Postdoctoral fellowship. The second author was supported in part by the International Business Machines Corporation under the IBM/MIT Joint Research Program, Faculty Development Award agreement dated August 9, 1983.

G.R. Blakley and D. Chaum (Eds.): Advances in Cryptology - CRYPTO '84, LNCS 196, pp. 276-288, 1985.

1. INTRODUCTION

In our paper "How to Construct Random Functions"([GGM]), we have

1) Introduced an algorithmic measure of the randomness of a function. (Loosely speaking, a function is random if any polynomial time algorithm, which asks for the values of the function at various points, cannot distinguish the case in which it receives the true values of the function, from the case in which it receives the outcome of independent coin flips.)

2) Constructed functions that are easy to evaluate and, nevertheless, achieve maximum algorithmic randomness, under the assumption that there exist one-way permutations.

In this paper, we describe in details 4 cryptographic applications of these "pseudo-random functions":Storageless ID Number Distribution, Dynamic Hashing, Deterministic Private-key Signature Scheme and Identify Friend or Foe. Before describing these applications, let us recall some of the definitions which appeared in [GGM].

1.1 Poly-Random Collections

Let I_k denote the set of all k-bit strings. Consider the set, H_k, of all functions from I_k into I_k. Note that the cardinality of H_k is 2^{k2^k}. Thus to specify a function in H_k we would need $k2^k$ bits: an impractical task even for a moderately large k. Even more, assume that one randomly selects subsets $H_k^\# \subseteq H_k$ of cardinality 2^k so that each function in $H_k^\#$ has a unique k-bit index; then there is no polynomial time Turing Machine that, given k, the index of a function $f \varepsilon H_k^\#$ and $\chi \varepsilon I_k$, will evaluate $f(\chi)$.

Our goal is to make "random functions" accessible for applications. I.e. to construct functions that can be easily specified and evaluated and yet cannot be distinguished from functions chosen at random in H_k. Thus we restrict ourselves to choose functions from a subset $F_k \subseteq H_k$ where the collection $F = \{F_k\}$ has the following properties:

1) Indexing: Each function in F_k has a unique k-bit index associated with it. (Thus picking randomly a function $f \varepsilon F_k$ is easy.)

2) Poly-time Evaluation: There exists a polynomial time Turing machine that given an index of a function $f \varepsilon F_k$ and an input χ, computes $f(\chi)$.

3) Pseudo-Randomness: No probabilistic algorithm that runs in time polynomial in k can distinguish the functions in F_k from the functions in H_k.

More precisely: if the collection F passes all polynomial time statistical tests for functions, where the notions "statistical test for functions" and "passes a test" are hereby defined.

A polynomial time statistical test for function is a probabilistic polynomial time algorithm T that, given an input k and access to an oracle O_f for a function $f:I_k \rightarrow I_k$, outputs either 0 or 1. Algorithm T can query the oracle O_f only by writing on a special query-tape some $y \varepsilon I_k$ and will read the oracle answer, f(y), on a separate answer-tape. As usual, O_f prints its answer in one step.

Let $F=\{F_k\}$ be a collection of functions. We say that F passes the test T if for any polynomial Q, for all sufficiently large k:

$$\left[p_k^F - p_k^R \right| < \frac{1}{Q(k)}$$

where p_k^F denotes the probability that T outputs 1 on input k and access to an oracle for a function f randomly chosen in F_k. p_k^R is the probability that T outputs 1 when given the input parameter k and access to an oracle O_f for a function f randomly picked in H_k(i.e. a random function).

Such a collection of functions F will be called a poly-random collection. Loosely speaking, despite the fact that the functions in F are easy to select and easy to evaluate, they will exhibit, to an examiner with polynomially bounded resources, all the properties of randomly selected functions.

The above definition is highly constructive. In[GGM] it was shown how to transform any one-to-one one-way function to an algorithm A_F for a poly-random collection of functions F. The construction is in two steps: first, using Yao's construction (see Appendix A) to transform a one-to-one one-way function into a Cryptographically Strong Pseudo Random Bit generator (CSPRB-generator); next, using ANY CSPRB-generator to construct a poly-random collection (see Appendix B). However, for practical purposes we will consider only poly-random collections whose underlying CSPRB generator is fast.

Efficiency considerations

In the recent years many CSPRB generators have been proposed ([BBS],[BM],[GMT],[Y]), based on various intractability assumptions and demonstrating various degrees of practicality.

Using the new results of Chor and Goldreich[CG] it is now possible to construct fast CSPRB generators which are "equivalent" to factoring: On input a k-bit long seed, these generators output log k bits at the price of one modular multiplication of two k-bit long integers. Factoring k-bit long integers is poly(k) reducable to distinguishing the sequence generated by these generators from truely random sequences.

Let T denote the average time needed for generating one bit in the underlying generator used in our construction of a poly-random collection. Then, evaluating a function chosen at random from F_k can be done in time $O(k^2T)$.

1.2 Comparison with CSPRB generators

The fundamental definitions and properties of Cryptographically Strong Pseudo-Random Bit(CSPRB) generators are given in Appendix A.

It is a theoretical challenge and an extremely useful task to find the most general properties of randomness that can be achieved by efficient pseudo-random programs.

Let us consider the effect of such programs on probabilistic computation.

CSPRB generators cut down the number of coin tosses

Performing a probabilistic polynomial-time computation that requires k^t random bits is trivial if we are willing to flip k^t coins. A very interesting feature of CSPRB generators is that they guarantee the same result of the computation by flipping only k coins!

Poly Random Collection cut down the storage as well

The existence of poly-random collections allows to successfully replace a random oracle(function), in any polynomial time computation, by k random bits.

It should be noticed that computing with a random oracle is different from computing with a coin. In fact, the bit associated with each string χ, not only is random, but does not change in time and is stored for free! The advantages of the random oracle model are clarified by all the applications discussed in the following sections.

Again, it is trivial(see Appendix C) to simulate a computation with a random oracle (function) that is queried on k^t inputs if one is willing to use k^t bits of storage. A very interesting feature of poly-random collections is that they guarantee the same result of the computation by using only k bits of storage!

Sharing Randomness in a distributed environment

An additional advantage of our solution is that it enables many parties to efficiently share such an f in a distributed environment. By sharing f we mean that if f is evaluated at different times by different parties on the same input χ, the same value $f(\chi)$ will be obtained. Such sharing is efficient as it can be achieved by an initial step which consists of (1) One party flipps k coins; and (2) All parties record the result. After this initial step no more coin flips or message exchanges are needed. The k bits stored by all determine a shared function of the poly-random collection.

Assume that in "situation" S, some party (processor) p_j wants to make a random choice so that the other processors will know it. Then it will simply compute f(j,S). Because of the "randomness" of f, such choices are as good as truly random choices. Note that any other processor p_i can compute the random choices processor p_j did in situation S, without any extra communication!

2. "STORAGELESS" DISTRIBUTION OF SECRET NUMBERS

2.1 The Problem

Consider the problem in distributing secret identification numbers (ID's). Every user in the system should receive a secret ID from the system, which is easily verifiable by the system, but hard to compute by anyone else. An example may be assigning calling card numbers to telephone customers. We assume there are no two users with the same name.

A possible solution could be to assign each user U a secret randomly selected number r, and store the pair(U,r) in a protected data base. This solution requires storage proportional to the number of users, which may be very large. Using our random functions, we propose a "storageless" solution to this problem.

2.2 Our Solution

Let F_k be a poly-random collection, and let the server pick $f \epsilon F_k$ at random. Then, the server assigns each user U, f(U) as her secret number. To verify whether(U,n) is a legal pair, the server computes f(U) and compares it with n. Now, suppose that Alice has such a secret ID and that all of her relatives(A_1,A_2 etc..), who possess their own secret ID's gang up to discover Alice's ID. They try to exploit the fact that their names $A_1,A_2...,A_q$ are similar to hers. However, for f picked by the server from a poly-random collection, they could not compute f(Alice) given $f(A_1),...,f(A_q)$.

This solution requires only k bits of storage, when the number of users in the system is bounded by a polynomial in k.

Notice that this solution also works in a distributed environment. If each "branch" of the server has a computer with the (shared k-bit) secret s embedded in it, a secret number can be handed out in San Francisco and be(locally) verified in Boston.

2.3 The Correctness Argument: Simulation

Assume that one-way permutations exists and that g is such a permutation. Let $F=\{F_k\}$ be a poly-random collection constructed using g and let f be a function randomly selected from F_k.

Assume that $A_1,A_2,\ldots,A_q$ have some advantage in guessing f(Alice) from $f(A_1),\ldots,f(A_q)$. Clearly, they could not have such an advantage if f were a truely random function. Thus, they can distinguish f from a truely random function. This, in turn, provides an algorithm for inverting g.

3. DYNAMIC HASHING

3.1 The problem

Consider the problem of hashing a few long keys (names) into shorter addresses (abbreviations), such that with very small probability two keys are hashed into the same address.
The classical purposes of hashing are:
(1) To save on memory space. (For example, assigning physical memory location to variables can be done by applying a hashing function to the variable names. This way there is no need to store the variable names, which may be long.)
(2) To allow fast retrieval of keyed information (hashing will help in applications where accessing the memory is slower than evaluating the function).

3.2 Our solution

In order to present our solution let us first generalize the definition of a poly-random collection. Let $p_1(.)$and $p_2(.)$ be two polynomials. A generalized poly-random collection is a collection, $F=\{F_{p_1(k),p_2(k)}\}$, of indexed and easy to evaluate functions from $I_{p_1(k)}$ into $I_{p_2(k)}$ such that a function chosen at random from $F_{p_1(k),p_2(k)}$ cannot be distinguished in poly(k) time from a random function from $I_{p_1(k)}$ into $I_{p_2(k)}$.

Our solution consists of using a function f chosen at random from $F_{p_1(k),p_2(k)}$ as a hashing function. (i.e. key K is hashed into address f(K)).

Note that our hashing function is much more roboust with respect to polynomial time computation than the Universal Hashing suggested by Carter and Wegman[CW]. In their scheme, the adversary picks an arbitrary key distribution and the hashing performance(expected number of collisions) is analyzed with respect to this fixed distribution.

Our scheme performs well even if the adversary does not fix his key distribution apriori, but can dynamically change the key distribution during the hashing process upon seeing the hashing function values on previous keys. In other words, even if an adversary can pick the keys to be hashed and examine the values of the hash functions on old keys, he cannot force collisions. Moreover, the adversary cannot distinguish the hashing value for a new key from a random value.

The roboustness of our hashing technique, makes it particularly suitable for cryptographic purposes. For example, Brassard ([B]) has pointed out the advantages of authentication schemes based on "cryptographically strong" hashing functions. This them is further developed in section 5.

4. MESSAGE AUTHENTICATION AND TIME-STAMPING

In this section we will show how to construct deterministic, memoryless, authentication schemes which are highly robust, as discussed in the following concrete setting.

Assume that all the employees of a large bank communicate through a public network. As an adversary may be able to inject messages, the employees need to authenticate the messages they send to each other (e.g. "transfer sum S from account A to account B"). A solution may consist of appending to the message m an authentication tag which is hard to compute by an adversary. In particular, we propose the following. Let all employees have access to authentication machines which compute a function f_s in a poly-random collection. The tag associated with a message m is $f_s(m)$. We can tradeoff security for the length of the tag. For example, if one uses only the first 20 bits of $f_s(m)$ as an authentication tag, then the chance that an adversary could successfully authenticate a message is about 1 in a million.

To avoid playback of previously authenticated messages, it is common practice to use time-stamps. Namely, authenticate m concatenated with date it was sent. So far, time-stamping was only a heuristic as an adversary who sees the message m authenticated with date D could conceivably authenticate m with another date (say D+1). Using our solution for message authentication, time-stamping makes playback provably hard. This is the case as for a random function $f(\chi)$ is totally unrelated to $f(\chi+1)$, and therefore the same holds (with respect to polynomial-time adversaries) for poly-random collections.

Another threat to the Bank's security is the loyalty of its own employees. They have the authenticating computer at their disposal and can use it to launch a chosen message attack against the scheme, so that when they are fired they can forge transactions. Our message authentication scheme remains secure even when the employees are not trustworthy, if each message to be authenticated is automatically time stamped by the computer. An employee who leaves the bank, after having widely experimented with the machine, will not be able to authenticate even one new message.

5. AN IDENTIFY FRIEND OR FOE SYSTEM

The members of a large but exclusive society are well known for their brotherhood spirit. Upon meeting each other, anywhere in the world, they extend hospitality, favors, advice, money, etc. Naturally they face the danger of imposters trying to take advantage of their generosity. Thus, upon meeting each other, they must execute a protocol for establishing membership. As they meet in public places (busses, trains, theatre), they must be careful not to yield information that can lead to future successful impersonations. They go around carrying pocket computers on which they may make calculations.

Clearly a password scheme will not suffice in this context, as the conversations are public. An interactive identification scheme is needed where the ability to ask questions does not enable future successful impersonations. Note that the questions that A may ask member B, must be picked from an exponential range to prevent an active imposter from asking all possible questions, receiving all possible answers and thereafter successfully impersonating as a member (or to prevent a passive imposter from having a non-negligible probability of being asked a question that he overheard the answer to).

Using our poly-random collection, we can fully solve this problem. Let the president of the society choose a k-bit random string s, specifying a function f_s in a poly-random collection. Each member receives a computer which calculates f_s. When member A meets B, he asks "z?" where $z \in_R I_k$. Only if B answers $f_s(z)$, will member A be convinced that B is a member. In addition, if the computers that calculate f_s can be manufactured so that they can not be duplicated, then losing a computer does not compromis the security of the entire scheme; it just allows one non-member to enjoy the privileges of the society.

6. SOLVING BLUM BLUM & SHUB OPEN PROBLEM

Blum, Blum and Shub [BBS] have presented an interesting CSPRB generator whose sequences pass all polynomial time statistical tests if and only if deciding Quadratic Residuosity modulo a Blum-integer[(1)] whose factorization is not known, is intractable.

[(1)] A Blum integer is an integer of the form p_1p_2 where p_1 and p_2 are distinct primes both congruent to 3 mod 4.

Notice that, even though a CSPRB sequence generated with a k-bit long seed is $P_1(k)$-bit long, a CSPRB generator and a seed s define an infinite bit-sequence $b_0, b_1, \ldots$ An interesting feature of Blum Blum Shub's generator is that knowledge of the seed and of the factorization of the modulus allows direct access to each bit in an exponentially long bit string (i.e. if k denotes the length of the seed and $i \leq k$, then the i-th bit in the string (b_i) can be computed in poly(k) time). This is due to the special weak one-to-one one-way function on which the security of their generator is based. However, this exponentially long bit string MAY NOT appear "random". Blum, Blum and Shub only prove that any SINGLE polynomially long interval of CONSECUTIVE bits in the string passes all polynomialt time statistical tests for strings. Indeed, it may be the case that, given $b_1, \ldots, b_k$ and $b_{2^{\sqrt{k}}+1}, \ldots, b_{2^{\sqrt{k}}+k}$ it is easy to compute any other bit in the string. Another CSPRB generator which possess the direct access property was suggested by Goldwasser, Micali and Tong GMT . Their generator is also based on a specific intractability assumption(factoring in a subset (of half) of the Blum integer). Also, it is not known whether direct access in the GMT generator preserves randomness.

The Blum Blum Shub open problem consists of whether direct access to exponentially far away bits in their pseudo-random pad is a "randomness preserving" operation. Or more generally, whether there exist generators which possess such a "randomness preserving direct access" property.

The Blum Blum Shub's generator, when fed with a k-bit long seed s, defines a function f_s in the following way: for each k-bit integer $\chi, f_s(\chi)$ is the χ-th block of k bits in the pad. I.e. $f_s(\chi) = b_{k.\chi+1} \cdots b_{k.\chi+k}$. Recall that the Blum Blum Shub generator is based on the intractability assumption of a special permutation and furthermore, even under this assumption, direct access was not proved to be a randomness preserving operation. As a consequence f_s may not be "random".

We solve the above problem in a very strong sense. In fact we construct random functions f, from k-bit strings into k-bit strings, given ANY one-way permutation. Having constructed such an f, we have virtually constructed the $k2^k$-bit long string $s_f = f(1)f(2)\ldots f(2^k)$. For the set $\{s_f\}$ we prove that direct access is a "randomness preserving" property.

APPENDICES

Appendix A: CSPRB Generators, One-Way Permutations and Yao's Construction.

Following the unpredictable number generators of Shamir [S], Blum and Micali [BM] have introduced the notion of Cryptographically Strong Pseudo-Random Bit (CSPRB) generators. They have also presented the first instance of it, relying on the intractability assumption of the discrete logarithm problem.

Let t be any fixed constant. A CSPRB generator is a deterministic program that receives as input a (random) k-bit long seed and outputs a k -bit long (pseudo-random) sequence such that the next bit in the sequence cannot be predicted in polynomial (in k) time from the preceeding bits. Yao [Y] introduces the notion of a polynomial-time statistical test and shows that the outputs of CSPRB generators pass all polynomial-time statistical tests. He also proves that one can construct CSPRB generators given any (weak) one-way permutation.

Let us be more formal. Let $f_k: I_k \to I_k$ be a sequence of permutations such that there is a polynomial-time algorithm that on input $\chi \varepsilon I_k$ computes $f_k(\chi)$. Let the function f be defined as follows: $f(\chi) = f_k(\chi)$ if $\chi \varepsilon I_k$. We say that f is a one-to-one one-way function if for all polynomial-time Turing Machines M there is a polynomial P such that, for all sufficiently large k

$M(\chi) \quad {f_k}^{-1}(\chi)$ for at least a fraction $\frac{1}{P(k)}$ of the $\chi \varepsilon I_k$.

LEMMA 1(Yao Y): Given a weak one-to-one one-way function, it is possible to construct CSPRB generators.

Sketch of the proof: Given a one-way permutation, f, Yao construct a hard to evaluate predicate by taking the exclosive-or of the inverse of f on polynomially many points. Namely,

$B_k(X_1, X_2, \ldots, X_{k^t}) = \text{XOR}\ f_k{-1}(X_1) f_k{}^{-1}(X_2) \ldots f_k{-1}(X_{k^t})$

where XOR s is the exclusive-or of all the bits of the string s.

Appendix B: The Construction of F(from any CSPRB Generator)(GGM)

Let G be a CSPRB-generator. Recall that G is a function defined on all bit strings such that if $\chi \varepsilon I_k, G(\chi) = b_1^{\chi}, \ldots, b_{P_1(k)}^{\chi}$. With no loss of generality, we can assume that $P(k) \geq 2k$.

(This is the case since Goldreich and Micali (GM) have shown that the existence of a CSPRB generator which expand a k-bit long seed into a (k+1)-bit output pad, yields the existence of a generator which expend a k-bit long seed into a 2k-bit long pad).

Let $S = {}_{k} \quad S_k$ be defined as follows. S_k is the set of all the first

2k bits output by G on seeds of length k. Then S passes all polynomial time statistical tests for strings.

Let $\chi \varepsilon I_k$ be a seed for G. $G_0(X)$ denotes the first k bits output by G on input X; $G_1(X)$ denotes the next k bits output by G. Let $\alpha = \alpha_1\alpha_2 \ldots \alpha_t$ be a binary string. We define $G_{\alpha_1\alpha_2\ldots\alpha_t}(\chi)=G_{\alpha_t}(\ldots(G_{\alpha_2}(G_{\alpha_1}(\chi)))\ldots)$.

Let $\chi \varepsilon I_k$. The function $f_\chi : I_k \text{->} I_k$ is defined as follows:

$$\text{For } y=y_1y_2\ldots y_k,\ f_\chi(y)=G_{y_1y_2\ldots y_k}(\chi).$$

Define $F_k=\{f_\chi\}_{\chi\varepsilon I_k}$ and $F=\{F_k\}$.

Note that a function in F_k needs not be one-to-one.

The reader may find it useful to picture a function $f_\chi : I_k \text{->} I_k$, as a full binary tree of depth k with k-bit strings stored in the nodes and edges labelled 0 or 1. The k-bit string χ will be stored in the root. If a k-bit string is s is stored in an internal node, v, then $G_0(s)$ is stored in v's left-son, v_l, and $G_1(s)$ is stored in v's right-son, v_r. The edge (v,v_l) is labelled 0 and the edge(v,v_r), is labelled 1. The string $f_\chi(y)$ is then stored in the leaf reachable from the root following the edge path labelled y.

It is easy to see that F satisfies properties (1) and (2) of poly-random collections. A proof that F satisfy also property (3) (pseudo-randomness) can be found in GGM (Main Theorem).

GENERALIZATIONS

In some applications, we would like to have random functions from $I_{P_3(k)} \text{->} I_{P_4(k)}$. E.g. in hashing we might want functions from I_k into I_{10}. We meet this need by introducing the collection $F=\{F_k\}$ defined as follows: For $\chi \varepsilon I_k$, f_χ F_k is a function from $I_{P_3(k)}$ into $I_{P_4(k)}$ defined as follows. Let $y=y_1\ldots y_{P_3(k)}$. Define $f_\chi(y) = \Gamma_{P_4(k)}\ G_{y_1\ldots y_{P_3(k)}}(\chi)$, where $\Gamma_{P_4(k)}(Z)$ are the first $P_4(k)$ bits output by G when fed input $z\varepsilon I_k$, where G is a CSPRB generator.

Such an F is also a poly-random collection: properties (1) and (2) trivially hold, and property (3) can be proved in a way similar to the proof of the Main Theorem in GGM .

Appendix C: An(unsatisfactory)straightforward simulation of random functions

Assume one needs to be able to evaluate a function that looks as if it is randomly selected from H_k. One can argue that since he will only need to evaluate the function on polynomially many(in k) inputs, it is sufficient that he proceeds as follows:

Choose a CSPRB generator G and a random k-bit long seed s. This choice specifies a k^{t+1}-bit long pseudo-random bit-sequence $b_1,\ldots,$ $b_{k^{t+1}}$ that can be used as securely as a truely random pad. Let $\chi_1,\ldots,$ χ_j denote the chronologically ordered sequence of inputs on which the "random function" f has already been evaluated.

Assume now that f needs to be evaluated on an input y. If y χ_i for i=1...j, then f(y) is set to be the j+1st block of k consecutive bits in the pseudo-random sequence. (I.e. f(y)=bk.j+1...$b_{k.j+k}$). Also, y is stored as the j+1st input(storing f(y) is optional). Otherwise, if y=χ_i for some i, f(y) is recomputed as the ith block of bits in the pseudo-random sequence (or is retrieved from memory).

Note that this procedure does not specify a function and thus does not meet the theoretical challenge. Furthermore, it wastes storage proportionally to the number of oracle queries(inputs on which the function has been evaluated). This is a strict lower bound! If the inputs are randomly chosen they cannot be compressed at all!

By means of a more clever use of CSPRB generators, our solution requires only k bits of storage. Thus it meets both the theoretical and the practical challenges.

ACKNOWLEDGEMENTS

We would like to thank Ron Rivest for suggesting the IFOF application.

REFERENCES

[AL] D. Angluin and D. Lichtenstein, Provable Security of Cryptosystems: a Survey, YaleU/DCS/TR-288, 1983

[BBS] L. Blum, M. Blum and M. Shub, A simple secure pseudo random number generator, Advances in Cryptology: Proc. of CRYPTO-82, ed. D. Shaum, R. L. Rivest and A.T. Sherman. Plenum press 1983, pp 61-78.

[BG] M. Blum and S. Goldwasser, An Efficient Probabilistic Public-Key Encryption Scheme Which Hides all Partial Information, preprint May 1984.

[BM] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. COMPUT., Vol 13, No. 4, Nov. 1984.

[B] G. Brassard, On computationally secure authentication tags requiring short secret shared keys, Advances in Cryptology: Proc. of CRYPTO-82, ed. D. Shaum, R.L. Rivest and A.T. Sherman. Plenum press 1983, pp 79-86.

[CG] B. Chor and O. Goldreich, RSA Rabin least significant bits are $\frac{1}{2}+\frac{1}{\text{poly(logN)}}$ secure, MIT/LCS/TM-260, May 1984.

[CW] J.L. Carter and M.N. Wegman, Universal classes of hash functions, Proc. 9th ACM Symp. on Theory of Computing, 1977, pp 106-112.

[GGM] O. Goldreich, S. Coldwasser and S. Micali, How to construc random functions, MIT/LCS/TM-244, November 1983.

[GM] O. Goldreich and S. Micali, The weakest CSPRB generator implies the strongest one, in preparation.

[GMT] S. Goldwasser, S. Micali and P. Tong. Why and how to establish a private code on a public network, Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp 134-144.

[RSA] R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Commun. ACM vol. 21, Feb. 1978, pp 120-126.

[S] A. Shamir, On the Generation of Cryptographically Strong Pseudo-random Sequences, 8th International Colloquium on Automata, Languages, and Programming, Lect. Notes in Comp. Sci. 62, Springer Verlag, 1981.

[Y] A.C. Yao, Theory and applications of trapdoor functions, Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp 80-91.