Electronic Colloquium on Computational Complexity - Reports Series 1995 - available via:
ECCC
TR95-045
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

Synthesizers and Their Application to the Parallel Construction of Pseudo-Random Functions *

Moni Naor † Omer Reingold †

Abstract

We present a new cryptographic primitive called *pseudo-random synthesizer* and show how to use it in order to get a parallel construction of a pseudo-random function. We show an NC^1 implementation of synthesizers based on the RSA or the Diffie-Hellman assumptions. This yields the first parallel (NC^2) pseudo-random function and the only alternative to the original construction of Goldreich, Goldwasser and Micali (GGM). The security of our constructions is similar to the security of the underling assumptions. We discuss the connection with problems in Computational Learning Theory.

*A preliminary version of this paper appeared at the *Proc. 35th IEEE Symp. on Foundations of Computer Science* 1995.

†Incumbent of the Morris and Rose Goldman Career Development Chair, Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Research supported by an Alon Fellowship and a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences. E-mail: naor@wisdom.weizmann.ac.il.

‡Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: reingold@wisdom.weizmann.ac.il.

1 Introduction

A pseudo-random function, as defined by Goldreich, Goldwasser and Micali [18], is a function that is indistinguishable from a truly random function to a (polynomial-time bounded) observer who can access the function as a black-box (i.e. can provide inputs of his choice and gets to see the value of the function on these inputs). Pseudo-random functions are the key component of private-key cryptography. They allow parties who share a common key to send secret messages to each other, to identify themselves and to authenticate messages [11, 19, 15, 33]. In addition they have many other applications, essentially in any setting that calls for a random function that is provided as a black-box [6, 9, 12, 17, 16, 34, 40].

Goldreich, Goldwasser and Micali provided a construction of such functions. This is the only known construction, even under specific assumptions, such as "factoring is hard". Their construction is sequential in nature and consists of n successive invocations of a pseudo-random generator (where n is the number of bits in the input to the function). Our goal in this paper is to present an alternative construction for pseudo-random functions that can be implemented in $\log n$ phases.

We introduce a new cryptographic primitive which we call *pseudo-random synthesizer*. A pseudo-random synthesizer is a two variable function $S(x, y)$, so that if many (but polynomially bounded) random assignments are chosen to x and to y, the output of S on all the combinations of these assignments is indistinguishable from random to a polynomial time observer. Our main results are:

1. A construction of pseudo-random functions based on pseudo-random synthesizers. Evaluating such a function involves $\log n$ phases where each phase consists of evaluating several synthesizers in parallel.

2. Constructions of parallel (NC^1) synthesizers based on standard assumptions such as RSA (it is hard to extract roots modulo a composite) and Diffie-Hellman and a very simple construction based on a problem from learning. The key generating algorithm of these constructions is sequential for RSA, non-uniformly parallel for Diffie-Hellman and parallel for the learning problem.

Taking (1) and (2) together we get a pseudo-random function that can be evaluated in NC^2. We note that our constructions do not weaken the security of the underlining assumption. For instance, in the RSA case, if there is an algorithm for breaking our construction in time t and success α (success α means that the observer has a bias of at least α in distinguishing the pseudo-random function from the random one), then there is an algorithm for breaking RSA that works in time $poly(t)$ and breaks RSA with probability $\alpha/poly(t)$. See [23, 33] for a discussion of security preserving reductions.

The class NC has been criticized as a model for parallel computation for two main reasons:

- It ignores communication delays and other parameters that determine the execution time on an actual parallel machine.

- It over-emphasizes latency rather than the speed-up of problems.

These criticisms seem less valid for the problem of constructing pseudo-random functions, since (a) it is likely that it will be implemented in a special purpose circuit (as there are DES chips) and (b) when used for the encryption of messages on a network, the latency of computing the function is added to the latency of the network and hence it makes sense to minimize it. Furthermore, if the complexity of evaluating a synthesizer on a given input is comparable to that of a pseudo-random

generator, then the work performed by our construction is comparable to the one in [18] and we can get optimal speed-up.

There is a deep connection between pseudo-random functions and hardness results for learning. Since a random function cannot be learned, if a concept class is strong enough to contain pseudo-random functions we cannot hope to learn it efficiently. Since no construction of pseudo-random functions in NC was known, several ways of bypassing this were suggested [29, 30]. It is still of interest to learning theory to find a distribution of *concepts* that is hard to learn [24]. By strengthening our assumptions, say, that it is hard to break RSA in time $n^{O(\log n)}$ we can get a pseudo-random function in NC^1. In [7] a way of using problems that are hard to learn for cryptographic purposes was proposed. We discuss the connection between our work and learning in Section 8. In general, we can use hard-to-learn problems to obtain synthesizers and thus pseudo-random functions.

Another application of pseudo-random functions in complexity was suggested by the work on Natural Proofs [42]. They showed that the existence of a pseudo-random function in NC^1 implies that there are no what they called Natural Proofs (which include all known lower bound techniques) for separating NC^1 from P. Our construction based on the strengthened Diffie-Hellman assumption satisfies that.

Previous work: Impagliazzo & Naor [26] have provided parallel constructions for several cryptographic primitives based on the hardness of subset sum (and factoring). The primitives include pseudo-random generators, universal one-way hash functions and strong bit-commitments.

An idea of Levin [31] is to construct from pseudo-random generators that expand the input by a factor of 2 (like the one in [26]) pseudo-random functions by selecting some secret hash function h and applying the GGM construction [18] to $h(x)$ instead of x. If $|h(x)| = \log^2 n$ then the depth of the tree is only $\log^2 n$ and presumably we get a pseudo-random function in NC. The problem with this idea is that we have decreased the security significantly: with probability $1/n^{\log n}$ the function can be broken, irrespective of the security guaranteed by the pseudo-random generator. To put this construction in the "correct" light, suppose that for security parameter k we have some problem whose solution requires time 2^k (on instance of size polynomial in k). If we would like to have security $1/2^k$ for our pseudo-random function, then the Levin construction requires depth k whereas our construction requires depth $\log k$.

Luby & Rackoff [34] have shown how to construct pseudo-random *permutations* from pseudo-random functions. Their construction is very simple and involves three or four invocations of a pseudo-random function in order to evaluate the pseudo-random permutation at a given point. Therefore, our constructions yield pseudo-random *permutations* in NC as well.

Organization of the paper: In section 2 we review the definition of pseudo-random functions, in section 3 we define pseudo-random synthesizers and collections of pseudo-random synthesizers and discuss their characteristics. In section 4 we present our parallel construction of pseudo-random functions from pseudo-random synthesizers, in section 5 we prove the security of this construction. In section 6 we discuss the relationships between pseudo-random synthesizers and other cryptographic primitives. In section 7 we present the constructions of pseudo-random synthesizers based on the Diffie-Hellman and the RSA assumptions. In section 8 we show how to construct pseudo-random synthesizers from hard-to-learn problems and consider a very simple concrete example, we also discuss the construction of pseudo-random functions in NC^1. In section 9 we suggest topics for further research.

2 Pseudo-Random Functions

For the sake of completeness and concreteness, we briefly review in this section the concept of pseudo-random functions as it appears in [15] (some of the definitions may have small variations). Informally, a pseudo-random function ensemble is a distribution of functions that cannot be efficiently distinguished from the uniform distribution. That is, when an efficient algorithm gets a function, as a black box, it cannot tell, with non-negligible probability of success, according to which of the distributions it was chosen. To formalize this, we first define function ensembles, (concentrating on length-preserving functions).

Definition 2.1 (function ensembles) *A* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables, so that the random variable F_n assumes values in the set of functions mapping n-bit long strings to n-bit long strings. The* uniform function ensemble, *denoted $R = \{R_n\}_{n\in\mathbb{N}}$, has R_n uniformly distributed over the set of functions mapping n-bit long strings to n-bit long strings.*

In our setting the distinguisher will have the form of an oracle machine that can make queries to a length preserving function, sampled from one of the two function ensembles. We assume that on input 1^n the oracle machine makes only n-bit long queries. In order for a pseudo-random function ensemble to be a practical substitute for the uniform function ensemble it must also be efficiently samplable and computable.

Definition 2.2 (efficiently computable pseudo-random function ensemble) *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is called* efficiently computable pseudo-random function ensemble *if the following conditions hold:*

1. (pseudo-randomness) *for every probabilistic polynomial-time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's*

$$|Prob[M^{F_n}(1^n) = 1] - Prob[M^{R_n}(1^n) = 1]| < \frac{1}{p(n)}$$

   where $R = \{R_n\}_{n\in\mathbb{N}}$ is the uniform function ensemble.

2. (efficient indexing) *There exists a probabilistic polynomial time algorithm, I, and a mapping from strings to functions, ϕ, so that $\phi(I(1^n))$ and F_n are identically distributed. We denote by f_i the $\{0,1\}^n \mapsto \{0,1\}^n$ function assigned to i (i.e. $f_i \stackrel{\text{def}}{=} \phi(i)$). We refer to i as the key of f_i and to I as the key-generating algorithm of F.*

3. (efficient evaluation) *There exists a polynomial time algorithm, V, so that $V(i,x) = f_i(x)$.*

At the following sections the term "pseudo-random functions" is used as an abbreviation for "efficiently computable pseudo-random function ensemble".

3 Pseudo-random Synthesizers

We begin by introducing some notations that are used in the paper.

- Let l and k be any two functions on the natural numbers, we denote by $F = \{F_{k(n)}^{l(n)}\}_{n\in\mathbb{N}}$ an ensemble of functions mapping $k(n)$-bit long strings to $l(n)$-bit long strings.

- Let X be any random variable, we denote by $X^{k\times l}$ the $k\times l$ matrix whose entries are independently identically distributed according to X. We denote by X^k the vector $X^{1\times k}$.
- U_n denotes the random variable uniformly distributed over $\{0,1\}^n$.
- We identify functions of two variables and functions of one variable in the natural way: We take $f:\{0,1\}^n\times\{0,1\}^n \mapsto \{0,1\}^k$ to be equivalent to $f:\{0,1\}^{2n}\mapsto\{0,1\}^k$, i.e. $f(x,y)$ is the same value as $f(x\circ y)$, where x and y are n-bit long strings and $x\circ y$ stands for x concatenated with y.

As mentioned above, we introduce in this paper a new cryptographic primitive called a pseudo-random synthesizer. Loosely speaking, pseudo-random synthesizers are efficiently computable functions of two variables that "merge" random bit sequences from two sources into one pseudo-random bit sequence. The significant feature of these functions is that they may reuse the input to each variable in all different combinations and their output still looks random. We first formalize the phrase "all different combinations".

Definition 3.1 *Let f be a function $f:\{0,1\}^{2n}\mapsto\{0,1\}^k$, X and Y two sequences of n-bit long strings $X=\{x_1,\ldots,x_k\}$ and $Y=\{y_1,\ldots,y_l\}$. We define $CONV_f(X,Y)$ to be the $k\times l$ matrix $(f(x_i,y_j))_{i,j}$.*

We can now define what a pseudo-random synthesizer is.

Definition 3.2 (pseudo-random synthesizer) *A* pseudo-random synthesizer *is a function $S:\{0,1\}^*\times\{0,1\}^*\mapsto\{0,1\}^*$ such that the following conditions hold:*

1. *S is polynomial-time computable.*
2. *There exists a function, $l_S:\mathbb{N}\mapsto\mathbb{N}$, such that $|S(x,y)|=l_S(n)$ for every $x,y\in\{0,1\}^n$.*
3. *For every probabilistic polynomial-time algorithm, D, every two polynomials $p(\cdot)$ and $m(\cdot)$, and all sufficiently large n's*

$$|Prob[D(CONV_S(X,Y))=1]-Prob[D({U_{l_S(n)}}^{m(n)\times m(n)})=1]|<\frac{1}{p(n)}$$

where X and Y are independently drawn from ${U_n}^{m(n)}$. (i.e. for random X and Y the matrix $CONV_S(X,Y)$ can not be efficiently distinguished from a random matrix.)

Notice that if S is a pseudo-random synthesizer and $l_S\geq 2n+1$ then S also defines a pseudo-random generator. This immediately follows from the definition if we take the sample size $m(n)$ to be 1. On the other hand, if G is a pseudo-random generator then we only get that for some function $l_G:\mathbb{N}\mapsto\mathbb{N}$ and for every polynomial $m(\cdot)$ the polynomial-size sample $\{G(U_n)^{m(n)}\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{U_{l_G(n)}^{m(n)}\}_{n\in\mathbb{N}}$. So if in the definition of pseudo-random synthesizer we require that $l_S\geq 2n+1$ it becomes a strengthening of the definition of pseudo-random generator, but how powerful are pseudo-random synthesizers if l_S is very small? The following lemma shows that any synthesizer S can be used to construct another synthesizer S', with a reasonably large output, in a way that preserves the parallel time complexity of S.

Lemma 3.1 *For every i and every constant $0<\epsilon<2$, if S is a pseudo-random synthesizer in NC^i (resp. AC^i) then there exists a pseudo-random synthesizer S^ϵ in NC^i (resp. AC^i) such that $l_{S^\epsilon}(n)=\Omega(n^{2-\epsilon})$*

Proof. For every constant $c > 0$ we can define S^c in the following way: Define $k_n \stackrel{\text{def}}{=} \max\{k : k^{c+1} < n\}$. For input $x, y \in \{0,1\}^n$, regard the first k_n^{c+1} bits of x and y as two k_n^c-long sequences X and Y of k_n-bit long strings. S^c then outputs $CONV_S(X,Y)$ (when we view it as a vector rather than a matrix). Notice that the following properties hold for S^c:

1. *S^c is indeed a pseudo-random synthesizer.* Let X' and Y' be independently drawn from ${U_n}^{m(n)}$ and let X and Y be independently drawn from ${U_{k_n}}^{m(n)k_n^c}$. From the definition of S^c, for every polynomial $m(\cdot)$, the distributions $CONV_{S^c}(X',Y')$ and $CONV_S(X,Y)$ are identical. Taking into account the fact that n is polynomial in k_n, we conclude that every polynomial-time distinguisher for S^c is also a polynomial-time distinguisher for S. Since S is a pseudo-random synthesizer so is S^c.

2. $l_{S^c}(n) = \Omega(n^{2-\frac{2}{c+1}})$. Since c is a constant and $n < (k_n+1)^{c+1}$ for every n it holds that $l_{S^c}(n) = k_n^{2c} l_S(k_n) \geq k_n^{2c} = (1-\frac{1}{k_n+1})^{2c}(k_n+1)^{2c} = \Omega(n^{2-\frac{2}{c+1}})$

3. *S^c is in NC^i (resp. AC^i).* Immediate from the construction of S^c.

Thus, by taking S^ϵ to be S^c for any $c > \frac{2}{\epsilon} - 1$ we obtain the lemma. □

In the construction of pseudo-random functions in NC we assume the existence of pseudo-random synthesizers with linear output size in NC. In order to complete the construction it is enough, by Lemma 3.1, to show the existence of synthesizers with constant output size in NC.

Nevertheless, the construction in Lemma 3.1 has an obvious disadvantage. The security of the synthesizer we construct is related to the security of the original synthesizer on much smaller input size. Thus, to preserve sufficient security we must work with larger numbers, resulting in a substantial increase in the time and space complexity of any construction using this synthesizer. If we assume the existence of a pseudo-random generator G in NC such that $|G(s)| = 2|s|$ for every s, then we can use a simplified variant of the construction of [18] for an alternative construction to the one in Lemma 3.1.

Theorem 3.2 *([18]) If G is a pseudo-random generator such that $|G(s)| = 2|s|$ for every s then for every polynomial $p(\cdot)$ there exists a pseudo-random generator G' such that $|G'(s)| = p(|s|)|s|$ for every s. Further more if G is in NC^i then G' is in NC^{i+1}.*

G' is defined as follows: On input s it computes $G(s) = s_0 \circ s_1$ and recursively generates $\frac{p(|s|)|s|}{2}$ bits from s_0 and from s_1. The number of levels required is $\lceil \log p(|s|) \rceil = O(\log |s|)$. Given a synthesizer S in NC, and G' as before, we can construct a new synthesizer S' in the following way: On input $x, y \in \{0,1\}^n$, first compute $X = G'(x) = \{x'_1, \ldots, x'_{p(n)}\}$ and $Y = G'(y) = \{y'_1, \ldots, y'_{p(n)}\}$ and then output $CONV_S(X,Y)$. It is easy to verify that S' is indeed a pseudo-random synthesizer in NC and that $l_{S'}(n) = p^2(n) l_S(n)$.

In this case, the security of S' relates to the security of S and G on the same input size. Nevertheless, the time complexity of S' is still substantially larger than the time complexity of S, and the parallel time complexity of S' might also be larger. We cannot assert that either of the two constructions of synthesizers with extended output size is superior to the other.

A natural way to relax the definition of a pseudo-random synthesizer is to allow a distribution of functions for every input size rather than a single function. To formalize this we use the concept of an efficiently computable function ensemble which was defined in section 2.

Definition 3.3 (collection of pseudo-random synthesizers) *An efficiently computable function ensemble $S = \{S_{2n}^{l(n)}\}_{n\in\mathbb{N}}$, where l is an $\mathbb{N} \mapsto \mathbb{N}$ function, is called* a collection of pseudo-random synthesizers *if for every probabilistic polynomial-time algorithm, D, every two polynomials $p(\cdot)$ and $m(\cdot)$, and all sufficiently large n's*

$$|Prob[D(CONV_{S_{2n}^{l(n)}}(X,Y)) = 1] - Prob[D(U_{l(n)}{}^{m(n)\times m(n)}) = 1]| < \frac{1}{p(n)}$$

where X and Y are independently drawn from $U_n{}^{m(n)}$.

As we shall see, a collection of pseudo-random synthesizers is sufficient for the construction of pseudo-random functions. Working with a collection of synthesizers, rather than a single synthesizer, enables us to extract some of the computations into a preprocessing stage during the sampling. This is especially useful if apart from a sequential preprocessing stage all other computations can be done in parallel.

Most of our observations regarding synthesizers are easily extended to collections of synthesizers. The only subtle point is that in order to construct a pseudo-random generator from a collection of synthesizers we should use some of the bits of the seed to sample a function from the collection. Since the number of bits needed in order to sample is polynomial, we can use a similar construction to the one in Lemma 3.1 to ensure that we have enough bits to sample and that our output is large enough.

4 A Parallel Construction of Pseudo-Random Functions

First, we define an operation on sequences we use in the construction.

Definition 4.1 *For every function $S : \{0,1\}^{2n} \mapsto \{0,1\}^n$ and every sequence of n-bit long strings $L = \{l_1, l_2, \ldots, l_k\}$ define $SQUEEZE_S(L)$ to be the sequence $L' = \{l'_1, \ldots, l'_{\lceil \frac{k}{2} \rceil}\}$ where $l'_i = S(l_{2i-1}, l_{2i})$ for $i \leq \lfloor \frac{k}{2} \rfloor$ and if k is odd then $l'_{\lceil \frac{k}{2} \rceil} = l_k$.*

We are now ready to present the construction of pseudo-random functions, using pseudo-random synthesizers as building blocks.

Construction 4.1 (Pseudo-Random Functions) *Let $S = \{S_{2n}^n\}_{n\in\mathbb{N}}$ be a collection of pseudo-random synthesizers and let I_S be a probabilistic polynomial-time key-generating algorithm for S (as in definition 2.2). The function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is constructed as follows:*

- *(*indexing*) We define a probabilistic polynomial-time key-generating algorithm I. On input 1^n I outputs a couple $(\vec{a}, \vec{k})$ where $\vec{a} = \{a_1^0, a_1^1, a_2^0, a_2^1, \ldots, a_n^0, a_n^1\}$ is generated according to U_n^{2n} and $\vec{k} = \{k_1, k_2, \ldots, k_{\lceil \log n \rceil}\}$ is generated by $\lceil \log n \rceil$ independent executions of I_S on input 1^n. (We later show how to reduce $\vec{a}$ from a $2n$ long sequence of n-bit long strings to a single n-bit long string.)*
- *(*evaluation*) For every possible output $(\vec{a}, \vec{k})$ of I on 1^n we define a function $f_{\vec{a},\vec{k}} : \{0,1\}^n \mapsto \{0,1\}^n$. On input $x = x_1 x_2 \ldots x_n$ the function outputs the single value in*

$$SQUEEZE_{S_{k_1}}(SQUEEZE_{S_{k_2}}(\ldots SQUEEZE_{S_{k_{\lceil \log n \rceil}}}(\{a_1^{x_1}, a_2^{x_2}, \ldots, a_n^{x_n}\})\ldots))$$

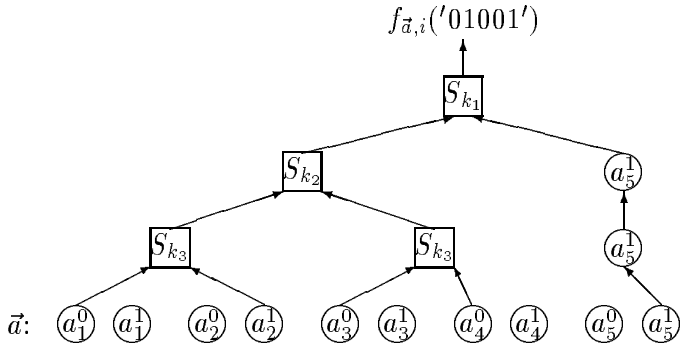

Figure 1: Computing the Value of the Pseudo-Random Function for $n = 5$

The evaluation of $f_{\vec{a},\vec{k}}(x)$ can be thought of as a recursive labeling process of a binary tree with n leaves and depth $\lceil \log n \rceil$ (in contrast to the depth-n binary tree in [18].) With each leaf we associate a bit of the input: this bit "chooses" one of two labels to its corresponding leaf. The label of each internal node at depth d is the value of $S_{k_{d+1}}$ on the labels of its children. The value of $f_{\vec{a},\vec{k}}(x)$ is simply the label of the root. (Figure 1 illustrates the evaluation of $f_{\vec{a},\vec{k}}$ for $n = 5$.)

Finally, we define F_n to be the random variable that assumes as values the functions $f_{\vec{a},\vec{k}}$ for every possible output $(\vec{a},\vec{k})$ of I on 1^n with the probability space induced by I.

Since the function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ was defined through the algorithms for sampling its functions and evaluating their values, it is clear that F is efficiently computable (given that S is efficiently computable). Furthermore, the parallel time complexity of $f_{\vec{a},\vec{k}} \in F_n$ is $O(\log n)$ times the parallel time complexity of $S_k \in S^n_{2n}$ and the parallel time complexity of I and I_S are identical.

Note that for some collections of synthesizers (as those presented in this paper) we can reduce the overhead of keeping $\lceil \log n \rceil$ keys of the collection. Certainly, this is true when we are using a single synthesizer instead of a collection. In this case we don't have to keep any key, since there is a single key for every input size. Moreover, if the collection of synthesizers remains secure even when it uses a public key (i.e. even if the distinguisher algorithm gets access to the key) then we can replace the $\lceil \log n \rceil$ keys with a single one.

Assume that the functions in S are in NC (so the pseudo-random functions we construct are in NC) and that there exists a pseudo-random generator G in NC, such that $|G(s)| = 2|s|$ for every s. In this case, a natural modification to the construction is to replace, for every $f_{\vec{a},\vec{k}}$, the sequence $\vec{a}$ with an n-bit long seed $\tilde{a}$. At the beginning of each application of $f_{\tilde{a},\vec{k}}$ we can obtain $\vec{a}$ by applying G' to $\tilde{a}$, where G' is the pseudo-random generator that can be constructed from G according to Theorem 3.2 (for $p(n) = 2n$, i.e. by using $\lceil \log n + 1 \rceil$ levels of the recursion.) Any efficient distinguisher between the new function ensemble, obtained by the modification, and the original one can be used to efficiently distinguish between the output of G' and the uniform distribution. Thus, the two function ensembles are computationally indistinguishable.

Actually, there is no need for the separate assumptions on the existence of G. Assume for example that we are working with a single synthesizer S. We can define $G(x_1 \circ \cdots x_4 \circ y_1 \circ \cdots \circ y_4)$ to be $CONV_S(X,Y)$, where $X = \{x_1,\ldots,x_4\}$ and $Y = \{y_1,\ldots,y_4\}$ (we assume for simplicity that 8 divides n and take $x_l, y_l \in \{0,1\}^{n/8}$ for $1 \le l \le 4$), and continue the modification as before. In the general case we have to replace S with a randomly chosen $S_j \in S^{n/8}_{n/4}$ independently at

every level of the recursion. By similar arguments the new function ensemble is computationally indistinguishable from the original one. The modification increases the parallel time complexity and amount of work, for functions in the ensemble, by a constant factor. On the other hand, if $|j| \ll \frac{2n^2}{\log n}$ it substantially reduces their key size.

Note that the length of the strings in $\vec{a}$ determines the security of the functions. There is no real reason for the strings to be n-bit long (where n is the length of the strings the functions are applied on).

5 Security of the Construction

To prove that the pseudo-random functions we construct are indeed secure we make use of an hybrid argument, and show that any distinguisher for the pseudo-random functions can be transformed into a distinguisher for the pseudo-random synthesizers.

Theorem 5.1 *Let S, I_S and F be as in Construction 4.1 , then F is an efficiently computable pseudo-random function ensemble. In particular a distinguisher between F and the uniform ensemble yields a distinguisher between the output of S and the uniform distribution. The success probability of the new distinguisher only decreases by a factor of $\frac{1}{\lceil \log n \rceil}$.*

Proof. As commented in section 4.1 it is obvious from the construction that F is an efficiently computable function ensemble. Assume that F is not pseudo-random. By the definition of pseudo-random function ensembles, there exists a polynomial-time oracle machine, M, and a polynomial $p(\cdot)$ so that for infinitely many n's

$$|Prob[M^{F_n}(1^n) = 1] - Prob[M^{R_n}(1^n) = 1]| > \frac{1}{p(n)}$$

where $R = \{R_n\}_{n \in \mathbb{N}}$ is the uniform function ensemble. Let $t(\cdot)$ be a polynomial that bounds the running time of $M(1^n)$, and define $m(n) = t(n)n$.

For every n, and every $0 \leq j \leq \lceil \log n \rceil$ we define the hybrid distribution H_n^j. Loosely speaking, the computation of functions in H_n^j is a generalization of the tree-labeling procedure of the computation of functions in F_n. Here, we start by labeling the nodes at depth $\lceil \log n \rceil - j$.If d is the number of leaves in the subtree rooted at such a node, then we associate with this node a d-bit substring of the input. We also choose an indexed set of 2^d random strings of length n and associate it with this node . The d-bit substring of the input "chooses" a label for its corresponding node out of the set of 2^d strings associated with that node in the natural way. The label of an internal node of smaller depth is recursively determined from the labels of its children by applying the synthesizer to those values. And, as before, the value of the function is simply the label of the root.

To simplify our notations we shall only formalize this description of the hybrid distribution H_n^j for $n = 2^\ell$.

Definition 5.1 *For every sequence of possible outputs of $I_S(1^n)$, $\vec{k} = \{k_1, k_2, \ldots, k_{l-j}\}$, and for every $2^{2^j} 2^{l-j}$ long sequence of n-bit long strings, $\vec{a} = \{a_r^s : 1 \leq r \leq 2^{l-j}, s \in \{0,1\}^{2^j}\}$, we define the function $f_{\vec{a},\vec{k}} : \{0,1\}^n \mapsto \{0,1\}^n$. On input $x = x_1 \circ x_2 \ldots \circ x_{2^{l-j}}$, where $x_i \in \{0,1\}^{2^j}$ for $1 \leq i \leq 2^{l-j}$, the function outputs the single value in*

$$SQUEEZE_{S_{k_1}}(SQUEEZE_{S_{k_2}}(\ldots SQUEEZE_{S_{k_{l-j}}}(\{a_1^{x_1}, a_2^{x_2}, \ldots, a_{2^{l-j}}^{x_{2^{l-j}}}\})\ldots))$$

H_n^j is the random variable that assumes as values the functions $f_{\vec{a},\vec{k}}$ defined above, where the k_i's are independently distributed according to the distribution induced by I_S and $\vec{a}$ is independently distributed according to $U_n^{2^{2^j}2^{l-j}}$.

It is immediate from the definition (for $n = 2^\ell$) and obvious from the description (for other n's) that H_n^0 and F_n are identically distributed and that $H_n^{\lceil \log n \rceil}$ and R_n are identically distributed. Thus, by the assumption, we can distinguish between the extreme hybrid distributions. Therefore, applying the standard hybrid argument we can distinguish between neighboring hybrid distributions. We show how to use this to construct a distinguisher D for the pseudo-random synthesizers.

Definition 5.2 *We define the probabilistic polynomial-time algorithm D. For every n the input of D is an $m(n) \times m(n)$ matrix $B = (b_{i,j})$ whose entries are n-bit long strings. On this input D first uniformly choose $0 \leq J < \lceil \log n \rceil$ and then generates $\vec{k} = \{k_1, k_2, \ldots, k_{\lceil \log n \rceil - J - 1}\}$ by independent executions of I_S on input 1^n. D invokes M on input 1^n and answers its queries as a function $f_{\vec{a},\vec{k}} \in H_n^{J+1}$ would, where the strings of $\vec{a} = \{a_r^s : 1 \leq r \leq 2^{\lceil \log n \rceil - J - 1}, s \in \{0,1\}^{2^{J+1}}\}$ take values, upon necessity, from the entries of B. D outputs whatever M outputs. Again, for simplicity, we only define the way D chooses the entries of B for $n = 2^\ell$.*

On the query $x = x_1 \circ x_2 \ldots \circ x_{2^{l-J}}$, where $x_i \in \{0,1\}^{2^J}$ for $1 \leq i \leq 2^{l-J}$, D defines $a_i^{x_{2i-1} \circ x_{2i}}$ to be some entry $b_{u,v}$ of B for every $1 \leq i \leq 2^{l-J-1}$. D manage this by associating a row u with the couple (i, x_{2i-1}) and a column v with the couple (i, x_{2i}) and recording these associations. If a row was previously associated with (i, x_{2i-1}) we take this row again, otherwise, we associate with (i, x_{2i-1}) the next free row in B, and similarly for columns. Finally D answer the query with the single value in

$$SQUEEZE_{S_{k_1}}(\ldots SQUEEZE_{S_{k_{l-J-1}}}(\{a_1^{x_1 \circ x_2}, \ldots, a_{2^{l-J-1}}^{x_{2^{l-J-1}} \circ x_{2^{l-J}}}\}) \ldots)$$

Since M makes no more than $t(n)$ queries and for each query we need no more than n new entries we get that the size of B is sufficient for this process (because $m(n) = t(n)n$).

It is obvious that D is a polynomial-time algorithm, we now show that D is also a distinguisher for the pseudo-random synthesizers.

Claim 5.1 *If $B = {U_n}^{m(n) \times m(n)}$ then $Prob[D(B) = 1 | J = j] = Prob[M^{H_n^{j+1}}(1^n) = 1]$.*

Proof. The entries of B are independent and uniformly distributed. Thus, no matter which queries M submits, in which order they are submitted and how D associates the entries of B with the strings in $\vec{a}$ (as long as each entry is associated with only one string), we get that these strings are independent and uniformly distributed as well. Thus, the distribution of $D(B)$ would not have changed if instead of taking values from B, D would generate all the values in $\vec{a}$ uniformly at random and therefore the claim follows. □

Claim 5.2 *If $B = CONV_{S_{2n}^n}(Z, Y)$ for $Z = \{z_1 \ldots z_{m(n)}\}$ and $Y = \{y_1 \ldots y_{m(n)}\}$ independently drawn from ${U_n}^{m(n)}$ then $Prob[D(B) = 1 | J = j] = Prob[M^{H_n^j}(1^n) = 1]$.*

Proof. Consider the vector $\vec{a'} = \{a_i'^s : 1 \leq i \leq 2^{l-j}, s \in \{0,1\}^{2^j}\}$, and assume that whenever D associates a row u with the couple (i, x_{2i-1}) it also gives the string $a_{2i-1}'^{x_{2i-1}}$ the value z_u and whenever D associates a column v with the couple (i, x_{2i}) it also gives the string $a_{2i}'^{x_{2i}}$ the value y_v.

Let k be the random key in S^n_{2n} that was used to generate B. By the definition of algorithm D and of matrix B, for every query $x = x_1 \circ x_2 \ldots \circ x_{2^{l-j}}$, where $x_i \in \{0,1\}^{2^j}$ for $1 \leq i \leq 2^{l-j}$, D answer with the single value in

$$SQUEEZE_{S_{k_1}}(\ldots SQUEEZE_{S_{k_{l-j-1}}}(\{a_1^{x_1 \circ x_2}, \ldots, a_{2^{l-j-1}}^{x_{2^{l-j-1}} \circ x_{2^{l-j}}}\})\ldots) =$$

$$SQUEEZE_{S_{k_1}}(\ldots SQUEEZE_{S_{k_{l-j-1}}}(SQUEEZE_{S_k}(\{a'^{x_1}_1, a'^{x_2}_2, \ldots, a'^{x_{2^{l-j}}}_{2^{l-j}}\}))\ldots)$$

By similar arguments to the proof of the previous claim, the distribution of $D(B)$ would not have changed if instead of taking values from Z and Y, D would have independently draw all the values in $\vec{a'}$ from the uniform distribution and then answer queries using $\vec{a'}$, $\vec{k}$ and k according to the last formula. This fact, again, implies the claim. □

We can now conclude, by the standard hybrid argument that for infinitely many n's

$$|Prob[D(CONV_{S^n_{2n}}(X,Y)) = 1] - Prob[D(U_n{}^{m(n)\times m(n)}) = 1]|$$

$$= \frac{|Prob[M^{F_n}(1^n) = 1] - Prob[M^{R_n}(1^n) = 1]|}{\lceil \log n \rceil} > \frac{1}{p(n)\lceil \log n \rceil}$$

where X and Y are independently drawn from $U_n{}^{m(n)}$.

This contradicts the assumption that S is a collection of pseudo-random synthesizers, therefore it completes the proof of Theorem 5.1 that F is indeed an efficiently computable pseudo-random function ensemble. □

Corollary 5.2 *For every collection of pseudo-random synthesizers S such that its functions are in NC^i there exists an efficiently computable pseudo-random function ensemble F such that its functions are in NC^{i+1}. Furthermore, the parallel time complexity of the key-generating algorithms of S and F is the same.*

Proof. By Lemma 3.1 we can construct from S a new collection of pseudo-random synthesizers $S' = \{S'^n_{2n}\}_{n\in\mathbb{N}}$ in NC^i, by Theorem 5.1 we can construct from S' an efficiently computable pseudo-random function ensemble F in NC^{i+1}. Both constructions preserve the parallel time complexity of the key-generating algorithms. □

Keeping in mind the last corollary, we shift our focus to the parallel construction of pseudo-random synthesizers.

6 Construction of Pseudo-Random Synthesizers Based on Cryptographic Primitives

In the next two sections we show constructions of pseudo-random synthesizers. In this section we provide constructions of pseudo-random synthesizers from other cryptographic primitives: what we call weak pseudo-random functions and trapdoor permutations. In the next section we show constructions based on concrete intractability assumptions. We do not know of efficient parallel constructions of pseudo-random synthesizers from pseudo-random generators, or directly from one-way functions.

The reason that pseudo-random functions are hard to construct is that they must endure very powerful attacks. Their adversary (the distinguisher) may query their values at every point, and

may adapt his queries based on the answers it gets. We can weaken the strength of the opponent by letting him access only to a polynomial sample of random points and the value of the function at these points. We show that even the functions obtained under this condition simply defines pseudo-random synthesizers.

For every function f and every sequence $X = \{x_1 \dots x_k\}$ of values in the domain of f we denote by $< f, X >$ the sequence $\{x_1, f(x_1), x_2, f(x_2) \dots x_k, f(x_k)\}$.

Definition 6.1 (collection of weak pseudo-random functions) *An efficiently computable function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, is a* collection of weak pseudo-random functions *if for every probabilistic polynomial-time algorithm, D, every two polynomials $p(\cdot)$ and $m(\cdot)$, and all sufficiently large n's*

$$|Prob[D(< F_n, {U_n}^{m(n)} >) = 1] - Prob[D({U_n}^{2m(n)}) = 1]| < \frac{1}{p(n)}$$

Let F be a collection of weak pseudo-random functions, and let I be the polynomial-time key-generating algorithm for F. Assume that, on input 1^n, I only uses n random bits (actually, there is no need for this assumption and, furthermore, since I only needs polynomial many random bits they can be obtained by a pseudo-random generator). For every $r \in \{0,1\}^n$ we denote by $I(r)$ the value of $I(1^n)$ for r as the random bits. We show how to construct a pseudo-random synthesizer from F.

Lemma 6.1 *For F and I as before if we define $S : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ such that $S(x,y) = f_{I(y)}(x)$ for every $x, y \in \{0,1\}^n$, then S is a pseudo-random synthesizer.*

Proof. Assume, in contradiction to the lemma, that S is not a pseudo-random synthesizer. Since it is obvious that S is efficiently computable, there exists a probabilistic polynomial-time algorithm, D, and polynomials $p(\cdot)$ and $m(\cdot)$, such that for infinitely many n's

$$|Prob[D(CONV_S(X,Y)) = 1] - Prob[D(U_n^{m(n)\times m(n)}) = 1]| > \frac{1}{p(n)}$$

where X and Y are independently drawn from ${U_n}^{m(n)}$.

For every n and every $0 \le i \le m(n)$ we define the hybrid distribution H_n^i. The values H_n^i assumes are $m(n)\times m(n)$ matrices whose first i columns are distributed according to $CONV_S(X,Y)$ where X is drawn from ${U_n}^{m(n)}$ and Y is independently drawn from ${U_n}^{i}$. The last $m(n)-i$ columns are distributed according to $U_n^{m(n)\times(m(n)-i)}$.

We now construct a distinguisher D' for F. For input $\{x_1, z_1, x_2, z_2, \dots, x_n, z_n\}$, D' defines $X = \{x_1, \dots, x_n\}$ and $Z = \{z_1, \dots, z_n\}$ and uniformly chooses $0 < J \le m(n)$. D' then generates an $m(n)\times m(n)$ matrix B whose first $J-1$ columns are distributed according to $CONV_S(X,Y)$, where Y is drawn out of ${U_n}^{J-1}$. The J's column is Z^t and the last $m(n)-J$ columns are distributed according to $U_n^{m(n)\times(m(n)-J)}$. D' then outputs $D(B)$.

It is easy to verify that D' is indeed polynomial time algorithm, that

$$Prob[D'(< F_n, {U_n}^{m(n)} >) = 1 | J = j] = Prob[D(H_j) = 1]$$

and that

$$Prob[D'({U_n}^{2m(n)}) = 1 | J = j] = Prob[D(H_{j-1}) = 1]$$

Thus, by the standard hybrid argument, for infinitely many n's

$$|Prob[D'(< F_n, {U_n}^{m(n)} >) = 1] - Prob[D'({U_n}^{2m(n)}) = 1]| > \frac{1}{p(n)m(n)}$$

in contradiction to the assumption that F is a collection of weak pseudo-random functions. Therefore, we can conclude the lemma. □

Notice that the pseudo-random synthesizer we construct in the previous lemma is even more powerful then we need: For random X and Y the matrix $CONV_S(X,Y)$ cannot be efficiently distinguished from a random matrix *even* if we allow the distinguisher access to X.

Defining weak pseudo-random functions as length-preserving is an arbitrary choice. We might as well consider weak pseudo-random functions of different output length. The construction of the pseudo-random synthesizer does not change.

From the previous lemma, if there exist weak pseudo-random functions that can be sampled and evaluated in NC then we also have a pseudo-random synthesizer in NC and therefore by Construction 4.1 we also have pseudo-random functions that can be sampled and evaluated in NC.

As a direct result from the previous lemma we can get a construction of a pseudo-random synthesizer, out of a collection of trapdoor permutations, via a collection of weak pseudo-random functions. The pseudo-random synthesizer constructed is in NC if the trapdoor permutations can be sampled and inverted in NC. (The additional requirement of an efficient hard-core predicate is already guaranteed by [20]). Since we have no concrete example of this sort, we give only a brief and informal description of the construction.

Let $\{f_i : D_i \mapsto D_i\}$ be a collection of trapdoor one way permutations and let b_i be an hard-core predicate for f_i. Assume that the collection is one-way when the input of f_i is uniformly distributed over D_i. For every function f_i (with security parameter n) we define $g_i : \{0,1\}^n \mapsto \{0,1\}$. For every $x \in \{0,1\}^n$ let $D(i,x)$ denotes the element in D_i sampled with x as the random bits, then $g_i(x)$ is defined to be $b_i(f_i^{-1}(D(i,x)))$. We claim (without proof) that the collection $\{g_i\}$ (with the same distribution over the keys as this of the f_i's) is a collection of weak pseudo-random functions.

7 Construction of Pseudo-Random Synthesizers Based on Concrete Intractability Assumptions

In this section we present two NC^1 constructions of pseudo-random synthesizers based on concrete, frequently-used, intractability assumptions: the Diffie-Hellman and the RSA assumptions. We first address issues that are common to both the constructions.

The functions in both the collections of pseudo-random synthesizers we construct can be evaluated in NC^1, the key generating algorithms, though, are sequential. We make use of the fact that some operations, like exponentiation, can be done efficiently in parallel, given an additional preprocessed data. In this idea we follow the work of Kearns and Valiant, [29]; in their context, the additional data is "forced" into the input, whereas in our context it is added to the key.

The efficient parallel evaluation of the synthesizers is based on known results in parallel computation of arithmetic operations (see Karp and Ramachandran [28] for a review). In particular we use the result of Beame, Cook and Hoover, [5], that enables modular multiplication of n numbers of length n bits (among other operations) by log-depth circuits. The construction of these circuits can be easily done in the sequential preprocessing stage of the sampling.

The pseudo-random synthesizers we construct are Boolean functions. In section 3 we showed two methods to expand the output size of pseudo-random synthesizers. The pseudo-random generator of Blum, Blum and Shub [8] and the one by Hastad, Schrift and Shamir [22] are natural candidates for the pseudo-random generator needed by the method based on Theorem 3.2. Both generators can be made to work efficiently in parallel (after a sequential preprocessing stage). And the security of

both of them is based on a principal number-theoretical intractability assumption, the intractability of factoring integers (Blum integers in [22]).

This seems to be a good place to make the comparison between the two methods of expansion, discussed above, more concrete. Assume that we are using Boolean synthesizers (instead of synthesizers with linear output size) for Construction 4.1 of pseudo-random functions. The method of Lemma 3.1 forces us to replace the n-bit long strings with n^2-bit long strings (in order to preserve the security). We also need $O(n^3)$ applications of the synthesizers instead of $O(n)$. The parallel time complexity is unchanged. Using the method based on Theorem 3.2 we can preserve the length of the strings we use, and we need $O(n^2)$ applications of the synthesizers and of the generator. The increase in the parallel time complexity is logarithmic (for the generators and synthesizers discussed in this section). This illustrates the advantage of a direct construction of parallel synthesizers with linear output size (rather than constructions that go through any one of the methods).

We use, in our constructions, the result of Goldreich and Levin, [20]. They showed an hard-core predicate for "any" one-way function.

Theorem 7.1 *([20]) Let f be any one-way function. For every probabilistic polynomial-time algorithm, A, for every polynomial, $p(\cdot)$ and all sufficiently large n's $Prob[A(f(x),r) = r \cdot x] < \frac{1}{2} + \frac{1}{p(n)}$, where x and r are independently drawn out of U_n and $r \cdot x$ denotes the inner product* (mod2) *of r and x.*

We use their result in a slightly different context, loosely speaking, if it is hard to compute $g(x)$, given $f(x)$, then it is also hard to guess $g(x) \cdot r$. We also use the next-bit prediction tests of Blum and Micali [10], the equivalence between pseudo-random ensembles and ensembles that pass all polynomial time next-bit tests was shown by Yao [46].

7.1 The Diffie-Hellman Assumption

In this subsection we define a collection of pseudo-random synthesizers based on the Diffie-Hellman assumption. For concreteness, we state the Diffie-Hellman assumption in terms of the group $\mathbb{Z}_P^*$, but, our construction works, just as well, for other groups.

Assumption 7.1 *(Diffie-Hellman [14]) For every probabilistic polynomial-time algorithm, A, for every polynomial, $q(\cdot)$ and all sufficiently large n's*

$$Prob[A(P, g, g^a \bmod P, g^b \bmod P) = g^{ab} \bmod P] < \frac{1}{q(n)}$$

where P is an n-bit prime, g a uniformly selected generator of $\mathbb{Z}_P^*$ and a and b chosen according to U_n.

We assumed nothing on the distribution of P, and, in a sense, we might consider P to be fixed for every output size. Note that the Diffie-Hellman assumption requires that The discrete log mod P be hard. The two problems are not known to be equivalent, but Maurer [35] showed an equivalence in a non-uniform model.

Definition 7.1 *For every n bit prime P, every generator, g, of $\mathbb{Z}_P^*$ and every $r \in \{0,1\}^n$ define $S_{P,g,r}(x,y) \stackrel{\text{def}}{=} (g^{xy} \bmod P) \cdot r$ for all $x, y \in \{0,1\}^n$. We define S_{2n}^1 to be the random variable that assume as values the functions $S_{P,g,r}$, where r is uniformly distributed over $\{0,1\}^n$ and g a uniformly selected generator of $\mathbb{Z}_P^*$. The distribution of P might be any samplable distribution we consider hard for the Diffie-Hellman assumption (for instance the uniform distribution). The function ensemble S_{DH} is defined to be $\{S_{2n}^1\}_{n \in \mathbb{N}}$.*

Lemma 7.2 *Given that the Diffie-Hellman assumption holds, S_{DH} is a collection of pseudo-random synthesizers.*

Proof. Assume that $S_{DH} = \{S^1_{2n}\}_{n\in\mathbb{N}}$ is not a collection of pseudo-random synthesizers. Since S_{DH} is efficiently computable, there exists a polynomial $m(\cdot)$ such that the ensemble $\{E_n\} = CONV_{S^1_{2n}}(X,Y)$, where X and Y are independently drawn from ${U_n}^{m(n)}$, is not pseudo-random. Thus, there exists an efficient next-bit prediction test T and a polynomial $q(\cdot)$ such that, for infinitely many n's, T succeeds to predict the next bit of a prefix (of uniformly chosen length) of $\{E_n\}$ with probability greater then $\frac{1}{2} + \frac{1}{q(n)}$.

Given P, g, $g^a \bmod P$ and $g^b \bmod P$ as in the Diffie-Hellman assumption and a uniformly chosen $r \in \{0,1\}^n$ we show below how to efficiently guess $(g^{ab} \bmod P)\cdot r$ with probability greater then $\frac{1}{2} + \frac{1}{q(n)}$ (for infinitely many n's). By Theorem 7.1, $g^{ab} \bmod P$ can be efficiently computed with non-negligible success probability in contradiction to the Diffie-Hellman assumption.

The guessing algorithm is extremely simple. Uniformly choose $1 \leq i,j \leq m(n)$. Define $X = x_1,\ldots,x_{m(n)}$ and $Y = y_1,\ldots,y_{m(n)}$ by setting $x_i = a$, $y_j = b$ (without actually knowing this values) and independently drawing all other values from U_n. Now feed T with the bits of $CONV_{S_{P,g,r}}(X,Y)$ until the entry (i,j) is reached (the algorithm can easily compute all $S_{P,g,r}(x_s,y_t)$ other than $S_{P,g,r}(x_i,y_j)$). Since the distribution of $CONV_{S_{P,g,r}}(X,Y)$, where P, g, r, X and Y are distributed as defined above, is exactly $\{E_n\}$ we get that T predicts $S_{P,g,r}(x_i,y_j)$ with probability greater then $\frac{1}{2} + \frac{1}{q(n)}$ (for infinitely many n's). As mentioned above, this contradicts the Diffie-Hellman assumption, thus, it proves the lemma. □

Corollary 7.3 *Given that the Diffie-Hellman assumption holds, there exist pseudo-random functions in NC^2.*

Proof. By the last lemma, given that the Diffie-Hellman assumption holds, S_{DH} is a collection of pseudo-random synthesizers. If the key-generating algorithm precomputes $g^{2^i} \bmod P$ for $1 \leq i \leq n$ then the functions of S_{DH} can be evaluated in NC^1. By Corollary 5.2, there exist pseudo-random functions in NC^2 (the key generating algorithms in both cases are sequential). Note that if we take a fixed P for every input size then S_{DH} is actually a synthesizer, rather then a collection of synthesizers. At that case the key-generating algorithm of the pseudo-random functions obtained is in "non-uniform" NC. □

7.2 The RSA Assumption

In this subsection we assume that the, extremely popular, RSA collection of functions, of Rivest, Shamir and Adleman [43], is indeed one-way.

Assumption 7.2 *(RSA [43]) For every probabilistic polynomial-time algorithm, A, for every polynomial, $q(\cdot)$ and all sufficiently large n's $Prob[A(N,e,m^e \bmod N) = m] < \frac{1}{q(n)}$, where $N = PQ$ is an n-bit integer, P and Q are two uniformly selected primes such that $|P| = |Q|$, e an n-bit prime (thus, e is larger than P and Q) and m uniformly chosen from $\mathbb{Z}^*_N$.*

We showed in section 6 a general construction of pseudo-random synthesizers out of trapdoor one-way permutations. Nevertheless, in the case of the RSA-functions we have to adjust this construction in order to get efficient parallel synthesizers. To enable preprocessing (and, thus, achieve the desired efficiency) we employ in our construction the subset product function. Let G be a finite group, for every n-tuple $\vec{y} = \{y_1,\ldots,y_n\}$ of elements in G and n-bit string $x = x_1\ldots x_n$

define $SP_{G,\vec{y}}(x)$ to be the product in G of the elements y_i such that $x_i = 1$. We use the following lemma that was shown by Impagliazzo and Naor in [26], and is based on the leftover hash lemma of [25, 27].

Lemma 7.4 *([26]) Let G be a finite group, $n > c \log|G|$ and $c > 1$. Then, for all but an exponentially small fraction of the choices of $\vec{y} \in (G)^n$, the induced distribution $SP_{G,\vec{y}}(U_n)$ is statistically indistinguishable within an exponentially small amount from the uniform distribution over G.*

We can now define the pseudo-random synthesizers.

Definition 7.2 *Let N be an n-bit integer $N = PQ$ such that P and Q are two primes, $|P| = |Q|$. Then for every $\vec{g} = \{g_1, \ldots, g_{2n}\} \in (\mathbb{Z}_N^*)^{2n}$ and every $r \in \{0,1\}^n$ we define $S_{N,\vec{g},r}(x,y) \stackrel{\text{def}}{=} ((g_x)^y \bmod N) \cdot r$, where $x, y \in \{0,1\}^{2n}$ and $g_x = SP_{\mathbb{Z}_N^*,\vec{g}}(x)$. We define S_{4n}^1 to be the random variable with value $S_{N,\vec{g},r}$, where N, $\vec{g}$ and r are uniformly distributed in the set of values they may assume. The function ensemble S_{RSA} is defined to be $\{S_{4n}^1\}_{n\in\mathbb{N}}$.*

Lemma 7.5 *Given that the RSA assumption holds, S_{RSA} is a collection of pseudo-random synthesizers.*

Proof. Assume that $S_{RSA} = \{S_{4n}^1\}_{n\in\mathbb{N}}$ is not a collection of pseudo-random synthesizers. Since S_{RSA} is efficiently computable, there exists a polynomial $m(\cdot)$ such that the ensemble $\{E_n\} = CONV_{S_{4n}^1}(X,Y)$, where X and Y are independently drawn from ${U_{2n}}^{m(n)}$, is not pseudo-random. Thus, there exists an efficient next-bit prediction test T and a polynomial $q(\cdot)$ such that, for infinitely many n's, T succeeds to predict the next bit of a prefix (of uniformly chosen length) of $\{E_n\}$ with probability greater then $\frac{1}{2} + \frac{1}{q(n)}$.

Let N, $m^e \bmod N$ and e be as in the RSA assumption. Following Shamir [44] we notice that given any z such that $\gcd(e,z) = 1$ and given $m^z \bmod N$ we can compute m. The reason is that if $gcd(e,z) = 1$ we can compute $a, b \in \mathbb{Z}$ such that $ae + bz = 1$, thus, we can compute $m = (m^e)^a (m^z)^b \bmod N$. We show below how to guess $(m^z \bmod N) \cdot r$, for r drawn from U_n and some z, with probability $\frac{1}{2} + \frac{1}{2q(n)}$, thus, by Theorem 7.1 we can efficiently compute m with non-negligible success probability in contradiction to the RSA-assumption.

The guessing algorithm uniformly chooses $1 \leq i,j \leq m(n)$ and $r \in \{0,1\}^n$. It defines the $m(n) \times m(n)$ matrix B where $b_{s,t} = (((m_s)^e)^{d_t} \bmod N) \cdot r$ and the choices of the m_s's and d_t's are as follows: For $s \neq i$, m_s is uniformly chosen from $\mathbb{Z}_N^*$, m_i is defined to be m. For $t \neq j$, d_t is drawn from U_{2n}, d_j is defined to be $\frac{z}{e} \bmod \varphi(N)$, where z is uniformly distributed over the set of $2n$-bit strings that are relatively primes to e. Notice, that although the algorithm do not know m_i and d_j it can still computes all values of B apart to $b_{i,j}$. We shall show that the distribution of B is statistically indistinguishable within an exponentially small amount from the distribution of E_n. Thus, if we feed T with the bits of B until the entry (i,j) it predicts $b_{i,j} = (m^z \bmod N) \cdot r$ with probability greater than, say, $\frac{1}{2} + \frac{1}{2q(n)}$.

In order to complete the proof all we have to show is that B is indeed statistically indistinguishable from E_n. Since e is relatively prime to $\varphi(N)$ (because it is a large prime) and for every s, m_s is uniformly distributed over $\mathbb{Z}_N^*$, we get that $(m_s)^e$ is also uniformly distributed over $\mathbb{Z}_N^*$. By Lemma 7.4, we have that the distribution of $(m_s)^e$ is statistically close to the distribution of $g_x = SP_{\mathbb{Z}_N^*,\vec{g}}(x)$ for uniformly chosen $x \in \{0,1\}^{2n}$ and $\vec{g} = \{g_1, \ldots, g_{2n}\} \in (\mathbb{Z}_N^*)^{2n}$. Notice also that for z that is chosen from U_{2n} the distribution of $\frac{z}{e} \bmod \varphi(N)$ and $U_{2n} \bmod \varphi(N)$ is statistically close. Since e is a large prime, even after restricting the z's to be relatively primes to e, the distributions are close. Given these two observations it is obvious that the distribution of B

is statistically indistinguishable within an exponentially small amount from the distribution of E_n, and the contradiction to the RSA-assumption follows. □

Corollary 7.6 *Given that the RSA assumption holds, there exist pseudo-random functions in NC^2.*

Proof. By the last lemma, given that the RSA assumption holds, S_{RSA} is a collection of pseudo-random synthesizers. If the key-generating algorithm precomputes $(g_i)^{2^j} \bmod N$ for $1 \leq i, j \leq 2n$ then the functions of S_{RSA} can be evaluated in NC^1. By Corollary 5.2, there exist pseudo-random functions in NC^2. □

An alternative construction based on RSA. We have shown that breaking S_{RSA} is computationally equivalent to taking the eth root $\bmod N$, where e is a large prime and N the product of two large primes. We now show an alternative construction such that breaking it is equivalent to taking the eth root for *any* e (in $\mathbb{Z}^*_{\varphi(N)}$). Nevertheless, this construction somewhat limits the set of "good" Ns.

We first notice that if $\varphi(N)$ has no "small" odd factors (say, smaller than n^2) then $2n$ random odd-values have non-negligible chance to be all in $\mathbb{Z}^*_{\varphi(N)}$. Thus, by Lemma 7.4 we may assume that the algorithm, trying to extract roots, can "almost uniformly" sample $\mathbb{Z}^*_{\varphi(N)}$. Sieve theory shows that the set of such Ns is not too sparse. For example, denote by $B(x)$ the number of primes p smaller than x such that $(p-1)/2$ is the product of two primes each larger than $p^{1/4}$. There exists a positive constant c such that $B(x) \geq \frac{cx}{\log^2 x}$. See [41] for several results of this kind (which are more than sufficient for our purpose). As a result we get that (a) If the RSA-assumption holds it also holds when the set of Ns is restricted as specified above. (b) The restricted set of N's can be efficiently sampled (using Bach's algorithm [4]).

For this construction we use the least-significant bit (LSB) instead of GL hard-bit. Alexi et. al. [1] showed that LSB is an hard-bit for RSA.

Definition 7.3 *Let N be an n-bit integer $N = PQ$ such that P and Q are two primes, $|P| = |Q|$ and $\varphi(N)$ has no odd factors smaller than n^2. For every $\vec{g} = \{g_1, \ldots, g_{2n}\} \in (\mathbb{Z}^*_N)^{2n}$ and every $\vec{d} = \{d_1, \ldots, d_{2n}\} \in (\mathbb{Z}^*_{\varphi(N)})^{2n}$ we define $S_{N,\vec{g},\vec{d}}(x,y) \stackrel{\text{def}}{=} LSB((g_x)^{d_y} \bmod N)$, where $x, y \in \{0,1\}^{2n}$, $g_x = SP_{\mathbb{Z}^*_N,\vec{g}}(x)$ and $d_y = SP_{\mathbb{Z}^*_{\varphi(N)},\vec{d}}(y)$. We define S^1_{4n} to be the random variable with value $S_{N,\vec{g},\vec{d}}$, where N, $\vec{g}$ and $\vec{d}$ are uniformly distributed in the set of values they may assume. The function ensemble $S_{RSA'}$ is defined to be $\{S^1_{4n}\}_{n\in\mathbb{N}}$.*

Lemma 7.7 *Given that the RSA assumption holds, $S_{RSA'}$ is a collection of pseudo-random synthesizers.*

Proof. Assume that $S_{RSA'} = \{S^1_{4n}\}_{n\in\mathbb{N}}$ is not a collection of pseudo-random synthesizers. Since $S_{RSA'}$ is efficiently computable there exists a polynomial $m(\cdot)$ such that the ensemble $\{E_n\} = CONV_{S^1_{4n}}(X,Y)$, where X and Y are independently drawn from ${U_{2n}}^{m(n)}$, is not pseudo-random. Thus, there exists an efficient next-bit prediction test T and a polynomial $q(\cdot)$ such that, for infinitely many n's, T succeeds to predict the next bit of a prefix (of uniformly chosen length) of $\{E_n\}$ with probability greater then $\frac{1}{2} + \frac{1}{q(n)}$.

Let N be a random composite as in the definition of $S_{RSA'}$, e a random value in $\mathbb{Z}^*_{\varphi(N)}$ and m random value in $\mathbb{Z}^*_N$. We define an algorithm that takes as input N, e and $m^e \bmod N$ and tries to guess $LSB(m)$.

The guessing algorithm uniformly chooses $1 \leq i, j \leq m(n)$ It defines the $m(n) \times m(n)$ matrix B where $b_{s,t} = LSB(((m_s)^e)^{d_t} \bmod N)$ and the choices of the m_s's and d_t's are as follows: For $s \neq i$,

m_s is uniformly chosen from $\mathbb{Z}_N^*$, m_i is defined to be m. For $t \neq j$, d_t is "almost uniformly" sampled from $\mathbb{Z}_{\varphi(N)}^*$, d_j is defined to be $\frac{1}{e} \bmod \varphi(N)$. The algorithm feeds T with the bits of B until the entry (i,j) and outputs T's prediction of $b_{i,j} = LSB(m)$. The non-negligible success probability of the algorithm and the contradiction to the RSA-assumption follows by similar arguments to those in the proof of Lemma 7.5.

If e is *any* value in $\mathbb{Z}_{\varphi(N)}^*$ (not necessarily random) we can still guess $LSB(m)$ - we just give the guessing algorithm N, $e \times d$ and $m^{e \times d} \bmod N$ where d is a random value in $\mathbb{Z}_{\varphi(N)}^*$. □

Since Alexi et. al. [1] showed that the $\log n$ least-significant bits are simultaneously hard for RSA we can adjust the functions in $S_{RSA'}$ to output $\log n$ bits. If we make a stronger assumption (not proven to be equivalent to the RSA-assumption), that $\Omega(n)$ bits are simultaneously hard for RSA, we get a direct construction of pseudo-random synthesizers with linear output size.

Note that all the pseudo-random synthesizers constructed in this section may allow their key to be public. This means that we can use a single synthesizer at all levels of the computations of the pseudo-random functions in Construction 4.1.

8 Pseudo-Random Synthesizers and Hard Learning Problems

In this section we discuss several aspects of the connection between pseudo-random synthesizers and hard-to-learn functions.

Blum, Furst, Kearns and Lipton [7], show how to construct several cryptographic primitives out of *hard-to-learn* functions, in a way that preserves the degree of parallelism of the functions. A major motivation for presenting such constructions is the simplicity of function classes that are believed to be hard for efficient learning. We show that, under the definitions of [7], pseudo-random synthesizers can easily be constructed from distributions on functions that are hard to learn. Thus, by the constructions showed in this paper we can add to the cryptographic primitives, constructed in [7], constructions of pseudo-random functions and of pseudo-random generators with large expansion ratio (without assuming, as in [7], that the functions are hard for learning when membership queries are allowed).

There is a difference between standard learning-theory definitions and standard cryptographic definitions. Loosely speaking, a collection of concepts is hard to learn if for every efficient algorithm there exists a distribution over the concepts that is hard for this specific algorithm to learn. In cryptographic settings the order of quantifiers is reversed: the hard distribution should be hard for *every* efficient algorithm. In order for hard-learning problems to be useful in cryptographic settings an average-case learning model is presented in [7].

Informally describing one of the definitions in [7], we can say that a distribution ensemble of functions, $F = \{F_n\}_{\in \mathbb{N}}$, is not *weakly predictable on the average* with respect to a distribution on the inputs D, if no efficient algorithm can predict $f(\tilde{x})$ with probability $\frac{1}{2} + \frac{1}{poly(n)}$, given $\tilde{x}$ and a polynomial sequence $\{< x_i, f(x_i) >\}$, where $f \in_R F_n$ and the inputs are independently distributed according to D.

An immediate observation is that a distribution ensemble of functions, F, is not weakly predictable on the average with respect to the uniform distribution if and only if it is a collection of weak pseudo-random functions. Thus by Lemma 6.1 such a distributions defines a pseudo-random synthesizer S: $S(x,y)$ is simply $f(x)$ where f is sampled from F using the bits of y. Using S we can construct pseudo-random generators and pseudo-random functions. Moreover, by Lemma 3.1. the pseudo-random generator we construct may have large expansion ratio ($n^{1-\epsilon}$ for every $\epsilon > 0$).

The pseudo-random generator constructed in [7] under the same assumptions has expansion ratio bounded by $1 + 1/n$.

Efficient synthesizer from a concrete hard-to-learn problem: Consider the following distribution on functions with parameters k and n. Select at random two disjoint sets $A, B \subset \{1, \ldots, n\}$ each of size k. Given input $x \in \{0,1\}^n$ compute the parity of the bits indexed by A and the majority of the bits indexed by B; Output the exclusive-or of these values. Blum et. al. [7], estimate that these functions, for $k = \log n$, cannot be weakly predictable without using "profoundly" new ideas. If this distribution of functions is not weakly predictable on the average, with respect to the uniform distribution, then it defines an extremely efficient synthesizer. Therefore, using the constructions of this paper, we get efficient parallel pseudo-random functions.

8.1 Pseudo-Random Functions in NC^1

Linial, Mansour and Nisan [32] show that there are no pseudo-random functions in AC^0 with security better than $n^{polylog(n)}$. Kharitonov [30] showed that after preprocessing, a polynomial size pseudo-random bit sequence (based on [8]) can be produced in NC^1 (the length of the sequence can stay undetermined at the preprocessing stage). Regarding these results, one may ask, are there pseudo-random functions in NC^1. We show that, under strong enough assumptions, our constructions yield a positive answer.

Note that, unlike the rest of the paper, the reduction in this section, from pseudo-random functions in NC^2 to pseudo-random functions in NC^1 substantially reduces the security of the functions.

Let $F = \{F_n\}_{n \in \mathbb{N}}$ be a pseudo-random function ensemble such that its functions are computable in NC^2. Assume also that no distinguisher with running time $n^{O(\log n)}$ can distinguish between F and the uniform function ensemble with success probability $n^{-\Omega(\log n)}$, we construct the ensemble $G = \{G_n\}_{\in \mathbb{N}}$. Using Levin's idea [31] (mentioned in the Introduction) the functions in G_n hash (with a secret hash function) n-bit long strings to $2\sqrt{\log n}$-bit long strings and apply the functions in $F_{2\sqrt{\log n}}$. The functions of G can be computed in NC^1, furthermore, as long as no two strings are hashed to the same value, the success probability of any distinguisher algorithm for distinguishing G_n and the uniform distribution, is not greater than for distinguishing $F_{2\sqrt{\log n}}$ and the uniform distribution. Since any polynomial-time distinguisher algorithm has probability $2^{-\Omega(2\sqrt{\log n})}$ to cause collisions for the hash function, we can conclude from the assumptions on F that no such algorithm can distinguish between G and the uniform functions ensemble with success probability $\frac{1}{poly(n)}$. Thus, G is a pseudo-random function ensemble in NC^1.

Notice that all the constructions in the paper (apart from the last one) are "security-preserving". In particular if we define the next property for cryptographic primitives: "no algorithm with running time $n^{O(\log n)}$ can break this primitive with success probability $n^{-\Omega(\log n)}$", then our constructions preserve this property. Thus, if for example, this property holds for the Diffie-Hellman assumption, we get a construction of pseudo-random functions in NC^1.

Note that, if we make stronger assumptions on the security of F, we can conclude stronger estimates for the security of G. The only upper bound on the security of G that holds regardless of the security of F is $2^{O(2\sqrt{\log n})}$. For example, if we assume that F has security 2^{n^ϵ} for some $\epsilon > 0$ we get that G has security of $2^{O(2^{\epsilon}\sqrt{\log n})}$.

9 Further Research

In Sections 6-8 we discuss the existence of pseudo-random synthesizers in NC. Additional work should be done in this area. The most obvious question is what are the general assumptions (in cryptography or in other fields) that imply the existence of pseudo-random synthesizers in NC. In particular, whether there exist parallel constructions of pseudo-random synthesizers out of pseudo-random generators or directly from one-way functions.

It is also of interest to find parallel constructions of pseudo-random synthesizers based on other concrete intractability assumptions. A task of practical importance is to derive more efficient concrete constructions of pseudo-random synthesizers in order to get efficient constructions of pseudo-random functions. As discussed in Section 7 direct constructions of synthesizers with linear output size will make an important contribution to the efficiency of the pseudo-random functions we construct.

An extensive research field deals with pseudo-random generators that "fool" algorithms performing space-bounded computations. This kind of generators can be constructed without any (unproven) assumptions; see [3, 36, 37, 39] for definitions, constructions and applications. It is possible that the concept of pseudo-random synthesizers and the idea of our construction can be applied to the "world" of space-bounded computations. As a motivation remark, note that the construction in [36] bares some resemblance to the GGM construction.

Let $IP(x,y)$ be the inner product of x and y (mod2) and let X and Y be random m-long sequences of n-bit strings. For some constant $0 < \alpha < 1$ and $s = \alpha n$ it can be shown that $CONV_{IP}(X,Y)$ is a pseudo-random generator for $SPACE(s)$ with parameter $\epsilon = 2^{-\Omega(s)}m^2$ (when $CONV_{IP}(X,Y)$ is given row by row). So, in some sense we can think of IP as pseudo-random synthesizer for space bounded computation. The only fact we use is that approximating IP is "hard" in the communication complexity model (see [13, 45]).

One might also try to apply the concept of pseudo-random synthesizers for other classes of algorithms. For example [2, 38] construct pseudo-random generators for polynomial-size constant-depth circuits.

Our primary motivation for introducing pseudo-random synthesizers is the parallel construction of pseudo-random functions. The special characteristics of pseudo-random synthesizers lead us to believe that other desired applications may exist. For instance, pseudo-random synthesizers easily define a pseudo-random generator with large output size and the ability to directly compute subsequences of the output. This suggests that pseudo-random synthesizers might be useful for software implementations of pseudo-random generators (or functions). Another possible application of our construction of pseudo-random functions based on synthesizers is to convert encryption methods that are not immune to chosen plaintext attacks into ones that are immune.

Acknowledgments

We thank Jon Sorenson who brought [41] to our attention.

References

[1] Alexi, W.B., B. Chor, O. Goldreich and C.P. Schnorr, RSA and Rabin functions: certain parts are as hard as the whole, *SIAM J. Comput.* 17(2) (1988) 194-209.

[2] Ajtai M. and A. Wigderson, Deterministic simulations of probabilistic constant depth circuits, *Proc. 26th Symp. on Foundations of Computer Science* (1985) 11-19.

[3] Babai, L., N. Nisan and M. Szegedy, Multiparty protocols, pseudorandom generators for logspace, and time-space tradeoffs *J. Comput. System Sci.* 45(2) (1992) 204-232.

[4] Bach, E., How to generate factored random numbers, *SIAM J. Comput.* 17(2) (1988) 179-193.

[5] Beame, P.W., S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems, *SIAM J. Comput.* 15 (1986) 994-1003.

[6] Bellare, M. and S. Goldwasser, New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs, in: G. Brassard, ed., *Advances in Cryptology - CRYPTO 89*, Lecture Notes in Computer Science, vol. 435, Springer-Verlag (1990) 194-211.

[7] Blum, A., M. Furst, M. Kearns and R.J. Lipton, Cryptographic primitives based on hard learning problems, in: D.R. Stinson, ed., *Advances in Cryptology, Proc. - CRYPTO 93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag (1994) 278-291.

[8] Blum, L., M. Blum and M. Shub, A simple secure unpredictable pseudo-random number generator, *SIAM J. Comput.* 15 (1986) 364-383.

[9] Blum, M., W. Evans, P. Gemmell, S. Kannan, M. Naor, Checking the correctness of memories, *Proc. 31st Symp. on Foundations of Computer Science* (October 1990). Full version: Algorithmica (1994) 225–244.

[10] Blum, M. and S. Micali, How to generate cryptographically strong sequence of pseudo-random bits, *SIAM J. Comput.* 13 (1984) 850-864.

[11] Brassard, G., **Modern cryptology**, Lecture Notes in Computer Science vol. 325, Springer-Verlag, 1988.

[12] Chor B., A. Fiat and M. Naor, Tracing traitors, *Advances in Cryptology - CRYPTO' 94*, Springer-Verlag (1994) 257–270.

[13] Chor B. and Goldreich, O., Unbiased bits from sources of weak randomness and probabilistic communication complexity, *Proc. 26th IEEE Symp. on Foundations of Computer Science* (1985) 429-442.

[14] Diffie, W. and M. Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory* 22(6) (1976) 644-654.

[15] Goldreich, O., **Foundations of Cryptography** (Fragments of a Book) 1995. Electronic publication: http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html (Electronic Colloquium on Computational Complexity).

[16] Goldreich, O., Towards a theory of software protection, *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 182-194.

[17] Goldreich, O., Two remarks concerning the Goldwasser-Micali-Rivest signature scheme, in: *Advances in Cryptology - CRYPTO' 86*, Lecture Notes in Computer Science, vol. 263, Springer-Verlag (1987) 104-110.

[18] Goldreich, O., S. Goldwasser and S. Micali, How to construct random functions, *J. of the ACM.* 33 (1986) 792-807.

[19] Goldreich, O., S. Goldwasser and S. Micali, On the cryptographic applications of random functions, *Advances in Cryptology - CRYPTO' 84*, Lecture Notes in Computer Science, vol. 196, Springer-Verlag (1985) 276-288.

[20] Goldreich, O. and L. Levin, A hard-core predicate for all one-way functions, in: *Proc. 21st Ann. ACM Symp. on Theory of Computing* (1989) 25-32.

[21] Goldwasser, S. and S. Micali, Probabilistic encryption, *J. Comput. System Sci.* 28(2) (1984) 270-299.

[22] Hastad, J., A.W. Schrift and A. Shamir, The discrete logarithm modulo a composite hides $O(n)$ bits, *J. of Computer and System Sciences* 47 (1993) 376-404.

[23] Herzberg A. and M. Luby, Public randomness in cryptography, *Advances in Cryptology - CRYPTO 92*, Lecture Notes in Computer Science, vol. 740, Springer-Verlag (1992) 421–432.

[24] Impagliazzo, R. and L. Levin, No better ways to generate hard NP instances than picking uniformly at random, *Proc. 31st IEEE Symp. on Foundations of Computer Science* (October, 1990) 812-821.

[25] Impagliazzo, R., L. Levin and M. Luby, Pseudo-random generation from one-way functions, *Proc. 21st Symposium on Theory of Computing* (1989) 12-24.

[26] Impagliazzo, R. and M. Naor, Efficient cryptographic schemes provably as secure as subset sum, Proc. of the 30th Symp. on Foundations of Computer Science (1989) 236–241. Full version: Technical Report CS93-12, Weizmann Institute, 1993.

[27] Impagliazzo, R. and D. Zuckermann, Recycling random bits, *Proc. 30th IEEE Symposium on Foundations of Computer Science* (1989) 248-253.

[28] Karp, R.M. and V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. van Leeuwen, ed., **Handbook of Theoretical Computer Science, vol.A** MIT Press (1990) 869-941.

[29] Kearns, M. and L. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata, *J. of the ACM.* 41(1) (Jan, 1994) 67-95.

[30] Kharitonov, M., Cryptographic hardness of distribution-specific learning, in: *Proc. 25th ACM Symp. on Theory of Computing* (May, 1993) 372-381.

[31] Levin, L. A., One-way function and pseudorandom generators, *Proc. 17th Ann. ACM Symp. on Theory of Computing* (May 1985) 363-365.

[32] Linial, N., Y. Mansour and N. Nisan, Constant depth circuits, Fourier transform, and learnability, *J. of the ACM.* 40(3) (Jul, 1993) 607-620.

[33] Luby M., **Pseudo-randomness and applications**, Princeton University Press, To appear.

[34] Luby, M. and C. Rackoff, How to construct pseudorandom permutations and pseudorandom functions, *SIAM J. Comput.* 17 (Apr, 1988) 373-386.

[35] U. Maurer, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, *Advances in Cryptology - CRYPTO 94*, Lecture Notes in Computer Science, vol. 740, Springer-Verlag (1994) 271–281.

[36] Nisan, N., Pseudorandom generators for space-bounded computation, *Combinatorica* 12(4) (1992) 449-461.

[37] Nisan, N., $RL \subseteq SC$, *Proc. 24th Ann. ACM Symp. on Theory of Computing* (1992) 619-623.

[38] Nisan, N. and A. Wigderson, Hardness vs. randomness *Proc. 29th IEEE Symp. on Foundations of Computer Science* (1988) 2-12.

[39] Nisan, N. and D. Zuckerman, Randomness is Linear in Space. Preliminary version: More deterministic simulations in logspace *Proc. 25th Ann. ACM Symp. on Theory of Computing* (1993) 235-244.

[40] Ostrovsky, R., An efficient software protection scheme, *Proc. 22nd Ann. ACM Symp. on Theory of Computing* (1990) 514-523.

[41] Ram Murty, M., Artin's conjecture for primitive roots, *The Mathematical Intelligencer* 10(4) Springer-Verlag (1988) 59-67.

[42] Razborov A. and S. Rudich, Natural proofs, *Proc. 26th Ann. ACM Symp. on Theory of Computing* (1994) 204-213.

[43] Rivest, R.L., A. Shamir, and L.M. Adleman, A method for obtaining digital signature and public key cryptosystems, *Comm. ACM* 21 (1978) 120-126.

[44] Shamir, A., On the generation of cryptographically strong pseudo-random number sequences, *ACM Trans. Comput. Sys.* (1983) 38–44.

[45] Vazirani U.V., Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources *Proc. 17th Ann. ACM Symp. on Theory of Computing* (1985) 366-378.

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