Eliminating the sequential nature in the construction of secure pseudo-random generators

A.M. Kourah and A.A. Belal
Computer Science of Dept., Faculty of Eng., Alexandria University, Alexandria, Egypt
ali_maher@acm.org and abelal@aast.edu

Other work showed how to construct a Pseudo-Random Generator (PRG), from any one-way function using the theory of hardcore predicates. This construction is generic and considered as the mapping between one-way functions and PRGs. This construction has two main inefficiencies. First, we can generate only a few number of bits per each computation of the one-way function. Second, we cannot generate the j^{th} block of pseudo-random bits without generating all the j-1 previous blocks. This means that the constructed generators are not parallelizable. In this paper, we propose a new construction method of PRGs using a combination of a one-way function and a simple deterministic sequence. This method results in PRGs that are fully-parallelizable, i.e., the cost of generating the i^{th} and j^{th} blocks are the same for all i and j. This method also generates a large number of bits per each computation of the underlying one-way function. We also put conditions for the combinations of one-way functions and deterministic sequences to result in provably secure PRGs. Of course, not all combinations satisfy these conditions. Hence, this construction is not intended to replace the original one. It is intended to construct fully-parallelizable PRGs. Searching for combinations of one-way functions and deterministic sequences satisfying the stated conditions is not an easy job. We examined a lot of cases. Some of them are provably secure PRGs. We present constructions based on block ciphers and secure hashing as examples for these cases. Other cases are totally insecure. Other cases are conjectured to be secure PRGs. We present constructions based on RSA and the subset sum problem as examples of conjectured secure PRGs.

إن استخدام السلاسل العشوائية في إجراء العمليات و الخوارزميات له استخدامات عديدة و تطبيقات متنوعة، ولكن الحصول على هذه السلاسل عن طريق بعض الظواهر الطبيعية التي تنتج العشوائية يعد أمرا بالغ الصعوبة هذا لكون هذه الظواهر نادرة التواجد و إن وجدت فإنها تنتج سلاسل متحيزة و مترابطة. هذا ما دعا الباحثين للبحث عن وسائل مختلفة للحد من استخدام الظواهر الطبيعية كمصدر للعشوائية للخوارزميات و يعد أهم ما توصل إليه الباحثين هو مولدات الأعداد العشوائية و هي كيانات تتلقى سلاسل عشوائية قصيرة و تولد سلاسل أكبر تستطيع أن توحي لمن ينظر أليها أنها عشوائية. يختلف تعريف مولدات الأعداد العشوائية تبعا للتطبيق الذي ستستخدم فيه، فمن هذه التطبيقات التطبيقات المتعلقة بالمحاكاة و هي تتطلب سلاسل ذات خواص احصائية معينة أما التطبيقات المتعلقة بالأمن فإنها تحتاج سلاسل غير قابلة للاستنتاج أي أن أي متفحص لجزء منها لا يستطيع التنبؤ بأي جزء مستقبلي منها. إن اثبات وجود مثل هذه الكيانات غير معروف حتى الآن لذا لجأ الباحثين إلى الربط بين وجودها و وجود الدوال ذات الإتجاه الواحد و هي دوال سهلة الحساب و صعبة الانعكاس و قد قام الباحثين بالربط بين هذه الدوال و ومولدات الأعداد العشوائية ربطا نتج عنه وضع طريقة لتكوين مولد عن طريق استخدام أي من هذه الدوال. تعد الطريقة التقليدية لتكوين مولدات الأعداد العشوائية طريقة ذات عيوب أساسية ظاهرة في أدائها تتلخص في أن عدد الأرقام الثنائية (Bits) المولدة في كل مرة يتم فيها حساب الدالة ذات الاتجاه الواحد عددا قليلا و إن كان كثيرا في بعض الدوال فإن الطريقة غير قابلة للعمل على التوازي بمعنى أنه كي يتم توليد أي جزء من السلسلة فيجب توليد كافة الأجزاء السابقة لها. في هذا البحث نعرض طريقة لاستخدام مجموعة من التتابعات البسيطة مع بعض الدوال ذات الاتجاه الواحد للتخلص من العيوب الأساسية المذكورة للطريقة التقليدية موضحين الشروط اللازمة الواجب توفرها في كل من الدالة ذات الاتجاه الواحد المستخدمة و التتابع البسيط المستخدم لضمان نجاح التركيبة في تكوين مولد مثبت الأمن و قابل للعمل على التوازي. نقوم أيضا في هذا البحث بعمل مجموعة من التجارب على تركيبات مختلفة من الدوال و التتابعات للحصول على نتائج عن إمكانية نجاح هذه الطريقة التي تبدو للوهلة الأولى بسيطة و بديهية ينتج عن هذه التجارب مجموعة من التركيبات مثبتة الأمن و تركيبات أخرى تفشل تماما و تركيبات غيرها ليست مثبتة الأمن و لكن يمكن اعتبارها آمنة إن لم يمكننا الحصول على أي طريقة لهدم أمنها.

Keywords: Secure pseudo-random generators, PRG, Fully-parallelizable, Block ciphers, RSA

1. Introduction

Secure Pseudo-Random Generator (PRGs) (or PRGs that are suitable for use in cryptographic applications) are generators for which no one (with limited resources) can predict even one bit of the generated sequence given a part of this sequence. For a good (practical) cryptographic PRG there are four main required properties: Simplicity, Efficiency, Provable Security, and Parallelizability.

It is proven that secure PRGs exist if and only if one-way functions exist [9]. The proof of this result is constructible. This construction is simple. Given a one-way function, f(x), with some proven simultaneous hardcore predicates, B(x). Apply the function, f, on a random input (the seed), x, and output B(x). Repeat the function with f(x) instead of x. This construction is applicable for any one-way function even if we cannot prove any bit of its input to be a hard core predicate. This is because we can construct a hardcore predicate for any one-way function [2]. Fig. 1 illustrates this original construction.

The main advantages of this construction are:

- *It maps one-way functions and PRGs:* This is a very important theoretical result. It says that we can construct a PRG from any one-way function.
- *It is a generic construction:* Any one-way function can be used in this construction to generate a pseudo-random bits string.
- Provable security*:* The generated PRG is secure as long as the underlying function is one-way.

There are two main inefficiencies in this original construction:

- *Only a few number of random bits can be generated per each computation of the one-way function:* most of the known one-way functions (mainly based on number theoretic problems that are assumed to be hard) generate only a few hardcore bits (usually $O(log\ n)$ LSBs or MSBs where n is the size of the input of the one-way function) per each computation.

The sequential nature of the construction itself: Namely, to generate the j^{th} block of the pseudo-random bits we have to generate all j-1 previous blocks.

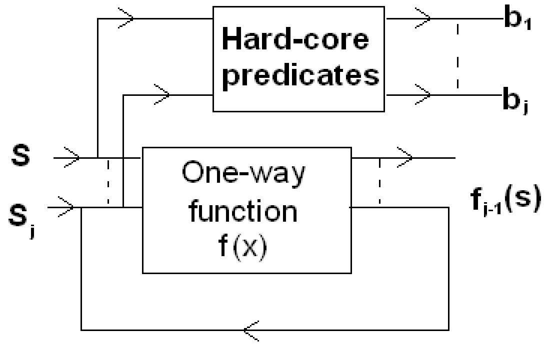

Fig. 1. The original construction of the PRGs.

In [3] Impagliazzo and Naor showed a very efficient construction for a PRG based on the intractability of the subset sum problem for certain dimensions. The *increase in efficiency* in their construction is due to the fact that many bits can be generated with one application of the assumed one-way function and the efficiency in computing the one-way function itself. The *security* of this construction does *not* depend on proving simultaneous hardcore predicates of the subset sum-based one-way function. Instead, it depends on the proof of the pseudo-randomness of the *output* of the one-way function. (The only known one-way function that is proven to simultaneously hide $O(n)$ of its input bits is the discrete log modulo composite [4]).

Although the subset sum-based one-way function is efficient to compute and is parallelizable (can be implemented in Nike's Complexity Class NC using an optimal number of processors), the nature of their PRG is still *sequential.* Blum, Blum, Shub (BBS) generator [5] is an example of PRGs that are *not pure sequential.* An interesting feature in the BBS generator is that if the factorization of n is known, the $2^{\sqrt{n}}$ th bit can be generated in time polynomial in $|n|$.

Now, we can see that the first mentioned inefficiency is solved by the construction of Impagliazzo and Naor [3]. The main interesting point is that they did not go through the original construction and they did not prove the simultaneous hard core predicates of the used one-way function. Their construction is also not a generic one and it is suitable (and proven) only for the subset sum-based one-way function.

In this paper we propose a technique to solve the second mentioned inefficiency, namely, to eliminate the sequential nature of the original construction. Using this technique we can construct *parallelizable* pseudo-random generators for which the cost of generating the j^{th} block of the pseudo-random bits equals the cost of generating any other block. This construction also increases the number of output bits per each computation of the used one-way permutation.

2. How to eliminate the sequential nature of the original construction

What do we mean by eliminating the sequential nature of constructing PRGs? The answer is: we want to construct a PRG for which computing the j^{th} block of pseudo-random bits does not require the computation of any previous block. This PRG can generate any number of blocks at a time by using the same number of processors. Our proposed method for constructing such generators is as follows:

- Choose x randomly and uniformly
- Generate any "simple" "deterministic" sequence $S_0(x)$, $S_1(x)$,…, $S_{n-1}(x)$.
- Input this sequence to $f(x)$ and output $f(S_0(x))$, $f(S_1(x))$,…, $f(S_{n-1}(x))$ as pseudo-random blocks.

This is illustrated in fig. 2.

Of course this is not a generic construction. One can easily show that there are many combinations of one-way functions and simple deterministic sequences that fail to produce a secure PRG when they are used in this way.

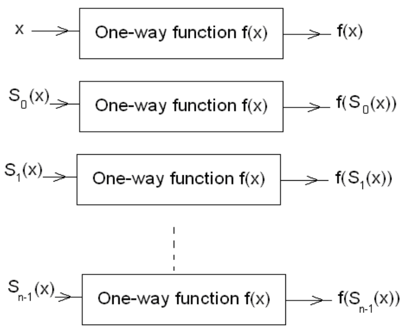

Fig. 2. The proposed construction.

Before going through which combination is successful and which is not let us first give conditions on the used one-way function and sequence that are necessary to produce a provably secure and fully-parallelizable PRG. These conditions are:

1. The sequence $S_j(x)$ is a deterministic sequence: This is obvious because the only input true randomness for a PRG must be in the seed x.
2. O(computing $S_j(x)$) = O(computing $S_i(x)$) for all i, j. This means that the used sequence is not sequential by nature.
3. O(computing $S_j(x)$) ≤ O(computing $f(x)$). This is an empirical condition to guarantee the efficiency of generating each block.
4. The sequence $S_j(x)$ does not belong to some *small* set with non-negligible probability for non-negligible number of seeds. If this condition is not satisfied one can use brute force attacks to scan the values of this small set as an input for every output block and he will succeed in finding $S_j(x)$ for some j and then x with non-negligible probability.
5. The structure of the one-way permutation hides the sequence $S_j(x)$. If $x \in U_k$ and $a = O(|x|)$ $\forall$ PPT Algorithm A, $\forall$ polynomial Q and $\forall$ sufficiently large k,

$$\Pr[A(f(S_0(x)),\ f(S_1(x)),\ \ldots,\ f(S_a(x))) = f(S_{a+b}(x))\ ,\ b>0] < 1/2+1/Q(k),$$

where the probability is taken over the random coin tosses of A, and random choices of x of length k. This condition means that no one can use some relationship between the bits of a given portion of the output blocks to find some next output.

If we look at the above conditions we will note that the first four conditions apply to the used sequence and the fifth condition is the one in which the combination of the sequence and the one-way function can succeed or fail to construct a PRG. As we mentioned, this construction is not a generic one. Not all one-way permutations can be successfully used with all sequences. In fact, there may be a permutation which has no suitable sequence at all.

In the rest of this paper we will present the results of the examination of some specific combinations of one-way permutations and simple deterministic sequences. These constructions are categorized into three main categories:

• *Unsuccessful constructions:* In these constructions although we use a sequence satisfying the first four conditions, the fifth condition is not satisfied when the sequence is combined with a specific one-way permutation. There are many examples of such combinations.

• *Successful constructions:* The security of the resulting PRG is proven and depends on the security of the underlying one-way permutation. These constructions and their proofs of security will be discussed in section III

• *Unproven and still unbreakable constructions:* For these constructions we cannot find successful attacks on the constructed PRGs. On the other hand, we cannot prove their security or link it to the security of the underlying one-way permutation or any other secure system. These constructions will be discussed in section IV.

3. Successful constructions

The Encryption function of a block cipher is a function that takes as an input a message m, and a key k, and outputs a cipher c, that is random looking. If we look at this function as a one-way function that takes a single input and outputs c we can use it with the sequence $S_j(x) = x+j$ to produce a secure PRG using our proposed method. This construction was proved to be secure [6]. The following theorem states this result:

Theorem 1: Let $F_k(x): \{0,1\}^L \times \{0,1\}^{|k|} \to \{0,1\}^L$ where $|x| = L$ be an encryption function of a block cipher and s be a random string then:

$$G(s)=F_s(0)F_s(1)\ F_s(2)\ldots\ F_s(n\text{-}1),$$

is a provably secure PRG. More precisely:

$$InSec_G^{prg}(t) \le InSec_F^{prf}(t',n),$$

where $t' = t + O(n(2L))$.

In a similar way we can prove that we can use a secure hash function combined with the same sequence $S_j(x) = x+j$ to produce a secure PRG. This is stated as follows:

Theorem 2: Let $H(x)$ be a secure hash function and s be a random string then,

$$G(s) = H(s)H(s+1)H(s+2),\ldots,H(s+n\text{-}1),$$

is a secure PRG.

4. Unproven and still unbreakable constructions

4.1. RSA

If we consider the PRG constructed by using the RSA encryption function x^e mod n with the modulus $n=pq$ and p and q are strong primes [7] and the encryption exponent e is large combined with the sequence $S_j(x) = x+j$ then the generated blocks will be a set of encrypted related messages. The only known attack on the RSA with related messages is when the encryption exponent is small [8]. No such attack is known on RSA with large encryption exponent, e. Note that when we implement such a system we have to avoid some other known attacks on the RSA function. For a survey of these attacks see [9].

If we cannot find an attack for such system this is by no means provide a proof for its security. We have to relate breaking such system to breaking the RSA itself. This system is conjectured to be a secure PRG. This conjecture is stated formally as follows:

Conjecture 1: Let $f(x) = x^e$ mod n where n is a product of two strong primes p and q where $|p\text{-}q|$ is not small, e is large ($e=O(n)$) and $d > \sqrt{n}$ where $d = e^{-1} mod\, \phi(n)$. Let x be a randomly chosen seed of the same length of n. Let $y_0=f(x)$, $y_1=f(x+1),\ldots,$ $y_i=f(x+i)$ where i is polynomial in $|n|$. Given $y_0,y_1,\ldots,y_i$ there is no PPT that can find $f(x+i+j)$ for some j polynomial in $|n|$ without inverting the RSA function.

A proof of this conjecture may be provided in the future. The report in the Appendix shows that this generator is only a candidate for a secure PRG. It does not prove its security.

4.2. Subset sum

Definition 3.4.1: The subset sum problem of dimensions n and l is: given n numbers, $a=(a_1, a_2,\ldots, a_n)$, each l bits long, and a number T, find a subset $S \subset \{1,\ldots,n\}$ such that $\sum_{i \in S} a_i = T \bmod 2^l$.

The one-way function that is based on the subset sum problem is defined as: $f:\{0,1\}^n \rightarrow \{0,1\}^l$ where the i^{th} bit in the input decides whether to take the element a_i into the summation or not. If the i^{th} bit is 1 then take a_i in the summation; otherwise do not take it. The output of the function is the resulting summation of the chosen elements. The Subset sum problem is one of the original problems that Karp [10] proved to be NP-Hard, (i.e., the corresponding decision problem is NP-Complete). We can also produce random instances of the subset sum problem that are hard on the average [11]. Hence, computing $f(x)$ is easy and inverting $f(x)$ is hard.

Theorem 3: The constructed PRG using the subset sum-based one-way function combined with the sequence Sj(x) = x+j is not secure.

Proof: Given $f(x)$, $f(x+1)$, $f(x+2)$ then we can find $f(x+3)$

- Either x or $x+1$ is even.
- This means either LSB(x)=0 or LSB(x+1)=0.
- This means either $f(x+1)-f(x) = a_0 \bmod N$ or $f(x+2)-f(x+1) = a_0 \bmod N$.
- If $f(x+1) - f(x) = a_0$ then $f(x+3) = f(x+2) + a_0$.

The subset sum based one-way permutation has a special property that is not found in any other one-way permutation (specially the permutations that are based on number theoretic hard problems). This property is: *the hard problem is to find a valid bit assignment to the input of the function not to invert some mathematical function.* This property helps in using the sequence jx mod n as an input sequence to this function. An interesting property of the sequence jx mod n is: If x is uniformly chosen then the j^{th} element of the sequence differs from the $(j+1)^{st}$ element by $O(|x|)$ bits (i.e., $O(|x|)$ bits will be converted from 0 to 1 or from 1 to 0.) Hence, to gain any information about x using the values of two successively generated blocks it is required from the attacker to decide which bits are inverted. Intuitively, it is required from the attacker to solve another subset sum problem.

We tried to map the security of this RPG to the security of the subset sum itself. All our attempts have unfortunately failed. We also tried to find attacks on this system but could find none. The security of this system is an open problem and may be solved in the future. The following conjecture states this result:

Conjecture 2: Let $f(x)$ be the subset sum-based one-way function and s be a random string. Given the sequence $f(s)f(2s)f(3s)\ldots f(is)$ where i is a polynomial in $|s|$ there is no PPT algorithm that can find $f((i+j)s)$ where j is a polynomial in $|s|$ without inverting $f(x)$.

A proof of this conjecture may be found in the future. The report in the Appendix shows that this generator is only a candidate for a secure PRG. It does not prove its security.

5. Conclusions

The original method of constructing a PRG from any one-way function has a sequential nature. One approach to eliminate this sequential nature is to search for suitable deterministic sequences for specific one-way permutations for which there is no PPT that can find $f(S_{n+a}(x)), a>0$ and a is a polynomial in $|n|$ by knowing $f(S_0(x)), f(S_1(x)),\ldots,f(S_n(x))$. If we use a suitable deterministic sequence with some one-way permutation then we can construct a PRG where the cost of computing the j^{th} block of the pseudo-random sequence is exactly the same as the cost for computing the next block of this sequence. This PRG is *Simple, Efficient, Provably Secure* and *Parallelizable.* We presented examples for provably secure PRGs constructed using the proposed method.

In two of the constructions given the resulting PRGs are conjectured to be secure. We encourage efforts to try to prove the security of these generators. We also encourage efforts to search for other examples of successful combinations or to introduce new solutions to produce provably secure and fully-parallelizable generators.

An open question is how to generalize this approach. Reaching a generalization of this approach will be a valuable result. By generalization we mean finding a secure sequence for every one-way function.

Appendix

This appendix presents the results of our implementations of the block ciphers, hash functions, RSA and subset sum-based constructions. The implementation is written in Java and run on Intel Pentium II 400 MHz machine with windows 2000 professional platform. The generated reports describe the results of experiments in which we sequentially generate 20000 random bits using the examined generator and measure the performance and randomness of this generator. Performance is measured by the time needed to generate the 20000 bits. Randomness is tested using (FIPS 140-1 *statistical tests for randomness*). If the generator passes these tests then it is a candidate to be a secure PRG. But, passing the tests does not prove the security of the generator. If the generator fails to pass one of these tests then it is completely insecure.

Testing Generator: AES_PRG

------------------------Initialization------------------
Key size: 16 bytes = 128 bits
Key:
5A920547BEECA6BA9B7B5773D0DBCB2F
Block size: 16 bytes = 128 bits
Initialization time: 37 milliseconds
--Sequential generation--
Number of generated bits: 20000
Number of random bits per calculation of the one way function: 128
Generation time: 42 milliseconds
Sample bits:
0000100001011110011100010111101100111
0110011011111110000000000010100000011
00001010011001100101110000000
Monobit test
Number of 1s: 9915 *passed*
Poker test
X3: 30.6496 *passed*
Run test
Gap 1: 2543
Gap 2: 1291
Gap 3: 583
Gap 4: 284
Gap 5: 180
Gap 6: 169
Block 1: 2601
Block 2: 1220
Block 3: 643
Block 4: 281
Block 5: 148
Block 6: 157
passed
Long run test
Max run length: 16 *passed*
The generator passes all tests
Testing Generator: MD5_PRG

--------------------Initialization----------------------
Initialization time: 3 milliseconds
------------------Sequential generation------------
Number of generated bits: 20000
Number of random bits per calculation of the one way function: 128
Generation time: 27 milliseconds
Sample bits:
0010100111000000100001100001011101100
0111111101011000110101101010100011010
000101110110001111001111010 0
Monobit test
Number of 1s: 10037 *passed*
Poker test
X3: 13.1648 *passed*
Run test
Gap 1: 2521
Gap 2: 1232
Gap 3: 642
Gap 4: 301
Gap 5: 154
Gap 6: 154
Block 1: 2510
Block 2: 1257
Block 3: 612
Block 4: 283
Block 5: 184
Block 6: 157
passed
Long run test
Max run length: 12 passed
The generator passes all tests
Testing Generator: SHA_PRG

------------------------Initialization------------------
Initialization time: 2 milliseconds
--------------------Sequential generation----------
Number of generated bits: 20000
Number of random bits per calculation of the one way function: 160
Generation time: 27 milliseconds
Sample bits:
0010010101011110111010100010000001100

110111000111000101111011101000110111
11110101001001100101110111110

Monobit test
Number of 1s: 10025 *passed*
Poker test
X3: 9.7856 *passed*
Run test
Gap 1: 2511
Gap 2: 1209
Gap 3: 609
Gap 4: 317
Gap 5: 163
Gap 6: 167
Block 1: 2475
Block 2: 1215
Block 3: 663
Block 4: 307
Block 5: 150
Block 6: 166
passed
Long run test
Max run length: 14 passed
The generator passes all tests
Generator: RSA_PRG

--------------------Initialization----------------------
number of bits of primes: 128
p:
282855661594162267458257527374267569
683
q:
242167371823603717744560872026395653
777
n:684984121736849197318145246791084803
69888665215009283247770510244311689
642691
phi(n):68498412173684919731814524679108480369363642181591517262567691844911026419232
e:681794914342531407894187579090562448
98518028569049219575958234216735296
410293
d:106269538216724083708498532716947239
70171323470866303210964508803483007
437661
Seed:090391268437273079319977599944363
60333988878824597130670571885285297
154805598
Initialization time: 481 milliseconds
--------------------Sequential Generation----------
Number of generated bits: 20000
Number of random bits per calculation of the one way function: 256
Generation time:894 milliseconds
Sample bits:
101110101101000100010111111011011011
100010110101001010011101011110011000
11111000010101111101010100000

Monobit test
Number of 1s: 9760 *passed*
Poker test
X3 : 36.1664 *passed*
Run test
Gap 1: 2427
Gap 2: 1197
Gap 3: 658
Gap 4: 300
Gap 5: 174
Gap 6: 195
Block 1: 2482
Block 2: 1276
Block 3: 615
Block 4: 307
Block 5: 127
Block 6: 145
passed
Long run test
Max run length: 14 *passed*
The generator passes all tests
Testing Generator: SSS_PRG

--------------------------Initialization----------------
$|S|$: 128
$l(|S|)$: 128
Seed:
303974519955064967531450883017475967
940
Initialization time: 8 milliseconds
-----------------------Sequential generation-------
Number of generated bits: 20000
Number of random bits per calculation of the one way function: 128
Generation time: 783 milliseconds
Sample bits:
010011101001011111111011000000000101
110100101000111001000111010010111101
01010001101000101100010010000

Monobit test
Number of 1s: 10028 *passed*
Poker test
X3: 19.9424 *passed*
Run test
Gap 1: 2461

Gap 2: 1273
Gap 3: 632
Gap 4: 305
Gap 5: 152
Gap 6: 154
Block 1: 2536
Block 2: 1163
Block 3: 630
Block 4: 319
Block 5: 156
Block 6: 173
passed
Long run test
Max run length: 12 *passed*
The generator passes all tests

References

[1] R. Impagliazzo, Leonid A. Levin, and Michael Luby, "Pseudo-random generation from one-way functions," Proc. (21)st ACM Symp. on Theory of Computing, Seattle, ACM. pp. 12-24 (1989).
[2] O. Goldreich and L. Levin, "A hard-core predicate for all one-way functions," In 21st ACM Symposium on Theory of Computing (1989).
[3] R. Impagliazzo, M. Naor, "Efficient Cryptographic Schemes Provably Secure as Subset Sum," Proc. 30th FOCS, pp. 236-241 (1989).
[4] J. Hastad, A.W. Schrift, and A. Shamir, "The discrete logarithm modulo a composite hides O(n) bits," Journal of Computer and Systems Sciences, Vol. 47, pp. 376-404 (1993).
[5] L. Blum, M. Blum and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," SIAM J. Computing, Vol. 15 (2), pp. 364-383 (1986).
[6] S. Goldwasser, and M. Bellare, "Lecture notes in cryptography," MIT, pp. 42-45 (1996).
[7] J. Gordon, "Strong primes are easy to find, In LNCS 209," Eurocrypt 84, pp. 216-223 (1984).
[8] M.K. Franklin and M.K. Reiter, "A linear protocol failure for RSA with exponent three," Presented at the CRYPTO'95 Rump session (1995).
[9] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," Notices Amer. Math. Soc. 46 (2), pp. 203-213 (1999).
[10] R.M. Karp, Reducibility among Combinatorial Problems, in Complexity of Computer Computation, ed. R.E. Miller and J.W. Thatcher, New York: Plenum Press (1972).
[11] M. Ajtai, "Generating hard instances of the lattice problem," Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 99-108, (1996).

Received February 14, 2004
Accepted August 12, 2004