Efficient and Secure Pseudo-Random Number Generation.

(Extended Abstract)

*Umesh V. Vazirani**

University of California

*Vijay V. Vazirani***

Harvard University

Cornell University

*Supported by NSF Grant MCS 82-04506, and by the IBM Doctoral Fellowship.
**Supported by NSF Grant MCS 81-21431.

Abstract: Cryptographically secure pseudo-random number generators known so far suffer from the handicap of being inefficient; the most efficient ones can generate only one bit on each modular multiplication (n^2 steps). Blum, Blum and Shub ask the open problem of outputting even two bits securely. We state a simple condition, the **XOR-Condition,** and show that **any generator** satisfying this condition can output *logn* bits on each multiplication. We also show that the *logn* least significant bits of RSA, Rabin's Scheme, and the x^2 mod N generator satisfy this condition. As a corollary, we prove that all boolean predicates of these bits are secure. Furthermore, we strengthen the security of the x^2 mod N generator, which being a Trapdoor Generator, has several applications, by proving it **as hard as Factoring.**

G.R. Blakley and D. Chaum (Eds.): Advances in Cryptology - CRYPTO '84, LNCS 196, pp. 193-202, 1985.

1. Introduction.

Recently, there has been a lot of interest in provably "good" pseudo-random number generators [10, 4, 14, 3]. These cryptographically secure generators are "good" in the sense that they pass all probabilistic polynomial time statistical tests. However, despite these nice properties, the secure generators known so far suffer from the handicap of being inefficient; the most efficient of these take n^2 steps (one modular multiplication, n being the length of the seed) to generate one bit. Pseudo-random number generators that are currently used in practice output n bits per multiplication (n^2 steps). An important open problem was to output even **two bits** on each multiplication in a cryptographically secure way. This problem was stated by Blum, Blum & Shub [3] in the context of their x^2 mod N generator. They further ask: how many bits can be output per multiplication, maintaining cryptographic security?

In this paper we state a simple condition, the **XOR-Condition** and show that any generator satisfying this condition can output *logn* bits on each multiplication. We show that the XOR-Condition is satisfied by the logn least significant bits of the x^2-mod N generator. The security of the x^2 mod N generator was based on Quadratic Residuosity [3]. This generator is an example of a *Trapdoor Generator* [13], and its trapdoor properties have been used in protocol design. We strengthen the security of this generator by proving it **as hard as factoring.** We also prove the XOR-Condition for logn least significant bits of RSA/Rabin Schemes. Our proofs are based on recent developments in RSA/Rabin Scheme bit security. We present a history of these recent developments in the next paragraph. More recently, by a different proof Alexi, Chor, Goldreich & Schnorr [1] also proved the simultaneous security of logn least significant bits of RSA/Rabin Schemes. Previously, Long & Wigderson [7] showed how to extract *logn* bits at each stage from the generator of Blum and Micali [4]; however, this gain in efficiency is not enough to compensate for the extra time taken by this generator ($O(n^3)$ steps for each stage).

The RSA-bit security problem has not only yielded several valuable proof techniques, but its two year history is also revealing in how mathematical progress is made - with successive partial solutions, simplifications and changes in point of view.

The first result on RSA bit security was proved by Goldwasser, Micali & Tong [6]. They proved that any oracle for RSA least significant bit (an efficent procedure which computes the least significant bit of the plaintext message when input the ciphertext) could be efficiently used to decrypt RSA messages, thus showing that RSA least significant bit is hard to compute unless RSA is easy to decrypt. However, the oracle was allowed to err on only $\frac{1}{logN}$ fraction of the inputs.

The next breakthrough came with the "binary gcd method" of Ben-Or, Chor & Shamir [2], which has been fundamental to all future developments. This procedure to decrypt RSA, probes the oracle at *pairs of points,* to determine the least significant bits of small messages. Each pair of probes is correct with probability $1/2+\epsilon$, provided the oracle is correct on $3/4+\epsilon$ fraction of inputs, where ϵ is any positive constant. They also showed that with more accurate oracles ($7/8+\epsilon$ correct) for other RSA bits they could

decrypt RSA.

At this stage it was not clear if even 3/4 security could be proved for the least significant bit. This question was resolved by Vazirani & Vazirani [12]. They showed that by *guessing* the least significant bits of *loglogN* random small messages (which can be done in polynomial time by considering all loglogN possibilities), they could randomize the oracle probes thereby decrypting with a less-than-3/4 oracle. They also give a method for extending the proof of security to loglogN least significant bits and the xor's of all non-empty subsets of these bits. Goldreich [6] analyzed their combinatorial problem exactly and showed that less-than-3/4 could be interpreted as $.725 + \epsilon$.

In the next major development, Schnorr & Alexi used the strong Chernoff bound along with guessing least significant bits of *loglogN* random messages to obtain a decryption procedure that used a *single oracle probe* for computing the least significant bit of small messages. Thus they proved $1/2+\epsilon$ security for any constant ϵ. However, this security was still not good enough for using RSA for direct pseudo-random number generation - $1/2+1/n^t$ security was needed.

By guessing *loglogN* most significant bits of only two random numbers, Chor & Goldreich [5] showed how to generate *logN pairwise independent numbers*, whose least significant bits were known. Thus they could ask the oracle logN pairwise independent questions. Then using the Chebychev inequality, they show that a $1/2+1/n^t$ oracle will suffice.

2. Extracting Two Bits from the x^2 mod N Generator:

The x^2 mod N generator [3] is the following: On input N, x_0 (where N is the product of two distinct primes each congruent to 3 mod 4, and x_0 is a quadratic residue mod N), it outputs $b_0 b_1 b_2 \ldots$ where $b_i = parity(x_i)$ and $x_{i+1} = x_i^2 \ mod \ N$. Its security was based on Quadratic Residuosity.

A variant of this generator outputs $b_i = location(x_i)$, where $location(x) = 0$ if $x < (N-1)/2$, 1 if $x \geq (N-1)/2$. The cryptographic security of this generator was also based on Quadratic Residuosity [3]. However, the generator which extracts parity *as well as* location at each stage may not be cryptographically secure, because revealing $parity(x_i)$ may make $location(x_i)$ predictable. Blum, Blum and Shub conjecture that this generator is also cryptographically secure, and ask the open problem: how many bits can be extracted at each stage, maintaining cryptographic security?

In this section we will prove their conjecture. In section 3 we will answer the open problem by giving a simple condition, the **XOR-Condition.** We will prove that *logn* bits ($n = |N|$) can be extracted at each stage from **any** generator satisfying this condition. We will also prove that the x^2 mod N generator as well as the generators based on RSA and Rabin's scheme satisfy this condition.

The following theorem will also give an intuitive idea for the general results of section 3, for which we will need to introduce some new definitions.

The 2-*Bit* x^2 mod N generator on input N, x_0 (N and x_0 as before), outputs $a_0b_0a_1b_1$ ... where $a_i = parity(x_i)$, and $b_i = location(x_i)$, $x_{i+1} = x_i^2 \ mod \ N$.

Theorem 1: The 2-*Bit* x^2 mod N generator is cryptographically secure.

Proof: Suppose the 2-*Bit* x^2 mod N generator is predictable to the left. There are two cases:

Case 1: It is predictable at an odd position, i.e. there is a probabilistic polynomial time procedure, P, which predicts b_{-1} [1] with probability $1/2 + \epsilon$, given $a_0b_0a_1b_1$ ... . Now we can use P to obtain $location(x_{-1})$: given any x_0, simply generate the sequence $a_0b_0a_1b_1$ ... , and use P to obtain $b_{-1} = location(x_{-1})$. Contradiction, since *location* is secure under the Quadratic Residuosity Assumption [3].

Case 2: It is predictable at an even position, i.e. there is a probabilistic polynomial time procedure, P, which predicts a_{-1} with probability $1/2 + \epsilon$, given $b_{-1}a_0b_0a_1b_1$ ... . Given x_0, we can generate $a_0b_0a_1b_1$, but not b_{-1}. Notice that P can be arbitrarilly bad at predicting a_{-1} if it is not provided with the correct bit b_{-1}. So instead we will use P to obtain two procedures, P_1 and P_2, such that either P_1 has an $\frac{\epsilon}{2}$-advantage in guessing $parity(x_{-1})$ or P_2 has an $\frac{\epsilon}{2}$-advantage in guessing $parity(x_{-1})$ *xor* $location(x_{-1})$.

Let u be the bit output by P on input $0a_0b_0a_1b_1$, and v be the bit output on $1a_0b_0a_1b_1$. If $u = v$, P_1 outputs u, else it outputs the flip of a fair coin. On the other hand, P_2 outputs the flip of a fair coin if $u = v$, else it outputs u (in this case, $u = (0 \ xor \ u) = (1 \ xor \ v)$). Notice the following facts:

1). On each input, x_0, exactly one of the two procedures, P_1 and P_2 uses the output of P, and the other one flips a coin.

2). Whenever P gives the correct answer, so does the procedure using its output; the other procedure, of course, flips a coin.

So the total number of correct answers output by both procedures is the number of correct answers output by P plus the number of correct answers output by the coin-flips. The fraction of total correct answers is $(1/2 + \epsilon) + 1/2 = (1 + \epsilon)$. So at least one of the two procedures must be correct on $1/2 + \epsilon/2$ fraction of inputs. In Theorem 3, we will show that $parity(x_{-1})$ *xor* $location(x_{-1})$ is also secure, thus contradicting the existence of P_1 and P_2 and therefore P.

---

[1] x_{-1} is the unique square root of $x_0 \ (mod \ N)$, which is a quadratic residue. $a_{-1} = parity(x_{-1})$ and $b_{-1} = location(x_{-1})$.

3. The XOR-Condition & Relative Security of Bits.

The difficulty in outputting two bits $b_1(x)$ and $b_2(x)$ at each stage (and the corresponding core of the above proof) lies in showing that there is no procedure that has any advantage in outputting bit $b_2(x)$, even though it is given $b_1(x)$ for free, i.e. in showing the **relative security** of b_2, given b_1. In general, in order to output k bits securely at each stage from a pseudo-random number generator, the main fact to be proved is that for all $i < k$, there is no procedure that outputs bit $b_{i+1}(x)$ given bits $b_1(x), \cdots, b_i(x)$. In this section we shall prove that the XOR-Condition suffices to prove the relative security of these bits.

Blum & Micali [4] give sufficient conditions for using a one-way function and a boolean predicate for cryptographically secure pseudo-random number generation. In the past the security of boolean predicates (bits) has been proved by assuming the intractability of the underlying one-way function (e.g. in proving the security of RSA least significant bits). There are several forms for these intractability assumptions (in terms of curcuit complexity, or Turing machine complexity, etc.). To make our theorems cleaner and independent of the nature of the intractability assumption, we shall in fact *define* the boolean predicate to be secure if the problem of inverting the underlying one-way function can be reduced in probabilistic polynomial time to the problem of computing the boolean predicate with a non-trivial advantage. We will require the reduction to be done uniformly (i.e. by the same Turing machine, for all N). As a result, any reasonable intractability assumption for the underlying one-way function will translate into a similar security for the boolean predicate. Since this reduction process is the only known technique for proving bit security, the proposed simplification does not sacrifice generality for all practical purposes.

First, we define formally the underlying one-way function:
let **N** be a set of positive integers, the *parameter values*, and for each $N \in \mathbf{N}$, let $n = |N|$ and $X_N \subset \{0,1\}^n$ be the *domain*. We will assume that a random element of X_N can be generated.
E_N: $X_N \rightarrow X_N$ is the one-way function with parameter N.
b: $(N,x) \rightarrow \{0,1\}$ is a *boolean predicate* computable in prob. poly. time.

Definition: Oracle $O_{b,N}$ has an $1/2 + \epsilon$ **advantage** in computing the boolean predicate b, if for $1/2 + \epsilon$ fraction of domain elements $x \in X_N$, $O_{b,N}$ outputs b(x) on input $E_N(x)$.

Definition: Boolean predicates $b_1,\ldots,b_{k(n)}$ are **inversion secure** if for each $t > 0$ there is a Las Vegas Algorithm T that runs in prob. poly. time:

$$T^{O_{b_i,N}}[i,E_N(x)] = x.$$

where $O_{b_i,N}$ is a $1/2 + 1/n^t$ advantage oracle for b_i with respect to N.

Definition: Oracle O_N has a $1/2 + \epsilon$ advantage for boolean predicate b_l **relative to** $b_1,\ldots,b_{l-1}$ if for at least $1/2 + \epsilon$ fraction of $x \in X_N$,

$$O_N[E_N(x),b_1(x),\ldots,b_{l-1}(x)] = b_l(x).$$

The behavior of O_N is unspecified, and may be arbitrarily bad, if any of the $l-1$ bits is incorrectly input.

Definition: b_l is **secure relative to** $b_1,...,b_{l-1}$ if for each t > 0, there is a Las Vegas algorithm T which runs in polynomial time:

$$T^{O_N}[E_N(x)] = x$$

where O_N is a $1/2 + 1/n^t$ advantage oracle for b_l relative to $b_1,...,b_{l-1}$.

Notice that T can use the oracle effectively only if it can guess correctly each of $b_1(x) \cdots b_{l-1}(x)$. But in our case, these boolean predicates are already inversion secure. For this reason, proving relative security (i.e. the existence of T) is considerably more difficult than proving simple bit security. We now give the **XOR-Condition,** and show (in Theorem 2) how it yields a proof of relative bit security from simple bit security.

The XOR-Condition: Boolean predicates $b_1 \cdots b_k$ satisfy the XOR-Condition if the XOR of each on-empty subset of these predicates is inversion secure.

Theorem: Let k(n) = O(logn). Let $b_1, \ldots, b_{k(n)}$ be boolean predicates which satisfy the XOR-condition. Then for every i < k(n), b_i is secure relative to $b_1,...,b_{i-1}$.

Proof: Suppose that O_N is a $1/2 + 1/n^t$ advantage oracle for b_i relative to $b_1,....,b_{i-1}$. Let T be the efficient procedure in the definition of the XOR-Condition for $b_1,...,b_{k(n)}$. Then T' is an efficient Las Vegas algorithm which uses the oracle O_N to invert E_N (see explaination at the end of the procedure).

T': On input N, i, E(x);
  O'_N <- Construct-Oracle[O_N, i-1, $1/n^t$];
  Run T with oracle O'_N to invert E_N.
end;

Construct-Oracle: On input O_N, j, ϵ;
  If j = 0 then return O_N.
  Else, let u = $O_N[E(x),b_1,...,b_{j-1},1]$
  v = $O_N[E(x),b_1,...,b_{j-1},0]$;
  Oracle $O_1[E(x),b_1,...,b_{j-1}] =$

$$\begin{cases} u \text{ if } u=v, \\ \textit{the flip of a fair coin otherwise} \end{cases}$$

  Oracle $O_2[E(x),b_1,...,b_{j-1}] =$

$$\begin{cases} v \text{ if } u \neq v, \\ \textit{the flip of a fair coin otherwise} \end{cases}$$

  Sample the two oracles on $8\epsilon^2 logn$ random elements of X_N, to determine the fraction of correct answers given by each.

If O_1 gives atleast $1/2 + \epsilon/4$ fraction correct answers then
return(Construct-Oracle[O_1, j-1, $\epsilon/4$]).
Else return(Construct-Oracle[O_2, j-1, $\epsilon/4$]).
end.

T' first calls the recursive procedure Construct-Oracle. By a proof similar to that of Theorem 1 either there is an oracle having $1/2+ 1/2n^t$-advantage for b_i relative to $b_1 \cdots b_{i-2}$ or there is an oracle having $1/2 + 1/2n^t$-advantage for b_{i-1} *XOR* b_i relative to $b_1 \cdots b_{i-2}$. Moreover, sampling the two oracles a polynomial $(8n^{2t}logn)$ number of times, Construct-Oracle can determine with high probability (1- $1/2logn$) which of the two oracles has the advantage. Continuing this procedure i-1 times, Construct-Oracle will obtain, with probability $\geq 1/2$, a $1/2 + 1/n^{t+c}$-advantage oracle for the XOR of some subset of $b_1 \cdots b_i$. T' can now use this oracle to invert E_N.

4. Proving the XOR-Condition, and Improving the Security of the x^2-mod N Generator.

In this section we state the theorems on the XOR-Condition, and briefly sketch the main ideas of their proofs. Detailed proofs will follow in the final paper.

Theorem 3: The *parity* function of the x^2-mod N generator is as hard as factoring, i.e. for any $t > 0$, an oracle which has a $1/2 + n^t$ advantage in guessing *parity(x)* on input x^2 mod N can be used to factor N. Here x is the unique square root of x^2 mod N which is a quadratic residue.

By modifying the algorithm of [1], we show how to use the *parity* oracle to extract square roots mod N efficiently. The main difficulty is that the parity oracle gives the least significant bit of the square root which is a quadratic residue. Thus on query x^2 mod N, if x is a quadratic residue mod N, the oracle will give lsb(x). Else, it gives the complement. In some sense the oracle gives the lsb(x) encrypted within the hard function - quadratic residuosity. How can the oracle's answers be interpreted correctly? The key idea is that a parity oracle with a small advantage can be used to implement a residuosity oracle which is correct with overwhelming probability [3]. This residuosity oracle may be used to "decrypt" the answers of the parity oracle.

This proves that the *location* function is also as hard as factoring since a $1/2 + 1/n^t$ oracle for *location* can be converted to a $1/2 + 1/n^t$ oracle for *parity:*
parity(x) = 0 iff location(x/2) = 0.

Theorem 4: The function *parity xor location* of the x^2-mod N generator is as hard as factoring.

We simply note that in the decryption algorithm [1], we already know the *location* of the numbers we query about. So the oracle for *parity xor location* is in effect giving us *parity,* which is sufficient for decryption.

Theorem 5: For each non-empty subset S, of the loglogN least significant bits, of the x^2-mod N generator: obtaining a $1/2 + 1/n^t$ advantage in guessing the XOR of these bits is as hard as factoring.

The idea presented in [12] will suffice along with the decryption algorithm [1]. Let the most significant bit being considered in S be the k^{th} bit, $k \leq loglogN$. Instead of running the gcd algorithm on "small messages" in the interval $[-\epsilon N,\epsilon N]$, we will choose the small messages in the interval $[-\epsilon N/2^{k-1}, \epsilon N/2^{k-1}]$. Now, for any x in this smaller interval, lsb(x)= 1 iff $\underset{S}{XOR} 2^{k-1}x = 1$ because the k-1 least significant bits of 2^{k-1} are all 0's. So, in order to obtain the lsb of a number in the interval $[-\epsilon N/2^{k-1}, \epsilon N/2^{k-1}]$, we simply multiply it by 2^{k-1}, and run the modified algorithm of Theorem 3. In a similar manner, we can prove the XOR-Condition for RSA and Rabin Scheme also:

Theorem 6: For each non-empty subset S, of the loglogN least significant bits, of RSA/Rabin Schemes: obtaining a $1/2 + 1/n^t$ advantage in guessing the XOR of these bits is as hard as decrypting RSA/factoring.

5. Going Beyond *logn* Bits.

How many bits can we hope to extract securely on each multiplication? We first make two simple observations. Certainly not all n bits. Because then all boolean predicates of x will be secure even though $E(x)$ is given. But, for example for RSA, we know that *Jacobi Symbol*(x) = *Jacobi Symbol*$(E(x))$, which is efficiently computable. Secondly, notice that in all the proofs, *logn* can be replaced by *clogn*, for any constant c.

In proving bit security, we limited the reductions (algorithms to decrypt the one-way funciton, using oracle for the bit) to be probabilistic polynomial time. If the computational complexity of the underlying one-way function is much more than a polynomial, then there is no reason to put this restriction. For example, if the intractability asssumption on the underlying one-way function states that its computational complexity is $o(n^{logn})$, then the reduction can be allowed $O(n^{logn})$ time. In this case our proofs can be modified to show that $\log^2 n$ least significant bits satisfy the XOR-Condition with $1/2 + 1/n^{logn}$ security for the bits and their XORs. These $\log^2 n$ bits can be output by a pseudo-random number generator, by a simple modification of the proof of Theorem 2. In general, if the the assumption on the complexity of the underlying one-way function is o(f(n)), then our proofs extend to showing that log(f(n)) bits can be securely output at each stage. For example, presently the fastest factoring algorithm runs in time

$O(2^{\sqrt{n \log n}})$ []. So, if we assume $f(n) = 2^{\sqrt{n}}$, we can securely extract $\sqrt{n}$ bits on each multiplication.

6. Acknowledgements: We are extremely grateful to Lenore Blum, Manuel Blum, Michael Rabin and Les Valiant for some very fruitful discussions.

7. References.

1). W. Alexi, B. Chor, O. Goldreich & C. Schnorr, "RSA/Rabin Bits are $1/2 + 1/poly(logN)$ Secure," this conference.

2). M. Ben-Or, B. Chor and A. Shamir, "On the Cryptographic Security of RSA bits," 1983 STOC.

3). L. Blum, M. Blum and M. Shub, "A Simple Secure Pseudo-Random Number Generator," to appear in SIAM Journal of Computing.

4). M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," 1982 FOCS.

5). B. Chor and O. Goldreich, " ," in preparation.

6). O. Goldreich, "On the number of Close-and-Equal Pairs of Bits in a String (with implications on the security of RSA's L.s.b.)", MIT/LCS/TM-256, March 1984.

7). S. Goldwasser, S. Micali and P. Tong, "Why and How to Estabish a Private Code on a Public Network," 1982 FOCS.

8). D. Long and A. Wigderson, "How Discreet is the Discrete Log?" 1983 STOC.

9). M. O. Rabin, "Digital Signatures and Public-key Functions as Intractable as Factorization," MIT/LCS/TR-212 Tech. memo, MIT, 1979.

10). C. Schnorr and W. Alexi, "RSA-bits are 0.5 + *epsion* secure," 1984 EUROCRYPT.

11). A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Random Sequences," 1981 ICALP.

12). U. Vazirani and V. Vazirani, "RSA bits are .732 + ϵ secure," CRYPTO-83.

13). U. Vazirani and V. Vazirani, "Trapdoor Pseudo-random Number Generators, with Applications to Protocol Design," 1983 FOCS.

14). A. Yao, "Theory and Applications of Trapdoor Functions," 1982 FOCS.