On the Security of Schnorr's Pseudo Random Generator

Rainer A. Rueppel
Crypto AG, P.O. Box 474, 6301 Zug
Switzerland

Abstract

At Eurocrypt 88 Schnorr [8] proposed a pseudo random generator for which he claimed that it could not be distinguished from a truly random source with less than $2^{o(n)}$ output bits, even when unlimited computing power was available. We show that this generator can, in fact, be distinguished with only $4n$ bits of output. Moreover, we present an efficient (linear-time) algorithm which recovers the key from a substring only slightly larger than the generator's keysize. Consequently, the generator is insecure.

1 Introduction

There are two ways in which one can (conceptually) limit the opponent's capabilities (we assume that the description of the generator is public, and that the opponent is allowed to obtain pure keystream).

1. The opponent has limited computing resources. For instance, one may assume that only polynomial-time attacks are feasible.

2. The opponent has limited access to the keystream bits. For instance, one may assume that it is unfeasible to collect more than l consecutive keystream bits, or, that it is unfeasible to collect a total of more than m keystream bits.

The basic notion of security for a pseudo random generator is that of indistinguishability [2, 3, 6, 10]. Ideally, one would like to design keystream generators that, below the unicity distance, cannot be distinguished from a random source, even with unlimited computing power; and that, beyond the unicity distance, cannot be distinguished from a random source under the assumption of reasonably bounded resources.

A complexity-theoretic framework has emerged over the past few years [2, 3, 6, 9, 10] that defines a pseudo random generator to be *perfect* or *cryptographically secure* if it passes all polynomial-time statistical tests. So far, it is not clear whether perfect generators do exist. Nevertheless, if one introduces some reasonable complexity hypothesis, one can prove "perfectness" of a generator. For instance, "perfect" generators have been postulated based on the discrete log [3], based on quadratic residuosity [2], based on one-way functions [10], based on RSA [1, 6]. At Eurocrypt 88 Schnorr proposed a pseudo random generator and a different notion of security such that the security of the generator is not based on any such unproven assumption [8]. His construction makes use of the permutation function generator proposed in [4]. This permutation generator consists of a m-round DES-like structure, where in each round i a different (pseudo) random function f_i is applied. It is shown that 3 rounds suffice to prove perfectness (polynomial-time indistinguishability) of the resulting permutation generator, provided the functions f_i are indistinguishable.

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 423-428, 1990.

Schnorr's pseudo random generator $G = \{G_k\}$ is defined as follows:
Input x: the key (seed) is a random function $f : I_n \to I_n$; size of description $n2^n$.

1. set $y_i^0 = i$ for $i = 0, 1, ..., 2^{2n} - 1$.
2. For $j = 0, 1, 2$ do
$$y_i^{j+1} = (R(y_i^j), L(y_i^j) \oplus f(R(y_i^j))))$$
{L and R mean left and right half of argument}.

Output $G_k(x)$: the sequence of $y_i^3, i = 0, 1, 2, ..., 2^{2n} - 1$. Note that the key x has size $k = n2^n$ (the function description) and that $G_k(x)$ has length $2n2^{2n}$. Thus, the generator stretches a seed of length k into roughly k^2 pseudo random bits.

The main claim in [8] is that this generator passes all statistical number tests (even those with unlimited time bound) that depend on at most $2^{n/3-(\log n)^2}$ bits of $G_k(x)$. This result refers to the situation where the opponent has limited access to the keystream bits, but otherwise has unlimited computing power. In Section 2 it is shown that Schnorr's claim is erroneous; more precisely, a statistical test is exhibited which distinguishes G with as little as $4n$ bits of $G_k(x)$. This fact was independently discovered by Ohnishi [7], Zheng, Matsumoto, and Imai [11], who were studying the construction of pseudo-random permutations, as proposed by Luby and Rackoff [4]. Maurer and Massey [5] identified and took up the constructive problem hidden in Schnorr's approach, namely to design what they called *perfect local randomizers*; these are algorithms that stretch a short string of random bits into a long pseudo-random sequence of bits with the property that every subset of e output bits is completely random (in the information-theoretic sense). They showed that such perfect local randomizers can elegantly be constructed using coding-theoretic tools. However, in a practical application the assumption that the opponent cannot observe more than e bits may be too restrictive. Therefore, if a local randomizer is proposed for direct use as keystream generator [8] , it is mandatory to analyze its computational security. For Schnorr's Generator we present an attack that recovers the key f from a small subset of $n2^n + O(n)$ bits of the output sequence of $G_k(x)$. Moreover, this algorithm runs in time $O(n2^n)$. Note, that this attack reaches the performance limits of any attack, since simply reading the key from memory requires linear time and linear space in the size of the key. It is also demonstrated that some generalizations of Schnorr's generator are prone to the same kind of efficient attack.

2 Results

Let some $y_0 = (l, r)$, and let F_f denote the permutation of I_{2n} as indexed by the key f. Then

$$Ly^3 = LF_f(l, r) = r \oplus f(l \oplus f(r))$$
$$Ry^3 = RF_f(l, r) = l \oplus f(r) \oplus f(r \oplus f(l \oplus f(r)))$$

We will drop the subscript f whenever the same key f is used in every round of the generator. Let O_F be the oracle that evaluates $F(y)$ at specific arguments y (in one computational step). We are interested in distinguishing the 2 cases, (1) F is a random function, (2) $F = F_f$, as used in G.

Theorem 1 *Schnorr's generator G can, for any number of rounds, m, be distinguished from a truly random source by 2 oracle calls, i.e., by observing $4n$ output bits of $G_k(x)$.*

Proof: let $F_m(l,r)$ denote the m-round permutation function on I_{2n}. Then, for any $m > 0$, F_m and its inverse F_m^{-1} are related as follows:

$$F_m^{-1}(l,r) = P_H(F_m(P_H(l,r))) \tag{1}$$

P_H denotes the permutation function that switches left and right half of its argument. The proof is by induction; assume (1) holds for m, and let $F_m^{-1}(l,r) = (a,b)$, then

$$F_{m+1}^{-1}(l,r) = (b \oplus f(a), a)$$

$$F_{m+1}(P_H(l,r)) = (a, b \oplus f(a))$$

Thus (1) holds also for $(m+1)$. For $m = 1$

$$F_1^{-1}(l,r) = (r \oplus f(l), l)$$

$$F_1(l,r) = (r, l \oplus f(r))$$

Thus 1 holds for any $m > 0$. It follows

$$(l,r) = F_m \circ F_m^{-1}(l,r) = F_m \circ P_H \circ F_m \circ P_H(l,r)$$

To distinguish $G_k(x)$ from a purely random string, we apply P_H to an arbitrary argument (l,r), call the oracle O_F, apply P_H a second time, and call the oracle a second time. The result will be (l,r) with probability 1 if F_m was used by the oracle. For a random function the probability that (l,r) will be the result is 2^{-2n}. □

Corollary 1 *If two functions f, g are used alternately, then G can still be distinguished by observing $4n$ output bits of $G_k(x)$, provided the number of rounds is odd.*

Proof: Let g be the function used in odd rounds 1,3,5,.. and let f be the function used in even rounds 2,4,6,..; then

$$F_{gf,1}(l,r) = (r \oplus g(l), l)$$

$$F_{gf,1}^{-1}(l,r) = (r, l \oplus g(r))$$

$$F_{gf,3}(l,r) = [r \oplus f(l \oplus g(r)), l \oplus g(r) \oplus g(r \oplus f(l \oplus g(r)))]$$

$$F_{gf,3}^{-1}(l,r) = [r \oplus g(l) \oplus g(l \oplus f(r \oplus g(l))), l \oplus f(r \oplus g(l))]$$

Hence, relation (1) still holds. □ Ohnishi [7] has independently discovered that this result holds for any palindromic arrangement of functions, that is, for any sequence of functions where reversing the order of the functions does not change the sequence of the functions. In the sequel, the computational security of G shall be analyzed for the case where the opponent has acquired a keystream-segment whose length is comparable to the keylength. This seems also interesting in the light of the generator's close structural ties to DES. The following algorithm will recover the key from a substring of size about $n2^n$ of $G_k(x)$. It is based on the observation that one can always find an $i \in [0,1,..,2^n - 1]$ such that

$$LF(i,r) = i \tag{2}$$

For any r there must exist a unique i such that $f(r) = i \oplus r$ which implies $f(i \oplus f(r)) = i \oplus r$ which in turn implies (2).The converse is not true in general, since f need not be invertible, that is, for an image $i \oplus r$ there may be more than one preimage. The case $LF(l,r) = l$ but $f(r) \neq l \oplus r$ is called a "false alarm". Such a "false alarm" can be easily eliminated by noting that

$$LF(LF(j,r), l \oplus r \oplus j) = j, \qquad j = 0,1,2,...,2^n - 1$$

has to hold with probability 1 if $f(r) = l \oplus r$. But if $f(r) \neq l \oplus r$ the check equation will only hold with probability 2^{-n} for any j. Thus, one additional test will usually discover a "false alarm". After verification that $f(r_0) = l_0 \oplus r_0$ key and keystream are related through the simple equation:

$$f(j) = LF(j \oplus f(r_0), r_0) \oplus r_0 \qquad \forall j \neq r_0$$

Algorithm: *Recover key f*

1. fix some r_0; set $i = 0$
2. test if $LF(i, r_0) = i$ {lock-in condition}
   if false set $i = i + 1$ and test again.
3. set $i_0 = i$
   {define $u_j = j \oplus i_0 \oplus r_0$ and $v_j = LF(j, r_0)$; these variables serve to check for correct lock-in}
   test if $LF(v_j, u_j) = j$ for $j = 0, 1, ..$
   {usually one test suffices to rule out a false lock-in}
   if false set $i = i_o + 1$ and go back to (2).
4. $f(r_o) = i_o \oplus r_o$
   for all $j \neq r_o$ set $f(j) = LF(j \oplus f(r_o), r_o) \oplus r_o$
   {the key f is recovered}

The number of bits required from the output sequence $G_k(x)$, in order to recover the complete key, is $n2^n + O(n)$, that is, just a little more than the keysize. The running time is $O(2^n)$, if n-bit operations are counted. The algorithm cannot be faster, since it has to look at each entry of the key f. From [4] it is known that the use of 3 independent random functions g, f, h results in a pseudo random permutation $F_{g,f,h}$. According to [11], it was proved in [7] that two independent random functions g, f used either in order g, f, f or g, g, f suffice to guarantee pseudo randomness of $F_{g,f}$. Consider the corresponding generalization of Schnorr's generator where F_f is replaced by $F_{g,f}$. Then the local randomness property of G' directly follows from the indistinguishability of $F_{g,f}$. Now the computational security of the generalized generator G' shall be addressed. For G' it holds

$$Ly^3 = LF_{g,f}(l,r) = r \oplus f(l \oplus g(r))$$

$$Ry^3 = RF_{g,f}(l,r) = l \oplus g(r) \oplus f(LF_{g,f}(l,r))$$

Algorithm (Sketch): *Recover key g,f*

1. fix some r_0;
   get $F_{g,f}(l, r_0)$ for $l = 0, 1, ..., 2^n - 1$.

2. define $f_0(l) = LF(l, r_0) \oplus r_0$ and $f_i(l) = f_0(l \oplus i)$;
   {if $g(r_0) = i$ then f must equal f_i and $RF_{g,f} = l \oplus i \oplus f_i(LF_{g,f})$}
   for $i = 0, 1, ..$ assume $g(r_0) = i$;
   if $RF_{g,f} = l \oplus i \oplus f_i(LF_{g,f})$ for $l = 0, 1, ..$ (only few checks are necessary) then $g(r_0) = i$ and $f = f_i$.

3. fix some l_0
   get $F_{g,f}(l_0, r)$ for $r = 0, 1, ..., 2^n - 1$
   compute $g(r) = RF_{g,f}(l_0, r) \oplus l \oplus f(LF_{g,f}(l_0, r))$ for all r.

The number of bits required from the output sequence $G'_k(x)$, in order to recover the complete key f, g, is $4n2^n$, that is, about double the keysize. The running time is $O(n2^n)$, that is, it is linear in the keysize. Again, the algorithm cannot be faster, since it has to look at each entry of the key f.

3 Concluding Remarks

Schnorr's generator offers neither a provable local randomization nor a reasonable resistance against cryptanalytic attacks. In fact, it can be predicted in linear time. Hence, it should not be used for encryption purposes. But it raises some interesting questions:

1. The local randomization problem, identified and taken up by Maurer and Massey [5]: Can one construct perfect local randomizers $G_k(x)$ which stretch a key x of length k into a pseudo random sequence of length $l > k$ in such a way that any $e < k$ bits of the pseudo random sequence are truly random? They showed that this problem can elegantly be solved using coding-theoretic tools. In fact, they obtained much better results than what Schnorr hoped for using the complexity-theoretic approach. The following example compares Schnorr's generator with a linear local randomizer of identical parameters:
   Example [5]: take an $[N, K]$ extended Reed-Solomon code over $GF(2^{2n})$ where $N = 2^{2n}$ and $K = 2^{n-1}$; if this code is mapped into $GF(2)$ a binary code with parameters $N' = 2n2^{2n}$ and $K' = 2n2^{n-1} = n2^n$ is obtained. Such a code has the property that every subset of $e \geq K = n2^{n-1}$ columns of the encoding matrix G is linearly independent. Consequently, if the $n2^n$ information bits are chosen independently and uniformly, the corresponding codeword of length $2n2^{2n}$ has the property that every subset of $e \geq 2^{n-1}$ bits is completely random. Note that this is about the third power of Schnorr's bound.

2. The pseudo random permutation construction problem [11]: is it possible to construct, with only one random function f, a permutation function F_f which is provably pseudo random? Then the local randomness property would be inherited by a bit generator construction such as Schnorr's.

But, what one would really like is a generator with both a proof of local randomization and a proof of unpredictability (under the assumption of reasonably bounded resources)

Acknowledgment

I wish to thank Jim Massey and Ueli Maurer for helpful discussions on the topic of the local randomizer. I would also like to thank Prof. C.P. Schnorr and Prof. T. Matsumoto for their comments.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts are as Hard as the Whole", SIAM Journal on Comput. 17 (1988), pp.194-209.

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

[3] M. Blum, S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits", SIAM J. Comput. 13 (1984), pp. 850-864.

[4] M.Luby, C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions", SIAM J.Comput. 17 (1988), pp. 373-386.

[5] U.Maurer, J.L.Massey, "Perfect Local Randomness in Pseudo-random Sequences", submitted to Crypto 89.

[6] S. Micali, C.P. Schnorr, "Efficient, perfect random number generators" Preprint MIT, University of Frankfurt, 1988.

[7] Y. Ohnishi, "A study on data security", Master Thesis (in Japanese), Tohuku University, Japan, 1988.

[8] C.P. Schnorr, "On the construction of random number generators and random function generators", Proc. Of Eurocrypt 88, Lecture Notes in Computer Science 330, Springer Verlag, 1988.

[9] A. Shamir, "On the generation of cryptographically strong pseudo-random sequences", 8th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 62, Springer Verlag, 1981.

[10] A.C.Yao, "Theory and applications of trapdoor functions", Proc. of the 25th IEEE Symp. on Foundations of Computer Science, New York, 1982.

[11] Y. Zheng, T. Matsumoto, H. Imai, "Impossibility and Optimality Results on Constructing Pseudorandom Permutations", Proceedings of Eurocrypt 89, this Volume, Lecture Notes in Computer Science, Springer Verlag, 1989.