Analysis of a Subset Sum Randomizer

Peter S. Gemmell [*]and Anna M. Johnston[†]

22 February 2001

Abstract

In [5] an efficient pseudo-random number generator (PRNG) with provable security is described. Its security is based on the hardness of the subset sum or knapsack problem. In this paper we refine these ideas to design a PRNG with independent seed and output generation. This independence allows for greater parallelism, design flexibility, and possibly greater security.

[*] University of New Mexico, Albuquerque, New Mexico. Email:{gemmell@cs.unm.edu}

[†] Sandia National Laboratories, Albuquerque, New Mexico 87185-0449. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.Email: {ajohnst@sandia.gov}.

1 Introduction

In [5] an efficient pseudo-random number generator with provable security is described. In this scheme the subset sum problem is used to generate m-random bits from a seed of n-random bits. It uses a set $A = \{a_i\}_{i=0}^{n-1}$ where each set element, a_i, is $l(n)$-bits long, where $l(n) = (1+c)n$, with $c > 0$. The subset sum function, f, operating on an n-bit integer $S = \sum_{i=0}^{n-1} s_i 2^i$ is defined as:

$$f(S) = \sum_{i=0}^{n-1} a_i s_i \bmod 2^{l(n)}.$$

In this model, one function generates both the randomizer output and the seed for the next iteration. The last $(m = cn)$-bits of this result are used as output for the PRNG, the remaining n bits are used as seed to generate the next cn bits of random data.

The randomizer we propose is very similar to the above scheme. It uses the subset sum problem to produce a secure pseudo-random number generator. But instead of using one function to generate both seed and outputted random data, our randomizer uses two subset sum sets. The first set generates a stream of seed bits while the second set generates the outputted data. Dividing the PRNG into these two separate sets neatly divides the analysis, allows for greater parallelism, may improve security and make for a more flexible design.

2 Dual Subset Sum Randomizer

A dual subset sum randomizer (further known as DSSR) is a pseudo-random number generator which uses two separate, independent subset sum problems to generate the internal (secret) seed and external (public) output. Each subset sum function operates on n-bits of input. The first subset sum function, G, produces an internal (secret) n-bit seed at each iteration. The second subset sum function, D, produces an external (public) cn-bit output at each iteration. Given a seed S^t, the seed for the next iteration is $S^{t+1} = G\left(S^t\right)$ and the output of the randomizer is $R^{t+1} = D\left(S^{t+1}\right)$. This definition of the PRNG allows for very flexible design. For example a subset sum problem similar to the Chor-Rivest knapsack ([4]) might be used for G and a knapsack modulo 2^{cn} could be used for D.

For now we define G, D as subset sum functions modulo 2^n and 2^{cn} respectively. Let G and D be based on the sets

$$\Gamma = \{g_i | g_i \in \{0, 1, \ldots, 2^n - 1\},\ i = 0, 1, \ldots, n-1\} \tag{1}$$

$$\Delta = \{d_i | d_i \in \{0, 1, \ldots, 2^{cn} - 1\},\ i = 0, 1, \ldots, n-1\} \tag{2}$$

Let the seed at time t be $S^t = \sum_{i=0}^{n-1} s_i 2^i$ with $s_i \in \{0, 1\}$. This defines G and D as

$$G\left(S^t\right) = \sum_{i=0}^{n-1} g_i s_i^t \bmod 2^n$$

$$D\left(S^t\right) \quad = \quad \sum_{i=0}^{n-1} d_i s_i^t \bmod 2^{cn}$$

The seed generated at time t is $S^t = G\left(S^{t-1}\right)$ and the PRNG output at time t is $R^t = D\left(S^t\right)$.

3 Background

The DSSR is designed for cryptographic applications. Strong cryptographic randomizers can be used to generate encryption keys, random data for public key cryptosystems, and can even be used as encryption systems themselves. In other words, strong cryptographic randomizers are needed and useful in almost all aspects of cryptography. If these randomizers are designed for efficency they also have numerous applications in supercomputing, modeling and simulation.

Because of the numerous applications, a strong cryptographic PRNG must satisfy requirements for non-cryptographic PRNG and be able to withstand cryptanalysis ([1]). In this paper we show that the DSSR satisfies both of these requirements.

Although the random subset sum problem is NP-complete, specific cases of the subset sum problem have been shown to be weak and easily broken. For example, Brickell broke the Merkle-Hellman subset-sum based public key encryption protocol using the Lenstra-Lenstra-Lovasc lattice basis reduction algorithm [3]. However the Merkle-Hellman protocol uses subset-sum problems of density 1/2 and L^3 appears to work only on low-density (less than .93) problems.

Impagliazzo and Naor [5] argue that if subset-sum is hard then it can be used for pseudo-random generation. They suggest also that the functions with density closest to 1 are the hardest functions to invert.

We organize the paper as follows:

- In section 4 we define variables and terms used throughout the paper.
- In section 5, we describe an approach to analysis that uses the Lenstra-Lenstra-Lovasc lattice basis reduction algorithm to solve subset sum problems.
- In section 6, we present a proof of security for $DSSR$ that is similar to a proof of security found in [5].

4 Definitions

While we will consider the security of $DSSR$ for a fixed input size ($n = 256$), we never-the-less can view it as part of larger function that takes various sizes of inputs and which might satisfy the following definition of pseudo-random function.

Definition 1 *Pseudo-random number generator A polynomial-time computable function $F : \{0,1\}^n \rightarrow \{0,1\}^M$ is a pseudo-random number generator if, for all polynomials p, there is an integer n_0 such that for all $n \geq n_0$ and for all polynomial-time algorithms, A:*

$$\left| Pr_{y \in_r \{0,1\}^M}[A \ accepts \ y] - Pr_{x \in_r \{0,1\}^n}[A \ accepts \ F(x)] \right| \leq 1/p(x)$$

5 Using LLL

Although the subset sum problem is NP-complete, it is not clear that it is hard to solve for either random (most) instances or that it is hard to solve for moderately-sized problems.

The Lenstra-Lenstra-Lovasc (L^3) [9] lattice basis reduction algorithm has played a major role in breaking several subset-sum related cryptographic protocols.

In section 6, we are only able to show that $DSSR$ is as secure as a particular class of density 0.5 subset sum inversion problems. However, it is not apparent (to us) how to attack $DSSR$ by inverting a density 0.5 subset sum problem. Instead, it seems that we must invert a 256-bit density 1 subset sum problem in order to deduce the internal seed stream. Never-the-less, we are designing a multi-precision rational number package that will enable L^3 to run without round-off error on 256-bit inputs. We will test this software on $DSSR$ instances to see if L^3 can be used to deduce the internal seed stream from several blocks of the output stream.

6 A Security Model for $DSSR$

We model the randomizer as a pseudo-random generator based on a pseudo-random subset-sum function of density just under 0.5.

Define $\Theta = \{c_i\}_{i=1}^{256}$ to be the concatenation of elements in Δ and Γ with lg(256) bits between to accommodate carries. Using $n = 256$:

$$c_i = d_i * 2^{256+8} + g_i$$

The elements in Θ are $(2 * 256 + 8)$ bits long. A new function which computes both the internal seed and the output of the randomizer simultaneously can now be created. Let:

$$F(S^t) = \sum_{i=1}^{256} s_i^t c_i \bmod 2^{2*256+8}$$

Note that F has density slightly less that 1/2.

Claim: If F were a one-way function, then [5] shows that F is also a pseudo-random function.

Define

$$
\begin{aligned}
F^1(S) &= \left\lfloor F(S)/2^{256+8} \right\rfloor \| F(S) \bmod 2^{256} \\
F^t(S) &= \left\lfloor F\left(F^{t-1}(S) \bmod 2^{256}\right)/2^{256+8} \right\rfloor \| \left(F\left(F^{t-1}(S) \bmod 2^{256}\right) \bmod 2^{256}\right)
\end{aligned}
$$

If words, F^t is F applied to the lower 256 bits of $F^{t-1}(S)$, with the middle 8 bits removed. In the original model this is equivalent to the concatenation of the $D(S^{t-1})$ and $G(S^{t-1})$.

> **Claim**: If F were a pseudo-random function, then for all $t > 0$, F^t would be a pseudo-random function producing $(t+1)n + \log 256$ bits.

Corollary 1 *If the function F were one-way, then the $DSSR$ would be a pseudo-random generator.*

7 A variant of $DSSR$ and its Security Model

We describe a parameterized (k) randomizer similar to the $DSSR$ and cite Impagliazzo and Naor in proving its security using the one way property of a random subset-sum function of density about $1/(1+k)$. The description and analysis are almost identical to the previous section. The only difference is that instead of generating 256 bits of random data at each iteration, only $k * 256$ bits, where $0 < k \leq 1$ bits are generated. Setting $k = 1$ will duplicate the previous analysis.

Let $k > 0$ be such that $k * 256$ is an integer and define sets similar to the previous model:

$$
\begin{aligned}
\Gamma &= \left\{g_i \middle| g_i \in_r Z/2^{256}Z,\ i = 0, 1, \ldots, 256-1\right\} \\
\Delta &= \left\{d_i \middle| d_i \in_r Z/2^{k*256}Z,\ i = 0, 1, \ldots, 256-1\right\} \\
\Theta &= \left\{c_i \middle| \left(c_i = d_i * 2^{256+8} + g_i\right) \in_r Z/2^{(1+k)256+8}\right\}
\end{aligned}
$$

As before the concatenated randomizer function can be defined as

$$
F_k(S) = \sum_{i=1}^{256} s_i c_i \bmod 2^{(1+k)256} + 8
$$

Note that F_k has density approximately $1/(1+k)$.

> **Claim**: If F_k were a one-way function, then [5] shows that F_k is also a pseudo-random function.

Define

$$
\begin{aligned}
F_k^1(S) &= \left\lfloor F_k(S)/2^{256+8} \right\rfloor \| F_k(S) \bmod 2^{256} \\
F_k^t(S) &= \left\lfloor F_k\left(F_k^{t-1}(S) \bmod 2^{256}\right)/2^{256+8} \right\rfloor \| F_k(F_k^{t-1}(S) \bmod 2^{256})
\end{aligned}
$$

If words, F_k^t is F_k applied to the lower 256 bits of $F_k^{t-1}(S)$, with the 8 bits following the first 256 bits removed. At each iteration, $k * 256$ new random bits will be created.

> **Claim**: If F_k were a pseudo-random function, then for all $k > 0$, F_k^t would be a pseudo-random function producing $(kt+1)256$ bits.

This modification of the DSSR has several benefits.

- A balance of the users need for security and efficency can be made by adjusting the density of the subset sum. The parameter k can be chosen so that $1/(1+k)$ is as close to 1 as needed for security purposes yet not so small that it runs too slow for the users efficency needs.
- The function F_k is uniformly distributed over all density $1/(1+k)$ subset-sum functions.

8 Two Examples

In this section we give to toy examples. The purpose of this is to clarify how the randomizers are used, not to demonstrate there qualities.

The first example is modeled like the original DSSR, but using only eight bits instead of 256. The second example also uses eight bits for the seed but is a modified version of DSSR, with modeled knapsack density approximately 0.8.

8.1 Example 1

The hexadecimal representations of Δ, Γ and the initial seed S^0 are:

	0	1	2	3	4	5	6	7
Δ	95	73	61	69	$A5$	$6E$	AE	FC
Γ	$A8$	FF	31	$9C$	13	$8A$	$0A$	CA
$S^0 = 7E$	0	1	1	1	1	1	1	0

This generates the following internal seed streams and random output:

$$S^t = \sum_{i=0}^{7} s_i^t d_i \bmod 2^8$$

t	S^t	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
1	FE	0	1	1	1	1	1	1	1
2	FA	0	1	0	1	1	1	1	1
3	99	1	0	0	1	1	0	0	1
4	$9F$	1	1	1	1	1	0	0	1
5	73	1	1	0	0	1	1	1	0
6	$C9$	1	0	0	1	0	0	1	1
7	$A8$	0	0	0	1	0	1	0	1
8	$D3$	1	1	0	0	1	0	1	1

$$R^t = \sum_{i=0}^{7} s_i^t g_i \bmod 2^8$$

t	R^t	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7
1	$3D$	1	0	1	1	1	1	0	0
2	$0C$	0	0	1	1	0	0	0	0
3	21	1	0	0	0	0	1	0	0
4	51	1	0	0	0	1	0	1	0
5	$4E$	0	1	1	1	0	0	1	0
6	18	0	0	0	1	1	0	0	0
7	$F0$	0	0	0	0	1	1	1	1
8	$8E$	0	1	1	1	0	0	0	1

8.2 Example Two

This example is very similar to the first example except that the elements of Γ are shortened to two bits. For this example I use the same Δ set as in the first example but use the following Γ set:

Γ	1	3	2	3	2	1	1	1

The 16 random bits generated are generated from eight iterations. Using the seed S^0 from the first example generates the following bits:

t	r_0^t	r_1^t
1	1	1
2	1	0
3	0	0
4	0	1
5	1	0
6	1	1
7	0	1
8	1	0

9 Final Comments

The DSSR is a simple fast way to generate random bits. The flexibility in the parameter choices allows the user to increase efficiency or security at will, though not both simultaneously. Although much of our analysis combined the two subset-sum sets into one set over one group, the division of these sets allows different subset sum problems to be used. Instead of doing the both subset sums over the integers modulo 2^{256}, the subset sum problems might be done over different groups. The internal seed subset sum problem might be done using elliptic curves while the external PRNG subset sum might be done multiplicatively modulo a prime. Even non-commutative groups, such as composition of permutations, could be used.

References

[1] *X9- Financial Services, X9.82-200x Random Number Generation*, January 2000. DRAFT.

[2] Alexi, Chor, Goldreich, and Schnorr, *Rsa/rabin bits are* $\frac{1}{2} + \mathit{poly}\,(\log n)^{-1}$, in Foundations of Computer Science, IEEE, 1984.

[3] E. F. Brickell, *Breaking iterated knapsacks*, in Advances in Cryptology: Proceedings of Crypto '84, G. R. Blakley and D. Chaum, eds., Berlin, 1985, Springer-Verlag, pp. 342–358. Lecture Notes in Computer Science Volume 196.

[4] B. Chor and R. L. Rivest, *A knapsack type public key cryptosystem based on arithmetic in finite fields*, in Advances in Cryptology: Proceedings of Crypto '84, G. R. Blakley and D. Chaum, eds., Berlin, 1985, Springer-Verlag, pp. 54–65. Lecture Notes in Computer Science Volume 196.

[5] R. Impaglazio and M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, in Foundations of Computer Science, IEEE, 1989.

[6] Kelsey, Schneier, Wagner, and Hall, *Cryptanalytic attacks on pseudorandom number generators*, in Fifth International Workshop on Fast Software Encryption, no. 1372 in Lecture Notes in Computer Science, 1998. Paris, France.

[7] J. Lagarias, *Pseudorandom number generators in number theory and cryptography*, in Proceedings of Symposia in Appied Mathematics, C. Pomerance, ed., vol. 42, AMS, 1990, pp. 114–144.

[8] J. C. Lagarias, *Knapsack public key cryptosystems and Diophantine approximation*, in Advances in Cryptology: Proceedings of Crypto '83, D. Chaum, ed., New York, USA, 1984, Plenum Publishing, pp. 3–24.

[9] A. Lenstra, H. L. Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen, (1982), pp. 515–534.