Some observations on the Bit-Search Generator

Chris J. Mitchell

Technical Report
RHUL–MA–2004–3
20 October 2004

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

In this short note an alternative definition of the Bit-Search Generator (BSG) is provided. This leads to a discussion of both the security of the BSG and ways in which it might be modified to either improve its rate or increase its security.

1 Introduction

In recent years there has been renewed interest in designing stream cipher keystream generators (KGs) capable of being implemented in small software or hardware and operating at very high rates. The *Bit-Search Generator* (BSG) [3] is a scheme designed to offer these characteristics when implemented as part of a KG. The BSG is a scheme that, like the Shrinking Generator (SG) [1] and Self-Shrinking Generator (SSG) [4], provides a method for irregular decimation of a pseudorandom sequence such as that generated by a linear feedback shift register (LFSR). The BSG has the advantage over the SG and SSG that it operates at a rate of 1/3 instead of 1/4 (i.e. producing n bits of the output sequence requires, on average, $3n$ bits of the input sequence).

In this short note we first provide a description of the BSG, based on that given by Gouget and Sibert [3]. Using this description we provide an equivalent and perhaps more 'natural' specification which operates on the *differential* of the original sequence. We then use this equivalent description as the basis for certain observations on the security of the BSG. Finally we also propose some possible modifications to the scheme, either to improve the rate or to increase the level of security provided.

2 The BSG

Suppose the sequence to which the BSG is to be applied is $(s_0, s_1, \ldots)$. Then, as described in [3], the output sequence (z_i) is generated as follows. (In this description, i and j denote input and output pointers, respectively).

Set: $i \leftarrow -1$; $j \leftarrow -1$;
Repeat the following steps:

1. $i \leftarrow i + 1$;
2. $j \leftarrow j + 1$;
3. $b \leftarrow s_i$;

4. $i \leftarrow i + 1$;

5. if $s_i = b$ then $z_j \leftarrow 0$ else $z_j \leftarrow 1$;

6. while $(s_i \neq b)$ $i \leftarrow i + 1$

Two examples of the BSG are provided by [3], as follows:

- If (s_i)=(0101001110100100011101) then (z_i)=(11011001).
- If (s_i)=(101001110100100011101) then (z_i)=(10010101).

3 An equivalent description of the BSG

We now provide a slightly different but equivalent specification of the BSG. We first define the *differential sequence* (d_i) as $d_i = s_i \oplus s_{i+1}$, $i \geq 0$, where $\oplus$ denotes bit-wise exclusive-or (or modulo 2 addition). The sequence (z_i) is derived from (d_i) as follows.

Set: $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:

1. $z_j \leftarrow d_i$;

2. if $(z_j = 1)$ then

   (a) $i \leftarrow i + 1$;

   (b) while $(d_i = 0)$ $i \leftarrow i + 1$;

3. $i \leftarrow i + 2$;

4. $j \leftarrow j + 1$

It is simple to verify that this modified description of the BSG generates the same output as the BSG as defined in section 2.

4 Reconstructing the input sequence from the BSG output

Given that the BSG is proposed as a component for constructing a KG, it is extremely important to know how simple it is to reconstruct parts of the input sequence from the output sequence (z_i). This arises in a natural context in a stream cipher application, where matching known plaintext and

ciphertext immediately gives keystream values, i.e. values of (z_i), and where knowledge of parts of the input sequence is a prerequisite to determining the secret key used to generate the sequence.

We focus here on the reconstruction of the differential sequence (d_i), using the description from section 3. It is important to note that recovering elements of (d_i) is likely to be of very similar significance to recovering elements of (s_i), for two main reasons.

- It is likely that information about elements of $(s_i \oplus s_{i+1})$ can be used to recover the entire sequence (s_i), although this will, of course, depend on the method used to generate (s_i).
- If (s_i) is generated using an LFSR, then the differential sequence (d_i) can also be generated using an identical LFSR (see, for example, [2, 5]). The transformation from (s_i) to (d_i) simple shifts the position of the start point of the sequence. Thus, in this case, recovering the entire sequence (d_i) from partial information has precisely the same difficulty as for the sequence (s_i).

From the BSG specification in section 3 it follows immediately that a single zero in the output sequence (z_i) corresponds to a pair of bits in the input sequence (d_i) of which the first is zero and no information is available about the second. I.e. every zero in the sequence (z_i) arises from a pair $(0, x)$ in (d_i), where x represents an unknown bit. Similarly, every one in the sequence (z_i) arises from a tuple $(1, 0^i, 1, x)$ in (d_i), where $i \geq 0$ and 0^i represents a tuple of i zeros. More specifically, the value i satisfies $i = j$ with probability 2^{-j-1}.

5 Some observations on security

First observe that, from the above analysis and as pointed out in [3], the rate of the BSG is clearly 1/3. This should be clear since the probability that the number of input bits required to produce one output bit is $(1 + i)$ with probability $1/2^i$, $i \geq 1$. Hence the rate is $\sum_{i=1}^{\infty}(1 + i)/2^i = 1/3$.

Second note that if an output bit is a zero, then one input bit is known. If an output bit is a one, then the following possibilities exist: two bits are known with probability 1/2, three bits are known with probability 1/4, ..., that is $(1 + i)$ bits are known with probability $1/2^i$ for $i \geq 1$. Hence the expected number of known bits is $\sum_{i=1}^{\infty}(1 + i)/2^i = 3$. The associated entropy is given by

$$\sum_{i=1}^{\infty} 2^{-i} \log_2(2^{-i}) = \sum_{i=1}^{\infty} i2^{-i} = 2.$$

Thus, assuming that the output sequence is evenly distributed, for each output bit the expected number of known input bits is 2, with an average entropy of 1.

So, if, for example, 64 consecutive output bits are known, the expected number of known input bits is 128, with an associated expected entropy of 64. That is, knowledge of 64 consecutive output bits and of the order of 2^{64} trials should enable the correct set of around 128 input bits to be discovered (assuming that some means exists for distinguishing correct from incorrect candidates for these input bits).

Probably the simplest strategy to search for the correct input string is to randomly generate a sequence of candidates for the input bits (where each candidate is consistent with the information derived from the output sequence). Each candidate input string should have a tuple $(1, 0^i, 1)$ inserted for every one occurring in the output sequence, where i is chosen independently at random for each output bit, such that $i = j$ with probability 2^{-j-1}. The probability of success of such a strategy will obviously depend on the number of ones amongst the 64 known output bits. To simplify matters suppose there are 32 ones, i.e. there are 32 places in the input sequence where a string of zeros of uncertain length may occur.

In each of these 32 locations, a string of i zeros will occur in the correct input sequence with probability 2^{-i-1}. We have also supposed that in any candidate input string, a string of i zeros will occur with probability 2^{-i-1}. The probability that the candidate string and the correct string agree in any one of the 32 positions is thus

$$\sum_{i=0}^{\infty}(2^{-i-1})^2 = 1/3.$$

That is, the probability that is a candidate input string will be a correct 'guess' for the input string is $3^{-32} \simeq 2^{50.7}$. That is, after a little over 2^{50} independent random trials, there is a good chance that the correct input string will have been found. Interestingly, this is rather less than the the 2^{64} trials suggested by the entropy calculations.

Note that, interleaved amongst the approximately 128 bits in (d_i) whose values can be determined, are a number of 'indeterminate' bits. These arise because of the penultimate $i \leftarrow i+2$ step in the BSG description in section 3. If the sequence (d_i) is an LFSR sequence, then the fact that unknown bits are interleaved with known bits does not make cryptanalysis significantly more difficult. This may also be true for other types of sequence used as inputs to the BSG. This observation motivates a simple modification to the BSG, described immediately below.

6 Improving the rate of the BSG

Next observe that a simple modification to the BSG enables its rate to be increased from 1/3 to 1/2, without necessarily any decrease in its effectiveness. This rate increase can be achieved simply by changing the penultimate step in the BSG description in section 3 from $i \leftarrow i + 2$ to $i \leftarrow i + 1$.

For this modified version of BSG it follows immediately that a single zero in the output sequence corresponds to a single zero in the input sequence. Similarly, every one in the output sequence arises from a tuple $(1, 0^i, 1)$ in the input sequence, where $i \geq 0$ and 0^i represents a tuple of i zeros (and $i = j$ with probability 2^{-j-1}). It should be clear that the rate of this modified scheme is precisely 1/2.

As far as the would-be reconstructor of the input sequence from the output sequence is concerned, the only difference is that the indeterminate values are removed. The uncertainty about the length of the zero-tuples remains precisely the same. Thus, if the input sequence is an LFSR sequence, the reconstruction problem has essentially the same difficulty. Hence, in some cases, this modified scheme has essentially the same level of security as the BSG but operates at a higher rate.

7 Improving the security of the BSG

The discussion in section 5 suggests that the security of the BSG rests on the indeterminate length of the input tuple required to generate an output of a one. By contrast, if a zero is output, then there is no ambiguity about the nature of the input string. This naive analysis suggests that security might be improved by introducing ambiguity no matter whether a zero or a one is output by the scheme.

This suggests the possibility of a generator (which we call the *Modified BSG*, or *MBSG*), defined as follows.

Set: $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:

1. $z_j \leftarrow d_i$;
2. $i \leftarrow i + 1$;
3. while $(d_i = 0)$ $i \leftarrow i + 1$;
4. $i \leftarrow i + 1$;
5. $j \leftarrow j + 1$

It is simple to verify that the MBSG has a rate of 1/3, i.e. the same as the BSG. It should also be clear that if the output bit is k then the input sequence used to generate this output bit must have the form $(k, 0^i, 1)$, where $i = j$ with probability 2^{-j-1}.

Following the same analysis as in section 5, for each output bit the expected number of known input bits is 3, with an average entropy of 2. Hence, if, for example, 43 consecutive output bits are known, the expected number of known input bits is 129, with an associated expected entropy of 86. If the random candidate strategy outlined in section 5 is used to conduct an exhaustive search, then around $3^{43} \simeq 2^{68.2}$ trials will be needed. This means that, in general, an exhaustive search through possible input sequences is likely to be significantly harder than for the BSG.

8 Conclusions

The BSG 'sequence filter' due to Gouget and Sibert [3] has been described, and an equivalent description given. This equivalent description has been analysed, resulting in some observations regarding the security of the BSG against naive attacks. Modified versions of the BSG have also been described which either offer an improved rate for a possibly equivalent level of security, or offer an improved level of security at the same rate.

References

[1] D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D. R. Stinson, editor, *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, Berlin, 1993.

[2] S. Golomb. *Shift Register Sequences*. Aegean Park Press, revised edition, 1982.

[3] A. Gouget and H. Sibert. The bit-search generator. In *The State of the Art of Stream Ciphers: Workshop Record, Brugge, Belgium, October 2004*, pages 60–68, 2004.

[4] W. Meier and O. Staffelbach. The self-shrinking generator. In A. De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer-Verlag, Berlin, 1994.

[5] R. A. Rueppel. *Analysis and design of stream ciphers.* Springer-Verlag, Berlin, 1986.