On the Construction of Run Permuted Sequences

CEES J.A. JANSEN
Philips Crypto B.V.
P.O. Box 218, 5600 MD Eindhoven
The Netherlands

Abstract

This paper describes the construction of classes of binary sequences, which are obtained by permuting the runs of zeroes and ones of some given periodic binary sequence $\underline{s} = (s_0, s_1, \ldots, s_{p-1})^\infty$, $s_i \in GF(2)$. A large class of sequences is constructed by permuting the runs of zeroes and ones of a DeBruijn sequence of given order. The properties of the sequences in this class are discussed. As is known, in this way all DeBruijn sequences of given order are obtained, but also many more sequences with higher complexities, all satisfying Golomb's first and second randomness postulates. It is shown how to generate the sequences in this class with the use of enumerative coding techniques. A more efficient sequence generator, employing shift registers is also introduced. The binary sequence generator obtained in this way can be useful for cryptographic purposes, e.g. in streamcipher systems.

1 Introduction

Run permuted sequences were introduced in [4] as sequences obtained through independently permuting the runs of ones and zeroes of a DeBruijn sequence of given order. DeBruijn sequences are well-known for their properties, i.e. for order n their period is 2^n, every n-tuple occurs exactly once in the sequence and by a result of DeBruijn [2] it is known that there are exactly $2^{2^{n-1}-n}$ such sequences. We will recall the properties of this sequence class, i.e. the maximum order complexities as introduced in [4] and the number of sequences obtained in this way. We will show how to generate the sequences in this class with the use of enumerative coding techniques. Also, a more efficient sequence generator, using shift registers, is given.

2 The Sequence Class $\mathcal{C}_n$

In [4, Ch. 6] a class of sequences is constructed through permuting the runs of ones and zeroes of a DeBruijn sequence of given order. The procedure described there is

I.B. Damgard (Ed.): Advances in Cryptology - EUROCRYPT '90, LNCS 473, pp. 196-203, 1991.

the following: the given sequence is written in its run-length representation and then the integers representing the runs of ones and the integers representing the runs of zeroes are permuted independently. The new sequence of integers obtained in this way is then transformed back into a binary sequence. The reason for doing this is twofold:

- The described procedure preserves the R1 and R2 properties of the original sequence ([3]), i.e. the number of ones and zeroes as well as the distribution of the runs remain unaltered.
- The sequences obtained in this way have interesting properties such as a good maximum order complexity.

Definition 1 *The class $\mathcal{C}_n$ of binary periodic sequences is defined as the class of cyclicly inequivalent sequences obtained through independently permuting the runs of ones and zeroes of a DeBruijn sequence of order n.*

It can be shown that the number of sequences in this class is given by the following equation:

$$|\mathcal{C}_n| = 2^{-n+2} \prod_{l=1}^{n-2} \binom{2^l}{2^{l-1}}^2. \tag{1}$$

If the binomial coefficients in (1) are approximated using Stirling's approximation formula, the result becomes:

$$|\mathcal{C}_n| = \rho_n \left(\frac{4}{\pi}\right)^{n-2} \prod_{k=1}^{n} G_k, \tag{2}$$

where $G_k = 2^{2^{k-1}-k}$ the number of binary DeBruijn sequences of order k and ρ_n is a correction factor which is less than 1 for all n and which converges to $0.61\cdots$ for large n.

Equation (2) clearly shows that the fraction of DeBruijn sequences contained in $\mathcal{C}_n$ goes to zero for large n. However, all DeBruijn sequences of order n are contained in $\mathcal{C}_n$, as DeBruijn sequences are by definition all those sequences in which all subsequences of length n occur exactly once. Hence, it is demonstrated that Golomb's statement [3, pg. 113] that the number of sequences in this class "is slightly larger" than the number of DeBruijn sequences of order n, is not very careful.

3 Complexity Properties of Sequences in $\mathcal{C}_n$

As all DeBruijn sequences of order n (i.e. maximum order complexity n) constitute only a small fraction of the entire class, the other sequences in $\mathcal{C}_n$ must necessarily have maximum order complexities higher than n. Concerning this complexity the following results are given in [4].

Proposition 1 *For all sequences $\underline{s} \in \mathcal{C}_n$, $n > 2$, the maximum order complexity $c(\underline{s})$ satisfies the inequality: $n \leq c(\underline{s}) \leq 2^{n-1} - 1$.*

Clearly, the lowerbound is attained by the DeBruijn sequences. For $n = 3$ the lowerbound coincides with the upperbound. For $n > 3$ the upperbound is obtained by sequences constructed as follows. All runs are divided into two sets; this is possible for all runs except for the two longest runs of both ones and zeroes, which are unique. With these two sets two identical sequences are constructed. Then the longest runs of ones and zeroes are placed in front of the first sequence and the longest but one runs in front of the second sequence. The two sequences are now concatenated to one sequence. The longest subsequence which occurs twice in this sequence has length $2^{n-1} - 2$, as can be seen from the next example.

Example 1 $\underline{s} \in \mathcal{C}_5$, $c(\underline{s}) = 15$

Two constructions:

11111 00000 11001010 00000 11001010 11111

111 000 11001010 000 11001010 111

Corollary 2 *For all sequences* $\underline{s} \in \mathcal{C}_n$, $n > 2$, *the maximum order complexity* $c(\underline{s})$ *cannot be equal to* $2^{n-1} - 2$.

The number of sequences with maximum order complexity of $2^{n-1} - 1$ (which is the maximum value) is given by the next proposition.

Proposition 3 *Let* $\mathcal{M}_n \subset \mathcal{C}_n$ *denote the subset of sequences in* $\mathcal{C}_n$ *having maximum order complexity of* $2^{n-1} - 1$. *The number of sequences in this set satisfies* $|\mathcal{M}_n| = 2^{-n+5}|\mathcal{C}_{n-1}|$, *for all* $n \geq 4$.

It appears not to be straightforward to find a general expression for the number of sequences in $\mathcal{C}_n$ with given complexity, other than for the three values shown above. However, some small numerical examples can give an impression of the distribution of complexity.

Example 2

$G_3 = 2$	$\|\mathcal{C}_3\| = 2$	$\|\mathcal{M}_3\| = 2$
$G_4 = 16$	$\|\mathcal{C}_4\| = 36$	$\|\mathcal{M}_4\| = 4$
$G_5 = 2048$	$\|\mathcal{C}_5\| = 88200$	$\|\mathcal{M}_5\| = 36$
$G_6 = 67108864$	$\|\mathcal{C}_6\| = 7304587290000$	$\|\mathcal{M}_6\| = 44100$

$n = 4$:

c	4	5	6	7
$\#(\underline{s})$	16	16	0	4

Average complexity $\bar{c} = 4.7778$

$n = 5$:

c	5	6	7	8	9	10	11	12	13	14	15
$\#(\underline{s})$	2048	17376	37824	19824	8048	1840	860	256	88	0	36

Average complexity $\bar{c} = 7.2892$

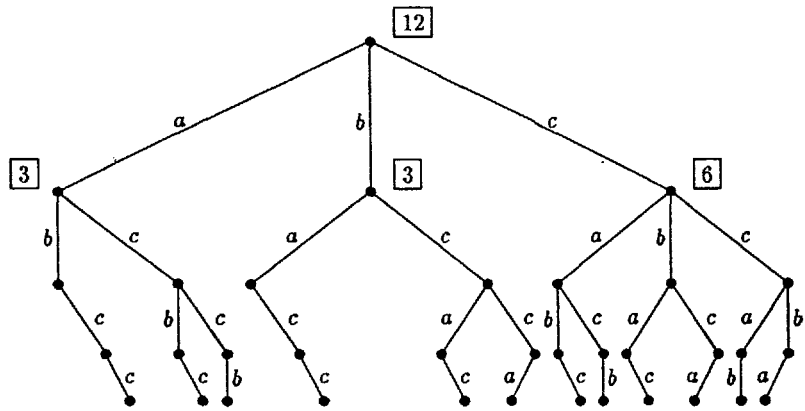

Figure 1: Ternary code tree for all permutations of *abcc*

4 Generation of Run Permuted Sequences

The sequences we have discussed so far in this section can also be generated efficiently. In order to be able to generate *all* sequences in C_n enumerative coding techniques (as in [1]) can be applied to generate the permutations.

An example demonstrates the enumerative encoding and decoding of permutations. Consider all sequences which comprise one a, one b and two c's; clearly there are $4!/2! = 12$ of these sequences. Let us assume a lexicographic ordering $a < b < c$ and define the leftmost character in the sequence to be the most significant character. All 12 sequences are represented by the ternary tree of Figure 1. The numbers written with the nodes of the tree, denote the number of leaves that can be reached from that node, i.e. the number of sequences starting with a, b, c, aa, ab, ... These numbers can easily be determined. For example, if N_x denotes the number of sequences starting with an x, where x is a sequence of length less than 4 from $\{a, b, c\}$, we have $N_a = 3!/2!$, $N_b = 3!/2!$, $N_c = 3!/1!$, $N_{ab} = 2!/2!$, and so on. From the code tree it can be seen that *cabc* is coded into $(3 + 3) + 0 + 0 = 6$. The codewords are often called the indices, written as $i(cabc) = 6$. The general expression for the indices is:

$$i(\xi_0\xi_1\xi_2\xi_3) = \sum_{n=0}^{2} \sum_{\nu_n < \xi_n} N_{\xi_0 \cdots \xi_{n-1}\nu_n},$$

where $\nu_n, \xi_n \in \{a, b, c\}$, for $n = 0, 1, 2, 3$. Using the above expression we see that $i(abcc) = 0$ and $i(ccba) = 11$.

The decoding process uses again the node numbers of the code tree. As an example, consider the decoding of an index with value 5. Clearly, the sequence cannot start with an a, as all such sequences have indices less than 3. Also, the sequence cannot start with a c, as all of these sequences have indices $\geq 3 + 3$. Hence, the sequence starts with a b. Next, the index is decreased by $N_a = 3$, yielding a new index with value 2. Applying the same procedure, c is obtained as the second character, yielding

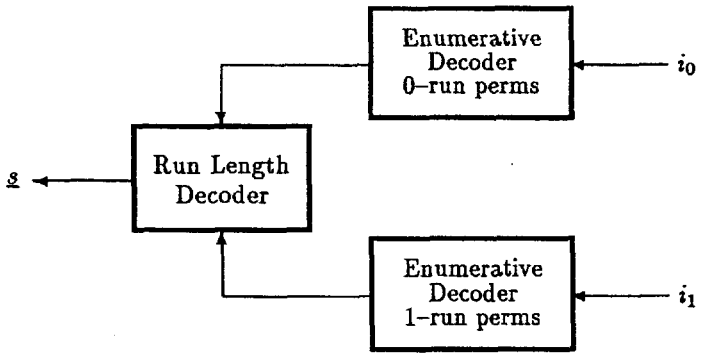

Figure 2: Generator structure for run permuted sequences

a new index with value 1. Continuing in this way we find that the sequence with index 5 is *bcca*.

It is obvious that the enumerative decoding process can be applied twice, once for the runs of ones and once for the runs of zeroes. In this way, all different run permuted sequences can be generated in their run-length representations. To generate the corresponding binary sequences it suffices to apply the run-length decoding algorithm. The structure of such a sequence generator is depicted in Figure 2

5 A Shift Register Construction

Although the enumerative decoding techniques are well understood, their implementational complexity is still quite substantial. Therefore, in this section we present a run permuted sequence generator, based on feedback shift registers; binary counting devices that are easy to implement.

The proposed method works because of the fact that the generator needs not be able to generate the entire class $\mathcal{C}_n$, but rather a smaller subclass of sequences, with sequences possibly occuring more than once, i.e. the class may contain phase shifted versions of one and the same sequence. Under these relaxed conditions the enumerative decoders, as shown in Figure 2, can be replaced by integer-sequence generators, being capable of generating a number of different sequences of integers.

These sequences of integers have to satisfy a number of conditions, viz.:

1. the period must be 2^{n-2}, as there are exactly that many runs of both ones and zeroes in a run permuted sequence of order n

2. because of the perfect run distribution, there must be 2^{n-3} integers with value 1, 2^{n-4} with value 2, etc.

Sequences satisfying these conditions can be constructed very efficiently by means of a nonlinear feedback shift register, generating a DeBruijn sequence of order $n-2$. An example is shown in Figure 3, which shows the complete binary sequence generator of

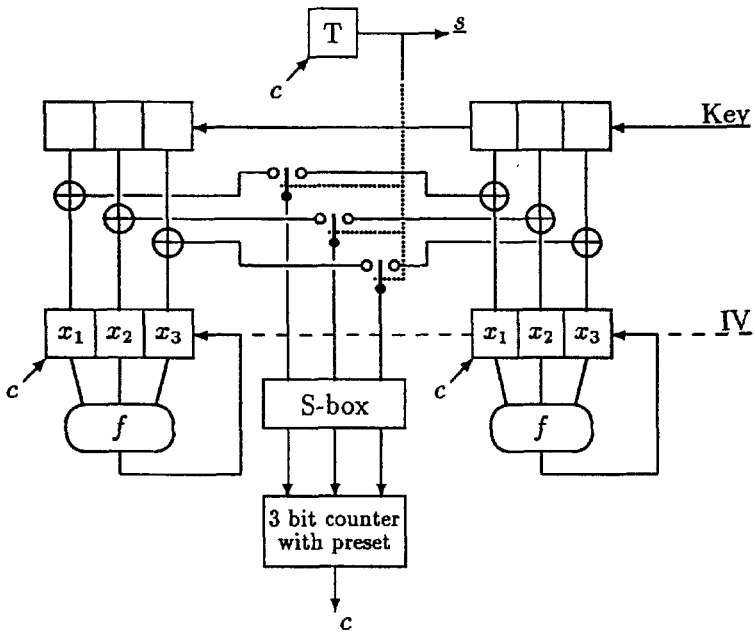

Figure 3: Generator example for run permuted sequences of order 5

order 5, being capable of generating 512 different run permuted sequences of period 32. The complexities of these sequences are listed below:

Total	$\#(\underline{s}) = 512$	(88200)
$c = 6$	128	(17376)
$c = 7$	168	(37824)
$c = 8$	168	(19824)
$c = 9$	48	(8048)
$\bar{c} = 7.2656$		(7.2892)

The numbers between brackets are for the entire class C_5.

From Figure 3 the two identical DeBruijn sequence generators of order 3 can be identified, i.e. one for the runs of ones and one for the runs of zeroes. The statesequences of each of these generators are permuted by the contents of the Key registers, thus giving rise to 8 different statesequences each. The initial vector IV causes the two statesequences to combine in 8 different ways. The statesequences are converted into appropriate sequences of integers by means of the S-box. Alternatively this S-box may be implemented separately for the two statesequences, thus allowing additional permutations. The following details apply to the generator of Figure 3:

- $f(x_1, x_2, x_3) = x_1 + x_3 + \overline{x_2 \vee x_3}$

- T: toggle flipflop $010101\cdots$
- T clocked by $c \wedge clock$
- NLFSR for 0-runs clocked by $c \wedge clock \wedge \underline{s}$
- NLFSR for 1-runs clocked by $c \wedge clock \wedge \bar{\underline{s}}$
- c is output of binary 8-counter (state 111)
- 8-counter counts from preset value to 111 and presets again
- S-box is substitution table, e.g.:

in:	0,	1,	2,	3,	4,	5,	6,	7
out:	7,	7,	7,	7,	6,	6,	5,	3

Example 3 Operating example: Key = 101001, IV = 111000

The table below shows the successive states of the DeBruijn generators and the corresponding integer values input to the S-box by the generators taking "Key" and "IV" into account.

DB–gen		G-r	G-l
000	0	1	2
001	1	0	3
011	3	2	0
111	7	6	7
110	6	7	1
101	5	4	5
010	2	3	4
100	4	5	6

```
Gr:  10.2.6.......7.....4...3..5....1.
Gl:  2.3.0...7.........1..5..4...6...2
ct:  7....5673456734567.6767.6767567..
s :  01010111000001111101100100110001 0
```

From the foregoing it should be clear that higher order sequences can be obtained analogously. In particular, for n^{th} order sequences, having period 2^n, the DeBruijn sequence generators must be of order $n-2$ and the presetable counter must comprise $\lceil \log_2 n \rceil$ bits. In this way 2^{3n-6} different sequences can be generated efficiently.

6 Conclusions

In this paper a construction method for binary sequences was described and its properties investigated. This construction method is based on source coding techniques. Starting with a DeBruijn sequence of given order we construct an entire class of sequences by permuting the runs of ones and zeroes. The class of sequences contains all DeBruijn sequences of the given order, but many other sequences as well, all satisfying Golomb's first and second randomness postulates.

It was demonstrated that all the sequences in the class can be generated by enumerative decoding techniques. Also, an efficient sequence generator based on shift

registers was shown, which seems very well suited for high speed applications. In the way described in this paper, many more classes of sequences can be constructed, e.g. if the starting sequence is a *Maximum Length Linear FSR sequence* or a *Legendre sequence* ([3, pg. 47]), which may be useful for cryptographic purposes. To this end, it suffices for the the shift register generator to simply use other S-boxes.

References

[1] T. Cover. "Enumerative Source Coding", *IEEE Trans. on Info. Theory*, vol. IT–19, pp. 73–76, January 1973.

[2] N. G. de Bruijn. "A Combinatorial Problem", *Nederl. Akad. Wetensch. Proc.*, vol. 49, pp. 758–754, 1946.

[3] S. W. Golomb. *Shift Register Sequences*, Holden–Day Inc., San Francisco, 1967.

[4] C. J. A. Jansen. *Investigations On Nonlinear Streamcipher Systems: Construction and Evaluation Methods*, PhD. Thesis, Technical University of Delft, Delft, April 1989.