Clock-controlled pseudorandom generators on finite groups*

Ulrich Baum[1] and Simon Blackburn[2]

[1] Institut für Informatik V, Universität Bonn,
Römerstraße 164, 53117 Bonn, Germany.
uli@leon.cs.uni-bonn.de
[2] Department of Mathematics, Royal Holloway,
University of London, Egham, Surrey TW20 0EX, UK.
uhah058@vax.rhbnc.ac.uk

Abstract. As a generalisation of clock-controlled shift registers, we consider a class of key-stream generators where a clocking sequence is used to control a "pseudorandom" walk on a finite group.

1 Introduction

Cascades of clock-controlled shift registers have been extensively studied as candidates for secure key-stream generators [2]. It is possible to regard any clock-controlled shift register used in this scheme as a finite state machine whose states are the elements of a finite cyclic group of order equal to the period of the register (see Section 2 for details). In this paper, we generalise this construction to the case where an arbitrary finite group is used as state space of each component of the cascade. Our motivation for this generalisation is twofold. Firstly since many finite groups can be implemented efficiently in both hardware and software, it is realistic to hope that some of the new generators constructed will be useful in practice. Secondly, the broader perspective gained should improve our understanding of the strengths and weaknesses of the generators currently in use.

The rest of this paper is organised as follows. Section 2 contains a description of our generalisation of a clock-controlled shift register. Sections 3 and 4 discuss the design criteria for such generators when used in a cascade. Next, we present some simple examples of our construction and discuss the statistical properties and security of their output sequences. Finally, we set out our conclusions and suggest areas for further research.

* This research was supported in part by Deutscher Akademischer Austauschdienst and the British Council under the British-German Academic Research Collaboration programme. Simon Blackburn is supported by E.P.S.R.C. Research Grant GR/H23719

2 The basic setup

A *(clock-controlled) group generator* consists of the following three components:

(1) A *control generator* C producing a periodic binary clocking sequence $(c_i)_{i\geq 0}$ of least period γ.
(2) A *finite group* $G = \langle g_0, g_1 \rangle$ generated by two elements g_0 and g_1.
(3) An *output function* $f : G \to GF(2)$.

The control generator may be any finite state machine which produces a binary output, e.g., an LFSR or a cascade of clock-controlled group generators. This allows us to build cascades from very simple group generators.

The clocking sequence is used to control a walk on the group G as follows. In each step, we move from $g \in G$ to gg_0 if the next bit of the clocking sequence is 0 or to gg_1 otherwise. Starting with the identity of G, this defines the *state sequence* $(q_i)_{i\geq 0}$ of our generator:

$$\begin{aligned} q_{-1} &:= 1 \\ q_i &:= q_{i-1} g_{c_i} \quad (i \geq 0) \end{aligned}$$

Hence $q_i = g_{c_0} \cdot g_{c_1} \cdots g_{c_i}$. In each step, the generator computes an output bit s_i by applying f to the current state:

$$s_i := f(q_i)$$

We note that clock-controlled LFSRs are a special case of our construction: Let $G = \langle g \rangle$ be cyclic of order $2^l - 1$ and choose a pair $(g_0 = g^{e_0}, g_1 = g^{e_1})$ generating G. For fixed $0 \neq \beta, \alpha \in GF(2^l)$, where α is a primitive element of $GF(2^l)$, define the output function f by

$$f(g^e) := Tr(\beta \alpha^e).$$

The resulting generator is a clock-controlled LFSR that is stepped by e_0 or e_1 steps according to the clocking sequence. In particular, setting $g_0 = 1$ yields the well-known stop-and-go generator.

We will end this section by giving a simple example of a non-abelian group generator: Let $G = S_3 = \langle (123), (12) \rangle$ be the symmetric group of order 6 and define the output function f by

$$\{(1), (23), (12)\} \mapsto 0, \qquad \{(123), (132), (13)\} \mapsto 1.$$

Clocked by the m-sequence 0111010... of period 7, we obtain the following pseudorandom sequence of least period 21:

$$101000111110110000110\ldots$$

This sequence has periodic linear complexity 18 and a good run length distribution.

3 The state sequence

It seems hard to prove general properties of the output sequence (s_i) produced by a clock-controlled group generator since these strongly depend on the chosen output function f. In this section, we are going to analyse the *state sequence* (q_i) of a clock-controlled group generator, which does not depend on the output function f.

PERIOD. First, we determine the least period ρ of the state sequence (q_i). Recall that γ denotes the least period of the clocking sequence.

Lemma 1. *$\rho = \gamma \cdot ord(q)$, where $q := q_{\gamma-1} = g_{c_0} \cdot g_{c_1} \cdot \ldots \cdot g_{c_{\gamma-1}}$.*

Proof. Since the clocking sequence (c_i) can be reconstructed from the state sequence (q_i), ρ equals the least period of the sequence of pairs (c_i, q_i), which is easy to determine: Obviously, it is a multiple of γ. By definition of q, we have $q_{k\gamma-1} = q^k$ for $k \in N$. Hence the sequence (c_i, q_i) has minimal period $l\gamma$, where $l = \min\{k \mid q^k = 1\} = \mathrm{ord}(q)$.

To apply this lemma, we have to find the order of q. In general, there seems to be no better way than explicitly computing q from one cycle of the clocking sequence. However, this is not necessary in some special cases: Modulo its derived subgroup G', G is a two-generator abelian group. The *coset* of G' containing q is uniquely determined by the number of zeroes and ones in one cycle of the clocking sequence. Hence the minimum order of an element of this coset is a lower bound on the $\mathrm{ord}(q)$. See Sections 5 and 6 for examples.

STATES REACHED. Next, we ask how many times each state occurs in one period of (q_i). In particular, we would like to know whether the sequence contains all possible states. Our intuition is that for good statistical properties of the output sequence, the state sequence should hit every element of G about equally often.

From the definition of q, it is clear that $q_{i+\gamma} = qq_i$ for all i. It follows that if we write a full period of the state sequence (q_i) as $(e \times \gamma)$-matrix , where $e := \mathrm{ord}(q)$, we obtain

$$(q_{i\gamma+j})_{0\le i<e, 0\le j<\gamma} = \begin{pmatrix} q_0 & q_1 & \ldots & q \\ qq_0 & qq_1 & \ldots & q^2 \\ q^2q_0 & q^2q_1 & \ldots & q^3 \\ \vdots & \vdots & \vdots & \vdots \\ q^{e-1}q_0 & q^{e-1}q_1 & \ldots & 1 \end{pmatrix}.$$

Each column of this matrix contains a coset of the cyclic subgroup $U := \langle q \rangle$ of G. Hence we have the following.

Lemma 2. *(a) The set $Q := \{q_i \mid i \ge 0\} \subseteq G$ of states reached by the generator is a union of cosets of the cyclic subgroup $U := \langle q \rangle$.*
(b) All elements of the same coset occur equally often in (q_i).

(c) $Q = G$ *iff* $(q_0, q_1, \ldots, q_{\gamma-1})$ *contains a transversal of the cosets of* U *in* G.
(d) (q_i) *contains every element of* G *equally often iff* $(q_0, \ldots, q_{\gamma-1})$ *contains the same number of elements from each coset of* U *in* G.

In general, one will have to construct the states $(q_0, \ldots, q_{\gamma-1})$ to check if the condition in part (d) is satisfied. In some special cases however, this can be easily seen *a priori* from simple conditions on the clocking sequence, see the examples in Sections 5 and 6.

STATISTICAL PROPERTIES. One can expect the state sequence to have good statistical properties if the clocking sequence has reasonable statistics. To see why, suppose that our generator's clocking sequence consists of independent and identically distributed random bits. Then the state sequence corresponds to a *random walk* on G. By general Markov chain theory, the random walk converges exponentially fast to a distribution that periodically cycles through the uniform distributions on all cosets of the unique normal subgroup U of G of the form $U = \{g_0, g_1\}^k$. For details and further reference on random walks, see [1]. This indicates that the state sequence should still have good statistics if the generator is clocked by a reasonable pseudorandom sequence.

4 The output function

The choice of the output function f is crucial for our generator's performance. Of the $2^{|G|}$ possible output functions, we should choose one that

- is easy and fast to evaluate,
- guarantees high period and linear complexity of the output sequence,
- yields an output sequence with good statistical properties and
- disguises the group structure of the generator as much as possible.

For cryptographic applications, there should be a sufficient number of good output functions for a given generator so the choice of f can be used as (part of) a secret key.

Our ultimate goal is to find an efficient algorithm which when given a suitable group G and generators g_0, g_1, finds a set of good output functions. As this appears to be beyond our reach at the moment, we will discuss some properties that we consider desirable.

Firstly, we consider how to choose f such that the output sequence has large period σ. Obviously, σ divides the period $\rho = \gamma \mathrm{ord}(q)$ of the state sequence. How can we make sure that σ has the maximum possible value ρ? A counting argument shows that if the set Q of states reached by the state sequence is large enough, then $\sigma = \rho$ for most choices of f. However, this is not very helpful since we have to *construct* a suitable output function that has additional properties. Hence we will try to find (simple) conditions on f that guarantee the maximal possible period $\sigma = \rho$.

PROPERTY 1. The least common multiple of the least periods of all sequences $(f(hq^i))_{i \geq 1}$, $h \in Q$, should equal $\mathrm{ord}(q)$.

Since these sequences are all shifted decimations of (s_i) by γ, Property 1 is a necessary condition for $\sigma = \rho = \gamma \mathrm{ord}(q)$ according to Lemma 6. Although it is not sufficient in general, there are some special cases where Property 1 guarantees $\sigma = \rho$:

Lemma 3. *Suppose that $\gamma = p^a$ and $ord(q) = p^b$ $(a, b \geq 1)$ are powers of the same prime p. Then Property 1 implies $\sigma = \rho$.*

Proof. As p is prime, $\sigma | \gamma \mathrm{ord}(q) = p^{a+b}$ must be a power of p as well: $\sigma = p^c$. Since $b > 1$, it follows from Lemma 7 that $c > a$ and $b = c - a$.

PROPERTY 2. The function f should be nearly balanced (so $||f^{-1}(0)| - |f^{-1}(1)||$ should be small).

We want our output sequence to contain about same number of zeroes and ones. Property 2 guarantees this if we assume that the state sequence hits every element of G equally often.

If the output sequence is to be cryptographically secure, it is desirable that deducing any information about the state sequence from the output sequence should be difficult. Thus f should disguise the group structure as much as possible.

PROPERTY 3. For any proper normal subgroup N of G, f should not be constant on every coset of N.

Otherwise, the generator is equivalent to one based on the smaller factor group G/N. In particular, f or its complement should not be group homomorphisms. In fact, if a close approximation to f fails Property 3, then f is also a poor choice because then the generator can be approximated by a smaller one.

We now introduce a much stronger property, which is easily seen to imply Property 3.

PROPERTY 4. The function f does not correlate with multiplication in G:

$$\forall g \in G \setminus \{1\} : |\{x \in G \mid f(xg) = f(x)\}| = |G|/2.$$

To see why this requirement makes sense, suppose that for some $g \in G$, $f(x) = f(xg)$ for significantly more (or less) than half of all $x \in G$. Then for all positions $i < j$ with $q_j = q_i g$, s_j can be predicted by s_i (resp. its complement), with high probability. Such correlations could also be used to gain information about the state sequence from the output sequence. Hence for cryptographic strength of our generator, f should – at least approximately – have Property 4.

Functions satisfying Property 4 have been studied in the case when G is an elementary abelian 2-group under the name of *bent functions* [5]. By analogy with this case, we say a function f is bent if it satisfies Property 4.

Unfortunately, bent functions do not exist for all groups and are never completely balanced:

Theorem 4. *Let G be a group of order n. Suppose that f is a bent function on G and define $z := |f^{-1}(0)|$ to be the number of zeroes of f. Then n must be an even square and $z = (n \pm \sqrt{n})/2$.*

Proof. For any elements $x, y \in G$, define the integer $t(x, y)$ by

$$t(x,y) := \begin{cases} 1 & \text{if } f(x) = f(xy) \text{ and} \\ -1 & \text{if } f(x) \neq f(xy) \end{cases}$$

We evaluate the sum $\sum_{x \in G} \sum_{y \in G} t(x, y)$ in two different ways: Firstly,

$$\sum_{x \in G} \sum_{y \in G} t(x,y) = \sum_{x \in G} t(x,1) + \sum_{x \in G} \sum_{y \in G \setminus \{1\}} t(x,y) = n + 0 = n$$

since because f is bent, $\sum_{x \in G} t(x, y) = 0$ for any $y \neq 1$. Secondly,

$$\begin{aligned} \sum_{x \in G} \sum_{y \in G} t(x,y) &= \sum_{x \in f^{-1}(0)} \sum_{y \in G} t(x,y) + \sum_{x \in f^{-1}(1)} \sum_{y \in G} t(x,y) \\ &= z(2z - n) + (n - z)(n - 2z) \\ &= (n - 2z)^2. \end{aligned}$$

Thus $n = (n - 2z)^2$ and hence n is a square and $z = (n \pm \sqrt{n})/2$. The theorem follows, since clearly n must be even.

It is not clear whether bent functions exist for every group whose order is an even square, nor do we know how to find them efficiently. If G is an elementary abelian 2-group, efficient constructions of bent functions are well known, see [5]. For small examples of non-abelian groups, we have found bent functions by exhaustive search. However, this approach is not feasible unless G is very small, say $|G| \leq 25$. The fact that bent functions are not perfectly balanced does not seem to be a problem in practice since if a balanced function is desired, we can complement an appropriate number of bits to produce a balanced function which still has reasonable correlation properties. Even if no bent function exists for a given group, we can search for "approximately bent" functions.

Despite their good correlation properties, we are not sure that bent functions are the ideal choice for cryptographically secure output functions. Because the property of being bent is so strong, it may well force extra structure upon the function which can be exploited in an attack. Some experimental evidence of such phenomena is presented in Sections 5 and 6. However, the structures we found would only exhibit themselves over long segments of the sequence, so they might not be detectable in practical situations.

5 Example 1: The quaternion group generator

In this section, we will have a closer look at a group generator based on the group of quaternions.

THE GENERATOR. In the notation of our basic setup, choose

$$G := \langle i, j \mid i^2 = j^2 = ijij = -1 \rangle = \{1, -1, i, -i, j, -j, ij, -ij\},$$

the quaternion group of order 8 with generators $g_0 := i$ and $g_1 := j$.

STATE SEQUENCE. Modulo the commutator subgroup $G' = \{1, -1\}$, the generators commute and have order 2. Hence the coset of G' containing a state q_i is uniquely determined by the number of zeroes and ones in the first i bits of the clocking sequence. Since all elements of $G \setminus G'$ have order 4, we know that the state q reached after a full period of the clocking sequence has order 4 iff one cycle of (c_i) contains an odd number of zeroes or ones, or both. In this case, the state sequence has period $\sigma = 4\gamma$.

From now on, suppose that the clocking sequence has least period $\gamma = 2^k$ and odd weight. Then it is clear that $q = \pm ij$ has order 4, so the state sequence has period $\rho = 4\gamma = 2^{k+2}$.

We claim that the state sequence hits every element of G the same number of times in each cycle. To see this, look at the subgroup generated by q:

$$U = \langle q \rangle = \{1, -1, ij, -ij\}.$$

Since U has index 2 in G and does not contain the generators $g_0 = i$ and $g_q = j$, the state sequence alternates between elements of U and elements of the coset $iU = jU = G \setminus U$. Since γ is even, it follows by Lemma 2 that every group element is reached exactly $\gamma/2$ times in one cycle of the state sequence.

OUTPUT FUNCTION. Next, we have to choose a suitable output function f satisfying Properties 1–4 of Section 4. Since 8 is not a square, no bent functions on G exist according to Theorem 4, so we have looked for functions that have Properties 1–3 and are approximately bent. By exhaustive search using the group theory package GAP [3], we have found the following 8 best candidates for f that look very much alike.

	1	−1	j	$-j$	i	$-i$	ij	$-ij$
f_1	1	0	1	1	0	0	1	0
f_2	0	1	1	1	0	0	1	0
f_3	1	0	0	0	1	1	1	0
f_4	0	1	0	0	1	1	1	0
f_5	1	0	1	1	0	0	0	1
f_6	0	1	1	1	0	0	0	1
f_7	1	0	0	0	1	1	0	1
f_8	0	1	0	0	1	1	0	1

If we use one of these output functions, the resulting binary output sequence of our generator has least period $\sigma = \rho = 4\gamma$ by Lemma 3. Since this is a power of 2, the linear complexity of the sequence is at least half the period according to Lemma 8.

IMPLEMENTATION. The quaternion group generator is very easy to implement in hard- or software: If we represent the elements of G by three bits according to

$$(a, b, c) \longleftrightarrow (-1)^a i^b j^c,$$

multiplication by i means $(a, b, c) \mapsto (a \oplus b \oplus c, \bar{b}, c)$ while multiplication by j amounts to $(a, b, c) \mapsto (a \oplus b, b, \bar{c})$. The output function can be implemented by a lookup table or a few boolean gates. In this implementation, we can expect our generator to be about as fast as a 3-bit LFSR.

CASCADES. Since one quaternion group generator is too small to be useful in practice, we are interested in cascading them. If we can make sure that the clocking input of each generator in the cascade has odd weight, we know that the period and linear complexity of our sequence is increased by a factor of 4 in each stage. Unfortunately, the output of our generator has even weight since the output function has weight 4 and every group element is reached the same number of times. A simple (but cryptographically questionable) way to overcome this difficulty is to change one output bit in each cycle of each stage.

When building cascades of LFSR, it is common to add the input of each stage to its output. Our experiments indicate that that the sequences obtained from cascades of quaternion group generators have better statistical properties if we do this as well.

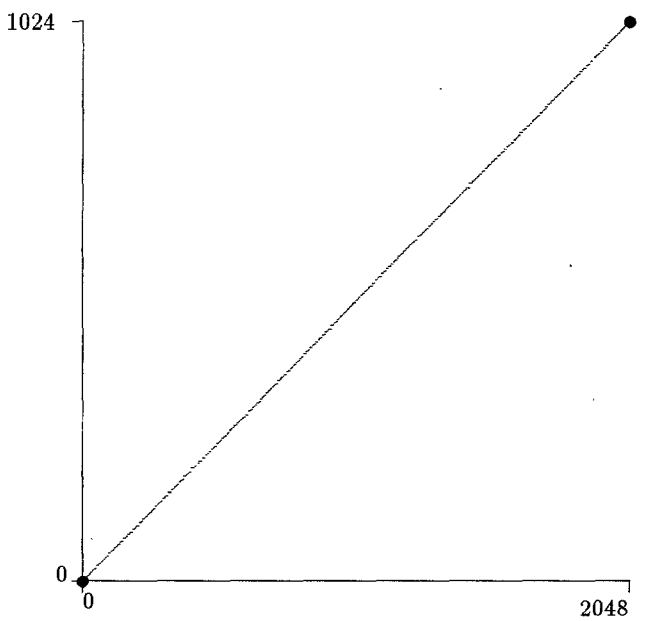
For cryptographic applications, a key of three bits per stage can be used to choose one of the 8 output functions $f_1, \ldots, f_8$ in each stage.

EXPERIMENTS. We have implemented a cascade of quaternion group generators in software using the group theory system GAP [3]. Each stage uses the output function f_1 from the table above and adds its input to its output. Between the stages, the first output bit is changed to guarantee that all clocking sequence in the cascade have odd weight. Clocking a four-stage cascade with the sequence $(1, 0, 0, 0)$, we obtain the following sequences:

```
(1011110001101001)

(10101111101110011010000011001110111101010001001100001010011001O)

(10100001000011111110000100110011100101110011111001100011000000001
 11001011101100000000101111011100001110011100011101101000101010 10
 01110100010110101011010001100110110000100110100110010011010101 00
 10011110111001010101111010001001011011001001001000111101111111 11)

(10100100101111101110010010101010110011000000010101010011000100 01
 10101000001010111011110100001111101011001111110000110011110010 01
 11000010100011001000001010001000101011110011001000100000001101 11
 11001110000010001101101100111100110111111100101001010000011101 10
 11001111010000010000111101010101011101101010111111111110110111 10
 01010010110101000100011110110000010101110000011011011100011100 10
 01101101001101110010110100110011010100011101110011001001011011 00
 01110000101100110110010111000111011000000011101001111101100010 01
```

```
01110001111010111011000111111111100110010101000000000011001000100
11111101011111101110100001011010111110011010100101100110100111100
10010111110110011101011111011101111110100110011101110101011000010
10011011010111011000111001101001100010101001111100000101101110 11
10011010000101000101101000000000001000111111101010100011101111
00000111100000010001001011100101000000100101001110001001001001 11
00111000011000100111100001100110000001001000100110011111110001101
00100101111001100011000010010010001101010010000110101110010001 00)
```

The output sequence of the fourth stage has period and linear complexity $4^5 = 1024$. The graph below shows that its linear complexity profile is nearly optimal, i.e. very close to the line $y = x/2$:



For all nonnegative integers k, the *k-error linear complexity* of a binary sequence is defined to be the smallest linear complexity that can be obtained by changing at most k bits of the sequence. The k-error linear complexity is an important indicator for the security of pseudorandom sequences: a sequence that has high linear complexity but is close to a sequence with low linear complexity, such as $(0^n 1)$, is cryptographically weak. See [6] for an algorithm which calculates the k-error complexity of a binary sequence of period 2^n. The k-error linear complexity of our sequence looks as follows:

k	k-error linear complexity
0	1024
1–254	514
255–256	513
257–510	257
511–512	1
≥ 513	0

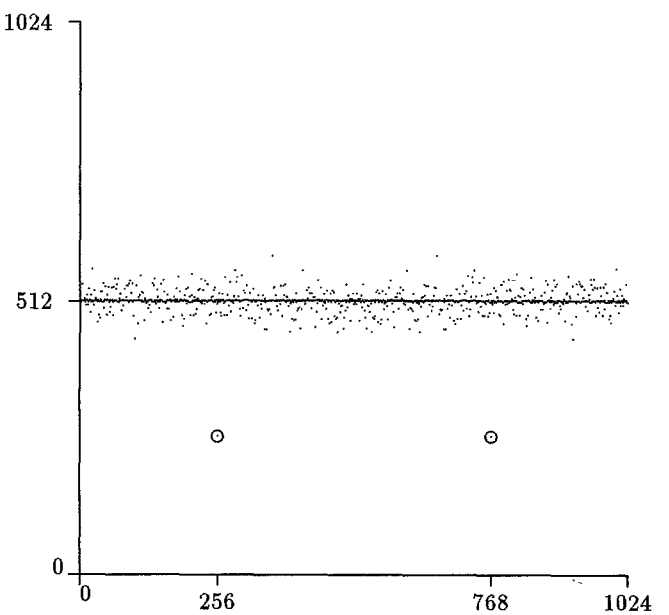
Note that the linear complexity drops down to 514 if a single error is allowed. This happens because we changed one bit of the output sequence in order to give it odd weight. So in fact, the output sequence has linear complexity 514 which is quite close to the minimum value 512 from Lemma 8 and certainly far from the expected value of 1023 for a random sequence. This indicates that the sequence might have some hidden structure.

The sequence contains 511 zeroes and 512 ones with the following run length distribution (compared to the ideal values for true random sequences):

l	1	2	3	4	5	6	7	8	9	10	11
# 0-runs of length l	126	71	31	13	8	4	2	0	1	0	1
# 1-runs of length l	131	61	36	12	8	5	2	0	0	2	0
ideal value	128	64	32	16	8	4	2	1	0	0	0

This run length distribution comes quite close to what might be expected from a random sequence.

Finally, we look at the autocorrelation function of the sequence. For a true random sequence, this should be constant at about 512 (half the length of the sequence):



From the distinctive low values at shifts of 256 and 768, it can bee seen that the sequence has some global structure. In fact, we found the following symmetries: If the first half of the sequence is XORed to the second, we obtain (except for the first bit which was changed in the output) the alternating sequence $0101\ldots$. This problem is caused by the output function: A shift of $512 = 2\gamma$ in the state sequence is equivalent to multiplication by $q^2 = -1$. Since $f(-g)$ equals $\overline{f(g)}$ if $g \in \langle q \rangle$ and $f(g)$ otherwise, the sequence $(s_i \oplus s_{i+512})_{i \geq 0}$ alternates between 0 and 1.

CRYPTOGRAPHIC STRENGTH. Our sequence has sufficiently large period and linear complexity, a good local linear complexity profile and a good run length distribution. Unfortunately, it shows some strong correlations over large distances, and will thus be cryptographically weak if large portions of the sequence are used. However, the experiments indicate that the sequence is quite secure if only segments up to length 256 are used.

6 Example 2: The quaternion group of order 16

In a similar way, we can define a group generator using the quaternion group of order 16.

THE GENERATOR. We use the quaternion group of order 16 defined by

$$G := \langle h, k \mid k^4 = h^2, h^4 = 1, (h,k) = k^{-2} \rangle$$

and the generators $g_0 := h$ and $g_1 := k^3 h$.

STATE SEQUENCE. Modulo the derived subgroup $G' = \langle k^2, h^2 \rangle$, the two generators commute and have order 2. The group G contains four elements of order 8, which form the coset $g_0 g_1 G' = \{k, kh^2, k^3, k^3h^2\}$. Hence q has order 8 iff one period of the clocking sequence contains an odd number of zeroes and an odd number of ones. In this case, the state sequence has least period 8γ.

The same argument as in Section 5 can be used to show that the state sequence hits every element of G the same number of times in one cycle.

OUTPUT FUNCTION. Again, we used GAP to search for a suitable output function. As the order of G is a square, bent functions on G may exist. By Theorem 4, such bent functions will not be balanced, but have either 6 or 10 zeroes. By exhaustive search, we have found 128 bent functions on G with 10 zeroes. The complements of these functions are the bent functions of G with 6 zeroes. Of these bent functions, we used the following as output function for the experiments:

1	h^2	h	h^3	k^2h	k^2h^3	k^2h^2	k^2	k	kh^2	kh	kh^3	k^3h	k^3h^3	k^3h^2	k^3
1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0

IMPLEMENTATION. There is an efficient four-bit representation of G similar to the one of the quaternion group of order 8 given in Section 5:

$$(a,b,c,d) \longleftrightarrow k^{2a} h^{2b} g_0^c g_1^d.$$

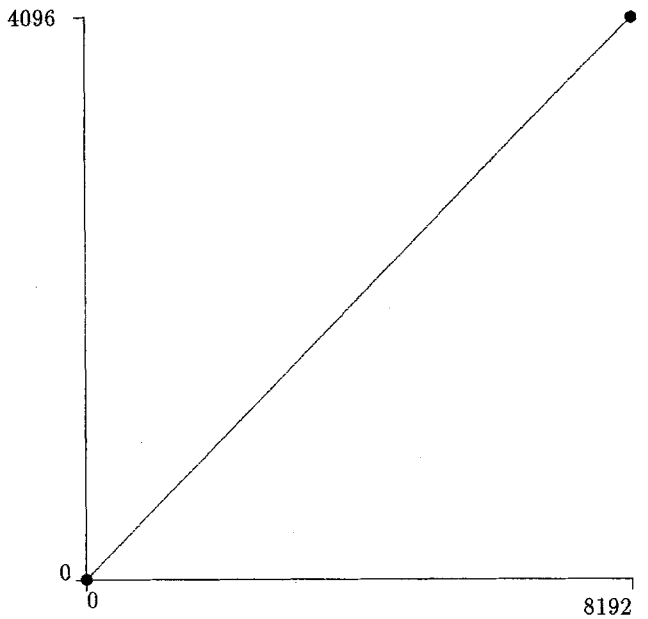
The choice of 256 possible output functions will give us up to 8 bits of key for each stage of a cascade.

Cascades. The comments in the previous section apply.

Experiments. We have also implemented a cascade of these generators in GAP. Each stage uses the output function from the table above. As with the Q_8-generator, we add the input of each stage to its output and change the first output bit to make sure that all sequences in the cascade have odd weight. Clocking a three-stage cascade with the sequence $(1,1,1,0,0,0,0,0)$, the output of the third stage is the following sequence of period and linear complexity $8^4 = 4096$.

```
( 1000100100011011010110111110101110000110000001000100100101110001010101100101101011000001110110000101
  0010101101001110110100110100101001000101010101100011001001010010101100100111111010001110110110111110
  0101001101000010110110100100100001010010110100010011001110000000110001100000100011001010111010011110
  0111011100000001100100000000101101011111110000010111001000010000001011001100010011010101100011110101 1
  1010111111100101111000100100101001011011010101011000110011111011000100011011111100001100011100100100 0
  0100110101001001100100101001111010111111101101011001011001000110110011100000101100010110111000000010
  1000101100000011011000000001001101110000011101000010101011001000111010000000110001010001111111000110 00
  0000000100010000000110100100010111101101100011111101110100101001011010001100011100011001011001010110
  1111101100010011110000000110010010011001000011011001010110011101001001110110001100100000100011101100 0
  1001001011100110110010101110101011010000111001100111010110000110010110100000000100101000101011000001
  1010110110001111110111011010010000110001111100010100000110001110011110111110010011001100001011011111
  0000001100110111100011111000000111100100011110011110000110110000000010101001011011111100000011000110 1
  0100001001000011000100101111100111101000011100011110010011110110100000111001100101000010010111001010
  0010011000000010001110010111011001100011101000001111011000001010110001000010100000010011100101111001 0
  0011000000011011010000010000010101101111011001101110000100101010000010110010011001010000110000110001
  0110011100100101111001001100100011100110101110000011010000010100110000011000010010001010101110000011
  1110011010110011000010000010000110101000110010101000110001001010001010001111111110111011011110101000
  1100101111010100111011110001110000111010101100011011001100101010101001010111011110000100101110100110
  1101011101101101001011000100101000111001100000100101101110010010011100011000000110001110111001000100
  1110110111101111001100100111100000111100011000000100000001111010001011101101111110011101001011111100
  0011100000100000010111101110101010100101011110101010111010001111010011000000011100100100110100010110
  1110011001101000011101100000100110001101001001010100011101011011110011110101001100011010101000000111
  1100010101000011100010000000001011001001001110000101011100101000111000110110111001011100001010001011
  1011111010011111010110100110111001110000110110000100011001101101111010110100010111001011001001001111
  1010100011011000010100100010000111101010101100110111001011000100110011000101001110000000101000010110
  1110111110010011010001110110011010010111101101110011000100110110000010100001000000011111001011110111 1
  1110001000000000100101001010011000001100110100100011111101111110100101111010001000110111101000001001 01
  0101000010101100111110100110000101111110010001011011011010001111101000110001100110000000000100001100
  1101010011100001000110101011110111000111100001001110000010001111001010011111000101111000110100111001
  1110101001000101101111110010001110100010110010010110110001001010011100111000001111110011101000010000 11
  0101001101111000010101101001010000100111111101000100000000010010110100000000000111010010111100110101 0
  1101001011001010111011000011110110111111110011001010011010100111001011111110110111110001011001011111
  1000111011111111110011011000111111101001001000101010100001100111100101001111110110000010010010001100
  0010111110000111101000010001001010110110111100001100010011011010001101001101110110101011001110101011
  1111011000010010111001001110100010101111001110001000101101000000000110011101100001001110010010001111
  1000111100011111111110001000101100110111010010001001011100001001100001011100111110000110011111010000
  1010100101011111101000101011000100111001101101011110010000111011010101110110101000101000111110100001 0
  0010011101000100001011111111001000000110010100011011111011010101101100101010101110100101000010001111 0
  1111010001110011101001100111111000111100001100000001101100111100111111100101000110100101011010110110
  0011110110001011100100110100000100001111000101101100111010111011011110000011001001001110010011011010
  1101011001000100110010011111001011001111000010101011111010001101011010110000011100111000011101 11 )
```

The linear complexity profile looks good as well:



Its k-error linear complexity looks similar to the one of the sequence in Section 5:

k	k-error linear complexity
0	4096
1–1022	2561
1023–1534	1026
1535–1790	321
1791–1918	41
1919–1982	7
1983–1984	2
≥ 1985	0

Again, one might argue that the relatively low linear complexity of 2561 obtained when changing the first bit back to its original value hints at some global structure.

The sequence contains 2111 zeroes and 1985 ones with the following run length distribution, which is close enough to the ideal distribution for true random sequences:

l	1	2	3	4	5	6	7	8			
# 0-runs of length l	522	269	114	61	33	12	8	2	1	1	1
# 1-runs of length l	486	267	137	66	32	15	11	4	3	3	0
ideal value	512	256	128	64	32	16	8	4	2	1	0

Finally, we have computed the autocorrelation function of the sequence. Over all shifts $\neq 0$, its minimum is 1920 and its maximum is 2222. All values being

CRYPTOGRAPHIC STRENGTH. So far, the test results indicate that we have found a good pseudorandom sequence. However, there are global correlations: When we change the first bit back to its original value, divide the sequence into four even parts and bitwise XOR them together, we obtain the all-zero sequence. In other words,

$$s_i \oplus s_{i+1024} \oplus s_{i+2048} \oplus s_{i+3072} = 0$$

for all i. Again, this shows that using large portions of the sequence should be avoided. This behaviour is caused by regularities in the output function similar to those discovered at the end of Section 5.

7 Conclusions and Open Problems

In this paper, we have generalised the notion of a clock controlled shift register to that of a register based on a finite group. We have shown how these registers can be cascaded to produce sequences of high period and linear complexity. Are the sequences produced by these cascades cryptographically secure ?

As pointed out in the sections above, the sequences produced by our generators do exhibit some structure. This structure certainly makes the use of especially long segments of the sequences unwise. However, if we suppose that only a portion of a period of the output sequence is used, the structures detailed in the previous sections do not seem to affect the security of the sequence since they involve terms of the output sequence that are very widely spaced. Since the local linear complexity profiles and run length distribution of the output sequences seem good, we have an indication that the sequences are secure when segments of reasonable length are used.

How can we maximise the assurance of security that our system gives ? The system depends on a careful choice of the output function of each generator in our cascade. We need to develop and expand the criteria given in Section 4. In particular, should we choose the output function to be bent? Maybe a function which only approximately satisfies the bent property but which performs better under other criteria is more appropriate. Further research – both experimental and theoretical – is needed on this matter. Is the form of the cascade we have used the most secure possible ? For example, we assume that the input to each stage is XORed with the output of the register. This operation seems to greatly improve the run length distribution of final output over that of a similar cascade with the XOR operation removed. Why is this, and can a different operation be introduced which increases the security of the output of the cascade ?

We believe that the concept of a bent function over a finite group is of interest in its own right, irrespective of its application in the situation outlined here. Do bent functions exist over any group whose order is an even square ? Certainly plenty of bent functions exist over the groups we have examined. Can large families of bent functions be constructed over certain families of groups ? The only constructions known so far apply only when the group is an elementary abelian 2-group.

In summary, the sequences produced by cascades of generators based on finite groups provide an interesting generalisation of the standard cascades of shift registers which can often be efficiently implemented. The output of such generators can have guaranteed minimum period and linear complexity. Experiments indicate that the sequences also have good linear complexity profiles and autocorrelation properties. However, further work is needed to establish that the sequences produced by cascades of this type are secure.

8 Appendix: well-known facts on period and linear complexity

For the convenience of the reader, this section contains a collection of well-known facts about period and linear complexity of binary sequences. These have been used in the proofs throughout this paper.

Lemma 5. *Let $(s_i)_{i\geq 0}$ be a periodic sequence of least period p. For $d, j \in N$, define the (shifted) d-decimation $(\sigma_i)_{i\geq 0}$ of (s_i) by $\sigma_i := s_{id+j}$. Then the following holds:*

(1) *The least period π of (σ_i) divides $p/(p,d)$.*
(2) *If $(p,d) = 1$ then $\pi = p$.*
(3) *For $(p,d) > 1$, π may be strictly less than $p/(p,d)$.*
(4) *If (s_i) has an irreducible minimal polynomial, then either $\pi = p/(p,d)$ or (σ_i) is the zero sequence.*

Proof. **(1)** For all i, we have

$$\sigma_{i+p/(p,d)} = s_{(i+p/(p,d))d+j} = s_{id+j+\text{ lcm }(p,d)} = s_{id+j} = \sigma_i,$$

hence $\pi | p/(p,d)$.

(2) For all i, we have

$$s_{(id+j)+\pi d} = s_{(i+\pi)d+j} = \sigma_{i+\pi} = \sigma_i = s_{id+j}.$$

As $(p,d) = 1$, $\{id + j \bmod p \mid i \geq 0\} = \{0, \ldots, p-1\}$, hence $s_{i+\pi d} = s_i$ for all $i \geq 0$. It follows that $p | \pi d$. Since $(p,d) = 1$, we have $p | \pi$. Together with (1), our claim follows.

(3) Example: Decimating the sequence 100010 by $d = 2$, we obtain 101 for $j = 0$ and 000 for $j = 1$.

(4) [4, Ex. 9.5,p. 364].

Lemma 6. *Let $(s_i)_{i\geq 0}$ be a periodic sequence of least period md. For $j \geq 0$, let k_j denote the least period of the decimated sequence $(s_{id+j})_{i\geq 0}$. (Obviously, $k_j = k_{j \bmod d}$.) Then $m = lcm(k_0, \ldots, k_{d-1})$.*

Proof. Let $l := \text{lcm}(k_0, \ldots, k_{d-1}) = \text{lcm}(\{k_j\}_{j\geq 0})$. For all $j \geq 0$, we have $s_{ld+j} = s_j$ since l is a multiple of the period k_j. It follows that (s_i) is ld-periodic, hence $md \mid ld$ and $m \mid l$.

On the other hand, $k_j \mid m$ for all j by Lemma 5(1), hence $l \mid m$.

Lemma 7. *Let $(s_i)_{i\geq 0}$ be a sequence of least period p^k. If we decimate this sequence by p^n, the least common multiple l of the least periods of all such decimations equals p^{k-n} for $k > n$ and 1 otherwise.*

Proof. Follows from the previous lemma and Lemma 5(1).

Lemma 8. *The linear complexity of a sequence $(s_i)_{i\geq 0}$ of least period p^n over a finite field of characteristic p is at least p^{n-1}. It equals p^n iff $s_0+s_1+\ldots+s_{p^n-1} \neq 0$.*

Proof. Since $x^{p^n} - 1 = (x-1)^{p^n}$ in characteristic p, the minimal polynomial of s has the form $(x-1)^l$ for some $l \leq p^n$. For $l < p^{n-1}$, it would divide $x^{p^{n-1}} - 1$, and the minimal period would be a divisor of p^{n-1}, contradicting our initial assumption. Hence $l \geq p^{n-1}$.

Let $s(x) := \sum_{i\geq 0} s_i x^i$ denote the generating function of s. Since $l \leq p^n$, we have $(x-1)^{p^n} s(x) = 0$. It follows that $l = p^n$ iff

$$(x-1)^{p^n-1} s(x) = \frac{x^{p^n}-1}{x-1} s(x) = (x^{p^n-1} + x^{p^n-2} + \ldots + x + 1)s(x) \neq 0,$$

which proves the second claim.

References

1. P. Diaconis, *Group Representations in Probability and Statistics*, Institute of Mathematical Statistics Lecture Notes – Monograph Series, 11, Hayward (CA), 1988.
2. D. Gollmann, W.G. Chambers, *Clock-Controlled Shift Registers: A Review*, IEEE J-SAC 7/4 (1989), 525–533.
3. M. Schönert et al., *GAP: Groups, Algorithms and Programming*, RWTH Aachen, 1992.
4. R. Lidl, H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, 1986.
5. R.A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer, 1986.
6. M. Stamp, C.F. Martin, *An Algorithm for the k-Error Linear Complexity of Binary Sequences with Period 2^n*, IEEE Trans. Inform. Theory 39/4 (1993), 1398–1401.