On Random Mappings and Random Permutations

W. G. Chambers

Department of Electronic and Electrical Engineering,
King's College London, Strand, London WC2R 2LS, UK

w.chambers@kcl.ac.uk

1 Introduction

Much work has been done by many people, including the present author, to prove that certain classes of sequence-generator have guaranteed periods [1]. Here we examine what happens at the other extreme, with sequence generators which are finite-state machines modelled as having random next-state tables. The advantages are a lack of mathematical structure which might provide an entry for the cryptanalyst, and a huge choice of possibilities; the disadvantages are that there are no guarantees on anything, and as is well known there is a risk of getting a very short period.

Thus we consider a finite-state machine whose state is specified by an integer in the range $0, ...N-1$, and which has a next-state function F which specifies the $n+1$-th state s_{n+1} as $F(s_n)$ with s_n the n-th state. The output can also be regarded as a function of the state, but we are not so much concerned with this. Evidently the function F can be represented by a look-up table with addresses and entries in the range $0, ...N-1$. Each function F corresponds to a state-diagram where the states correspond to N points $0, ...N-1$, with point i (the predecessor) joined to point j (the successor) by an arrowed line if $F(i) = j$. Points lying on a loop or *cycle* are called cyclic points; a point i satisfying $F(i) = i$ is regarded as lying on a cycle of length 1. Every point has a unique successor, but some points have no predecessors and some have more than one predecessor. In general the state diagram will consist of a number of cycles, plus a number of directed trees rooted on cyclic points; in these trees the arrows are pointing to the root.

If the next-state function F is invertible, so that for every j in $0, ...N-1$ there is an i satisfying $F(i) = j$, then the function will be called a permutation (an N-permutation), and the state-diagram will consist of a number of cycles without any trees rooted in them.

There are altogether N^N N-functions, and $N!$ N-permutations. Typically the state is represented by a number of bits, say n, so that we have $N = 2^n$. Normally n would be of the order of hundreds.

There is a considerable literature on this topic [2], [3], [4], [5], [6], [7].

2 Problems with Random Mappings

Random mappings in cryptography have attained a considerable notoriety, perhaps akin to the "Fool's Mate" in chess, as a trap for novices [2], [8]. Here are three reasons why random N-functions are best avoided:

1) On average, if we start from a specified state or point, we traverse of the order of $\sqrt{N}$points before reaching a cyclic point, and the cycle we find ourselves on is typically of length of order $\sqrt{N}$.

2) A large number of starting points, of the order of $N/2$, are liable to end up on the same cycle.

3) It is not unlikely that the terminating cycle is a small fraction of $\sqrt{N}$. In fact the probability of finding a cycle of size $\beta\sqrt{N}$ is of the order of β, for small β.

Thus it can be shown that the probability of choosing at random an N-function with a tree exceeding ρN in size $(0.5 < \rho < 1)$ is asymptotically (for large N) equal to $\rho^{-1/2}-1$, so that we can find a tree of size $0.8N$ with probability 11.8%. It can also be shown [7] that the probability of finding a tree of size $> N/2$ rooted on a cycle of size $< \beta\sqrt{N}$ with $\beta \ll 1$ is about $\beta\sqrt{(2/\pi)}$.

On the other hand there are some points in favour of using general non-invertible mappings. We may be able to compensate for the "square-root" effect by doubling the number of bits specifying the state. Moreover the proof that a mapping is invertible is often constructive, telling us how to do it. This may be a source of cryptographic weakness.

3 Simple Properties of Random Permutations

The properties of random N-permutations are much more satisfactory than those of general random mappings. Thus we have the following elementary results:

1) The probability that two points lie on the same cycle of a randomly chosen N-permutation is $1/2$.

2) The probability of a given point lying on a cycle of length $\leq r$ (with $1 \leq r \leq N$) is simply r/N.

Thus although it is possible to choose a point on a 1-cycle the probability of this occurring is $1/N$, which is normally very tiny.

4 Simulating Cycle-lengths

In order to compare the cycle-structure of a given permutation with that of a typical random permutation it is useful to have a way of simulating the numbers and sizes of cycles in a random permutation. This can be done as follows: Suppose we are building up a permutation cycle by cycle and we have so far built up cycles containing all but R points. We now wish to find the probability that the next cycle to be formed has length l. From the points not yet included in the previous cycles we choose the unique point P whose numerical value is the least. Then we

have to choose $l-1$ further points not equal to P from the R points not included in the previous cycles. These are used to build up a string starting at P which is finally closed by the choice of P. The probability of obtaining a cycle of given length l in this way is $1/R$ if $l = 1$, and is

$$\frac{R-1}{R}\frac{R-2}{R-1}\cdots\frac{R-l+1}{R-l+2}\cdot\frac{1}{R-l+1} = \frac{1}{R}$$

if $l > 1$. Thus the probability is independent of l for $1 \leq l \leq R$, and this suggests the following Nim-like construction for simulating the numbers and the lengths of the cycles in a random N-permutation. We start with N objects. At any stage when there are R objects left ($R \geq 1$) we choose at random an integer U in the range $1, \ldots, R$ with uniform probability. This gives the length of the next cycle. Then we decrease R by U and iterate unless R is now 0, when we stop. It is quite easy to simulate 10^8 permutations in this way.

5 Test Results with 2^{24} States

We carried out a test run on the generator

$$x_{n+3} = P[(ax_{n+2} + bx_{n+1} + x_n) \bmod 256], \tag{1}$$

where the 8-bit invertible mapping $P[.]$ was based on a method by Wheeler [9]. The x_n are 8-bit unsigned integers, and a and b are randomly chosen odd integers. This generator has 2^{24} states. We grouped the cycle-lengths l into ranges, Range-0 for $l = 1$, Range-1 for $l = 2$, Range-2 for $l = 3$ or 4, Range-3 for l from 5 to 8, up to Range-24 for l from $2^{23} + 1$ to 2^{24}. For each permutation we counted the number of cycle-lengths in each range. For the ranges corresponding to the shorter cycle lengths the distribution of counts is approximately Poisson, with a mean for any such range equal to the sum of the reciprocals of the lengths in the range. This was done for about 5000 permutations altogether, and the results scaled remarkably well with the results obtained by simulating the cycle-lengths of random permutations as described in the previous section. (See Table 1 for the counts found in this case.) Unstructured random permutations for P were also tried out, with similar results.

6 Generators Driven by a Periodic Input

One obvious way of reducing the risk of a short period is to drive the finite-state machine with a periodic input of period P from a sequence generator of guaranteed period. If P is very large, then nothing much can be said in general terms, but if P is very much less than the lengths of typical output sequences, then we have the following curious situation. The output can be regarded as the interleaving of P sequences generated by P finite-state machines with distinct structures. Thus if it is possible to de-interleave the sequences there is roughly a P-fold increase in the probability of one of them being found with a period

R	c=0	c=1	c=2	c=3	c=4	c=5	c=6	c=7	c=8	c=9	c=10
0	36792375	36781875	18396794	6129424	1533219	306707	51345	7226	919	109	7
1	60653544	30326549	7580746	1263713	158201	15826	1308	109	3	1	0
2	55810493	32542111	9496555	1847368	269100	31109	2994	239	29	2	0
3	53013429	33639358	10681920	2255302	359219	45440	4831	461	39	1	0
4	51529292	34170772	11323363	2501571	413340	54986	6042	582	51	1	0
5	50772158	34419894	11659964	2633092	447037	60395	6749	636	70	5	0
6	50391787	34533668	11836403	2704233	462232	63659	7261	682	70	5	0
7	50187758	34601249	11925662	2740190	471433	65429	7518	678	72	11	0
8	50096923	34628447	11968518	2756208	475991	65603	7474	775	56	5	0
9	50051307	34635363	11993991	2766419	477998	66537	7527	786	69	3	0
10	50020982	34647034	12006632	2769755	480499	66553	7788	685	69	3	0
11	50015585	34646548	12010145	2771860	480311	67100	7651	733	62	4	1
12	49996515	34663898	12010142	2773626	480899	66338	7779	736	64	3	0
13	49991107	34666006	12012180	2774882	480802	66399	7791	757	71	4	1
14	50001789	34657477	12006765	2776918	482266	66412	7567	732	69	5	0
15	49988613	34662906	12014393	2777989	481106	66386	7761	776	65	4	1
16	49997512	34656565	12012673	2777731	481101	65970	7633	741	69	3	2
17	49983626	34663103	12018489	2779363	480395	66495	7709	740	71	9	0
18	49985423	34657053	12019573	2779867	482147	67377	7735	736	82	6	1
19	49974912	34662769	12022805	2782080	481685	67147	7776	759	62	5	0
20	49969519	34664176	12024126	2784710	481899	66919	7770	798	73	10	0
21	49960804	34659303	12031792	2788560	483573	67427	7712	764	59	6	0
22	49968911	34645032	12027232	2836971	417030	101925	2899	0	0	0	0
23	53206024	25733028	19571408	1489540	0	0	0	0	0	0	0
24	30739151	69260849	0	0	0	0	0	0	0	0	0

Table 1. Table 1: Results from 10^8 simulated permutations on 2^{24} objects. The column labelled R gives the range-number (Section 5). A column labelled c=n gives the number of permutations with n cycles of lengths in the appropriate range. Thus the number 740 under c=7 with R=17 means that 740 permutations were found with 7 cycles of lengths in Range-17, that is of lengths from $2^{16}+1$ to 2^{17}.

shorter than some stipulated value, in comparison with the original undriven machine. Thus suppose we have a 32-bit random finite-state machine; then there is a chance of roughly 2^{-16} of ending up in a 1-cycle. If this machine is driven in some way with a sequence of period $P = 2^{16}$ then the output will consist of P interleaved sequences, one of which may well have final period 1. Thus in the output there may be a value which is repeated every 2^{16} iterations.

On the other hand, the input period gives an almost certain guarantee on the output period, so that one way of allaying worries about short periods is to use an input sequence of very long period.

7 Generators that Modify their own Look-up Tables

Oddly enough, at the December 1993 workshop on "Fast Software Encryption" [10] there were no suggestions for generators that modify their own look-up tables

as they run. There are certain practical objections to such a scheme of course, for instance if the generator needs to be restarted frequently or if random access to the key-stream is needed, but if one simply needs a long key-stream without any restarts such a method is feasible and of course it makes it harder for the cryptanalyst by presenting him as it were with a moving target.

If we have such a scheme then the internal description of the machine-state must include the look-up table. Now the point is the following: if the transformation from one state to the next is invertible, then we have a permutation for the next-state function, but otherwise we may be dealing with a general finite-state machine with its likelihood of much smaller loop sizes. Thus it would seem that the modification of the look-up tables should be carried out in a way that enables one to undo the transformation in a unique manner.

Thus as a simple example consider the following random-number generator proposed by Bob Jenkins and publicised on the Usenet service. The state variables are

`a, b`: Unsigned 32-bit integers
`m[0]..m[255]`: a lookup table of unsigned 32-bit integers
`p`: a counter cycling from 0 to 255

Initially `p` is set to zero and the other variables to random values. There are two internal unsigned 32-bit integer variables, `x` and `y`. Addition is carried out "modulo 2^{32}". We loop indefinitely on the following instructions:

```
x = m[p];
y = m[RS(x)]; /* RS(x) is a right-shift of x by 24, leaving an 8-bit result */
a = R(a); /* R() is a rotation left by 27 bits */
a = a + y; /* addition is mod 2^32 */
m[p] = a + b; /***/ /* see below */
output(b+y); /* put the value b+y on the output stream */
b = x;
p = (p + 1) mod 256; /* Step p */
```

The instruction marked /***/ evidently changes the lookup table. It is not hard to see that if we know the state variables at the end of this loop we could determine them at the start, including the value of the modified table-entry; thus we have what is in effect a permutation. The same thing would apply if we replaced /***/ by `m[p] = m[p] + a + b`, or `m[p] = m[p] XOR (a + b)`, but not if we replaced /***/ by `m[p] = m[p] + a`, as we could not then deduce the initial value of `b` from the final values of the state variables.

8 A Cipher Claimed to Resemble A5

Two other encryption algorithms have recently been publicised on the Internet, without much theoretical backing. The first is "alleged RC4", which has similarities to the algorithm just described. Here the next state function is invertible.

The second (announced by Ross Anderson and Michael Roe) is purported to be very similar to the A5 cipher used in the GSM mobile telephone system [11]. It uses three binary linear feedback shift-registers with known (key-independent) primitive polynomials of degrees 19, 22 and 23 respectively. These registers are initially set in a key-dependent manner to non-zero values, and on each iteration they are stepped as follows: A control-bit is taken from a known position near the centre of each register. If two or three of the control bits are equal to 1, then the registers producing these bits are stepped. On the other hand if two or three of the control bits are 0, then the registers producing *these* bits are stepped. In effect the registers are mutually clock-controlled in a stop/go fashion, and it is easy to see that there is a probability of 3/4 that a register is stepped on any iteration of the algorithm. Thus the longest register would be expected to go through a complete cycle in roughly $P = (2^{23} - 1).4/3$ iterations.

Regarded as a finite-state machine the system has just under $2^{19+22+23} = 2^{64}$ states, and the next-state function is non-invertible. Thus we would expect to find eventual periodicities of the order of 2^{32}, after a precursor sequence of the same order of length. Instead, in a search for eventual periodicities the author found 237 cases (all distinct), and all of them had periods very close to small multiples of $P = (2^{23}-1)\cdot 4/3$; moreover just over 40% of these cases had periods very close to P itself. (The precursor sequences were on the whole a little longer, of lengths something like $10P$ on average.) Evidently the shorter registers, one with a period very close to $P/16$ and the other with a period very close to $P/2$, are locking on to the period of the longest register, with respectively 16 cycles and 2 cycles for every cycle of the longest register. Further investigations have shown that this "lock-in" is a robust phenomenon, occurring independently of the choice of primitive polynomials, and even occurring if the three sequences of control bits are chosen as random periodic sequences with periods equal to numbers of the form $2^n - 1$. This topic is being pursued further.

The shortness of the period is probably not a hazard in normal use [11], where only a few hundred bits of output are required between key-changes, but perhaps one would be advised not to use this scheme as a random-number generator without further study.

9 Concluding Remarks

Encryption algorithms in which the look-up tables are continuously modified have the attractions of high speed and of making the analyst's task harder, but there may be a lingering doubt about the period, particularly when there is no significant theory available. The above discussion strongly suggests that the next-state function should be invertible, but one might like some further reassurance. The use of a "rekeyed" cipher is a good way of obtaining a guaranteed huge period. Here there is "driving" sequence generator D whose output is used to modify the state of a second generator G which provides the final output. The generator D has a known or lower-bounded huge value for its period. Thus a 32-bit cascade generator with a cascade of length 8, as suggested in [1], will have

a period of $(2^{32} - 1)^8$. The speed of the driving generator D is not so critical, since the output generator G need not be rekeyed on every iteration. In this way we combine a generator D with fixed (but key-dependent) lookup tables with a generator G where the tables are continuously modified, and keep the advantages of both.

References

1. Chambers WG, "Two stream ciphers", *Lecture Notes in Computer Science*, **809**, 51-55 (1994)
2. Knuth D, *The art of computer programming, Vol 2: Seminumerical algorithms*, 2nd ed., Chapter 3 (Reading, Mass: Addison-Wesley) 1981
3. Kolchin VF, *Random Mappings*, (Optimization Software Inc., 1986), para 3.3
4. Flajolet P, Odlyzko AM, "Random mapping statistics", *Lecture Notes in Computer Science*, **434**, 329-354 (1990)
5. Arratia R, Tavare S, "The cycle structure of random permutations", *Annals of Probability*, **20**, 1567-1591 (1992)
6. Mutafchiev L, "Large trees in a random mapping pattern", *European Journal of Combinatorics*, **14**, 341-349 (1993)
7. DeLaurentis JM, "Components and cycles of a random function", *Lecture Notes in Computer Science*, **293**, 231-242 (1988)
8. Lewin R, *Complexity*, (London: J M Dent Ltd) 1993, p28
9. Wheeler D, "A bulk data-encryption algorithm", *Lecture Notes in Computer Science*, **809**, 127-134, (1994)
10. Anderson R (Ed.), "Fast Software Encryption", *Lecture Notes in Computer Science*, **809** (1994, Springer-Verlag)
11. Mouly M, and Pautet M-B, *The GSM System for Mobile Communications*, (1992: published by M. Mouly and Marie-B. Pautet, 49, rue Louise Bruneau, F-91120 Palaiseau, France), p249 and p481