Bad and Good Ways of Post-Processing Biased Physical Random Numbers

Markus Dichtl

Siemens AG
Corporate Technology
81730 München
Germany
Email: Markus.Dichtl@siemens.com

Abstract. Algorithmic post-processing is used to overcome statistical deficiencies of physical random number generators. We show that the quasigroup based approach for post-processing random numbers described in [MGK05] is ineffective and very easy to attack. We also suggest new algorithms which extract considerably more entropy from their input than the known algorithms with an upper bound for the number of input bits needed before the next output is produced.

Keywords: quasigroup, physical random numbers, post-processing, bias

1 Introduction

It seems that all physical random number generators show some deviation from the mathematical ideal of statistically independent and uniformly distributed bits.

Algorithmic post-processing is used to eliminate or reduce the imperfections of the output. Clearly, the per bit entropy of the output can only be increased if the post-processing algorithm is compressing, that is, more than one bit of input is used to get one bit of output.

Bias, that is a deviation of the 1-probability from 1/2, is a very frequent problem. There are various ways to deal with it, when one assumes that bias is the only problem, that is, the bits are statistically independent.

In the rest of the paper, it is assumed that the physical random number generator produces statistically independent bits. It is also assumed that the generator is stationary, that is, the bias is constant.

We show that the post-processing algorithm based on quasigroups suggested in [MGK05] is ineffective. We describe an attack on the post-processed output bits which requires very little computational effort.

We show that post-processing algorithms for biased random numbers which produce completely unbiased output cannot have upper bounds on the number of input bits required until the next output bit is produced. We describe new algorithms for post-processing biased random numbers which have a fixed number of input bits. The new algorithms extract considerably more entropy from the input than the known algorithms with a fixed number of input bits.

2 An ineffective post-processing method for biased random numbers

At FSE 2005, S. Markovski. D. Gligoroski, and L. Kocarev suggested in [MGK05] a method based on quasigroups for post-processing biased random numbers. In this paper, we only give results for their E-algorithm, but all results can be transferred trivially to the E'-algorithm.

2.1 The E-transform

A quasigroup $(A, *)$ of finite order s is a set A of cardinality s with an operation $*$ on A such that the operation table of $*$ is a Latin square (that is, all elements of A appear exactly once in each row and column of the table).

The mapping $e_{b_0,*}$ maps a finite string of elements $a_1, a_2, \ldots, a_n$ of A to a finite string $b_1, b_2, \ldots b_n$ such that $b_{i+1} = a_{i+1} * b_i$ for $i = 0, 1, \ldots, n-1$. b_0 is called the leader of the mapping $e_{b_0,*}$. The leader must be chosen in such a way that $b_0 * b_0 \neq b_0$ holds.

For a fixed positive integer k, the E-transform of a finite string of elements from A is just the k-fold application of the function $e_{b_0,*}$. Here we deviate slightly from the terminology of [MGK05]. There, the E-transform allows different leaders for the k applications of the function e, but later in the description of the post-processing algorithm the leader is a fixed element.

2.2 True and claimed properties of the E-transform

Why the suggested quasi group approach is ineffective for post-processing biased random numbers Theorem 1 of [MGK05] states correctly that the E-transform is bijective. As a consequence of this theorem the E-transform is ineffective for post-processing biased random numbers. As a bijection, it cannot extract entropy from its input bits. The entropy of its output bits is just the same as the entropy of the input bits. Of course, applying a bijection to some random bits does not do any harm either.

The output of the E-transform is not uniformly distributed Theorem 2 of [MGK05] claims incorrectly that the substrings of length $l \leq k$ are uniformly distributed in the E-transform of a sufficiently long arbitrary input string.

Let x be an element of the quasigroup A. The E-transform is a bijection, so for each string length n, there is an input string w which is mapped by the E-transform to the string with n times x in sequence. Clearly, all substrings of the E-transform of w are just repetitions of x. Hence, the distribution of substrings of length $l \leq k$ in the E-transform of w is as far from uniform as possible.

When we look at the proof of the theorem, we see that the authors do not deal with an arbitrary input string, but with one where all elements of the string are chosen randomly and statistically independently according to a fixed distribution. In their proof, the authors do not try to show that the distribution of the output substrings is exactly uniform, but refer to the stationary state of a Markov chain. So their result could hold at most asymptotically. However, the Lemma 1 they use in their proof, is completely wrong. Let x be the input string for the first application of $e_{b_0,*}$. Then the lemma states that for all m the probability of a fixed element of A at the mth position of $e_{b_0,*}(x)$ is approximately $1/s$, s being the order of the quasigroup. To see how wrong this is, we consider a highly biased generator suggested in [MGK05]. We assume that the probability of 0-bits is 999/1000 and the probability of 1-bits is 1/1000. The bits are assumed to be statistically independent. We name this generator HB (highly biased). (The value of the bias is -0.499 .) For post-processing, we use a quasigroup $(\{0, 1, 2, 3\}, *)$ of order 4. We map each pair of input bits bijectively to a quasigroup element by using the binary value of the pair. Hence the post-processing input 0 has probability 0.998001, 1 and 2 have probability 0.000999, and the probability of 3 is 0.000001. The first element of $e_{b_0,*}(x)$ depends only on the first element of x. So, one element of A appears as first element of the output string with probability 0.998001, two with probability 0.000999, and one with probability 0.000001. All these values are very far away from the value 1/4 suggested by the lemma.

One should note that the generator HB, which we consider several times in this paper, was not constructed to demonstrate the weakness of the quasigroup post-processing, but that the authors of [MGK05] claim explicitly that their method is suited for post-processing the output of HB.

2.3 What is really going on in the E-transform

The elements at the end of the output string of the E-transform approach the uniform distribution very slowly, when the input string from a strongly biased source is growing longer. This is shown in Figure 1. We used the quasigroup of order 4 from the Example 1 in [MGK05] and the leader 1 with $k = 128$ (as suggested by the authors of [MGK05] for highly biased input) to post-process 10000 samples of 100000 bits (50000 input elements) from the generator HB.

To achieve an approximately uniform distribution, about 50000 input elements, or 100000 bits have to be processed.

Now, it would be wrong to conclude that everything is fine after 100000 bits. Since the E-transform is bijective for all input lengths, the entropy of the later output elements is as low as the entropy of the first ones. How can the entropy of the later output bits be so low if they are approximately uniformly distributed? The low entropy is the result of strong statistical dependencies between the output elements. For the later output elements, the E-transform just replaces one statistical problem, bias, with another one, dependency. Of course one cannot expect anything better from a bijective post-processing function.

2.4 Attacking the post-processed output of the E-transform

Since the E-transform is bijective, it is, from an information theoretical point of view, clear that its output resulting from an input with biased probabilities can

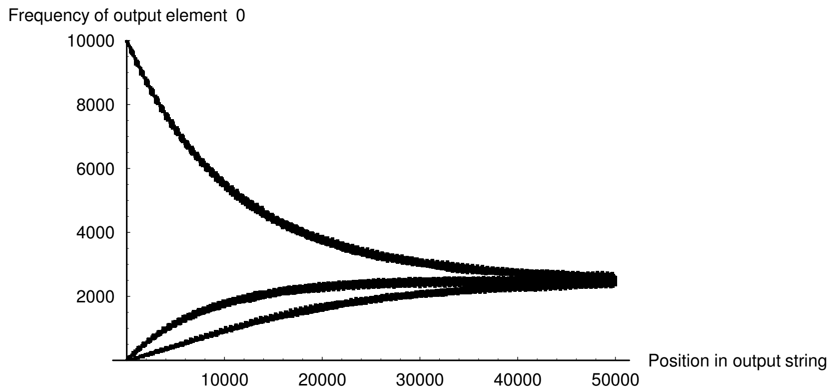

Fig. 1. Frequencies of the element 0 in the post-processed output of the generator HB. For each position in the output string, one dot indicates the frequency of the output element 0 at that position in 10000 experiments. To be visible, the dots have to be so large the individual dots are not discernible and give the impression of three "curves". For subsequent output elements, the output frequency sometimes jumps from one of the three "curves" visible in the figure to another one.

be predicted with a probability greater than $1/s$, where s is the cardinality of A. We will show now that the computation of the output element with the highest probability is very easy.

The first output element of the E-transform is the easiest to guess. We just choose the string containing the most frequent input symbol and apply the E-transform to get the most probable first output element. For predicting later output elements, we have to know the previous input elements. They can be easily computed from the output elements. Due to the Latin square property of the operation table of $*$, the equation $b_{i+1} = a_{i+1} * b_i$, which defines $e_{b_0,*}$ can be uniquely resolved for a_{i+1} if b_{i+1} and b_i are given. To the previous input elements we append the most probable input symbol and apply the E-transform. The last element of the output string is the most probable next output element. Our prediction is correct if the most probable input symbol occurred. For the generator HB and a quasigroup of order 4, this means that we have a probability of 0.998001 to predict the next output element correctly. The computational effort for the attack is about the same as for the application of the E-transform.

2.5 Attacking unknown quasigroups and leaders

The authors of [MGK05] do not suggest to keep secret the quasigroup and the leader chosen for post-processing the random numbers. We show here for one set of parameters suggested in [MGK05] that even keeping the parameters secret would not prevent the prediction of output bits. For the attack, we assume that the attacker gets to see all output bits from post-processing, and that she may restart the generator such that the post-processing is reinitialized. Since the post-processing is bijective, an attacker can learn nothing from observing the post-processed output of a perfect random number generator. Instead, we consider again the highly biased generator HB.

We choose the parameters for the post-processing function according to the suggestions of [MGK05]. The order of the quasigroup was chosen to be 4, and the number k of applications of $e_{b_0,*}$ as 128.

Normalised Latin squares of order 4 have 0, 1, 2, 3 as their first row and column. By trying out all possibilities, one sees easily that there are exactly 4 normalised Latin squares of order 4. By permuting all four columns and the last three rows of the 4 normalised Latin squares we find all possible Latin squares of order 4. Hence, there are $4 \cdot 4! \cdot 3! = 576$ quasigroups of order 4. With 4 leaders for each, this would mean 2304 choices of parameters to consider for the E-transform. However, we have to take into account that the leader l may not have the property $l * l = l$. After sorting out those undesired cases, we are left with 1728 cases.

From the observed post-processed bits, we want to uniquely determine the parameters used.

The attack is very simple. We apply the inverse E-transform for all 1728 possible choices of quasigroup and leaders to the first n of the observed postprocessed bits. With high probability, the correct choice of parameters is revealed by an overwhelming majority of 0 bits in the inverse. When choosing $n = 32$, the correct parameters are uniquely identified by an inverse of 32 0 bits in 61 % of all cases. Using more post-processed bits, we can get arbitrarily close to certainty about having found the right parameters. This variant of the attack is due to an anonymous reviewer of FSE 2007, mine was unnecessarily complicated.

When we have determined the parameters of the E-transform, the attack can continue as described in the previous section.

This finishes our treatment of quasigroup post-processing for true random numbers.

3 Two classes of random number post-processing functions

We have seen that bijective methods of post-processing random numbers are not useful, as they can not increase entropy. We have to accept that efficient post-processing functions have less output bits than input bits. This paper only deals with the post-processing of random bits which are assumed to be statistically independent, but which are possibly biased. The source of randomness is assumed to be stationary, that is the bias does not change with time. Although the bias is constant, it is unrealistic to assume that its numerical value, that is the deviation of the probability of 1 bits from 1/2, is exactly known. A useful post-processing function should work for all biases from a not too small range.

Probably the oldest method of post-processing biased random numbers was invented by John von Neumann [vN63]. He partitions the bits from the true random number generator in adjacent, non-overlapping pairs. The result of a 01 pair is a 0 output bit. A 10 pair results in a 1 output bit. Pairs 00 and 11 are just discarded. Since the bits are assumed to be independent and the bias is assumed to be constant, the pairs 01 and 10 have exactly the same probability. The output is therefore completely unbiased.

However, this method has two drawbacks: firstly, even for a perfect true random number generator it results in an average output data rate 4 times slower than the input data rate, and secondly, the waiting times until output bits are available can become arbitrarily large. Although pairs 01 or 10 statistically turn up quite often, it is not certain that they will do so within any fixed number of pairs considered. This is a unsatisfactory situation for the software developer who has to use the random numbers. In many protocols, time-outs occur when reactions to messages take too long. Although the probability for this event can be made arbitrarily low, it can not be reduced to zero when using the von Neumann post-processing method.

The low output date rate may be overcome by using methods like the one described by Peres [Per92]. A very general method for post-processing any stationary, statistcally independent data is given in [JJSH00]. The algorithm described there can asymptotically extract all the entropy from its input. So the data rate is asymptotically optimal. The problem of arbitrarily long waiting times, however, can not be solved. This is proved in the following theorem:

Theorem 1. *Let f be an arbitrary post-processing function which maps n bits to one bit, n being a positive integer. Let $X_1, \ldots, X_n$ be n independent random bits with the same probability p of 1-bits. Let S be an infinite subset of the unit interval. Then $Pr[f(X_1, \ldots, X_n) = 1] \neq 1/2$ holds for infinitely many $p \in S$.*

Proof: $\Pr[f(X_1, \ldots, X_n) = 1]$ is a polynomial of at most n-th degree in p. Its value for $p = 0$ is either 0 or 1. If the function is constant, it is unequal 1/2 on the whole unit interval. If it is a non-constant polynomial, there are at most n p-values, for which it assumes the value 1/2. For all other $p \in S$ holds $\Pr[f(X_1, \ldots, X_n) = 1] \neq 1/2$. q. e. d.

So we have two classes of functions for post-processing true random numbers: those with bounded numbers of input bits to produce one bit of output, and those with unbounded. We can reformulate our theorem as

An algorithm for post-processing biased, but statistically independent random bits with a bounded number of input bits for one output bit cannot produce unbiased output bits for an infinite set of biases.

For practical implementations, an unbounded number of input bits means an unbounded waiting time until the next output is produced.

So we have to accept some output bias for bounded waiting time post-processing functions, but we will show subsequently how to make this bias very small.

Although not as well known as the von Neumann post-processing algorithm, some algorithms with a fixed number of input bits are frequently used to improve the statistical quality of the output of true random number generators.

Probably the simplest method is to XOR n bits from the generator in order to get one bit of output where n is a fixed integer greater than 1.

Another popular method is to use a linear feedback shift register where the bits from the true random number generator are XORed to the feedback value computed according to the feedback polynomial. The result of this XOR operation is then fed back to the shift register. After clocking m bits from the physical random number generator into the shift register, one bit of output is taken from a cell of the shift register. This method tries to make use of the good statistical properties of linear feedback shift registers. Even when the physical source of randomness breaks down completely and produces only constant bits, the LFSR computes output which at first sight looks random. When the bits from the physical source are non-constant but biased, some of the bias is removed, but additional dependencies between output bits are introduced by this LFSR method.

4 Improved random number post-processing functions with a fixed number of input bits

4.1 The concrete problem considered

We consider the following problem: we have a stationary source of random bits which produces statistically independent, but biased output bits. Let p be the probability for a 1 bit. The post-processing algorithm we are looking for must not depend on the value of p. The post-processing algorithm uses 16 bits from the physical source of randomness in order to produce 8 bits of output. Our aim is to choose an algorithm such that the entropy of the output byte is high.

The number of input bits of the function we are looking for is 16. This is a trivial upper bound on the number of input bits needed before the next output is produced. Clearly, any decent implementation will also produce the output after a bounded waiting time.

4.2 A solution for low area hardware implementation

Let $a_0, a_1, \ldots, a_{15}$ be the input bits for post-processing. We define the 8 bits $b_0, b_1, \ldots, b_7$ by $b_i = a_i \text{ XOR } a_{(i+1) \bmod 8}$. We note that the mapping from $a_0, \ldots, a_7$ to $b_0, \ldots, b_7$ defined in this way is not bijective. Only 128 different values for $b_0, b_1, \ldots, b_7$ are possible. When the input is a perfect random number generator, we destroy one bit of entropy by mapping $a_0, \ldots, a_7$ to $b_0, \ldots, b_7$. The output $c_0, c_1, \ldots, c_7$ of the suggested post-processing function is defined by

$c_i = b_i$ XOR a_{i+8}. We call this function, which maps 16 input bits to 8 output bits, H.

If we split the 16 input bits of H into two bytes we name $a1$ and $a2$, we can write H in pseudocode as

```
H(a1,a2)= XOR(XOR(a1,rotateleft(a1,1)),a2)
```

Figure 2 shows the design of this post-processing function.

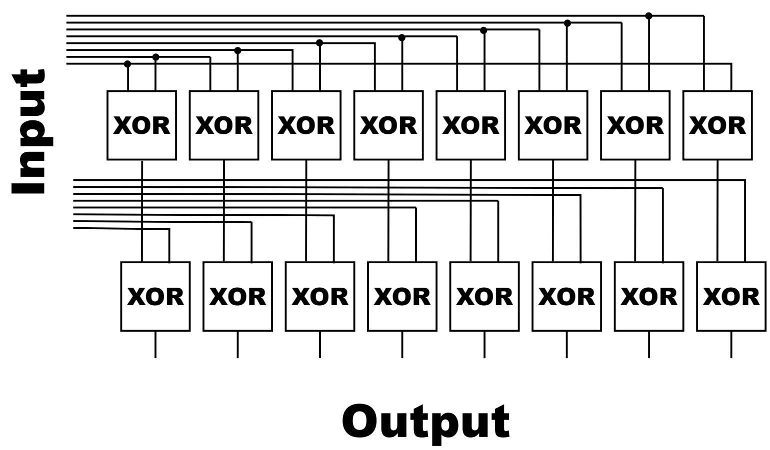

Fig. 2. The post-processing function H

The entropy of the output of H is shown in Figure 3 as a function of p.

The dashed line in the figure shows, for comparison, the entropy one gets when compressing two input bytes to one output byte by bitwise XOR. We see that H extracts significantly more entropy from its input than the XOR.

Figure 3 might suggest that for low biases the entropies of H and XOR are quite comparable. Indeed both are close to 8, but H is considerably closer. Let's look at a generator which produces 1-bits with a probability of 0.51. This is quite good for a physical random number generator. When we use XOR to compress two bytes from this generator to one output byte, the output byte has an entropy of 7.9999990766751 . Using H, we get $e_H = 7.9999999996305$. So in this case the deviation from the ideal value 8 is reduced by a factor more than 2499 if we use H instead of XOR. How good must a generator be to produce output bytes with entropy e_H when using XOR for postprocessing? From 2 input-bytes with a 1-probability of 0.501417 we get an output entropy of e_h if we XOR them. With XOR the 1-probability must be seven times closer to 0.5 than with H to get the output entropy e_H.

The improved entropy of H is achieved with very low hardware costs, just 16 XOR gates with two inputs are needed.

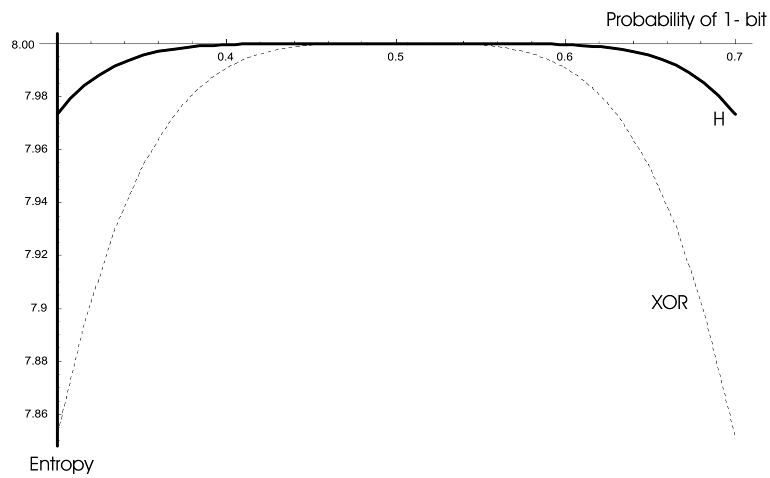

Fig. 3. The entropy of H

4.3 Analysis of H

What is going on in H? Why is its entropy higher than the entropy of XOR although one first discards one bit of entropy from one input byte? Let us look at the probabilities in more detail.

We define the quantitative value of the bias as

$$\epsilon = p - 1/2.$$

First we compute the probability that the source produces a raw (unprocessed) byte B. Since the bits of B are independent, the probability depends only on the Hamming weight w of B. It is

$$(1/2 - \epsilon)^{8-w} \cdot (1/2 + \epsilon)^{w}.$$

For the different values of w we obtain

w	byte probability for a raw data byte
0	$\frac{1}{256} - \frac{\epsilon}{16} + \frac{7\,\epsilon^2}{16} - \frac{7\,\epsilon^3}{4} + \frac{35\,\epsilon^4}{8} - 7\,\epsilon^5 + 7\,\epsilon^6 - 4\,\epsilon^7 + \epsilon^8$
1	$\frac{1}{256} - \frac{3\,\epsilon}{64} + \frac{7\,\epsilon^2}{32} - \frac{7\,\epsilon^3}{16} + \frac{7\,\epsilon^5}{4} - \frac{7\,\epsilon^6}{2} + 3\,\epsilon^7 - \epsilon^8$
2	$\frac{1}{256} - \frac{\epsilon}{32} + \frac{\epsilon^2}{16} + \frac{\epsilon^3}{8} - \frac{5\,\epsilon^4}{8} + \frac{\epsilon^5}{2} + \epsilon^6 - 2\,\epsilon^7 + \epsilon^8$
3	$\frac{1}{256} - \frac{\epsilon}{64} - \frac{\epsilon^2}{32} + \frac{3\,\epsilon^3}{16} - \frac{3\,\epsilon^5}{4} + \frac{\epsilon^6}{2} + \epsilon^7 - \epsilon^8$
4	$\frac{1}{256} - \frac{\epsilon^2}{16} + \frac{3\,\epsilon^4}{8} - \epsilon^6 + \epsilon^8$
5	$\frac{1}{256} + \frac{\epsilon}{64} - \frac{\epsilon^2}{32} - \frac{3\,\epsilon^3}{16} + \frac{3\,\epsilon^5}{4} + \frac{\epsilon^6}{2} - \epsilon^7 - \epsilon^8$
6	$\frac{1}{256} + \frac{\epsilon}{32} + \frac{\epsilon^2}{16} - \frac{\epsilon^3}{8} - \frac{5\,\epsilon^4}{8} - \frac{\epsilon^5}{2} + \epsilon^6 + 2\,\epsilon^7 + \epsilon^8$
7	$\frac{1}{256} + \frac{3\,\epsilon}{64} + \frac{7\,\epsilon^2}{32} + \frac{7\,\epsilon^3}{16} - \frac{7\,\epsilon^5}{4} - \frac{7\,\epsilon^6}{2} - 3\,\epsilon^7 - \epsilon^8$
8	$\frac{1}{256} + \frac{\epsilon}{16} + \frac{7\,\epsilon^2}{16} + \frac{7\,\epsilon^3}{4} + \frac{35\,\epsilon^4}{8} + 7\,\epsilon^5 + 7\,\epsilon^6 + 4\,\epsilon^7 + \epsilon^8$

For a good generator, ϵ should be small. As a consequence, the terms with the lower powers of ϵ are the more disturbing ones, the linear ones being the worst.

Before we start the analysis of H, we consider the XOR post-processing. The XOR of two raw input bits results in a 1-bit with a probability of $2p(1-p) = 1/2 - 2\epsilon^2$. That is, the bias of the resulting bit is $-2\epsilon^2$. When we insert this bias term for the ϵ in the table above we obtain the byte probabilities for the bitwise XOR of two raw data bytes:

w	byte probability for the XOR of two raw data bytes
0	$\frac{1}{256} - \frac{\epsilon^2}{8} + \frac{7\,\epsilon^4}{4} - 14\,\epsilon^6 + 70\,\epsilon^8 - 224\,\epsilon^{10} + 448\,\epsilon^{12} - 512\,\epsilon^{14} + 256\,\epsilon^{16}$
1	$\frac{1}{256} - \frac{3\,\epsilon^2}{32} + \frac{7\,\epsilon^4}{8} - \frac{7\,\epsilon^6}{2} + 56\,\epsilon^{10} - 224\,\epsilon^{12} + 384\,\epsilon^{14} - 256\,\epsilon^{16}$
2	$\frac{1}{256} - \frac{\epsilon^2}{16} + \frac{\epsilon^4}{4} + \epsilon^6 - 10\,\epsilon^8 + 16\,\epsilon^{10} + 64\,\epsilon^{12} - 256\,\epsilon^{14} + 256\,\epsilon^{16}$
3	$\frac{1}{256} - \frac{\epsilon^2}{32} - \frac{\epsilon^4}{8} + \frac{3\,\epsilon^6}{2} - 24\,\epsilon^{10} + 32\,\epsilon^{12} + 128\,\epsilon^{14} - 256\,\epsilon^{16}$
4	$\frac{1}{256} - \frac{\epsilon^4}{4} + 6\,\epsilon^8 - 64\,\epsilon^{12} + 256\,\epsilon^{16}$
5	$\frac{1}{256} + \frac{\epsilon^2}{32} - \frac{\epsilon^4}{8} - \frac{3\,\epsilon^6}{2} + 24\,\epsilon^{10} + 32\,\epsilon^{12} - 128\,\epsilon^{14} - 256\,\epsilon^{16}$
6	$\frac{1}{256} + \frac{\epsilon^2}{16} + \frac{\epsilon^4}{4} - \epsilon^6 - 10\,\epsilon^8 - 16\,\epsilon^{10} + 64\,\epsilon^{12} + 256\,\epsilon^{14} + 256\,\epsilon^{16}$
7	$\frac{1}{256} + \frac{3\,\epsilon^2}{32} + \frac{7\,\epsilon^4}{8} + \frac{7\,\epsilon^6}{2} - 56\,\epsilon^{10} - 224\,\epsilon^{12} - 384\,\epsilon^{14} - 256\,\epsilon^{16}$
8	$\frac{1}{256} + \frac{\epsilon^2}{8} + \frac{7\,\epsilon^4}{4} + 14\,\epsilon^6 + 70\,\epsilon^8 + 224\,\epsilon^{10} + 448\,\epsilon^{12} + 512\,\epsilon^{14} + 256\,\epsilon^{16}$

Indeed, all the linear terms in ϵ are gone.

We also analysed some linear feedback shift registers used for true random number post-processing. In the cases we considered, the ratio of the numbers of input and output bits was 2. We observed that the output probabilities contained no linear powers of ϵ, but squares occurred. So XOR and LFSRs with an input/output ratio of 2 seem to be random number post-processing functions of similar quality.

Now we compute the probabilities of the output bytes of H. One approach is to consider every possible pair of input bytes to determine its probability as the product of the byte probabilities from the table for raw bytes above, and to sum up these probabilities separately for all input byte pairs which lead to the same value of H. We obtain 30 different probabilities. Formulas for theses probabilities are shown in Table 1. In all these probabilities, there are no linear or quadratic terms in ϵ! This explains why H is better than the simple XOR of 2 input bytes.

But now, there is a challenge: Can we eliminate further powers of ϵ?

4.4 Improving

Can we eliminate further powers of ϵ? Surprisingly, slight modifications of H also eleminate the third and forth power of ϵ. Again, we split up the 16 input bits

output byte probability for H
$\frac{1}{256}+\frac{\epsilon^3}{16}-\frac{\epsilon^4}{4}-\frac{3\,\epsilon^5}{4}+\frac{\epsilon^6}{2}+3\,\epsilon^7+3\,\epsilon^8-4\,\epsilon^9-8\,\epsilon^{10}$
$\frac{1}{256}-\frac{\epsilon^3}{16}-\frac{\epsilon^4}{4}+\frac{3\,\epsilon^5}{4}+\frac{\epsilon^6}{2}-3\,\epsilon^7+3\,\epsilon^8+4\,\epsilon^9-8\,\epsilon^{10}$
$\frac{1}{256}-\frac{\epsilon^3}{16}-\frac{\epsilon^5}{4}-\frac{\epsilon^6}{2}+5\,\epsilon^7-\epsilon^8-12\,\epsilon^9+8\,\epsilon^{10}$
$\frac{1}{256}+\frac{\epsilon^3}{16}+\frac{\epsilon^5}{4}-\frac{\epsilon^6}{2}-5\,\epsilon^7-\epsilon^8+12\,\epsilon^9+8\,\epsilon^{10}$
$\frac{1}{256}-\frac{\epsilon^3}{16}+\frac{\epsilon^4}{4}-\frac{\epsilon^5}{4}-\frac{3\,\epsilon^6}{2}+\epsilon^7-5\,\epsilon^8+20\,\epsilon^9+24\,\epsilon^{10}-64\,\epsilon^{11}$
$\frac{1}{256}+\frac{3\,\epsilon^3}{16}+\frac{\epsilon^4}{4}+\frac{\epsilon^5}{4}+\frac{5\,\epsilon^6}{2}+3\,\epsilon^7-5\,\epsilon^8-20\,\epsilon^9-40\,\epsilon^{10}-32\,\epsilon^{11}$
$\frac{1}{256}+\frac{\epsilon^3}{16}-\frac{\epsilon^4}{4}-\frac{\epsilon^5}{4}+\frac{\epsilon^6}{2}-3\,\epsilon^7+3\,\epsilon^8+20\,\epsilon^9-8\,\epsilon^{10}-32\,\epsilon^{11}$
$\frac{1}{256}-\frac{\epsilon^3}{16}+\frac{\epsilon^5}{4}-\frac{\epsilon^6}{2}-\epsilon^7-\epsilon^8+12\,\epsilon^9+8\,\epsilon^{10}-32\,\epsilon^{11}$
$\frac{1}{256}-\frac{3\,\epsilon^3}{16}+\frac{\epsilon^4}{4}-\frac{\epsilon^5}{4}+\frac{5\,\epsilon^6}{2}-3\,\epsilon^7-5\,\epsilon^8+20\,\epsilon^9-40\,\epsilon^{10}+32\,\epsilon^{11}$
$\frac{1}{256}-\frac{\epsilon^3}{16}-\frac{\epsilon^4}{4}+\frac{\epsilon^5}{4}+\frac{\epsilon^6}{2}+3\,\epsilon^7+3\,\epsilon^8-20\,\epsilon^9-8\,\epsilon^{10}+32\,\epsilon^{11}$
$\frac{1}{256}+\frac{\epsilon^3}{16}-\frac{\epsilon^5}{4}-\frac{\epsilon^6}{2}+\epsilon^7-\epsilon^8-12\,\epsilon^9+8\,\epsilon^{10}+32\,\epsilon^{11}$
$\frac{1}{256}+\frac{\epsilon^3}{16}+\frac{\epsilon^4}{4}+\frac{\epsilon^5}{4}-\frac{3\,\epsilon^6}{2}-\epsilon^7-5\,\epsilon^8-20\,\epsilon^9+24\,\epsilon^{10}+64\,\epsilon^{11}$
$\frac{1}{256}-2\,\epsilon^6+9\,\epsilon^8-32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^4}{4}+\epsilon^6-3\,\epsilon^8+16\,\epsilon^{10}-32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^3}{8}+\frac{\epsilon^5}{2}+2\,\epsilon^7-7\,\epsilon^8-8\,\epsilon^9+32\,\epsilon^{10}-32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^3}{8}-\frac{\epsilon^5}{2}-2\,\epsilon^7-7\,\epsilon^8+8\,\epsilon^9+32\,\epsilon^{10}-32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^3}{8}+\frac{\epsilon^4}{4}+\frac{\epsilon^5}{2}+\epsilon^6+2\,\epsilon^7+5\,\epsilon^8-8\,\epsilon^9-48\,\epsilon^{10}-64\,\epsilon^{11}-32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^3}{8}+\frac{\epsilon^4}{4}-\frac{\epsilon^5}{2}+\epsilon^6-2\,\epsilon^7+5\,\epsilon^8+8\,\epsilon^9-48\,\epsilon^{10}+64\,\epsilon^{11}-32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^4}{4}-2\,\epsilon^6+\epsilon^8+32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^4}{4}+9\,\epsilon^8-32\,\epsilon^{10}+32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^4}{2}+3\,\epsilon^6-3\,\epsilon^8-16\,\epsilon^{10}+32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^3}{8}+\frac{\epsilon^5}{2}-\epsilon^6+2\,\epsilon^7+5\,\epsilon^8-8\,\epsilon^9-16\,\epsilon^{10}+32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^3}{8}-\frac{\epsilon^5}{2}-\epsilon^6-2\,\epsilon^7+5\,\epsilon^8+8\,\epsilon^9-16\,\epsilon^{10}+32\,\epsilon^{12}$
$\frac{1}{256}+\epsilon^6-11\,\epsilon^8+16\,\epsilon^{10}+32\,\epsilon^{12},$
$\frac{1}{256}-\frac{\epsilon^3}{4}+\frac{\epsilon^4}{2}-\epsilon^5+7\,\epsilon^6-20\,\epsilon^7+45\,\epsilon^8-112\,\epsilon^9+176\,\epsilon^{10}-128\,\epsilon^{11}+32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^5}{2}-\epsilon^6+2\,\epsilon^7+5\,\epsilon^8+8\,\epsilon^9-16\,\epsilon^{10}-32\,\epsilon^{11}+32\,\epsilon^{12}$
$\frac{1}{256}-\frac{\epsilon^3}{8}+\epsilon^6+4\,\epsilon^7-11\,\epsilon^8+16\,\epsilon^{10}-32\,\epsilon^{11}+32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^5}{2}-\epsilon^6-2\,\epsilon^7+5\,\epsilon^8-8\,\epsilon^9-16\,\epsilon^{10}+32\,\epsilon^{11}+32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^3}{8}+\epsilon^6-4\,\epsilon^7-11\,\epsilon^8+16\,\epsilon^{10}+32\,\epsilon^{11}+32\,\epsilon^{12}$
$\frac{1}{256}+\frac{\epsilon^3}{4}+\frac{\epsilon^4}{2}+\epsilon^5+7\,\epsilon^6+20\,\epsilon^7+45\,\epsilon^8+112\,\epsilon^9+176\,\epsilon^{10}+128\,\epsilon^{11}+32\,\epsilon^{12}$

Table 1. Probabilities of the outputs of H for bias ϵ

into two bytes $a1$ and $a2$ and define the functions $H2$ and $H3$ by the following pseudocode:

```
H2(a1,a2)= XOR(XOR(XOR(a1,rotateleft(a1,1)),rotateleft(a1,2)),a2)

H4(a1,a2)= XOR(XOR(XOR(XOR(a1,rotateleft(a1,1)),
                        rotateleft(a1,2)),rotateleft(a1,4)),a2)
```

For $H2$ we find that the lowest powers of ϵ in the probabilities of the output bytes are ϵ^4. For $H3$ they are ϵ^5.

4.5 An even better solution

It seems that the linear methods which could eliminate all powers of ϵ up to the forth do not work for the fifth. Whereas the linear solutions turned up rather wondrously, we have to labour now. What must we do in order to eliminate the first through fifth powers of ϵ? We have to partition the set of 65536 inputs into 256 sets of cardinality 256 in such a way that the first through fifth powers of ϵ disappear in all 256 sums over the 256 input probabilities. Fortunately, we do not have to deal with too many different input probabilities, since they depend only on the Hamming weight w of the 16 bit input. The probabilities and their numbers of occurrences are given in Table 2.

A careful look at this table shows that all odd powers of ϵ disappear in the probabilities, if we partition the inputs in such a way that 16 bit input values and their bitwise complements are always in the same partition. This makes our problem significantly easier and leads to the much simpler Table 3.

For the Hamming weight 8, the number of occurrences in Table 3 is only half of the number in Table 2, because the Hamming weight of complements of inputs with Hamming weight 8 is also 8.
Now, we have to partition the 32768 values from this table into 256 sets with 128 elements, such that the first through fifth powers of ϵ eliminate each other. Here is the solution we found:

Among the 256 sets, we distinguish 7 different types:

Type A consists of once the Hamming weight 0, 112 times the Hamming weight 6, and 15 times the Hamming weight 8. There is only one set of type A.

Type B consists of once the Hamming weight 1, 42 times the Hamming weight 5, and 85 times the Hamming weight 7. There are 16 sets of type B.

Type C consists of 14 times the Hamming weight 4, 28 times the Hamming weight 5, 36 times the Hamming weight 7, and 50 times the Hamming weight 8. There are 46 sets of type C.

Type D consists of twice the Hamming weight 2, 37 times the Hamming weight 5, 16 times the Hamming weight 6, 43 times the Hamming weight 7, and 30 times the Hamming weight 8. There are 60 sets of type D.

Type E consists of 5 times the Hamming weight 3, 7 times the Hamming weight 4, 58 times the Hamming weight 6, 43 times the Hamming weight 7, and 15 times the Hamming weight 8. There are 112 sets of type E.

w	Occurrences	Probability of 16 bit input with Hamming weight w
0	1	$\frac{1}{65536}-\frac{\epsilon}{2048}+\frac{15\,\epsilon^2}{2048}-\frac{35\,\epsilon^3}{512}+\frac{455\,\epsilon^4}{1024}-\frac{273\,\epsilon^5}{128}+\frac{1001\,\epsilon^6}{128}-\frac{715\,\epsilon^7}{32}+\frac{6435\,\epsilon^8}{128}-\frac{715\,\epsilon^9}{8}+\frac{1001\,\epsilon^{10}}{8}-\frac{273\,\epsilon^{11}}{2}+\frac{455\,\epsilon^{12}}{4}-70\,\epsilon^{13}+30\,\epsilon^{14}-8\,\epsilon^{15}+\epsilon^{16}$
1	16	$\frac{1}{65536}-\frac{7\,\epsilon}{16384}+\frac{45\,\epsilon^2}{8192}-\frac{175\,\epsilon^3}{4096}+\frac{455\,\epsilon^4}{2048}-\frac{819\,\epsilon^5}{1024}+\frac{1001\,\epsilon^6}{512}-\frac{715\,\epsilon^7}{256}+\frac{715\,\epsilon^9}{64}-\frac{1001\,\epsilon^{10}}{32}+\frac{819\,\epsilon^{11}}{16}-\frac{455\,\epsilon^{12}}{8}+\frac{175\,\epsilon^{13}}{4}-\frac{45\,\epsilon^{14}}{2}+7\,\epsilon^{15}-\epsilon^{16}$
2	120	$\frac{1}{65536}-\frac{3\,\epsilon}{8192}+\frac{\epsilon^2}{256}-\frac{49\,\epsilon^3}{2048}+\frac{91\,\epsilon^4}{1024}-\frac{91\,\epsilon^5}{512}+\frac{143\,\epsilon^7}{128}-\frac{429\,\epsilon^8}{128}+\frac{143\,\epsilon^9}{32}-\frac{91\,\epsilon^{11}}{8}+\frac{91\,\epsilon^{12}}{4}-\frac{49\,\epsilon^{13}}{2}+16\,\epsilon^{14}-6\,\epsilon^{15}+\epsilon^{16}$
3	560	$\frac{1}{65536}-\frac{5\,\epsilon}{16384}+\frac{21\,\epsilon^2}{8192}-\frac{45\,\epsilon^3}{4096}+\frac{39\,\epsilon^4}{2048}+\frac{39\,\epsilon^5}{1024}-\frac{143\,\epsilon^6}{512}+\frac{143\,\epsilon^7}{256}-\frac{143\,\epsilon^9}{64}+\frac{143\,\epsilon^{10}}{32}-\frac{39\,\epsilon^{11}}{16}-\frac{39\,\epsilon^{12}}{8}+\frac{45\,\epsilon^{13}}{4}-\frac{21\,\epsilon^{14}}{2}+5\,\epsilon^{15}-\epsilon^{16}$
4	1820	$\frac{1}{65536}-\frac{\epsilon}{4096}+\frac{3\,\epsilon^2}{2048}-\frac{3\,\epsilon^3}{1024}-\frac{9\,\epsilon^4}{1024}+\frac{15\,\epsilon^5}{256}-\frac{11\,\epsilon^6}{128}-\frac{11\,\epsilon^7}{64}+\frac{99\,\epsilon^8}{128}-\frac{11\,\epsilon^9}{16}-\frac{11\,\epsilon^{10}}{8}+\frac{15\,\epsilon^{11}}{4}-\frac{9\,\epsilon^{12}}{4}-3\,\epsilon^{13}+6\,\epsilon^{14}-4\,\epsilon^{15}+\epsilon^{16}$
5	4368	$\frac{1}{65536}-\frac{3\,\epsilon}{16384}+\frac{5\,\epsilon^2}{8192}+\frac{5\,\epsilon^3}{4096}-\frac{25\,\epsilon^4}{2048}+\frac{17\,\epsilon^5}{1024}+\frac{33\,\epsilon^6}{512}-\frac{55\,\epsilon^7}{256}+\frac{55\,\epsilon^9}{64}-\frac{33\,\epsilon^{10}}{32}-\frac{17\,\epsilon^{11}}{16}+\frac{25\,\epsilon^{12}}{8}-\frac{5\,\epsilon^{13}}{4}-\frac{5\,\epsilon^{14}}{2}+3\,\epsilon^{15}-\epsilon^{16}$
6	8008	$\frac{1}{65536}-\frac{\epsilon}{8192}+\frac{5\,\epsilon^3}{2048}-\frac{5\,\epsilon^4}{1024}-\frac{9\,\epsilon^5}{512}+\frac{\epsilon^6}{16}+\frac{5\,\epsilon^7}{128}-\frac{45\,\epsilon^8}{128}+\frac{5\,\epsilon^9}{32}+\epsilon^{10}-\frac{9\,\epsilon^{11}}{8}-\frac{5\,\epsilon^{12}}{4}+\frac{5\,\epsilon^{13}}{2}-2\,\epsilon^{15}+\epsilon^{16}$
7	11440	$\frac{1}{65536}-\frac{\epsilon}{16384}-\frac{3\,\epsilon^2}{8192}+\frac{7\,\epsilon^3}{4096}+\frac{7\,\epsilon^4}{2048}-\frac{21\,\epsilon^5}{1024}-\frac{7\,\epsilon^6}{512}+\frac{35\,\epsilon^7}{256}-\frac{35\,\epsilon^9}{64}+\frac{7\,\epsilon^{10}}{32}+\frac{21\,\epsilon^{11}}{16}-\frac{7\,\epsilon^{12}}{8}-\frac{7\,\epsilon^{13}}{4}+\frac{3\,\epsilon^{14}}{2}+\epsilon^{15}-\epsilon^{16}$
8	12870	$\frac{1}{65536}-\frac{\epsilon^2}{2048}+\frac{7\,\epsilon^4}{1024}-\frac{7\,\epsilon^6}{128}+\frac{35\,\epsilon^8}{128}-\frac{7\,\epsilon^{10}}{8}+\frac{7\,\epsilon^{12}}{4}-2\,\epsilon^{14}+\epsilon^{16}$
9	11440	$\frac{1}{65536}+\frac{\epsilon}{16384}-\frac{3\,\epsilon^2}{8192}-\frac{7\,\epsilon^3}{4096}+\frac{7\,\epsilon^4}{2048}+\frac{21\,\epsilon^5}{1024}-\frac{7\,\epsilon^6}{512}-\frac{35\,\epsilon^7}{256}+\frac{35\,\epsilon^9}{64}+\frac{7\,\epsilon^{10}}{32}-\frac{21\,\epsilon^{11}}{16}-\frac{7\,\epsilon^{12}}{8}+\frac{7\,\epsilon^{13}}{4}+\frac{3\,\epsilon^{14}}{2}-\epsilon^{15}-\epsilon^{16}$
10	8008	$\frac{1}{65536}+\frac{\epsilon}{8192}-\frac{5\,\epsilon^3}{2048}-\frac{5\,\epsilon^4}{1024}+\frac{9\,\epsilon^5}{512}+\frac{\epsilon^6}{16}-\frac{5\,\epsilon^7}{128}-\frac{45\,\epsilon^8}{128}-\frac{5\,\epsilon^9}{32}+\epsilon^{10}+\frac{9\,\epsilon^{11}}{8}-\frac{5\,\epsilon^{12}}{4}-\frac{5\,\epsilon^{13}}{2}+2\,\epsilon^{15}+\epsilon^{16}$
11	4368	$\frac{1}{65536}+\frac{3\,\epsilon}{16384}+\frac{5\,\epsilon^2}{8192}-\frac{5\,\epsilon^3}{4096}-\frac{25\,\epsilon^4}{2048}-\frac{17\,\epsilon^5}{1024}+\frac{33\,\epsilon^6}{512}+\frac{55\,\epsilon^7}{256}-\frac{55\,\epsilon^9}{64}-\frac{33\,\epsilon^{10}}{32}+\frac{17\,\epsilon^{11}}{16}+\frac{25\,\epsilon^{12}}{8}+\frac{5\,\epsilon^{13}}{4}-\frac{5\,\epsilon^{14}}{2}-3\,\epsilon^{15}-\epsilon^{16}$
12	1820	$\frac{1}{65536}+\frac{\epsilon}{4096}+\frac{3\,\epsilon^2}{2048}+\frac{3\,\epsilon^3}{1024}-\frac{9\,\epsilon^4}{1024}-\frac{15\,\epsilon^5}{256}-\frac{11\,\epsilon^6}{128}+\frac{11\,\epsilon^7}{64}+\frac{99\,\epsilon^8}{128}+\frac{11\,\epsilon^9}{16}-\frac{11\,\epsilon^{10}}{8}-\frac{15\,\epsilon^{11}}{4}-\frac{9\,\epsilon^{12}}{4}+3\,\epsilon^{13}+6\,\epsilon^{14}+4\,\epsilon^{15}+\epsilon^{16}$
13	560	$\frac{1}{65536}+\frac{5\,\epsilon}{16384}+\frac{21\,\epsilon^2}{8192}+\frac{45\,\epsilon^3}{4096}+\frac{39\,\epsilon^4}{2048}-\frac{39\,\epsilon^5}{1024}-\frac{143\,\epsilon^6}{512}-\frac{143\,\epsilon^7}{256}+\frac{143\,\epsilon^9}{64}+\frac{143\,\epsilon^{10}}{32}+\frac{39\,\epsilon^{11}}{16}-\frac{39\,\epsilon^{12}}{8}-\frac{45\,\epsilon^{13}}{4}-\frac{21\,\epsilon^{14}}{2}-5\,\epsilon^{15}-\epsilon^{16}$
14	120	$\frac{1}{65536}+\frac{3\,\epsilon}{8192}+\frac{\epsilon^2}{256}+\frac{49\,\epsilon^3}{2048}+\frac{91\,\epsilon^4}{1024}+\frac{91\,\epsilon^5}{512}-\frac{143\,\epsilon^7}{128}-\frac{429\,\epsilon^8}{128}-\frac{143\,\epsilon^9}{32}+\frac{91\,\epsilon^{11}}{8}+\frac{91\,\epsilon^{12}}{4}+\frac{49\,\epsilon^{13}}{2}+16\,\epsilon^{14}+6\,\epsilon^{15}+\epsilon^{16}$
15	16	$\frac{1}{65536}+\frac{7\,\epsilon}{16384}+\frac{45\,\epsilon^2}{8192}+\frac{175\,\epsilon^3}{4096}+\frac{455\,\epsilon^4}{2048}+\frac{819\,\epsilon^5}{1024}+\frac{1001\,\epsilon^6}{512}+\frac{715\,\epsilon^7}{256}-\frac{715\,\epsilon^9}{64}-\frac{1001\,\epsilon^{10}}{32}-\frac{819\,\epsilon^{11}}{16}-\frac{455\,\epsilon^{12}}{8}-\frac{175\,\epsilon^{13}}{4}-\frac{45\,\epsilon^{14}}{2}-7\,\epsilon^{15}-\epsilon^{16}$
16	1	$\frac{1}{65536}+\frac{\epsilon}{2048}+\frac{15\,\epsilon^2}{2048}+\frac{35\,\epsilon^3}{512}+\frac{455\,\epsilon^4}{1024}+\frac{273\,\epsilon^5}{128}+\frac{1001\,\epsilon^6}{128}+\frac{715\,\epsilon^7}{32}+\frac{6435\,\epsilon^8}{128}+\frac{715\,\epsilon^9}{8}+\frac{1001\,\epsilon^{10}}{8}+\frac{273\,\epsilon^{11}}{2}+\frac{455\,\epsilon^{12}}{4}+70\,\epsilon^{13}+30\,\epsilon^{14}+8\,\epsilon^{15}+\epsilon^{16}$

Table 2. Probabilities of 16-bit-vectors with bit bias ϵ

w	Occurrences	Probability of input + probability of complement
0	1	$\frac{1}{32768} + \frac{15\,\epsilon^2}{1024} + \frac{455\,\epsilon^4}{512} + \frac{1001\,\epsilon^6}{64} + \frac{6435\,\epsilon^8}{64} + \frac{1001\,\epsilon^{10}}{4} + \frac{455\,\epsilon^{12}}{2} + 60\,\epsilon^{14} + 2\,\epsilon^{16}$
1	16	$\frac{1}{32768} + \frac{45\,\epsilon^2}{4096} + \frac{455\,\epsilon^4}{1024} + \frac{1001\,\epsilon^6}{256} - \frac{1001\,\epsilon^{10}}{16} - \frac{455\,\epsilon^{12}}{4} - 45\,\epsilon^{14} - 2\,\epsilon^{16}$
2	120	$\frac{1}{32768} + \frac{\epsilon^2}{128} + \frac{91\,\epsilon^4}{512} - \frac{429\,\epsilon^8}{64} + \frac{91\,\epsilon^{12}}{2} + 32\,\epsilon^{14} + 2\,\epsilon^{16}$
3	560	$\frac{1}{32768} + \frac{21\,\epsilon^2}{4096} + \frac{39\,\epsilon^4}{1024} - \frac{143\,\epsilon^6}{256} + \frac{143\,\epsilon^{10}}{16} - \frac{39\,\epsilon^{12}}{4} - 21\,\epsilon^{14} - 2\,\epsilon^{16}$
4	1820	$\frac{1}{32768} + \frac{3\,\epsilon^2}{1024} - \frac{9\,\epsilon^4}{512} - \frac{11\,\epsilon^6}{64} + \frac{99\,\epsilon^8}{64} - \frac{11\,\epsilon^{10}}{4} - \frac{9\,\epsilon^{12}}{2} + 12\,\epsilon^{14} + 2\,\epsilon^{16}$
5	4368	$\frac{1}{32768} + \frac{5\,\epsilon^2}{4096} - \frac{25\,\epsilon^4}{1024} + \frac{33\,\epsilon^6}{256} - \frac{33\,\epsilon^{10}}{16} + \frac{25\,\epsilon^{12}}{4} - 5\,\epsilon^{14} - 2\,\epsilon^{16}$
6	8008	$\frac{1}{32768} - \frac{5\,\epsilon^4}{512} + \frac{\epsilon^6}{8} - \frac{45\,\epsilon^8}{64} + 2\,\epsilon^{10} - \frac{5\,\epsilon^{12}}{2} + 2\,\epsilon^{16}$
7	11440	$\frac{1}{32768} - \frac{3\,\epsilon^2}{4096} + \frac{7\,\epsilon^4}{1024} - \frac{7\,\epsilon^6}{256} + \frac{7\,\epsilon^{10}}{16} - \frac{7\,\epsilon^{12}}{4} + 3\,\epsilon^{14} - 2\,\epsilon^{16}$
8	6435	$\frac{1}{32768} - \frac{\epsilon^2}{1024} + \frac{7\,\epsilon^4}{512} - \frac{7\,\epsilon^6}{64} + \frac{35\,\epsilon^8}{64} - \frac{7\,\epsilon^{10}}{4} + \frac{7\,\epsilon^{12}}{2} - 4\,\epsilon^{14} + 2\,\epsilon^{16}$

Table 3. Probabilities for combining 16 bit inputs and their complements (bit bias ϵ)

Type F consists of 13 times the Hamming weight 4, 30 times the Hamming weight 5, 8 times the Hamming weight 6, 2 times the Hamming weight 7, and 75 times the Hamming weight 8. There are 4 sets of type F.

Type G consists of 20 times the Hamming weight 4, 4 times the Hamming weight 5, 24 times the Hamming weight 6, 60 times the Hamming weight 7, and 20 times the Hamming weight 8. There are 17 sets of type G.

The following table shows the output probability for each type:

Type	Probability of output byte
A	$\frac{1}{256} + 28\,\epsilon^6 + 30\,\epsilon^8 + 448\,\epsilon^{10} + 256\,\epsilon^{16}$
B	$\frac{1}{256} + 7\,\epsilon^6 - 112\,\epsilon^{10} - 256\,\epsilon^{16}$
C	$\frac{1}{256} - \frac{21\,\epsilon^6}{4} + 49\,\epsilon^8 - 168\,\epsilon^{10} + 224\,\epsilon^{12} - 64\,\epsilon^{14}$
D	$\frac{1}{256} + \frac{37\,\epsilon^6}{16} - \frac{33\,\epsilon^8}{4} - 78\,\epsilon^{10} + 312\,\epsilon^{12} - 112\,\epsilon^{14} - 64\,\epsilon^{16}$
E	$\frac{1}{256} + \frac{7\,\epsilon^6}{16} - \frac{87\,\epsilon^8}{4} + 134\,\epsilon^{10} - 248\,\epsilon^{12} + 48\,\epsilon^{14} + 64\,\epsilon^{16}$
F	$\frac{1}{256} - \frac{45\,\epsilon^6}{8} + \frac{111\,\epsilon^8}{2} - 212\,\epsilon^{10} + 368\,\epsilon^{12} - 288\,\epsilon^{14} + 128\,\epsilon^{16}$
G	$\frac{1}{256} - \frac{15\,\epsilon^6}{4} + 25\,\epsilon^8 - 24\,\epsilon^{10} - 160\,\epsilon^{12} + 320\,\epsilon^{14}$

All ϵ powers up to the fifth are gone!

In order to define a concrete post-processing function, we have to fix which inputs are mapped to which outputs. We can do this arbitrarily, taking into account the rules indicated above. The many choices we have for defining the concrete function do not influence the statistical quality of the function, but they can be used to ease the implementation of the function. Let S be a fixed function constructed according to the principles described above.

4.6 What about going further?

Naturally, we want to go on to eliminate the sixth powers of ϵ. Linear programming shows that the probability for Hamming weight 1 can be used at most 296/891 times when we want to arrange the probabilities in such a way that for sets of 256 probabilities the ϵ-powers cancel out up to the sixth. As a fraction does not make sense here, we have shown that it is impossible to eliminate the sixth powers of ϵ. We do not claim that S is optimal, as there might be solutions with sixth powers of ϵ with smaller absolute values of the coefficients than S.

We used the linear progamming approach to prove another negative result: When considering postprocessing with 32 input and 16 output bits, the probability of the outputs contains nineth or lower powers of ϵ.

4.7 The Entropy of S

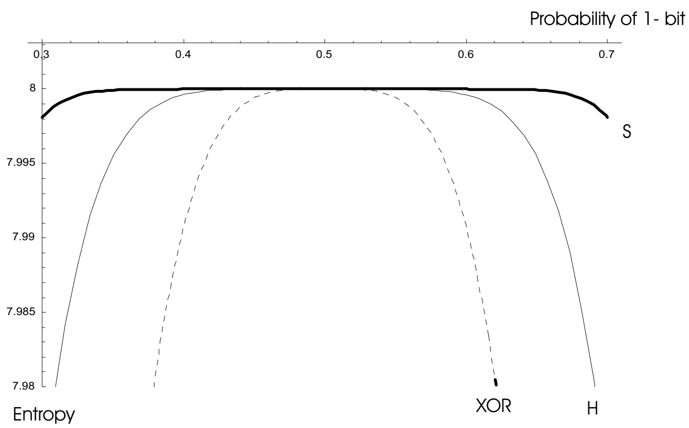

Fig. 4. The entropy of S

Figure 4 shows, as expected, that S extracts even more entropy from its input than H.

In the evaluation directive AIS 31 for true random number generators of the German BSI [KS01], a bias of 0.02 is still considered acceptable. This bias corresponds to an entropy of 0.998846 per bit, or 7.99076 per byte. This entropy is achieved by a source with bias 0.1, if each output bit is the XOR of two input bits. With the post-processing function H, the same entropy is achieved with a source bias of 0.16835. And for the post-processing function S, we obtain this entropy even for a source bias of 0.23106.

4.8 On the Implementation of S

A hardware implementation of the post-processing function S would probably require a considerable amount of chip area. The easiest way to implement S is just a lookup table, which assigns output bytes to all possible 16 bit inputs. Such a table requires 64 kBytes of ROM. This should be no problem on a modern PC, but could be prohibitive for some smart-card applications. We can halve this storage requirement, if we take into account that inputs and their bitwise complements lead to the same output. If the leading bit of the input is 1, we use the output for its bitwise complement from the table. Therefore, we can use a table of 32 kBytes.

We can implement really compact software for S, if we make use of the freedom we have for fixing the values for the function S. We will only sketch the principle of such a software implementation. The main idea is to use the lexicographically first possible solution for all situations, where one has a choice. Following this principle, the only instance of type A leads to the output of 0, the 16 instances of type B to the output values 1, ..., 16. The 46 instances of type C are assigned the output values 17, ..., 62, and so on. The assignment of input values to instances of types also follows this lexicographic principle. As an example, we consider inputs of Hamming weight 8. The 15 lexicographically lowest 16 bit values with Hamming weight 8 are assigned to the only instance of type A. Type B does not use inputs of Hamming weight 8, so the lexicographically next inputs with this Hamming weight are assigned to instances of type C. The lexicographically next 50 inputs with Hamming weight 8 are assigned to the first instance of type C, the next 50 to the second instance of type C, and so on. When we follow this lexicographic design principle, the tables needed for the computation of S can be stored very compactly in about 50 bytes.

This lexicographic principle can only be used profitably for a software implementation, if we are able to determine the rank of a given 16 bit value of Hamming weight w in the lexicographic order of the inputs of this Hamming weight. Let $p_0, p_1, \dots, p_{w-1}$ with $p_0 < p_1 < \cdots < p_{w-2} < p_{w-1}$ be the bit positions of the 1-bits of the input. Let the bit position of the least significant bit be 0, and let the bit position of the most significant bit be 15. Then the lexicographic rank is given by

$$\sum_{i=0}^{w-1} \binom{p_i}{i+1}.$$

5 Conclusion and further research topics

We have shown that the quasigroup post-processing method for physical random numbers described in [MGK05] is ineffective and very easy to attack.

We have also shown that there are post-processing functions with a fixed number of input bits for biased physical random number generators which are much better than the ones usually used up to now.

The results of section 4 of this paper can be extended in many directions: e. g. compression rates greater than 2, other input sizes, systematic construction of good post-processing functions.

6 Acknowledgment

The author would like to thank Bernd Meyer (Siemens AG) for his help and for many valuable discussions.

References

[JJSH00] A. Juels, M. Jakobsson, E. Shriver, and B. K. Hillyer, *How to turn loaded dice into fair coins*, IEEE Trans. Information Theory **46** (2000), no. 3, 911–921.

[KS01] Wolfgang Killmann and Werner Schindler, *A proposal for: Functionality classes and evaluation methodology for true (physical) random number generators*, `http://www.bsi.bund.de/zertifiz/zert/interpr/trngk31e.pdf`, 2001.

[MGK05] Smile Markovski, Danilo Gligoroski, and Ljupco Kocarev, *Unbiased random sequences from quasigroup string transformations*, Proceedings of FSE 2005 (Henri Gilbert and Helena Handschuh, eds.), Lecture Notes in Computer Science, vol. 3557, Springer-Verlag, 2005, pp. 163–180.

[Per92] Yuval Peres, *Iterating von Neumann's procedure for extracting random bits*, The Annals of Statistics **20** (1992), no. 3, 590–597.

[vN63] J. von Neumann, *Various techniques for use in connection with random digits*, von Neumann's Collected Works, vol. 5, Pergamon, 1963, pp. 768–770.