How to Predict the Output of a Hardware Random Number Generator

Markus Dichtl

Siemens AG, Corporate Technology
Email: Markus.Dichtl@siemens.com

Abstract. A hardware random number generator was described at CHES 2002 in [Tka03]. In this paper, we analyze its method of generating randomness and, as a consequence of the analysis, we describe how, in principle, an attack on the generator can be executed.

1 Introduction

Both designs for hardware random number generators and the evaluation of hardware random number generators have not been treated often in publications. This means a serious contrast between – on the one hand – the importance of hardware random number generation and their evaluation for the security of applications and – on the other hand – the little attention this topic has found in the published literature.

One case where the absence of a suitable physical random number generator received considerable public attention were the defects found in the random number generation for SSL implemented in an early version of the Netscape browser. The time of the day used as the only source of true randomness did not provide enough entropy. This lack of entropy could be used for a spectacular attack [GW96].

Physical random number generators deriving their randomness from a physical random process, are also called true random number generators (TRNGs). TRNGS have to be distinguished from pseudo random number generators (PRNGs). PRNGs derive their output algorithmically from a secret initial state. The unpredictability of PRNGs relies on the computational infeasibilty of trying all possible initial states, and on some assumptions on the algorithm used.

We will see that the hardware random number generator described at CHES 2002 in [Tka03] is a combination of TRNG and PRNG elements. Therefore we just call it the RNG (random number generator) in this paper.

We will show that the RNG in some cases produces very little entropy, so that its output can be predicted. This is in contrast with one of the design requirements for the generator cited in [Tka03].

This paper provides theoretical analysis based on the properties of the RNG described in [Tka03]; no experiments on a real chip were made.

2 A hardware random number generator

The RNG described in [Tka03] uses two free running oscillators, implemented as ring oscillators, to clock two deterministic finite state machines.

One of the finite state machines is a binary linear feedback shift register (LFSR) of length 47. The feedback polynomial of the LFSR is primitive.

The other finite state machine is a one dimensional binary cellular automaton (CA) with a neighbourhood of 3. The CA consists of 37 cells. All cells except one follow one rule to derive their next state: The new value is the XOR of the old values of the two neighbouring cells. Only cell 28 is subject to another, albeit similar, rule: Its new value is the XOR sum of its old value and the old values of its two neighbours. For the two cells at the border of the CA, null boundary conditions are used; that is those neighbouring cells which are required by the CA rules, but which

are beyond the limits of the CA, are assumed to have the fixed value 0. If started from a not all zeros state, this CA has a cycle length of $2^{37} - 1$.

When the RNG has to produce a random number, 32 bits of the LFSR and 32 bits of the CA are taken from fixed positions of those finite state machines. The 32 bits from the LFSR are XORed with the 32 bits from the CA in order to receive the 32 bit output word.

3 Where does the randomness in the RNG come from?

The main elements of the RNG are two free running oscillators, a linear feedback shift register, and a cellular automaton. Free running oscillators can be a basis of TRNGs, whereas linear feedback shift registers and cellular automata, which are deterministic, are frequently used in the construction of PRNGs. In order to get a clear understanding of the origin of randomness in the RNG, the TRNG parts and the PRNG parts have to be separated mentally.

The linear feedback shift register used in the RNG can be seen as a special counter with a period length of $2^{43}-1$. The counter states are not represented as the familiar binary numbers, but are encoded as subsequent shift register states. Clearly the conversion between the representation of a counter state as a binary number and a shift register state is completely deterministic.

Analogously, the cellular automaton used in the RNG can be seen as a special counter with a period length of $2^{37} - 1$. Here, the counter state is represented as a cellular automaton state, but again, the conversion between these cellular automaton states and the familiar binary numbers is completely deterministic.

Hence, the only source of entropy in the RNG are the initial states of the registers in the linear feedback shift register and the cellular automaton, and the number of clocks that occurred for the linear feedback shift register and for the cellular automaton. The number of clocks for the linear feedback shift register is only relevant modulo $2^{43} - 1$, the number of clocks for the cellular automaton modulo $2^{37} - 1$.

4 How random is this?

As we have identified the sources of entropy in the RNG, the question arises how much entropy they provide.

The first source of randomness are the undetermined initial states of the registers. Even if each register assumed the 0 and 1 state with probability 1/2 each time the RNG is initialized, and even if there were no dependencies between the states of the registers at different initializations, this would not help the RNG much on the long run, because a good TRNG has to must produce continually new entropy as it runs, and not rely on an initial stock of entropy. However, the initial states of flip flops turn out to be no reliable source of entropy. Due to manufactoring variations, they are not completely symmetrical and as a consequence, most flip flops have an initial state which they initially take with a probability close to 1.

The other sources of randomness are the number of clocks occurring for the cellular automaton (modulo $2^{37} - 1$) and for the linear feedback shift register (modulo $2^{43} - 1$) since the initialization.

When an attacker Alice knows the frequencies of the free running oscillators clocking the LFSR and the CA only with limited precision, the RNG becomes completely unpredictable after a sufficiently long waiting time.

Let us assume Alice knows the frequencies of the free running oscillators with a precision of 10 percent, and that she also knows the initial state of the CA. Roughly speaking she looses all information about the state of the CA after about $10 \cdot (2^{37} - 1) \approx 10^{12}$ clocks of the CA. Even if the CA were clocked with 1 GHz, this would mean a waiting time of about 22 minutes. And in order to achieve completely independent CA states, one would also need waiting times of about 22 minutes between each 32 bit block of random values generated.

In [Tka03], too, a minimum sampling period for the subsequent generation of 32 bit blocks of random values is given. It is considerably smaller than 10^{12}, namely 86 cyles of the oscillator clocking the LFSR. Subsequently, we shall study the predictability of the RNG when this minimum sampling period is used.

5 How to predict the RNG bits

5.1 How well does an attacker know the frequencies of the free running clocking oscillators?

Evidently, the better the attacker Alice knows the frequencies of the free running oscillators clocking the LFSR and the CA, the better she can predict the numbers of clocks occurring for the LSFR and the CA.

The knowledge of these frequencies depends heavily on the circumstances of the attack. The main environmental parameters influencing the frequencies of free running oscillators are the temperature and the supply voltage. Sometimes these parameters are difficult to predict for Alice. In other applications, she may know these parameters precisely , or may even choose them. For example, professionally run trust centers tend to have their computers in stable air conditioned environments without much variation in temperature or supply voltage. On the other hand, smartcards will encounter enormous variations in environmental conditions, but when the user of a smartcard wants to attack the physical random number generator of the smartcard, she may choose the environment temperature and the supply voltage at her will.

But even when Alice knows the operating conditions of the oscillators perfectly, their frequencies cannot be predicted perfectly, because of non-deterministic effects in the oscillators. For example, there are physically unavoidable noise voltages in the transistors of the oscillator. This noise influences to some degree the exact moments when the transistors switch.

In order to infer the clocking frequencies from the environmental data, the attacker can either perform experiments on a chip with the hardware random number generator, or she must know the design details of the oscillators. The manufacturer of the chip of course knows these design details, and the outcome of a good hardware random number generator should be unpredictable even for the manufacturer of the hardware.

We will not elaborate a statistical model for the attacker's knowledge of the clocking frequencies, because this is not crucial for the attack. As we will see later on, we can easily increase the number of tries to guess the number of clocks occurring for the LFSR and the CA if our assummptions about the knowledge of the clocking frequencies are wrong.

In order to make the attack as efficient as possible, we concentrate on the case where the RNG is subsequently sampled as fast as possible. In [Tka03], the minimum time between the sampling of two output words is defined by the requirement that both state machines (CA and LFSR) clock at least twice their length.

For our attack we also need to assume an upper bound on the ratio of the frequency of the faster free running oscillator and the slower oscillator. This is not an arbitrary restriction, but performance and power consumption considerations make it advisable to choose the clocking frequencies of both oscillators in the same order of magnitude. If a very fast oscillator is used for one finite state machine and a slow oscillator for the other, one gets a RNG with a low data rate but high power consumption. The low data rate is caused by the slow oscillator and the design rule that the finite state machine must be clocked twice its length before it can be sampled again. The high power consumption is due to the fast oscillator, because power consumption and clocking frequency of the state machine are roughly proportional. Subsequently, we assume an upper bound of 3 for the frequency ratio of the oscillators. If the frequency ratio were higher, more guesses would be needed to find the correct number of clocks. A lower bound would speed up the attack.

In the scenario where the attacker defines the environmental conditions, she should be able to know the clock frequencies with a precision of 1 percent. If the attacker does not control the environmental conditions, she might be able to determine the clocking frequencies with a precision of 10 percent.

5.2 Guessing the number of clocks

Subsequently, we will consider three 32 bit words sampled from the RNG at top speed. This means that we have to consider the number of clocks occurring between the first and second sample, and

between the second and third sample, for each oscillator. We use f_{CA} and f_{LFSR} to denote the clock frequencies of the CA and the LFSR, respectively.

Case A Here we consider the case $f_{\mathrm{LFSR}} \leq f_{\mathrm{CA}}$. IN this case, the maximum sampling frequency is limited by the rule that the LFSR must clock twice its length before it can be sampled again. This means that at top speed the LFSR clocks 86 times, or, since Alice knows the frequency only with a precision of one percent in the scenario of an environment controlled by her, there may also occur 85 or 87 clocks. By our bound of 3 on the frequency ratio between the oscillators, the number of CA clocks is bounded by 258. With an error of one percent in Alices knowledge of the frequency, this leads to at most 7 possible numbers of CA clocks. Analogously, with a 10 percent insecurity for the frequencies, there are 19 possibilities for the number of LFSR clocks, and at most 53 for the number of CA clocks.

Case B Here we consider the case $f_{\mathrm{LFSR}} > f_{\mathrm{CA}}$. We have to distinguish two subcases.

Case B1 When $37 f_{\mathrm{LFSR}} \leq 43 f_{\mathrm{CA}}$ holds, that is f_{LFSR} is only slightly larger than f_{CA}, the maximum sampling rate allowed for the RNG is still determined by the LFSR frequency. As in case A, we have 3 possibilities for the number of LFSR clocks, if we know the frequencies with a precision of 1 percent. Since the CA is clocked at a lower rate, at most 3 numbers of CA clocks are possible. For the scenario of a 10 percent precision in the knowledge of the frequencies, we get 19 possible numbers of LFSR clocks and also 19 possible numbers of CA clocks.

Case B2 When $37 f_{\mathrm{LFSR}} > 43 f_{\mathrm{CA}}$ holds, the maximum sampling rate is determined by the CA. If the attacker knows the frequencies with a precision of 1 percent, this leads to 3 possible numbers of CA clocks, and to at most 7 possible numbers of LFSR clocks. With a 10 percent accuracy in the frequencies, there are 19 possible numbers of CA clocks, and 46 possible numbers of LFSR clocks.

In the case of frequencies known with a precision of 1 percent, the worst case is that we have a total of 21 possibilities for the numbers of clocks for both finite state machines. With a precsision of 10 percent, the worst case are 1007 possibilities.

Since we need the numbers of clocks occurring between the first and second sample, and between the second and third sample, we get a total of 441 cases (1 percent case) or 1014049 (10 percent case). This numbers are just a very coarse upper bound on the number of cases to consider, because the numbers of clocks between the different samples are strongly dependend. If, for example, the attacker knows that the number of LFSR clocks is between 200 and 240, she should not begin with 200 for the number of clocks between the first and second sample, and 240 for the number of clocks between the second and third sample. This combination is quite improbable to occur, because the frequencies of the oscillators do not change suddenly from very low to very high. Instead, the best strategy for the attacker is to assume that the clock frequency changed only very little from the second to the third sample. So, combinations of numbers of clocks with small differences should be tried first.

5.3 Determining the internal states of the CA and the LFSR

In this section we assume that we have correctly guessed the number of clocks of both the CA and the LFSR occurring between three top speed samplings of the RNG. We try to find out the internal states of the machines from the three 32 bit output words.

Since we assume that we know the number of clocks occurring we could try a brute force approach. The almost 2^{43+37} possible intial states make this quite impractical. An efficient solution must rely on the properties of the state machines.

A closer inspection of the two finite state machines makes the solution very easy: both are linear in GF(2). The function combining bits from each finite state machine to compute the RNG

output is also linear. We have to solve a system of 96 linear equations in order to determine the 80 bits of the states of the CA and the LFSR.

The fact that the number of equations exceeds the number of variables by 16, helps to eliminate wrong guesses of the number of clocks of the finite state machines. With a probability of $1-1/2^{16}$, a wrong guess results in a system of linear equations without a solution.

When one tries to write down the linear equations, one encounters a minor problem: [Tka03] does not specify which 32 bits from each finite state machine are used and how they are permuted. An attacker could reverse engineer the chip in order to receive this information. The information is also known to the manufacturer of the chip. And, as already mentioned above, the output of a good RNG should be unpredictable even for the manufacturer of the chip. In our further analysis, we assume that the attacker knows which bits of the finite state machines are used for the output, and how they are permuted.

To determine the time required to find the solution a system of equations as described above, fixed random choices of bits and fixed random permutations were used. Clearly these choices do not have essential influence on the complexity of solving the system of linear equations.

On a 400 MHz Pentium II, Mathematica 4.2 solved the system of equations in 0.06 seconds using the function LinearSolve[] . This time can definitively be improved significantly by using a faster PC or dedicated software for solving systems of linear equations over GF(2). But even when it takes 0.06 seconds to solve the system of linear equations, in the scenario of clock frequencies known with a precision of 1 percent, all 441 possible systems can be solved in 27 seconds. In the 10 percent scenario, it takes 17 hours to try all 1014049 possibilities. But as pointed out above, many combinations of numbers of clocks are quite improbable, so a good strategy for ordering the tries will enable the attacker to find the solution much faster. If the attacker tries all 1014049 possibilties, she will find about 15 solutions not corresponding to the internal states of the finite machines. The reason is that wrong guesses lead to a solvable system of linear equations with a probabilty of $1/2^{16}$. The attacker should prefer solutions for which the differences in the number of clocks are small.

5.4 Predicting bits

Once the attacker knows the internal states of the finite machines, she is well off. In order to predict the next output bits, she just has to guess the numbers of clocks of the finite state machines until the time the next random sample was generated. We have seen above that the number of cases to consider is quite small. But now the task of finding the right number of clocks is easier in two ways, compared to finding the correct number of clocks to determine the state. To be able to get the equations, Alice had to guess the right number of clocks for two samples. Here the number of clocks for one sample is sufficient. Alice can also profit from knowledge aquired when finding out the internal states of the finite machines. She may have started with little knowledge of the oscillator frequencies, but now she knows them with high precision, because she knows for which numbers of clocks the system of linear equations could be solved. This good knowledge of the oscillator frequencies leads to very few possibilities for the numbers of clocks for the finite state machines. Alice applies these numbers of clocks to a simulation of the finite state machines in order to compute the next output of the RNG.

6 Is the described attack practically relevant?

The attack described above enables an attacker to predict output bits from the RNG after having seen some earlier output bits. The question is whether there are practical security applications where such an attack could be applied.

One straight forward application of cryptographic RNGs is the generation of keys for symmetric cryptography. When a number of keys is generated subsequently for different users, the recipient Alice of a key could find out the key generated for the next user by applying the technique described above. Today, symmetric keys usually have 128 bits or more, so Alice can use her own

key to determine the state of the RNG and only has to try a very small number of possible keys for the next user. Of course she does not have to stop there, she can continue with the next user but one, and so on.

In the scenario just described, the attacker had to participate actively in a protocol in order to get her own key, from which she could derive they keys of other users. Can the attack of section 5 also be used by a passive attacker? For such an attack we need a protocol which generates and communicates random numbers in plaintext, and subsequently uses the RNG to generate a secret. This turns out to occur very often, namely the generation of a random challenge for challenge and response authentication, and subsequently the generation of a session key.

7 Conclusion

We showed that the random number generator described in [Tka03] is a combination of TRNG and PRNG elements. The TRNG elements produce little entropy when the random number generator is sampled at top rates. The output of the device can be predicted by taking into account both the small amount of entropy generated and the linearity of the PRNG elements.

How can these problems be overcome? Obviously by strengthening the TRNG elements and/or the PRNG elements.

The problem with the TRNG elements is that at top sampling rates the amount of state information it outputs largely exceeds the amount of entropy it generates. This can be cured by sampling less frequently, or by sampling less bits each time. We have seen in section 4 that the required reduction of the sampling frequency is rather impractical. To sample less bits each time the RNG is invoked, is more efficient. For example, if only one bit of output is generated in each output of the RNG, the data rate drops to 1/32 of the original design. But attacks like the one described above are impossible, because the device produces more entropy than it outputs state information.

Concerning the PRNG elements, non-linear components could be used to prevent attacks like the one described above. The disadvantage of only fixing the PRNG parts of the RNG is that this provides only computational security. Attacks are in principle still possible but require a large – hopefully too large for practical application – computational effort. In contrast, TRNGs provide information theoretical security.

References

[GW96] I. Goldberg and D. Wagner, *Randomness and the Netscape browser*, Dr. Dobb's Journal (1996), 66–70.

[Tka03] T. E. Tkacik, *A hardware random number generator*, Cryptographic Hardware and Embedded Systems - CHES 2002 (B. S. Kaliski Jr., C. K. Koç, and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 2523, Springer-Verlag, 2003, pp. 450–453.