A Provably Secure True Random Number Generator with Built-in Tolerance to Active Attacks

B. Sunar, W. J. Martin, D. R. Stinson
{sunar,martin}@wpi.edu
Electrical & Computer Engineering
Mathematical Sciences
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
dstinson@uwaterloo.ca
School of Computer Science
University of Waterloo
Waterloo Ontario, N2L 3G1
Canada

June 10, 2005

Abstract

This paper is a contribution to the theory of true random number generators based on sampling phase jitter in oscillator rings. After discussing several misconceptions and apparently insurmountable obstacles, we propose a general model which, under mild assumptions, will generate provably random bits with some tolerance to adversarial manipulation and running in the megabit-per-second range. A key idea throughout the paper is the fill rate, which measures the fraction of the time domain in which the analog output signal is arguably random. Our study shows that an exponential increase in the number of oscillators is required to obtain a constant factor improvement in the fill rate. Yet, we overcome this problem by introducing a post-processing step which consists of an application of an appropriate resilient function. These allow the designer to extract random samples only from a signal with only moderate fill rate and therefore many fewer oscillators than in other designs. Lastly, we develop fault-attack models, and we employ the properties of resilient functions to withstand such attacks. All of our analysis is based on rigorous methods, enabling us to develop a framework in which we accurately quantify the performance and the degree of resilience of the design.

Key Words: True (and pseudo-) random number generators, resilient functions, cryptography.

1 Introduction

Random number generators have numerous applications in a diverse set of areas ranging from statistics to cryptography to art. For many applications, pseudo-random number generators (PRNGs) — which take a short random string an expand it into a stream of "random looking" bits using a deterministic algorithm — are quite satisfactory. However, for cryptographic applications it is crucial to generate pseudo random bits which will be unpredictable even by the strongest adversary. Furthermore, even if high quality PRNGs, with their output being computationally indistinguishable from the output of a true random number generator (TRNG), may be built, these PRNGs will still need to be seeded using TRNGs.

Good TRNG design rests on the quality of three components:

- **Entropy Source:** Various TRNG designs have been proposed for harvesting randomness present in physical processes such as thermal and shot noise in circuits, brownian motion, or nuclear decay. The entropy source is the most critical component as it determines the available entropy. Some sources exhibit biases; these should be eliminated in the collection or post-processing steps.

- **Harvesting Mechanism:** The entropy source is tapped using a harvesting mechanism that does not disturb the physical process above yet "collects" as much entropy as possible. Various designs have been proposed to realize this step. A decent harvesting mechanism should come with rigorous justification with the underlying assumptions regarding the source explicitly stated.

- **Post-Processing:** Although this component is not needed in all designs, many TRNG designers strengthen their designs by post-processing the output bits. A post-processor may be applied to mask imperfections in the entropy source or harvesting mechanism, or to provide tolerance in the presense of environmental changes and tampering. A post-processor may be as simple as a von Neumann corrector [3] or may be as complicated as an extractor function [1] or a one-way hash function such as SHA-1 [3]. One should scrutinize post-processors which modify the output conditional on its statistical properties. There is great danger in deterministic methods aimed at improving the "appearance of randomness".

From a practical standpoint it is essential that TRNGs are built using a commonly available cheap silicon process. Moreover, it is highly desirable to implement TRNGs using purely digital design technique. This allows for easier integration with digital microprocessors, and also makes it possible to implement TRNGs on popular reconfigurable platforms (i.e. FPGAs and CPLDs). To date, various random number generator designs have been proposed based on mixed or pure digital electronics. These designs vary significantly according to their entropy sources and the harvesting techniques they employ. For instance, the design introduced in [5] uses a combination of analog and digital components for amplification and sampling of white noise. The main problem with this type of circuit is the amplification stage, which requires significant power to bring the noise level up a few orders of magnitude to the digital logic level. A similar design was developed by Intel Corp. [3], where the thermal noise on a junction is amplified and used to drive a voltage-controlled oscillator which is then sampled by another oscillator. The output sequence is post-processed using the von Neumann corrector and SHA-1. The design in [6] samples the jitter in a phase-locked loop (PLL) – an analog component – on a specialized reconfigurable logic platform. The innovative design introduced in [7] randomly samples the output of an LFSR and a cellular automaton. The randomness comes from the jitter in the two oscillator circuits which are used to clock the two deterministic circuits. Although the design is purely digital and no amplification is needed, it is difficult to verify the harvesting/sampling technique due to a complicated harvesting scheme. Moreover, despite the size of the design the entropy source is limited to the two oscillators. In another design [10], a simple architecture based on metastable circuits is proposed. The design passes the statistical tests only when a large number of such circuits are combined.

Indeed, all of these designs are validated by running the DIEHARD [11] or NIST Test Suites [12]. Typically, the optimal sampling frequency is determined by trial and error: the sampling frequency is decreased until the output sequence starts to pass the NIST or DIEHARD tests. Furthermore, due to the complex harvesting schemes employed, it is very difficult to give a mathematical justification for proper collection of the entropy.

Inspired by these observations we develop a set of requirements for our TRNG design:

- The design should be purely digital (no analog components are allowed),
- The harvesting mechanism should be simple (easy to analyze); it should preserve and optimally sample the entropy source. In other words, the unpredictability of the TRNG should not be based on the complexity of the harvesting mechanism, but only on the unpredictability of the entropy source.
- A strict mathematical justification of the entropy collection mechanism should be given, with all assumptions clearly stated and at least empirically justified. The design should be sufficiently simple to allow rigorous analysis.
- No correction circuits are allowed. Many times an adaptive correction circuit is used either to "adjust the sampling frequency" or to "smooth the output distribution". Since most of such circuits use the characteristic of the output to adaptively process the entropy source, they introduce further correlations. For instance, a correction circuit that counts the number of ones and zeroes and accordingly compensates the delay of a sampler will clearly introduce further bias to the output sequence.
- Compact and efficient design, (high throughput per area and energy spent). No amplifiers or other analog components are allowed. This essentially means that we have to sample variations in the time domain (such as the design in [7] does) rather than the variations in the voltage levels. This also means that we cannot use complicated post-processing schemes (e.g. SHA-1).

We propose a design that meets all of these requirements and allows for fault-tolerant extensions.

In the remainder of this paper we introduce a TRNG design that uses jitter (or random vibration) in clock signals present in all digital clocked circuits. Since such a signal will be available in most applications, this approach is particularly attractive. We outline several harvesting techniques and introduce a sampling method based on resilient functions.

2 Oscillator Rings

Oscillators provide a simple and effective method to build TRNGs [4]. A simple digital oscillator may be built by chaining an odd number of inverter gates in a ring configuration. Due to the instability, the output of any of the inverters will oscillate from a logic one to a logic zero and back. Hence, a square wave signal is obtained by tapping into the oscillator at any point in the ring. The signal has several characteristics:

- Ideally the output signal Ψ is a periodic square wave with its period determined by the number of inverters and and the delay of an inverter, i.e. $T = n \cdot \tau$ and $\Psi(t) = \Psi(t + T)$, where T denotes the period and τ denotes the delay of a single inverter.
- The output signal is *not* a perfect square wave. For instance, the period vibrates in a random manner $T = T + \hat{T}$ where $\hat{T}$ represents a random variable that takes values in the range $(-T/2, T/2)$. In a high quality digital circuit the range of $\hat{T}$ is quite small compared to the period. The vibration in the clock signal denoted by the random variable $\hat{T}$ is commonly called *jitter*. Jitter is the source of entropy we wish to harvest.

In Figure 1 a typical oscillator output is depicted. The jitter is represented via extra lines at each transition of the waveform.

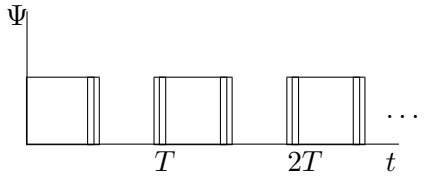

Figure 1: Periodic oscillator signal with transition zones marked as multiple lines.

A practical configuration for harvesting jitter is based on the idea of sampling the output of a ring oscillator using the output of another oscillator. This configuration is commonly referred to as *coupled oscillators*. If the periods of the two oscillators are well matched then it should be that with high probability we are sampling from the transition zones and not the deterministic part of the waveform. Unfortunately, problems arise in practical realization of coupled oscillators:

- Exactly matching the period of the two oscillators is fairly difficult and requires the use of special layout design techniques at the VLSI level.
- Due to imperfections, the two signals may *drift* relative to one another. This makes for very fragile TRNG designs.

To alleviate these two problems, additional circuitry (e.g. a compensator) is sometimes used to adjust the waveforms to match the transitions. This solution, however, *corrupts* the randomness of the jitter and introduces biases. In the next section we consider more sophisticated methods for combining and sampling oscillators.

3 Combining signals to exploit randomness of phase jitter

Our setting is as follows. We have r distinct oscillator rings arranged on a chip design. The i^{th} ring is composed of n_i inverters where the optimal parameters n_i and r are to be determined below. The analog output signal Ψ of the circuit is the XOR of the outputs Ψ_j of the r individual rings $R_1, \ldots, R_r$. The XOR function is typically implemented as a binary XOR-tree. The output Ψ is converted into a digital signal by sampling it at a regular clock frequency f_s.

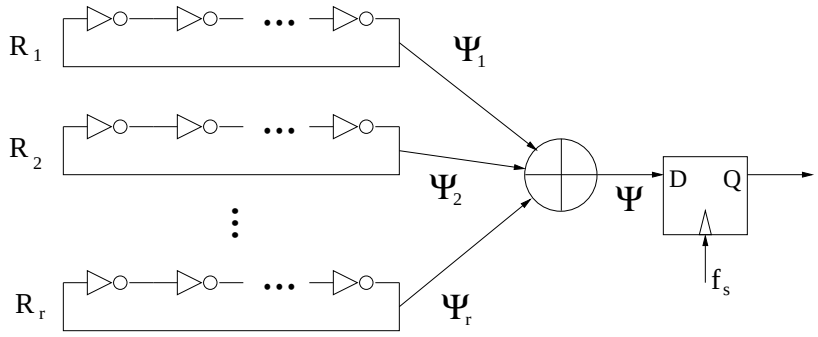

Figure 2: Design for combining and sampling oscillator rings

The output of the XOR combines periodic transition zones contributed by each ring. Since the entire waveforms are XOR-ed the output will also have deterministic regions, and occasionally

zones where jitter contributed from different rings overlaps. The resulting waveform exhibiting this effect with only two rings is depicted below in Figure 3. The deterministic parts of the waveform are not shown for clarity. The two oscillators are on identical phase and their ideal periods are related by $2T_2 = 3T_1$. Both oscillators contribute a transition zone at $t = 2T_2$ and any other even multiple of T_2 (marked by arrows).

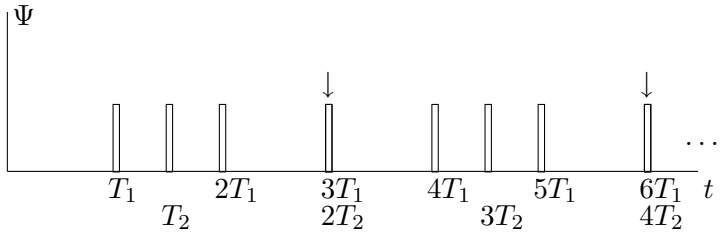

Figure 3: Output waveform of the XOR of two oscillator outputs with periods T_1 and T_2.

Our purpose is to populate the entire spectrum with such transition zones and then sample the waveform only in such zones. Due to the such overlaps we lose out on the entropy. Later we will tackle this problem, introducing several approaches to its resolution.

4 The Urn Model – a Combinatorial Approach

The combined signal is sampled to provide a random bitstream. The sampling of this combined signal Ψ is modeled as follows. The behavior of the XOR gate on signals which lie strictly between zero and one is well-documented to behave as a smooth interpolation. The combined signal is assumed to lie between L and H volts where $L < H$. A signal value closer to L is deemed to give rise to a zero bit and a signal value closer to H volts is recorded as a one.

We take the following axiomatic approach, which for the time being neglects phase drift. Suppose R_j is an oscillator ring with period $T = T_j$. That is, at times $0, T, 2T, \ldots$, the signal is designed to switch from low to high and at times $T/2, 3T/2, 5T/2, \ldots$ the signal is designed to switch from high to low. (Note that we aim to harvest jitter only from the "up" transition zones, allowing a full period before sampling jitter from the same ring.)

Assumption: *In any open time interval $(mT - T/4, mT + T/4)$, there is a unique point t where the signal crosses $(L+H)/2$ volts and this t behaves as a normally distributed random variable with mean mT and some variance σ_j^2.*

This hypothesis will be the basis of our claims regarding the random behavior of the overall system. Of course, such an assumption is unavoidable as any device which purports to produce random bits must be assumed to have access to some random physical source. Fortunately, the claim in our axiom is strongly supported by empirical evidence. In [2], for example, over a million oscilloscope captures (sampling in the GHz range) of jitter from a single oscillator ring with 83 inverters having $T \approx 150$ns were displayed and exhibited classical bell curve behavior.

Consider a time interval $I = [a, b]$ such as $[100T, 1100T]$ and suppose, for the sake of simplicity, that our trigger voltage is $(L+H)/2 = 2.5V$. Our goal is to "fill up this interval with randomness" using our combined signal composed of the output of k oscillator rings. More precisely, we would like to ensure that, for some threshold p, at any point t in interval I, there exists some ring R_j such that $1/4 < \mathsf{Prob}[\Psi_j < 2.5] < 3/4$. That is, for some integer m, $|t - mT_j/2| < .6745\sigma_j$. We will later relax this condition by simply asking that it hold with at least some probability q (e.g., $q = 0.9$) when t is chosen uniformly at random from interval I:

Criterion A: *With t chosen uniformly at random from the interval I, the probability that there exist integers j and m with $|t - mT_j| < .6475\sigma_j$ is at least q.*

In an ideal model, the above design criterion is not even sufficient to ensure that the output bitstream be balanced. Indeed, if the circuit consists of a single oscillator ring and the clock is perfectly calibrated to sample at times $mT + \epsilon$, then the above model holds but the output will consist almost entirely of ones. While the model can be refined to eliminate the possibility for such bias, we will work with it as given with the knowledge that phase drift and the large number of rings used makes such biased behavior highly improbable.

An extremely useful simplification in this regard is to reduce the problem to a combinatorial one. Let us consider a long time interval I and partition it into small equal-length subintervals in such a way that Criterion A is satisfied (with $q = 1$) for any time t in a subinterval J provided some ring R_j is in transition at some point t' in the same subinterval. That is, we discretize the time domian in such a way that any jitter event occuring in any subinterval J makes Criterion A hold (with $q = 1$) for all times t in interval J. We will call these subintervals "urns" because, as we shall soon see, the problem which now faces us is one well-known to probabilists who study urn models.

Here are the precise details of how the interval is to be subdivided into "urns". Let the interval $I = [a, b]$ be subdivided into ℓ equal length subintervals $J_1, \ldots, J_\ell$ where $J_h = [a + (h-1)\frac{b-a}{\ell}, a + h\frac{b-a}{\ell}]$. For the purpose of this discussion, we will refer to each J_h as an *urn*. An urn J is to be considered full if there is some ring R_j in the circuit whose signal satisfies $\Psi_j(t) = 2.5$ for some real number t in the urn J. Otherwise, the urn is empty.

Now we see that Criterion A is satisfied provided

- at least $q\ell$ of the urns are filled, and
- $\ell > (b-a)/.6475\sigma_j$ for all $j = 1, \ldots, k$.

Ignoring phase drift, we see that ring R_j will fill roughly one out of every $\pi_j = \ell T_j/2(b-a)$ urns. (We call π_j the "combinatorial period" of the ring R_j.) The next step is to decide how many inverters each ring should have in order to efficiently populate such an interval I with jitter events.

5 Relatively Prime Ring Lengths

Looking at Figure 3, one notices that there is "waste" of entropy when two rings are in transition at the same point in time. One is therefore tempted to find ways to minimize such overlap. It has been proposed that, in order to fill as many urns as possible and in order to make the behavior of the r rings as independent as possible, the ring lengths $n_1, \ldots, n_r$ should be pairwise relatively prime integers. While one experiment with such a design produced a digital output with good randomness properties (according to the NIST tests), the randomness is apparently not attributable to the above model. We now show this using the Chinese Remainder Theorem.

Suppose that ring R_j fills urns $d_j + m\pi_j$ ($m \geq 0$). It is reasonable to assume that, if the ring lengths n_j are pairwise relatively prime, then the combinatorial periods π_j are close to relatively prime as well. Then, with $N = \pi_1\pi_2\cdots\pi_r$, we have an urn J_h filled if there exists a j with

$$h \equiv d_j \pmod{\pi_j}.$$

The urn is therefore empty whenever

$$h \not\equiv d_j \pmod{\pi_j}$$

for all $j = 1, \ldots, r$. Modulo period π_j, there are $\pi_j - 1$ values not equal to d_j. So an integer h with $0 \leq h < N$ represents an unfilled urn whenever

$$h \equiv e_j \pmod{\pi_j}$$

for all j unless some $e_j = d_j$ modulo π_j. For any such system of congruences, there is a unique solution h modulo N. So among the N urns,

$$\prod_{j=1}^{r} (\pi_j - 1)$$

will be left unfilled. In other words, a randomly chosen urn will be left unfilled with probability

$$\left(1 - \frac{1}{\pi_1}\right)\left(1 - \frac{1}{\pi_1}\right) \cdots \left(1 - \frac{1}{\pi_r}\right).$$

Suppose that all $\pi_j \geq 25$ and we desire 2/3 of the urns to be filled. If the π_j's are required to be relatively prime, then it is not possible to do much better than choosing $\pi_1 = 25$, and choosing the rest of the π_j's to be distinct primes. It can be verified numerically that almost 900 rings will be required, due to the growth rate of prime numbers. Even for a fill rate of 50%, the number of gates on the chip quickly becomes infeasible. Moreover, the entire motivation for the relatively prime ring length model is unclear. Under our axiomatic assumption, the location of the transition point for the j^{th} ring is a normally distributed random variable. Ignoring phase drift, it carries no memory; there is no correlation between consecutive bits influenced by this single ring. Thus this feature of the model is not only impractical, but its value to the model is questionable to begin with.

6 Identical Ring Lengths

Our proposal is to take advantage of the random delays d_j as well as the phase drift of the signals relative to one another by making all ring lengths identical. Indeed, it is possible that two rings with identically many inverters will exhibit a great deal of overlap in their transition zones. These non-deterministic elements (initial delay and phase drift) make it highly likely that two such rings will have *no* overlap in their contributions to the jitter.

In what follows we will show that, using identical ring lengths, we can achieve reasonable fill rates with only a moderate number of rings, thereby keeping the total number of inverters down and allowing for portability of the hardware to fairly compact platforms.

Based on our combinatorial urn model, the problem of filling a long time interval I with jitter using identical ring lengths becomes one quite familiar to probabilists. This is the so-called "Coupon Collector's Problem" and it has been well-studied in discrete probability.

The Coupon Collector's Problem falls under the category of "urn models" in probability and is given as follows. There are N different coupons, but an unlimited supply of each of these. A coupon collector collects one coupon per day, choosing a coupon from among the N uniformly at random, with replacement. The challenge is to compute the expected number of days, r, which will pass before the collector has at least one copy of each of the N distinct coupons. We will need to delve deeper and estimate the number of days before the collector has 80% confidence that she has at least 95% of the coupons, or more generally confidence c out of 100 that the current number of coupons in her possession is at least fN where $0 < f \leq 1$.

6.1 Urn width

Before we get into the combinatorics of the coupon collector's problem, we need to know a bit about the actual numbers involved. A crucial parameter for the design is the width of these intervals we are calling urns. Suppose we have a design where each ring has a fixed number n of inverters. This determines a period T for the ring and, depending on the specifics of the hardware, a "jitter width" w which is technically the standard deviation of the associated random variable. Based on our desired entropy per bit of output, our urn width may differ from this value. Here is a small table:

Target entropy	$p = H_2^{-1}(E)$ min $Pr[X = 0]$	$A^{-1}(p)$ tolerance	num. urns ($w = 0.02T$)
0.99	0.441198	0.145σ	173
0.97	0.398388	0.258σ	97
0.95	0.269128	0.335σ	75
0.90	0.316019	0.479σ	52
0.80	0.243004	0.699σ	36
0.50	0.110028	1.229σ	20

Table 1: Urn width for various levels of entropy per output bit

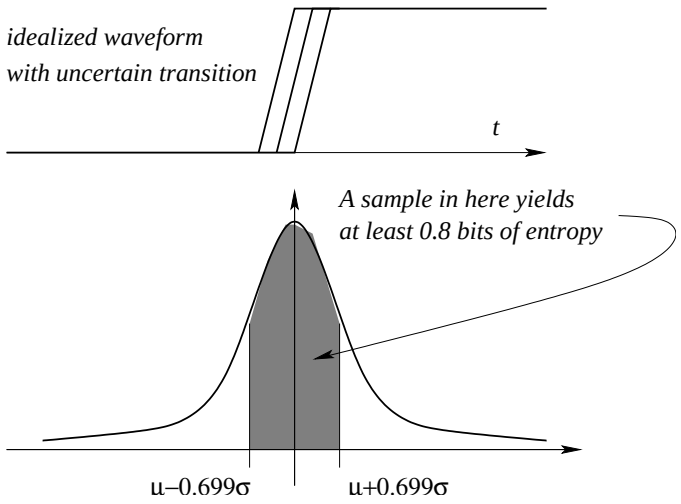

Figure 4: A single jitter event; sampling close to the mean yields a bit with high entropy

For a target entropy E per sampled bit, we take the inverse of the binary entropy function $H(p) = -p \log_2 p - (1-p)\log_2(1-p)$ to find how close to 0.5 the probability of sampling a zero must lie. The column $A^{-1}(p)$ is half the desired width of our urn. If sampling occurs within the interval $(\mu - A^{-1}(p), \mu + A^{-1}(p))$, then we are guaranteed the prescribed entropy per sampled bit. So the number of urns (using the example where the standard deviation w is two percent of the period), is given by $50/2A^{-1}(p)$. For other ratios of standard deviation to period, $\omega = \sigma/T$, we simply multiply the above number of urns by $0.2/\omega$.

6.2 Phase Drift

Whereas in the model involving relatively prime ring lengths phase drift of one component relative to another constitutes a major complication, phase drift can only improve the randomness of the identical ring length model. Indeed, in each sequence of π consecutive urns, we are assuming that each ring fills one randomly chosen urn (that urn whose index is congruent to d_j modulo π) and that these d_j are independently chosen uniformly at random. Experimental evidence suggests exactly that phase drift, if it does occur in any non-negligible way, is independent from one ring to another. In our model, this is simply interpreted as ring R_j choosing a slightly different delay d_j in each interval I of π urns. The delays remain uniformly distributed and independently chosen, so the model still applies.

The only complication is that, in the event that some ring has its transition point close to the end of I (i.e., that d_j is close to zero modulo π), a ring may fill no urns or two urns in some consecutive sequence of N urns. This can be compensated for by making a slight increase in the total number of rings so that in any such sequence of N consecutive urns, the number of rings filling at least one urn is at least the desired number in the above table with very high probability.

6.3 Coupon Collecting

With all ring lengths identical, the time spectrum gets partitioned into equal-length subintervals, which we call "urns". If we focus on the urns within one period of the ring, we wish to fill as many of these as possible using as few rings as possible. Allowing for the randomness of phase shifts and owing to uncontrollable phase drift, we model each oscillator ring as filling one urn per period with a jitter event in such a way that sampling at any point in this subinterval is assumed to yield a random bit.

Our problem can therefore be modeled as one in discrete probability: the coupon collector problem, where one selects one urn from a set of N uniformly at random until each urn has been "filled", or selected at least once. The power and limitation of this approach is already evident from the most elementary analysis of the coupon collector problem.

Let us add rings to the design one at a time and view these as uniformly random selections of one urn from among N. Suppose that, at some point in time, all but a fraction qN of the N urns have been filled. Then there is a probability q that the next selection will fill a new urn and a probability $(1-q)^{k-1}q$ that it will require exactly k more selections before a new urn is filled. So the expected number of selections required to fill the next urn is

$$\sum_{k=1}^{\infty}(1-q)^{k-1}\cdot q\cdot k=\frac{1}{q}.$$

For example, when half the urns are filled, the expected number of additional rings required to fill the next one is two and when 3/4 are filled, it is four.

Therefore the expected number of rings needed to fill all N urns is

$$r=\sum_{s=1}^{N}\frac{N}{s}=N\sum_{s=1}^{N}\frac{1}{s}\approx N\log N.$$

where log(.) denotes the natural logarithm function.

Clearly this suggests that we need a very large number of rings. Of course, we are penalized the most for our desire to fill even the last few urns. For example, with $N = 50$, 100 and 200, Table 6.3 gives the expected number of rings required to fill fN urns for various values of f. The table is

	f					
N	0.50	0.60	0.70	0.80	0.90	1.00
50	34.2	45.1	59.0	78.5	110.8	224.9
100	68.8	90.9	119.2	159.0	225.8	518.7
200	138.1	182.5	239.6	319.9	456.0	1175.6

Table 2: The expected number of rings required to fill fN urns for various values of the fill rate f.

built by computing the exact summation. For instance, to compute the number of rings r needed for $N = 100$ urns, to obtain a fill rate of $f = 0.70$, we use the summation $r = 100 \cdot \sum_{s=31}^{100} 1/s$.

So our strategy, of course, is to aim for a fill rate, f, less than 1.0 and to compensate for the resulting fraction of non-random samples.

6.4 A more detailed analysis of fill rates

For a given number N of urns, a given fill rate $0 < f \leq 1$, and a given level of confidence $0 < p < 1$, we would like to determine the minimum number $r = M(N, f, p)$ of rings necessary so that, among the m urns, the event that at least fN are filled has probability at least p. For $f = 1$, this is the Coupon Collector's Problem.

For our application, the numbers $M(N, 1, p)$ are too high for practical implementation unless p is taken so small as to seriously impact the robust design features we seek. So we will prefer to keep the confidence p close to one while decreasing the fill rate f and we will present a rigorous post-processing strategy in Section 7 to recover full confidence in the quality of the bits generated by the overall design.

Unfortunately, there is no known closed form expression for $M(N, f, p)$. But the following related function is well-studied. Let $P(N, r, f)$ denote the probability that at least fN out of N urns are filled with exactly r rings. Then

$$M(N, f, p) = \min \{r : P(N, r, f) \geq p\}$$

and these values can be quickly generated by a computer algebra system.

Let us re-phrase this last problem in the language of surjective functions. There are N^r functions from an r-set to an N-set. This is the total number of ways to associate each ring in our design to one of the N urns. (Of course, with variability in periods, there is a non-zero probability that some ring will not be in transition in a given interval and also a chance that some ring will exhibit two transitions in a given interval; we take these as negligible.) The number of surjections (onto functions) among these N^r is given by inclusion-exclusion:

$$J(r, N) := \sum_{h=0}^{N} (-1)^h \binom{N}{h} (N - h)^r.$$

To prove this, simply note that each function $f : [r] \to [N]$ is a surjection onto some non-empty subset of $[N] := \{1, 2, \ldots, N\}$ and that this is a partition of all functions into surjections. It immediately follows that the number of functions $f : [r] \to [N]$ with an image of size at least fN is

$$\sum_{k=\lceil fN \rceil}^{N} \binom{N}{k} J(r, k).$$

The probability that the image has size at least fN is then this value divided by N^r, the total number of functions. This gives us our desired value $P(N, r, f)$.

In Table 6.4, we give $M(N, f, p)$ for $N = 36$ urns. (See Appendix for similar tables for $N = 52$ and 100.)

	f									
p	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.50	26	30	35	41	48	52	61	74	94	142
0.55	27	31	36	42	49	53	62	76	96	147
0.60	27	31	37	42	50	54	64	77	99	153
0.65	27	32	37	43	51	55	65	79	101	159
0.70	28	33	38	44	52	56	66	81	104	165
0.75	29	33	39	45	53	57	68	83	108	173
0.80	29	34	40	46	54	59	70	86	111	182
0.85	30	35	41	48	56	61	72	89	116	193
0.90	31	36	42	49	58	63	75	93	122	208
0.95	33	38	45	52	61	67	80	99	132	233
0.99	36	42	50	58	68	75	90	113	153	291

Table 3: For the case $N = 36$ urns shows the number r of rings necessary to fill at least fN of the urns with probability at least p.

7 Resilient Functions

Even in our model with identical ring lengths, in order to keep the number of rings within a practical range, we have been forced to compromise on the number of filled urns. A sampler which merely measures the signal at regular intervals and turns these readings into zeros and ones will have, say, 90% random bits (each with at least 25% chance of being a zero and at least 25% chance of being a one) and 10% of the samples will hit every ring in a "flat range". That is, these few samples will be deterministic functions of time. But we don't know which are the random ones and which are deterministic. In what follows, we based our model on the assumption that the random bits behave as unbiased coin flips, while being aware that is a non-trivial simplification.

While this level of randomness may be satisfactory for some applications, we seek to eliminate these non-random components to our output by boiling down the bitstream to a shorter, more random sequence of bits. We accomplish this using resilient functions.

Definition 1 *An (n, m, t)-resilient function is a function*

$$F(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_m)$$

from $\mathbf{Z}_2^n$ to $\mathbf{Z}_2^m$ enjoying the property that, for any t coordinates $i_1, \ldots, i_t$, for any constants $a_1, \ldots, a_t$ from $\mathbf{Z}_2$ and any element y of the codomain

$$\mathsf{Prob}[F(x) = y | x_{i_1} = a_1, \ldots, x_{i_t} = a_t] = \frac{1}{2^m}.$$

In the computation of this probability all x_i for $i \notin \{i_1, \ldots, i_t\}$, are viewed as independent random variables each of which takes on the value 0 or 1 with probability 0.5.

In more informal terms, knowledge of any t values of the input to the function does not allow one to make any better than a random guess at the output. Resilient functions are used in a number of cryptographic applications.

In our setting, the input to the resilient function is the sequence of bits generated by the chip as described above. We have argued that the output is composed of a large number (e.g., $0.9L$) of random bits infiltrated by a small number (e.g., $0.1L$) of bits which are deterministic functions of the circuit. If we feed these L bits into an $(L, m, L/10)$ resilient function, the output must necessarily consist of m truly random bits.

We now show how resilient functions may be obtained from error-correcting codes. A simple technique for constructing resilient functions is given in the following theorem:

Theorem 1 *(e.g., [14]) Let G be a generator matrix for an $[n, m, d]$ linear code C. Define a function $f : \{0,1\}^n \mapsto \{0,1\}^m$ by the rule $f(x) = xG^T$. Then f is an $(n, m, d-1)$-resilient function.*

For more information on resilient functions, and their connections to codes and designs see [8] and [9].

Equipped with a construction for resilient functions, we can choose from two approaches in our TRNG design:

- In the urn model, since we do not have control over which urns will filled by the jitter from any given ring, we may over-compensate by adding a large number of extra rings so that we have a reasonable chance of filling out the spectrum. From the above discussion of the coupon collector problem, we need to use $N \log_2 N$ rings to fill N urns. To be precise, N denotes the ratio between a clock period and the standard deviation of the transition at the edges (i.e., the effective jitter width).

- Alternatively, we can use a reduced number of rings and have only a portion of the spectrum filled with jitter. This means that we will occasionally sample from the deterministic portions of the signal. To filter out these deterministic bits we use a resilient function with appropriate parameters. The output of the resilient function will give us purely random bits with high probability.

The second approach is clearly preferable, since it drastically reduces the number of rings we need to use in our design. On the other hand, we now have to implement a resilient function. Note that we do not necessarily need to build the resilient function in hardware. Since it is a purely deterministic function, it may easily implemented in software as a post-processing step. On the other hand, if one wishes to implement the resilient function in hardware this can still be achieved efficiently. All one has to do is to implement a vector times a (constant) matrix product as described in Theorem 1. Hence, the presented method does *not* require full error decoding, but is rather akin to calculation of a syndrome. Moreover, if a cyclic code is used to construct the resilient function, one may use a simple linear feedback shift register (LFSR) to obtain an even more compact implementation. Since such constructions are well known and published we do not go into the implementation details any further.

There is a trade-off between code length the size of the buffers we need to use in the implementation of the resilient function. A code of short length is easy to implement and requires smaller buffers, but using such a code runs a higher risk of being compromised when there is a burst of errors created by natural causes or by an intelligent attacker. For example, a code with minimum distance $d = n/2$ used with 80% fill rate has a per-block failure rate at most 7×10^{-52} for $n = 1024$. But a code with $n = 8$ will fail roughly 13.2% of the time with same input stream, giving a 4-bit

sequence of entirely deterministic output. (The 80% fill rate on a long input stream is here modeled as a 0.2 per bit probability of determinism so that the resilient function fails on an n-bit input block with probability $\sum_{k=n/2}^{n} \binom{n}{k} 0.2^k$.)

At this point we introduce a very simple code that is suitable for our application.

Definition 2 *The simplex code $H_m^{\perp}$, the dual of the Hamming code H_m, is a $[2^m - 1, m, 2^{m-1}]$ linear code.*

Note that the minimum distance and the resilience of a resilient function build from a simplex code is roughly half of the block length. Although the output length is logarithmically related to the length of the input, this provides a class of resilient functions with particularly strong resistance against faults.

8 Tolerance Against Attacks

We only consider fault-introduction attacks since they are significantly more powerful and more easily implemented on true random number generators than bit-deduction attacks. Simply stated, one can say that it is sufficient to introduce a bias in the output of a TRNG for an attack to be considered successful. We analyze the tolerance of our design against attacks in two categories:
Non-invasive attacks: The types of attacks we consider under this category have to do with *external* influences that an attacker might employ including but not limited to, improperly biasing input and/or output bits (e.g. introducing spikes in the power supply), applying electromagnetic shocks to the chip, forcing abrupt temperature changes etc. For most of the attacks in this category the duration of the adversaries' influence is limited. Since our design is stateless, only a finite number of bits at the output are influenced. This allows the design to eliminate the *corrupted* bits by simply using a stronger resilient function (i.e., employ an error-correcting code with larger minimum distance).
Invasive attacks: Attacks of this type are more demanding of resources from the attackers point of view. However, invasive attacks are more powerful and usually have permanent effect on a circuit. For instance, an attacker may insert a pin (or burn a hole by using a strong laser) into the chip and thereby introduce permanent defects, e.g. cause a short or open circuit by taking out a transistor. Since the attacker's goal is to damage the TRNG and thereby bias its output bits, his job is easy — complex under-the-microscope work that requires high precision is not necessary for him.

To counter such powerful attacks we need to analyze our design in parts and make sure all parts display the same level of resistance.

- **Oscillator Rings:** An attacker may introduce a permanent fault by taking out one or more oscillators, and thereby turning one or more of the ring outputs into deterministic/constant signals. As in the case of non-invasive attacks our design can easily handle a number of such faults determined by increasing the strength of the resilient function, i.e., by a certain amount of over-engineering in the original design.

- **XOR-tree:** We assume that, in the realization of the design, an XOR-tree is used to hash the ring outputs into a single bit which is later sampled. The intuition behind using such a structure is that, as long as any one of the input bits is random, the output of the tree will be random. A four bit version of a binary XOR-tree is shown on the left in Figure 5. In the first level the ring outputs are XOR-ed together in a pairwise manner. If any single one of these XOR gates are faulted by the adversary fixing its output to either a zero or one output bits,

the effect may be seen as two of the ring outputs have become deterministic and as before the bias will be eliminated by the resilient function as much as the built-in strength parameter t permits. If any single XOR-gate in the second level is faulted, the effect is stronger since now the output of that XOR gate comes from four rings outputs. The effect of such faults clearly doubles at each level and reaches its optimum (as a threat to performance) at the last level, where there is a single XOR-gate. This is a serious problem, since now we have a much weaker spot in the design than the rest of the circuit. One way to get around this problem is to simply replicate the XOR-tree and generate the output bit multiple times to meet any desired fault-tolerance criterion, but this is a highly wasteful method. For example, a single XOR-tree to hash outputs from r rings will require r XOR-gates (assuming r is a power of two). The direct r-fold replication would require r^2 gates to provide the same level of fault-tolerance at the last level as at the top.

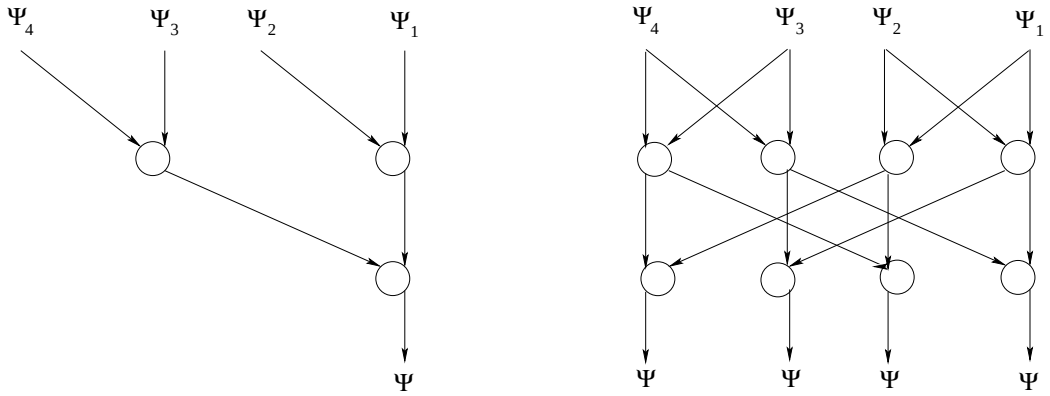

Figure 5: The original binary tree (left) and the redundant tree (right)

To alleviate this problem we propose to replace of the binary XOR-tree with the expanded XOR trellis shown on the right of Figure 5. As easily seen from the diagram, each XOR-gate contributes now the same level entropy. However, since we still need a single stream of output bits, we need to change the sampling mechanism. That is, the output lines should be sampled one after another in a sequential scheme. Hence, each output line is sampled once in each clock cycle.

We briefly remark that the trellis depicted is easily scalable. For $r = 2^k$, we have one node for each pair (a, i) where a is a binary k-tuple and $0 \leq i \leq k$. Node $(a, 0)$ corresponds to the signal Ψ_a and there is a wire from node (a, i) to $(b, i + 1)$ when the binary k-tuples a and b agree in all but possibly the i^{th} position.

- **Sampler:** The sampler for the original binary XOR-tree circuit output would be implemented as a simple D-flip flop with its clock input set to a signal switching at the sampling frequency. The problem with this approach is, as before, we have a single point of failure. Since we want to protect the circuit from invasive attacks, the only way to get around this problem is to sample all output lines in the expanded tree by simultaneously using r flip-flops, and then connecting their output lines to the chip pins. Consequently, the number of output lines may become too excessive even for a moderate number of rings. In this case, one may employ a strategy where a pruned expanded trellis is employed which would have a reduced number of output lines. Such a circuit can be easily constructed by removing some of the output lines

from the trellis in Figure 5 as well as the sub-circuits feeding them in the diagram. In the resulting design, the weakest gates in the circuit will again be those gates at the output stage.

- **Resilient function:** If the resilient function is implemented as part of post-processing in software then there is no problem since we assume that the post-processing step is performed on a trusted system. Otherwise, similar precautions need to be taken to implementation of the resilient function as well. Since the resilient function, in general, will have an irregular structure compared to the XOR-tree, making it tolerant to invasive attacks is more of an ad hoc process. For example, one may use redundant logic expressions (by avoiding logic minimization) as a means of making it fault-tolerant. On the other hand, if cyclic codes are employed, the resilient function may be realized by using an LFSR circuit. If this is the case, taking advantage of the regularity, one may easily replicate the storage elements to achieve fault-tolerance.

9 Putting it all together – a sample design

In this section we develop a sample design by picking realistic parameters. We quantify the entropy rate and the throughput of the random bit stream generated by our circuit.

If our rings use 13 inverters, then experimental evidence shows that the period is roughly 25 ns and the standard deviation for the jitter random variable is $\sigma = 0.5$ ns [2]. So $\sigma = 0.02T$. Now we want all samples to be within $\frac{1}{4}\sigma$ of the mean of some jitter event. So, with tolerance $(\mu - \frac{1}{4}\sigma, \mu + \frac{1}{4}\sigma)$, we can say that 1% of the spectrum is filled with jitter for each ring. So we have $N = 100$ urns in our combinatorial model. We note that the tolerance $\frac{1}{4}\sigma$ ensures that each generated bit will yield at least 0.97 bits of entropy per sampled bit.

With these parameters, the formulae developed in Section 6.4 (or last row, third entry in Table 5) tell us that $r = 114$ rings will be enough to fill at least $0.60N$ of the urns with probability at least 0.99. The output is sampled from the 114 rings and fed into a resilient function. Now, for our resilient function, we employ a $[256, 16, 113]$-code which is a known extended BCH code [13]. [1] This means that the samples are grouped into blocks of 256 bits and fed into the resilient function which returns only 16 bits. The code we selected has minimum distance $d = 113$ and therefore the resilience of the associated resilient function is $t = 112$. With a block length of 256 and fill rate of 0.60, out of the 256 bits which go into our resilient function $(1 - 0.60) \cdot 256 = 102.4 \approx 103$ bits will be deterministic. Since our resilient function can tolerate up to 112 corrupted bits, the design has an additional margin to resist additional (adverserial and non-adverserial) faults and errors of up to 9 bits. The error margin or tolerance may be increased by either increasing the number of rings or by using an error-correcting code with larger minimum distance.

The output is 16 bits per each 256 bits sampled. These 16 bits each have 0.97 bits of entropy. Since the frequency of the circuit is $\frac{1}{25ns} = 40$ Mhz, this model gives us a random stream with bit rate of $\frac{16}{256}40 = 2.5$ Mbps where each bit carries 0.97 bits of entropy. Thus, with modest parameters, we obtain a rather fast random bit generator with provably secure behavior.

10 Conclusion

In this work we have outlined the design of a true random number generator which can be built using purely digital logic. We developed design principles that lead to a small footprint design. The

[1] Although there are many choices for the code we use as our resilient function as discussed above, we choose here a good cyclic code which is not a simplex code.

elements are simple to analyze and implement. At the same time, our scheme allows one to increase fault tolerance of the random bit generator against active adversaries to any desired level. One rare feature of the design is that it is given in a framework which allows one to exactly quantify the performance and the degree of resilience of the design. The small set of basic assumptions regarding the physical properties of the platform are experimentally supported. We identify further experimentation, especially across a variety of reconfigurable (FPGA, CPLD) and ASIC platforms, as future work.

References

[1] Boaz Barak, Ronen Shaltiel, and Eran Tomer. True Random Number Generators Secure in a Changing Environment. In Çetin K. Koç and Christof Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, pages 166–180, Berlin, Germany, LNCS 2779 2003. Springer-Verlag.

[2] Abcunas Brian, Sean Patrick Coughlin, Gary Thomas Pedro, and David C. Reisberg. Evaluation of RNGs Using FPGAs, May 2004. Worcester Polytechnic Institute, Major Qualifying Project Report.

[3] Benjamin Jun and Paul Kocher. The Intel random number generator, April 1999. White Paper Prepared for Intel Corporation.

[4] Raymond A. Schulz. Random Number Generator Circuit. United States Patent, Patent Number 4905176, February 27 1990.

[5] Vittorio Bagini, and Marco Bucci A Design of Reliable True Random Number Generator for Cryptographic Applications. In Çetin K. Koç and Christof Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 1999*, pages 204–218, Berlin, Germany, LNCS 1717 1999. Springer-Verlag.

[6] Viktor Fischer, and Miloš Drutarovský True Random Number Generator Embedded in Reconfigurable Hardware In B.S. Kaliski Jr. et al., editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, pages 415–430, Berlin, Germany, LNCS 2523 2003. Springer-Verlag.

[7] Thomas E. Tkacik A Hardware Random Number Generator In B.S. Kaliski Jr. et al., editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, pages 450–453, Berlin, Germany, LNCS 2523 2003. Springer-Verlag.

[8] C. J. Colbourn, J. H. Dinitz, and D. R. Stinson, Applications of combinatorial designs to communications, cryptography and networking, Surveys in Combinatorics, 1999, pages 37–100, (1999 British Combinatorial Conference).

[9] D. R. Stinson and K. Gopalakrishnan, Applications of Designs to Cryptography, In: CRC Handbook of Combinatorial Designs (C. D. Colbourn, and J. H. Dinitz, eds.) CRC Press 1996.

[10] Michael Epstein, Laszlo Hars, Raymond Krasinski, Martin Rosner, and Hao Zheng Design and Implementation of a True Random Number Generator Based on Digital Circuit Artifacts C.D. Walter et al. editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, LNCS 2779, pages 152-165, 2003. Springer-Verlag Berlin Heidelberg 2003.

[11] Marsaglia, G. DIEHARD: A Battery of Tests of Randomness, http://stat.fsu.edu/ geo, 1996.

[12] NIST Special Publication 800-22 A Statistical Test Suite for Random and Pseudorandom Numbers. December 2000.

[13] Andries E. Brouwer Server for bounds on the minimum distance of q-ary linear codes, $q = 2, 3, 4, 5, 7, 8, 9$. URL: http://www.win.tue.nl/ aeb/voorlincod.html

[14] B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, R. Smolensky The bit extraction problem or t-resilient functions, in: 26th IEEE Symposium on Foundations of Computer Science, 1985, pages 396–407.

Appendix

	f									
p	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.50	37	44	51	57	66	74	88	107	126	179
0.55	38	44	52	57	67	75	89	109	128	183
0.60	38	45	53	58	68	76	91	111	131	188
0.65	39	46	53	59	69	78	93	113	134	194
0.70	39	46	54	60	71	79	94	116	137	199
0.75	40	47	55	61	72	80	96	118	140	206
0.80	41	48	56	63	74	82	99	121	144	213
0.85	42	49	58	64	75	84	101	125	149	223
0.90	43	51	59	66	78	87	105	130	155	235
0.95	45	53	62	69	81	91	110	137	164	255
0.99	49	57	68	75	89	100	121	152	185	299

Table 4: For the case $N = 52$ urns shows the number n of rings necessary to fill at least fN of the urns with probability at least p.

	f									
p	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
0.50	70	81	93	106	122	140	163	192	234	307
0.55	71	82	94	108	123	142	165	195	237	312
0.60	72	83	95	109	125	144	167	197	241	317
0.65	73	84	96	110	126	145	169	200	244	323
0.70	73	85	97	111	128	147	171	203	248	329
0.75	74	86	98	113	129	149	174	206	253	336
0.80	75	87	100	114	131	152	177	210	258	344
0.85	77	88	101	116	134	155	180	214	263	354
0.90	78	90	104	119	137	158	185	220	271	366
0.95	81	93	107	123	141	164	191	228	283	386
0.99	86	99	114	131	151	175	205	246	307	427

Table 5: For the case $N = 100$ urns shows the number n of rings necessary to fill at least fN of the urns with probability at least p.