Robust, low-cost, auditable random number generation for embedded system security

Ben Lampert$^{\star\circ}$ lampert.b@gmail.com
Riad S. Wahby* rsw@cs.stanford.edu
Shane Leonard* shanel@stanford.edu
Philip Levis* pal@cs.stanford.edu

*Stanford University
$^{\circ}$NAUTO, Inc.

Abstract

This paper presents an architecture for a discrete, high-entropy hardware random number generator. Because it is constructed out of simple hardware components, its operation is transparent and auditable. Using avalanche noise, a non-deterministic physical phenomenon, the circuit is inherently probabilistic and resists adversarial control. Furthermore, because it compares the outputs from two matched noise sources, it rejects environmental disturbances like RF energy and power supply ripple. The resulting hardware produces more than 0.98 bits of entropy per sample, is inexpensive, has a small footprint, and can be disabled to conserve power when not in use.

1. Introduction

Random numbers are fundamental to cryptography and computer security. Virtually every cryptographic primitive (symmetric ciphers, public key cryptography, signatures, certificates) depends on random bits. Poor random number generation leaves systems and applications open to attack [11].

Embedded systems, including sensor networks and the Internet of Things, are increasingly reliant on cryptography. While early sensor networks depended on link-layer encryption and used pre-installed keys [27], modern best practices require devices to use randomness, e.g., for periodic key rotation and per-session keys [35]. Because these devices are deployed for years (even decades) and difficult to patch, a vulnerability can be especially disastrous; thus, following best practices is vitally important in the embedded setting.

In secure systems, an important metric is *entropy*, informally, the number of bits of information not known to an adversary. Given sufficient entropy, a device can generate an unpredictable sequence using a deterministic algorithm called a pseudorandom number generator (§2.1). However, if entropy is limited, an adversary might be able to predict the algorithm's output after searching only a small space.

A conservative approach employed in many systems is to combine entropy from many different sources. In the non-embedded context, systems often run on complex hardware with many peripherals. In these systems, disk seeks, network packet arrivals, and keyboard interrupts can all contribute entropy. Unfortunately, the constraints of embedded devices make this approach challenging: many embedded systems are simple, low-power devices with few entropy sources.[1]

Another approach to entropy gathering is to use a purpose-built hardware random number generator. These on-chip random number generators, which are commonly included on modern processors and high-end microcontrollers, use a physical process such as thermal noise [19, Ch. 11] to generate random bits. But integrated random number generators pose two problems when building a trustworthy, secure system. First, in most cases there is no description or specification of their operation, and thus no way to know how much entropy they provide or how to use them safely. Second, even in cases where the circuit's design is documented (as with Intel's RDRAND and RDSEED [20]), verifying that a chip uses the documented design is prohibitively expensive [6, 53], and bugs that subtly alter the circuit's operation are easy to introduce and extremely difficult to detect [5]. As Linux kernel developer Ted Ts'o comments, "Relying solely on the hardware random number generator which is using an implementation sealed inside a chip which is impossible to audit is a BAD idea" [58].

As an alternative to using an on-chip generator, an embedded system designer might build her own circuit.[2] This approach has the advantage that the circuit's implementation and operation can be examined and tested. Further, there are many such circuits to choose from, both paper and real. However, many of these designs do not respond to the unique constraints of embedded systems with regard to size or power consumption. Others are inadequately specified or tested and might fail, for example, when deployed in a cold environment. We leave a detailed discussion of these limitations to Section 2.2.

This paper presents the Lampert circuit, an entropy generator designed for embedded systems. Section 3 explains how a Lampert circuit uses avalanche noise as its source of entropy. Avalanche noise is caused by the behavior of electrons in a reverse biased diode (§3.1) and is therefore fundamentally random. However, turning avalanche noise into a high-quality

Sensys '16, November 14–16, 2016, Stanford, CA, USA

ISBN 978-1-4503-4263-6.

DOI: http://dx.doi.org/10.1145/2994551.2994568

[1] While there are techniques for extracting randomness from low-entropy sources [29, 55], the cost of this extraction is prohibitive in resource-constrained embedded environments.

[2] We assume that building an integrated circuit is out of scope for many embedded systems designers, and thus limit our focus to discrete circuits.

entropy source in an embedded device involves several technical challenges. First, the design must quantize the analog noise into a stream of random bits. Existing designs compare the noise source's output to a reference voltage, which can result in sensitivity to environmental conditions. A key insight, described in Section 3.3, is that comparing the outputs from two matched noise sources greatly reduces sensitivity to external disturbances.

Another challenge is that generating avalanche noise requires a high voltage (12–18 V). Power converter circuits for generating this voltage from the 1.8–3.3 V typical on embedded devices introduce strong disturbances that influence the circuit's output, reducing entropy. Section 3.4 describes how the Lampert circuit carefully times the operation of the power converter to eliminate these disturbances.

A Lampert circuit is small: it occupies less than 1.5 cm^2 of board area. It is inexpensive: its components cost less than \$1.50 at reasonable scale. It is simple and therefore simple to test and audit: each component has very well-defined behavior. In Section 7 we evaluate the entropy of its output and find that it generates > 0.98 bits per sample, with little sensitivity to environmental effects such as temperature. We also find that a device using a Lampert circuit can generate more than 1000 bits of entropy in 25 ms at boot, seeding a pseudorandom number generator for its entire lifetime.

This paper makes three contributions:

- a new circuit design for generating entropy in an embedded device, based on the comparison of two balanced sources of avalanche noise,
- an end-to-end analysis of the tolerances and issues in incorporating the circuit into practical designs, and
- a careful evaluation of the circuit, showing that it produces > 0.98 bits of entropy per sample and is insensitive to temperature.

2. Background

In this section, we describe the role of random numbers in computer security (§2.1), study the design space of hardware entropy generation by briefly surveying existing designs (§2.2), and distill our findings into requirements for random number generator suitable for embedded systems (§2.3).

2.1 Randomness and computer security

Because the role of entropy in random number generation for computer security is often misunderstood, we provide a brief overview of the principles. Interested readers may refer to Corrigan-Gibbs et al. [11] for a more detailed discussion.

A crucial property of secure random number generation is unpredictability: an adversary has negligible probability of guessing the output of a secure random number generator. This is a stronger requirement than, for example, uniformity. The additive feedback generator used in GNU libc's `rand` illustrates the difference: its output is close to uniform, but an adversary observing about 30 outputs in sequence can trivially predict future outputs [46].

Other, more sophisticated statistical tests might reveal the weakness of `rand`, but they still provide no guarantees about unpredictability. As an example, π is hypothesized to be a normal number [36], meaning that its digits will pass statistical tests for self-correlation, periodicity, and bias. But an adversary who knows that a system's "random number generator" just computes digits of π will have no trouble predicting future PRNG outputs.

The requirement for unpredictability has driven the development of cryptographically secure pseudorandom number generators (CSPRNGs, or just PRNGs). Informally, a PRNG is a deterministic algorithm that produces a sequence whose future outputs cannot be predicted by any efficient algorithm given access to past outputs. As with other cryptographic primitives, an adversary is assumed to have access to the algorithm, but not to a secret key, called the seed. Thus, given an ideal PRNG, the seed's entropy—the number of bits of the seed that the adversary does not know—measures the difficulty of guessing the PRNG's future outputs.

In practice, secure systems gather entropy from their environment, for example, by measuring the timing of unpredictable events or the evolution of physical processes. This raises the question: why not gather fresh entropy each time a random number is needed? The answer is that gathering entropy is often slow and energetically costly compared to computing the next output of a PRNG.

Finally, we note that secure stream ciphers (including block ciphers in counter mode) are suitable PRNGs. For example, given a 128-bit seed k, encrypting the sequence $\{0, 1, 2, \ldots\}$ under k using AES-128 is a secure PRNG, under standard cryptographic assumptions. Further, frequent re-seeding is unnecessary: given current notions of AES-128's security, a conservative estimate is that this arrangement can be used to generate 2^{60} bytes without re-seeding.

2.2 Related work

There is a large body of commercial and academic work on hardware RNGs. We decompose this work into four categories: circuits integrated on processors and systems-on-chip (SoCs), commercial RNG peripheral devices, designs for FPGAs, and discrete designs.

At a high level, all of these circuits extract randomness from an underlying physical process. These include circuit noise phenomena (thermal noise, shot noise, flicker noise, Zener noise, or avalanche noise) [19, Ch. 11], and quantum phenomena (photon beam splitting [33], photoelectric effect [47, Ch. 40], or radioactive decay [47, Ch. 44]). The physical process drives nondeterministic circuit behavior, which is measured and converted to bits.

Processors and SoCs. Some CPUs and SoCs include built-in RNG circuits. Intel's Ivy Bridge random number generator uses a latch which is repeatedly driven to metastability [39], then allowed to settle to a logic value [20]. The noise source here is implicit in the circuit components comprising the metastable latch; it is likely dominated by flicker and thermal noise. Several SoCs, including products from Broadcom [4] and TI [1], also integrate RNGs; to our knowledge, none of the underlying circuit designs are documented. Chips with integrated RNGs are convenient; on the other hand, manufacturers do not provide entropy guarantees for these designs, and the lack of design details or access to circuit internals makes effective auditing and monitoring impossible.

Commercial devices. A number of commercial devices are available, relying on a range of noise sources including circuit noise [2, 7, 10, 28, 32, 49, 52, 56, 59] and quantum phenomena [25, 38]. A few devices are available in form factors suitable for use in an embedded system [32, 56], but most are intended to act as entropy sources for personal computers or servers. Thus, almost all are optimized for very high bit rates. In addition, these designs resist auditing and monitoring (because their designs are proprietary, and they often incorporate black-box integrated circuits); many are not designed with an eye to energy efficiency; and most cost tens or hundreds of dollars.

FPGA-based designs. Several recent works have focused on purely digital designs suitable for implementation on FPGAs [43]. These circuits use either metastability [34] or oscillator jitter [44, 51]. As with the Intel RNG described above, these circuits implicitly measure the noise of their constituent transistors, which are presumably dominated by thermal and flicker noise. While these circuits are not black-box (since the FPGA designer has control over their implementation), they are still difficult to audit or monitor. This is because, first, all of the critical nodes are internal to the FPGA (and thus not observable), and second, the noise sources are implicit (and therefore cannot be measured separately from the rest of the circuit, even if internal nodes could be measured). Moreover, instantiating these circuits requires that the embedded system use an FPGA.

Discrete designs. Several discrete RNG designs have been proposed, based on diode breakdown noise [21–23, 37, 41, 60], noisy amplifiers [12, 17, 40], incident RF noise [37, 57], and even chaotic attractors [13, 61].

Because the behavior of a chaotic system is determined by its initial conditions, chaos circuits are only nondeterministic to the extent that their initial conditions are influenced by circuit noise. Circuits based on RF noise are similarly problematic, because their input can be influenced by a remote adversary.

Noisy amplifier–based circuits employ some mix of circuit noise processes which depends on which amplifier a given design uses. (This is because the noise characteristic of a given amplifier is peculiar to its design.) Since the noise source cannot be separated from the rest of the circuit, these designs make auditing and monitoring difficult.

Diodes in reverse breakdown are strong, self-contained noise sources, but existing circuits have a range of practical issues, including operating point instability and susceptibility to power supply noise. We discuss further in Sections 3.2–3.3.

2.3 Assumptions and requirements

Assumptions. We assume that an adversary has only a limited ability to manipulate the circuit using targeted RF energy, for three reasons. First, high attenuation RF shielding is inexpensive and easy to incorporate. Second, if the circuit is physically small, it is an antenna only for very high frequencies and will pick up only a small amount of energy. Finally, targeted RF attacks are coarse-grained, and will therefore affect all parts of the circuit in an approximately uniform way. We discuss such disturbances in Section 3.3.

We also make standard cryptographic assumptions. Under these assumptions, a software system does not need to use a hardware random number generator every time it needs a random number. Instead, it can generate a single random seed and use a PRNG [11], as we discuss in Section 2.1.

Finally, we assume that the embedded circuit is physically secure against an adversary when it is operating. This rules out, for example, an adversary who can remove or replace circuit components, or place leads on circuit traces. This also means that the embedded processor is physically secure: an adversary cannot (say) attach a debugger and retrieve the seed or other state from the processor's memory.

Requirements. NIST sets forth guidelines for designing random number generators [14, 15]. These include using a noise source whose behavior results from fundamentally probabilistic behavior; protecting the noise source from adversarial observation and influence; and various documentation and testing requirements. Beyond these requirements, embedded applications impose limitations on energy, cost, and size.

Our survey of existing work (§2.2) reveals that existing non-proprietary designs do not meet these guidelines. Several commercial designs claim to, but they cost far too much to incorporate into tiny embedded devices. As a result, our goal is to design a hardware random number generator that meets the following requirements:

1. *Simple to audit and monitor.* A designer implementing the circuit should be able to inspect the state of critical nodes in the circuit. This includes the output of the noise source, any nodes that hold state, and all power supply and reference voltages. Further, these nodes should be compatible with continuous monitoring in the case of security-critical designs.

2. *Discrete circuit.* A discrete circuit (i.e., a circuit built of individual components, as opposed to an integrated circuit) is simple to construct, and is compatible with our requirement that the circuit should be auditable. Further, a discrete circuit can be incorporated into existing embedded system designs.

3. *Low cost.* The total price for all components should be low when produced at moderate scale (say, thousands of devices).

4. *Small size.* Because space is at a premium in embedded systems, the circuit should be as small as possible. In particular, it should occupy minimal printed circuit board area.

5. *Sufficiently high output rate.* Randomness is critical for security, so an embedded system should not enter normal operation until it has gathered sufficient entropy. We therefore require the circuit to produce entropy fast enough that the embedded system's boot sequence is not unduly delayed. Concretely, we aim to gather about one thousand bits in a few tens of milliseconds.

6. *Energy efficient.* Power consumption is critical in many embedded applications, so the energy cost per bit should be minimal, and the circuit should draw little or no power when not in use.

7. *Robust, practical noise source.* The noise source should practical, that is, a device that is easy to obtain and incorporate into a wide range of designs. It should also be robust, meaning several things. First, the noise source should have high immunity to environmental factors, including varying temperature and radio frequency (RF) interference. Second, the noise source should be an explicit device in the circuit such that its output can be inspected directly. Third, the noise source should be as strong as possible, and in particular should need no

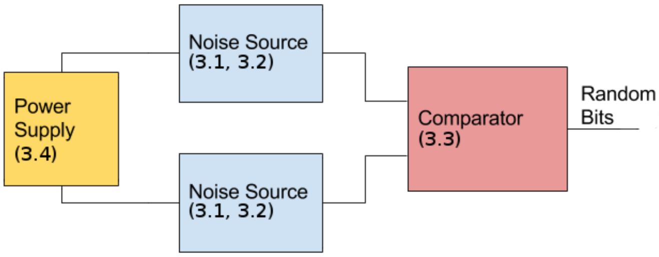

Figure 1: Block diagram of the Lampert circuit. Random bits are sampled from the comparison between two noise sources.

amplification before being converted to bits. This both reduces circuit complexity (by eliminating components) and relaxes the specifications for the conversion circuit (because a strong signal is less susceptible to being corrupted by nonideal circuit components).

3. Design

This section describes the Lampert circuit design, building up from a single noise source to the full final circuit. At each step, it describes the issues that arise, with measurements as needed, and the approach the design takes to address them. Figure 1 shows the circuit's block diagram, noting which subsection discusses each component.

At a high level, the goal of a hardware random number generator is to produce a high-entropy bitstream. Roughly speaking, such circuits comprise two blocks: a *noise source*, whose output is a nondeterministic signal; and a *conversion circuit*, which captures, amplifies, and conditions the output of the noise source to produce bits. Sections 3.1–3.2 discuss the Lampert circuit's noise source; 3.3 discusses the conversion circuit; and 3.4–3.5 discuss practical considerations.

3.1 Choosing a noise source

Every random number generator requires a source of entropy. Section 2.2 mentions several options. In designing the Lampert circuit, we choose avalanche noise because:

1. it is internally generated (unlike RF energy) and cannot be manipulated by an adversary;
2. it is from a cheap, commodity part (unlike radioactive noise) and can be easily incorporated into any design;
3. it provides a very strong signal (unlike thermal noise or RF energy) which requires no amplification and is robust to circuit nonidealities;
4. it is insensitive to temperature and other effects (unlike thermal noise or RF energy) and so once characterized can be used in wide range of environments; and
5. its generating process, avalanche breakdown, can be safely relied upon as it is fundamentally probabilistic and well characterized.

The rest of this subsection explains, at a high level, what causes avalanche noise and why this leads to the above five properties. Readers familiar with avalanche diodes can skip to the experimental characterization at the end of this subsection; for more detail refer to Zeghbroeck [63].

A diode is a semiconductor device with a directional current-voltage relationship. In one direction (the *forward* direction), a diode conducts current freely. In the other direction (*reverse*), the diode does not conduct until the applied voltage exceeds the diode's *reverse breakdown threshold*.

Several types of diodes with controlled reverse breakdown characteristics are widely available. They are commonly used, for example, to protect other circuit components from excessive voltage. One type of diode with controlled reverse breakdown is the avalanche diode; its reverse breakdown is the result of a physical process called avalanche breakdown.[3] Under avalanche breakdown conditions, these diodes exhibit random behavior called *avalanche noise*.

Avalanche breakdown is the result of electron collisions in the crystalline lattice of a semiconducting material. In brief, each electron in a semiconductor is associated with one of several discrete bands of energy levels. Electrons occupying energy levels in the valence band are tightly bound to atoms in the material's crystalline structure; thus, they cannot move and do not conduct current. Highly energetic electrons occupy energy levels in the conduction band; these electrons are loosely bound to the crystalline structure, and are free to move, conducting electricity. In a semiconducting material at room temperature, a tiny fraction of electrons are in the conduction band as a result of thermal excitation.

An electron in the conduction band is accelerated by an incident electric field (i.e., an applied voltage). When this field is small, the resulting current is negligible. However, for an electric field of sufficient magnitude, electrons in the conduction band are strongly accelerated, gaining kinetic energy. These electrons become so energetic that, upon colliding with an electron in the valence band, the latter is "knocked loose" into the conduction band. The electric field can then accelerate this second electron, which will transfer energy to other valence electrons through similar collisions. In this manner, a single electron can create an "avalanche" of charge carriers. The resulting avalanche current is probabilistic due to the randomness of electron collisions; the variability of avalanche current is avalanche noise.

Figure 2 shows a test circuit built with the avalanche diode referenced in Section 6, and Figure 3 shows its noise at a microsecond scale with V_{cc} set to 12.16 V. This results in an 160 mV average at V_{noise}, with ±60 mV deviation from mean. This noise voltage is orders of magnitude larger than other sources of noise in the circuit, and so these other sources can be ignored. Figure 4 shows a histogram of V_{noise} deviation from 160 mV for 500 000 samples taken at 1 ns intervals.

3.2 Setting the operating point

Figures 2–4 show a test in which V_{cc} is carefully adjusted to produce the desired *operating point*, i.e., the average voltage

[3] Diodes exhibit another reverse breakdown phenomenon called Zener breakdown. Reverse breakdown results from a combination of Zener and avalanche behavior, but Zener current dominates below 5.6 V, while avalanche current dominates above [50]. We measured reverse breakdown diodes with voltages ranging from 2.7 V to 18 V, and found that avalanche-dominated devices generated significantly more noise than Zener-dominated devices. In our tests, a 12 V avalanche diode generated noise power sufficient to obviate amplification, consistent with our requirements (§2.3).

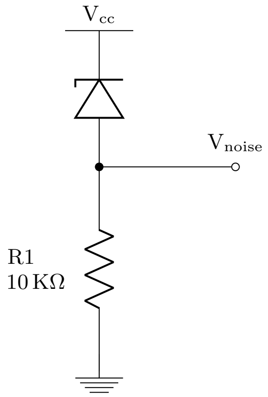

Figure 2: Avalanche noise test circuit.

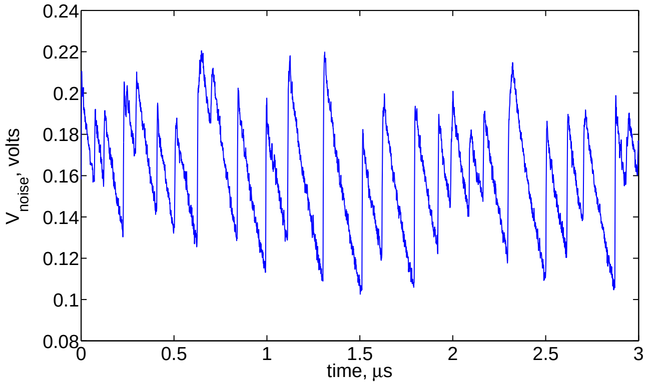

Figure 3: V_{noise} vs. time for Figure 2, V_{cc} = 12.16 V.

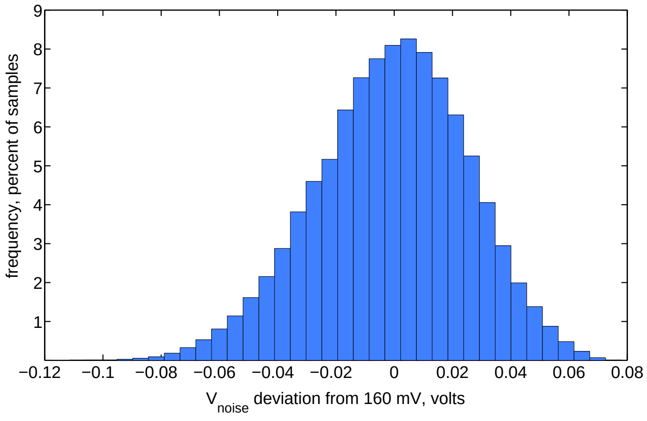

Figure 4: Histogram of deviation of V_{noise} from 160 mV for the test setup in Figure 2, V_{cc} = 12.16 V, for 500 000 samples taken at 1 ns intervals.

at V_{noise}. This test makes two simplifying assumptions. First, it assumes that the required supply voltage, V_{cc}, is fixed. In practice, the circuit's operating point will change with time, temperature, and other factors: at another time, the circuit might need a different value of V_{cc} to establish the desired operating point. Second, the test assumes that there are no external disturbances in the system that affect V_{noise}. We address the first assumption immediately below, and the second in the next subsection.

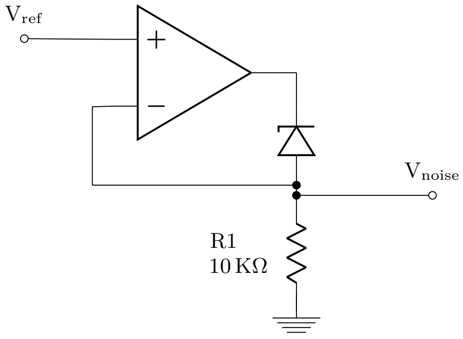

Figure 5: Bias circuit for an avalanche diode.

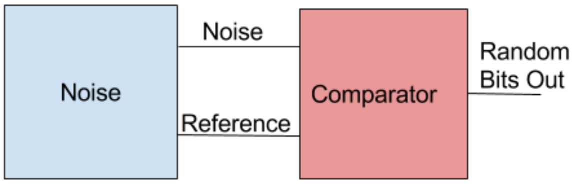

Figure 6: Block diagram of a system that uses a single noise source to generate bits.

A standard means of counteracting operating point variability is to use *negative feedback* [19, Ch. 8]. The Lampert circuit uses negative feedback to set the average value of the V_{noise} with an *operational amplifier* (op-amp), as depicted in Figure 5. In this configuration, the function of the op-amp is to drive the average value of V_{noise} to be equal to the value of V_{ref}.[4] The result is that this circuit corrects for manufacturing variability among avalanche diodes, variation of a given diode over time due to the effects of temperature and aging, and variation due to changing V_{cc} values.

3.3 Rejecting disturbances

Figure 6 shows one approach to generating bits from the circuit of Figure 5 using a *comparator*, a circuit that indicates if the difference between its inputs is positive or negative. Since the op-amp causes the average voltage at V_{noise} to be equal to V_{ref}, the comparator's output will be 0 and 1 with equal probability. However, this arrangement is susceptible to disturbances from nearby signals in the system.

To see how, consider what happens if V_{noise} is affected by another signal in the circuit in such a way that its average value equals V_{ref}, but its instantaneous value depends on the disturbing signal. First, note that the op-amp will not reject this disturbance, since the average value of V_{noise} is not disturbed. On the other hand, the probability that the noise value is (say) greater than the reference value at any instant depends on both the noise source and the disturbance; as a result, the probability distribution of the comparator's output value also depends on the disturbance. If the disturbance is strong and predictable, it will overwhelm the influence of the noise source on the comparator's output and cause the output bits to be predictable.

[4] This equality is not perfect because op-amps do not behave ideally; in our implementation (§6) we are careful to choose an op-amp with sufficient precision for this task.

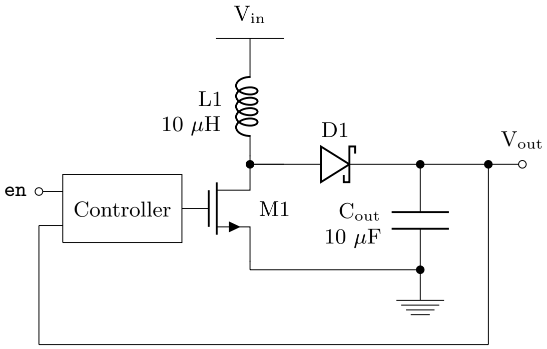

Figure 7: A boost converter takes a DC input voltage, V_{in}, and produces a higher DC output voltage, V_{out} (§3.4). The circuit is enabled via the logic-level `en` input. In our implementation (§6) the switch (M1) and the controller are integrated into a single chip.

In practical systems, one strong source of disturbances that matches the above description is the power supply. Concretely, for the circuit of Figure 5, it is extremely difficult to ensure that V_{cc} has no effect on either V_{ref} or V_{noise}. In fact, since a comparator measures the difference between its two inputs, merely ensuring that V_{cc} had identical effect on V_{ref} and V_{noise} would suffice (since adding the same value to two numbers does not affect their difference)—but this, too, is extremely difficult to achieve in practice.

The Lampert circuit addresses this problem by instantiating a second copy of the circuit from Figure 5, resulting in a circuit with two noise sources (as depicted in Figure 1). This approach (an example of a *differential circuit*) is effective because the influence of the power supply on the two noise circuits is nearly identical. Indeed, nearly any external influence on the circuit of Figure 1 will have identical effects on the two copies of the noise source; meanwhile, because the comparator measures only the difference between the two circuits, such effects have little influence on the output bits.

3.4 Power supply design

One final issue is generating the supply voltage. The Lampert circuit needs a supply voltage greater than 12 V, but most embedded systems have power supplies in the range of 1.8–3.3 V. This is a common problem, for example, for LED drivers and displays. The standard approach is to use a *boost converter*, a circuit that produces an output voltage whose value is greater than its input voltage [16]. Figure 7 illustrates a boost converter.

A significant challenge to using a boost converter in the Lampert circuit is that converters produce very strong disturbances in the circuit. Indeed, despite the fact that the Lampert circuit uses a differential noise source (§3.3), we found that the boost converter's disturbances were strong enough to affect the comparator output, substantially degrading the entropy of the resulting bitstream.

To overcome this issue, the Lampert circuit uses two interleaved phases of operation, controlled by a microcontroller. In the first phase, the boost converter circuit is enabled, charging an output capacitor, C_{out}. Once the voltage across C_{out} reaches 18 V, the first phase ends. In the second phase, the boost converter is disabled and thus produces no disturbance. Now the comparator's output is sampled. Once the voltage across C_{out} reaches 12 V, the second phase ends and the first phase begins again.

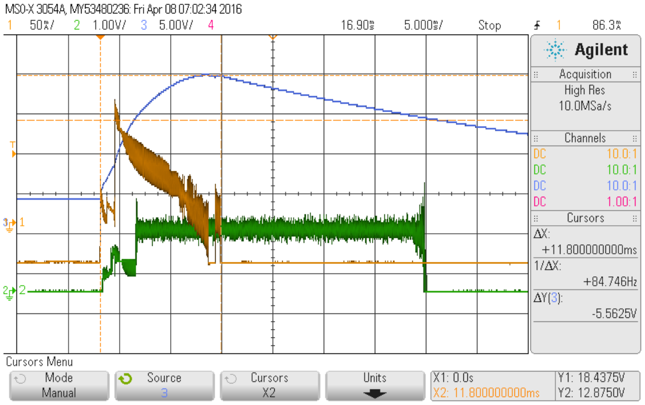

Figure 8: Waveforms showing boost and comparator operation. The blue line is the boost converter's output, which takes 13 ms to charge to 18 V, and 20 ms to discharge to 12 V. The yellow line is the input current to the boost converter. While the boost converter runs, its input current changes rapidly. After the boost converter's output reaches 18 V ($t = -5$ ms), the converter is disabled and its input current goes to zero. The green line is the comparator's output (at this time scale, individual logic levels are not visible). Once the boost converter output (blue) drops below 12 V, the comparator no longer switches because the noise source stops operating.

The duration of the first phase depends on the particulars of the boost converter. The duration of the second phase is given by the current consumption of the Lampert circuit and the size of C_{out}. For example, if the Lampert circuit draws 5 mA and $C_{out} = 10\ \mu$F, then the second phase lasts for

$$\frac{(18\text{ V} - 12\text{ V}) \cdot 10\ \mu\text{F}}{5\text{ mA}} = 12\text{ ms}$$

Figure 8 shows the waveforms for both phases. The green signal shows the random bit stream operating for 17.5ms after power is turned off, slightly more than estimated.

3.5 Putting it together

Figure 9 shows a full Lampert circuit. Once the output of the boost converter (details in Fig. 7) reaches 18 V, the boost converter is disabled and its output capacitor (C_{out}, Fig. 7) powers the circuit. Op-amp 1 and Op-amp 2 set the operating point (§3.2) for the two avalanche diodes, D2 and D3 (§3.3). The noise signals generated by D2 and D3 are fed to a comparator, whose output is simply a measure of which of these two random variables is higher.

As long as it is sampled below its bandwidth (§7.1), this circuit produces very nearly 1 bit of entropy per sample. It cannot produce exactly 1 bit because comparators have limited precision. If the two inputs are within some tiny ϵ, then the output is undefined and may be biased.

The next two sections discuss how to integrate the Lampert circuit into an embedded system and how to test that it is working properly.

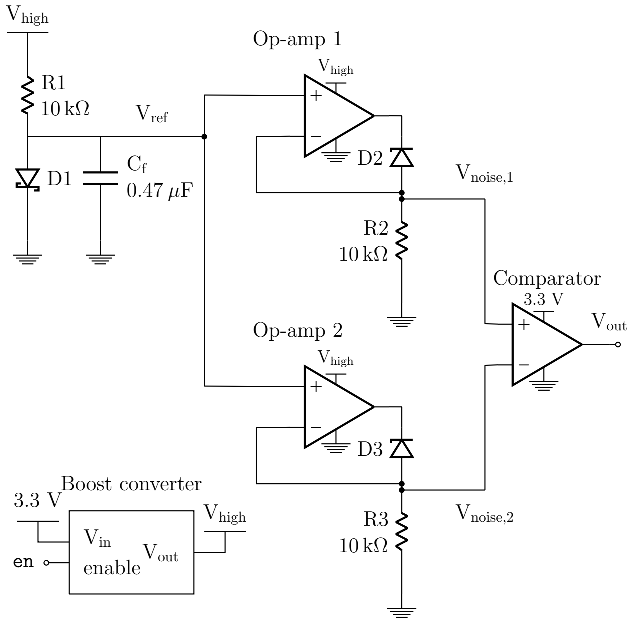

Figure 9: The Lampert circuit. The boost converter is shown in Figure 7. A microcontroller controls the boost converter and samples bits from V_{out} (§3.5).

4. Using the Lampert circuit

This section describes how to integrate the Lampert circuit into an embedded design and how software should use it.

The circuit should be as dense and symmetric as possible. For example, the traces around the two noise sources should be parallel, close, of the same length, and separated by a co-planar waveguide to minimize coupling. This is to ensure that any RF interference affects them equally and they do not affect each other. The circuit should be placed away from strong signals; for example, it should be isolated from power converters (other than its own) and any RF circuits.

The software random number subsystem requires three components: a way to read the output of the circuit (e.g. a GPIO line) a cryptographic hash function, and a pseudorandom number generator. When the software boots, and before operations that require randomness can commence, software should sample the circuit's output to gather a string of high-entropy bits. According to NIST recommendations [15, 45], the required number of bits required is $^{512}/_{e}$, where e is the entropy per bit. Based on our evaluation (§7), we assume that the entropy of each bit is $e \geq 0.9$. Thus, the system should sample at least 570 bits; we recommend using 768 bits, which provides at least 690 bits of entropy.

Once these 768 bits have been gathered, software should pass them through a cryptographic hash function. We recommend using SHA256. This mixes the ≥ 690 bits of entropy into 256 bits, providing a full-entropy, 256-bit seed.[5]

The PRNG's state comprises a key and a counter whose width is the AES block size (128 bits). If the system has an efficient (e.g., hardware) AES-256 implementation, the output of the hash from the previous step should be used as the AES key, and the counter should be initialized to zero. If only AES-128 is supported, 128 bits of the hash output should be used as the AES key, and the other 128 bits should be used to initialize the counter. Each time the system needs to generate a random number, the generator encrypts the current value of the counter under the PRNG key, increments the counter, and returns the AES output, truncated as necessary (e.g., return the least significant 32 bits for a 32-bit random number).

5. Testing and monitoring

As we argue in Sections 1 and 2, a trustworthy source of high-entropy bits must allow for acceptance testing after manufacture and auditing during operation. In this section, we discuss how to test and audit the Lampert circuit for proper operation.

Acceptance testing. After assembling an embedded system containing a Lampert circuit, the circuit should be thoroughly tested. This testing is designed to detect both manufacturing flaws in the circuit components, and assembly defects at the circuit board level. Note that, because they are designed to be performed once at manufacture, we allow these tests to use external test equipment, e.g., oscilloscopes and power supplies. The following tests should be performed (all node names refer to Figs. 7 and 9).

1. *Operating point test.* Using an external power supply, apply a range of voltages from 12 V to 18 V to V_{high} in 0.5 V steps; the `en` input should be de-asserted. Ensure that the circuit draws an appropriate level of quiescent current (approximately 5 mA); that V_{ref}, $V_{noise,1}$ and $V_{noise,2}$ have average values within 1 mV of one another.

   This test establishes proper connection and gross operation of the op-amps, V_{ref} circuit, and avalanche diodes (§3.2).

2. *Avalanche noise.* As above, use an external power source to supply a range of voltages at V_{high}. For each setting of V_{high}, sample the values on $V_{noise,1}$ and $V_{noise,2}$, ensuring that these voltages have standard deviation of at least 10 mV when sampled over a 100 MHz bandwidth. Compute the Fourier transform of the voltages at each node, ensuring that the spectrum is flat (and, in particular, has no tones).

   This test establishes that the avalanche diodes are generating the expected level of noise, and that this noise has the proper spectrum.

3. *Entropy testing.* As above, use an external power source to supply a range of voltages at V_{high}. For each setting of V_{high}, sample V_{out} at the sample rate used by the embedded system (§7.1), ensuring that the density of ones in the resulting bit stream is 50%±0.5%. Compute the Shannon entropy of this bit stream; the result should be > 0.98.

   This test establishes the Lampert circuit's core functionality across power supply range.

4. *Power supply operation.* Apply 3.3 V to the circuit and assert `en`. Measure the time for V_{high} to reach 18 V, ensuring that it is approximately 20 ms. De-assert `en`

[5] It is not necessary to apply other whitening or de-biasing techniques, e.g., von Neumann's procedure [62]; under common assumptions, a cryptographic hash function suffices for randomness extraction [29].

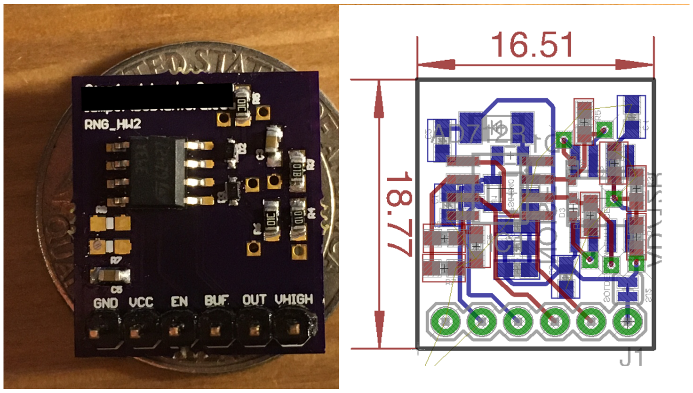

Figure 10: Standalone RNG board. On the left is final design over a US quarter; on the right is the printed circuit board layout with dimensions in mm.

and ensure that the time for V_{high}'s value to reach 12 V is close to 12 ms.

This test establishes that the boost converter is working properly.

5. *End-to-end testing.* To test the embedded system's end-to-end functionality, we recommend creating testing software that repeatedly exercises the seed generation functionality (§4), saving the high-entropy bit string generated in each run. These strings should be tested with the statistical test suites described above.

   This test establishes that the embedded system's random number subsystem will be properly seeded in operation.

As with all acceptance testing, data for all boards should be carefully logged such that outliers can be identified. An ideal test suite should include multiple temperatures (cold, room temperature, and hot, as determined by the temperature grade of the embedded system), but in practice, industrial testing regimes often phase out multi-temperature testing once a design has been thoroughly vetted and strong correlation across temperature can be established from historical data.

Online auditing. In addition to acceptance testing, an embedded system can audit the Lampert circuit online to ensure that it has not failed. The following three tests can help to detect a failed RNG:

1. *Operating point test.* In this test, the system measures the average value of the V_{ref} node and the V_{noise} nodes. Note that this test does not require a high-performance ADC, since the average value is slowly varying.

   This test establishes that the reference and operating point circuits are working properly.

2. *Boost converter test.* This test is essentially the same as test #4, above. During operation, the embedded system measures the amount of time for V_{high} to reach 18 V with the converter active, and the amount of time to reach 12 V with the converter inactive.

   This test establishes that the boost converter is working properly.

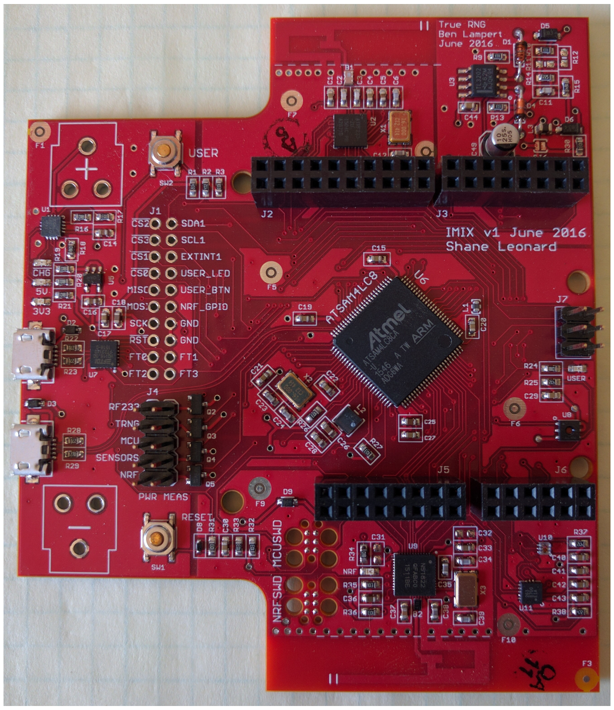

Figure 11: imix circuit board. The Lampert circuit is instantiated in the top right corner.

3. *Bit string health test.* In this test, the system computes a simple statistical measure of the output of the Lampert circuit. Specifically, it counts the occurrence of a set of bit patterns, and discards bit strings that fail to fall within bounds. We recommend testing in the same way as Intel's Ivy Bridge RNG [20].

   This test establishes that the system has not failed in a mode that results in a continuous stream of 0 or 1, or a simple alternating pattern.

We note that these tests may not be necessary in all applications. However, as they are relatively simple to implement, we recommend that an embedded system incorporating the Lampert circuit implement these tests, and use them to sanity check its operation.

6. Implementation

We built two implementations of the Lampert circuit. Both implement the design of Figure 9, where the boost converter is a single-chip solution based on the TPS61041 [54]. In total, this circuit has 22 components, including additional decoupling capacitors [24] on power supply and reference nodes. We list the bill of materials and corresponding costs in Table 1. Note that we do not include the cost of the printed circuit board, as this will vary by application.

Standalone implementation. The first implementation is a standalone test board, depicted in Figure 10. We designed printed circuit boards in CadSoft EAGLE [9]. These were manufactured by OSH Park on a 2-layer, 2 oz. copper board. Boards were hand assembled. The standalone printed circuit board measures 1.42 × 1.9 cm, for a total area of 1.45 cm^2. Our standalone design is freely available [31].

Description	Value	Circuit component	Unit cost	#	Total cost
Capacitor	10 μF	C1 (Fig. 7)	\$0.05075	1	\$0.05075
Comparator	TLV3201	Comparator (Fig. 9)	\$0.45000	1	\$0.45000
Dual op-amp	LM358	Op-Amp 1, 2 (Fig. 9)	\$0.07650	1	\$0.07650
Boost converter	TPS61041	Controller, M1 (Fig. 7)	\$0.65520	1	\$0.65520
Inductor	10 μH	L1 (Fig. 7)	\$0.03230	1	\$0.03230
Diode	MBR0530	D1 (Figs. 7 and 9)	\$0.04505	2	\$0.09100
Avalanche diode	1N759 (12 V)	D2, D3 (Fig. 9)	\$0.01780	2	\$0.03560
Miscellaneous passive components (capacitors and resistors)					\$0.05000
Total					\$1.44000

Table 1: Bill of materials for Lampert circuit implementation (§6). Costs are per-unit, quoted at quantity 10 000. We do not include the incremental cost of printed circuit board area because it will vary by application.

To control this board as described in Sections 3.4 and 4, we connected it to an Arduino microcontroller [3]. After assembly, we followed the acceptance test flow described in Section 5.

Embedded system integration. We have also integrated the Lampert circuit into a multi-radio ARM Cortex-M4–based platform called imix, whose design and software are open [26]. This board is shown in Figure 11. We designed these boards in CadSoft EAGLE. Sierra Circuits manufactured them in 4 layers of 1 oz. copper, and assembled our engineering prototypes. The Lampert circuit implementation on the imix board occupies approximately the same area as the standalone board.

7. Experimental results

This section evaluates the Lampert circuit with respect to the requirements set forth in Section 2.3. Specifically, we evaluate output rate, energy efficiency, and the robustness of the Lampert circuit's entropy generation. We evaluate in three steps. First, we establish a conservative rate at which to sample the output (§7.1). Next, using this sample rate, we run a battery of statistical tests on the raw output of the Lampert circuit taken over several days (§7.2) and across temperature conditions (§7.3). We also measure the average time to gather sufficient entropy for seeding a PRNG (§2.1, §4), and the energy that this entails (§7.4).

We find that the Lampert circuit requires relatively little energy to seed a PRNG, that it produces a high-entropy seed quickly, and that the entropy is insensitive to temperature and stable over time.

Metrics. In Sections 7.1–7.3, we evaluate the output of the Lampert circuit using the ent test suite [30], which estimates the entropy and serial correlation of the raw bitstream. In contrast, other randomness test suites [8, 42] are designed for processed bitstreams. As we discuss in Section 2.1, producing a high-entropy seed (i.e., one that cannot be guessed by an adversary) is sufficient to ensure security.

7.1 Establishing sample rate

In this subsection, we measure the effect of sample rate on the quality of the Lampert circuit's output, and establish a conservative sample rate.

Method. Using the standalone Lampert circuit implementation (§6), we collected several sequences of 500,000 samples of the raw bitstream at sampling rates of 100 kHz, 10 MHz, and 500 MHz, using an oscilloscope. We then subsampled these data at intervals of 1× to 100× to produce collections of raw bits. Finally, we computed entropy and serial correlation for the unbiased bitstream using ent.

Results. The subsampled bitstreams cover sample rates from 1 kHz to 500 MHz. The results from running the ent tests are shown in Figures 12 and 13.

We measure the average transition rate at V_{out} (Fig. 9) to be roughly 6MHz, which roughly corresponds to the maximum frequency at which the system's state can change. This is consistent with the results showing strong serial correlation for sample rates at and above 6 MHz: at these frequencies, each sample captures essentially the same state of the system as the prior sample. On the other hand, at sample rates well below the transition rate (say, 400 kHz and lower), the serial correlation is negligible.

From these data we make the conservative choice to use a 128 kHz sample rate for gathering bits. This decision represents a tradeoff between serial correlation and time to gather a random seed. Sampling 1024 raw bits at 128 kHz requires 8 ms, and can therefore be done with a single boost-discharge cycle (§3.4).

7.2 Long-term testing

In this subsection, we evaluate the quality of the Lampert circuit's output versus time with statistical testing.

Method. To evaluate the performance of the Lampert circuit versus time, we gathered data over approximately 10 days using the standalone implementation (§6). Once per minute, we ran 1000 on/off cycles (§4), gathering 512 bits per cycle. For each data point, we also recorded the ambient temperature. We used ent to compute the entropy and serial correlation of each set of 51 200 bits.

Results. Figure 14 shows the results. Over the course of the test, entropy varied about 1%. We confirmed with the temperature readings that the periodicity in the entropy was the result of normal temperature fluctuations; in the next subsection we look at temperature further. The data show several outlier events during which entropy dropped as low as 0.986. We do not yet know the cause of these outliers. We note that the minimum entropy is well above 0.9, the value we assumed in Section 4.

Serial correlation is consistently low: generally it is below 1%, and it is strictly less than 2%. This is in line with the results of Section 7.1.

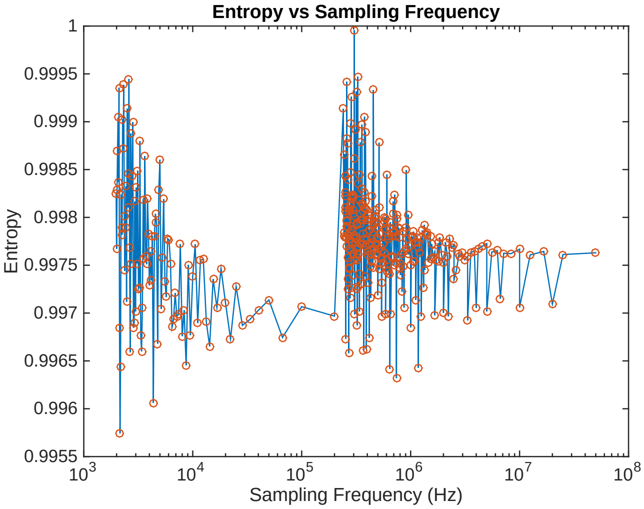

Figure 12: Entropy vs. sample rate (§7.1).

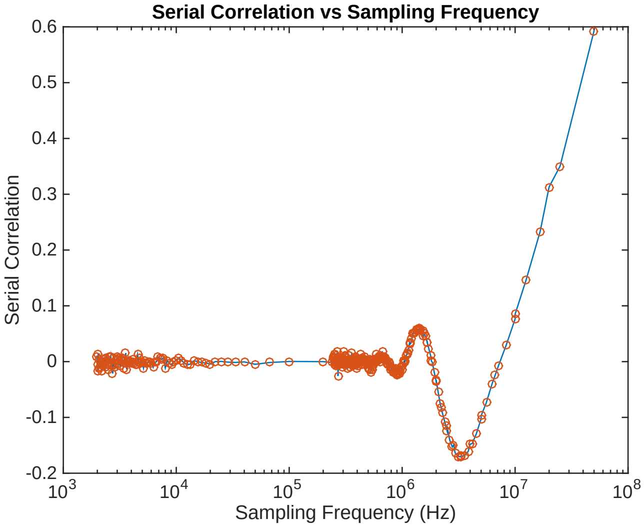

Figure 13: Serial correlation vs. sample rate (§7.1).

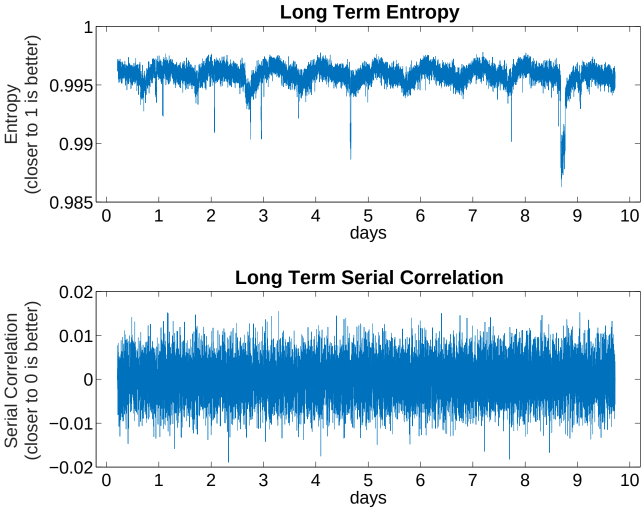

Figure 14: Statistical tests vs. time (§7.2).

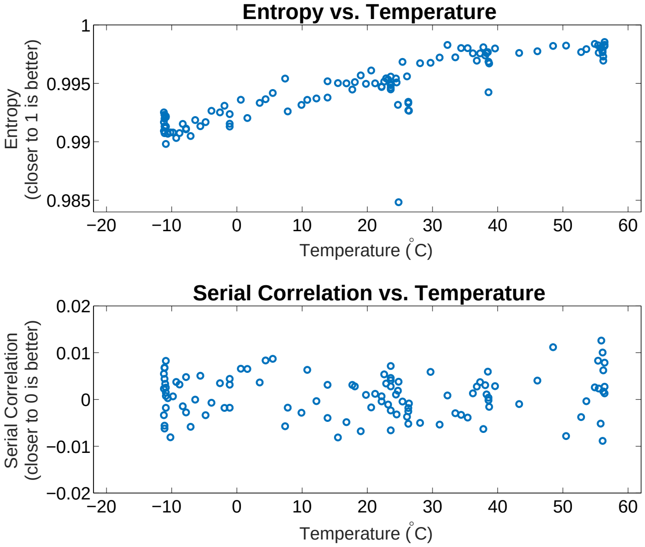

Figure 15: Statistical tests vs. temperature (§7.3).

We ran similar statistical tests against the imix platform's Lampert circuit implementation (§6) with similar results.

7.3 Temperature testing

In this subsection, we evaluate the quality of the Lampert circuit's output versus temperature with statistical testing.

Method. To evaluate the performance of the Lampert circuit versus temperature, we gathered data over a range of temperatures from −11 °C to 56 °C. We sealed a standalone board, a microcontroller, and a temperature probe in an airtight container and placed this container in a temperature-controlled environment. For each temperature point, we ran 1000 on/off cycles, gathering 512 bits per cycle. We used ent to evaluate each set of 51 200 bits.

Results. Figure 15 shows the results. The difference in entropy between the lowest and highest temperatures is about 1%. This is expected, for two reasons: first, the avalanche phenomenon has a very weak but nonzero temperature dependence [50], and second, at high temperatures the op-amps and comparator (§3, Fig. 9) contribute a very small amount of thermal noise. As in the previous subsection, we see an outlier event where entropy drops to about 0.984. We believe this outlier was caused by an error in our test setup when the device was initially placed in the oven. Again, this is well above 0.9, the value that we assumed in Section 4.

We find that serial correlation is essentially independent of temperature and shows a similar spread to the previous two experiments.

7.4 Energy and time costs

The boot process described in Section 4 requires the microcontroller to pause until the Lampert circuit delivers enough entropy to seed a PRNG. We now discuss the expected delay and energy cost of this sequence.

Method. Using the standalone Lampert circuit implementation (§6), we follow the boot procedure described in Section 4 to gather 1024 bits. We start with the boost converter's C_{out} fully discharged to simulate conditions at boot.

Results. Collecting 1024 raw bits requires 8 ms at 128 kHz. This is less than the $\approx$ 12 ms that the boost converter's output takes to discharge from 18 V to 12 V (§3.4), meaning that 1024 bits can be collected in one cycle. The boost converter's output takes $\approx$ 13 ms to charge from 0 V to 18 V (§3.4, Fig. 8), meaning that the process of gathering a random bit string requires $\approx$ 25 ms.

In charging its output to 18 V, the boost converter requires $\approx$ 3 mJ. This translates to a cost of < 3 μJ per bit, at an average power consumption of 120 mW over the period of gathering bits. To put this number in context, a ZigBee radio requires about 400 nJ per bit to transmit [48], or about 10× less than the Lampert circuit.

Since the time and energy cost of generating random bits is paid only at boot (§4), and can be amortized over the entire duration of an embedded system's operation, we conclude that these costs are reasonable.

8. Discussion and conclusion

Computer security is often an arms race: new attacks and vulnerabilities emerge, and system designers patch them, securing systems until the next wave of more complex attacks, which require correspondingly more complex solutions. This paper takes a different approach: rather than waiting until an attack emerges, we argue that low-level primitives like random number generators should be secure by design.

In the current climate of government surveillance and compromised cryptography, security by trust is not security. For a primitive as fundamental as a random number generator, one needs to know not just how it is supposed to work, but also that it actually works that way. Using the Lampert circuit, one can physically inspect a device to check that it has the right components in the right configuration. One can test the output of the circuit as well as the output of each of its subcomponents (e.g., test that the diodes are producing the expected noise distribution). One can run open source software that correctly whitens the circuit output and uses it to seed a PRNG. One can test and check the entire system, from end to end. While this is admittedly overkill for many applications, it sets a bar which any application can easily reach. And "overkill" is a moving target: transport layer security (TLS) seemed like overkill for social networking until Firesheep showed the havoc one could wreak without it [18].

The Lampert circuit satisfies all of the requirements set forth in Section 2.3: it uses fewer than 25 components, costs less then $1.50 to build, and requires less then 1.5 cm^2 of area. Still, it can be improved: lower cost, lower power consumption, and smaller size are all future work. We hope that in the near future, incorporating a hardware random number generator with strong entropy guarantees is cheap and easy enough that every system concerned with security does so.

Acknowledgments

We thank Dan Boneh, Henry Corrigan-Gibbs, Greg Kovacs, Rishab Mehra, Keith Winstein, and the anonymous reviewers. This work was supported by Intel/NSF CPS Security grant #1505728, the Secure Internet of Things Project, and gifts from VMware and Analog Devices.

Board designs and software are available from
https://github.com/helena-project/imix
https://github.com/lampertb/LampertCircuitRNG

References

[1] AM335x cryptography users guide. http://processors.wiki.ti.com/index.php/Cryptography_Users_Guide.

[2] Araneus Alea II TRNG. http://www.araneus.fi/products/alea2/en/.

[3] Arduino. https://www.arduino.cc/.

[4] BCM2835. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md.

[5] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans. In *CHES*, Aug. 2013.

[6] S. Bhunia, M. Hsiao, M. Banga, and S. Narasimhan. Hardware Trojan attacks: threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, Aug. 2014.

[7] BitBabbler. http://www.bitbabbler.org/.

[8] R. G. Brown. dieharder test suite. http://www.phy.duke.edu/˜rgb/General/dieharder.php.

[9] Cadsoft EAGLE. http://www.cadsoftusa.com/.

[10] ComScire. https://comscire.com/.

[11] H. Corrigan-Gibbs and S. Jana. Recommendations for randomness in the operating system: How to keep evil children out of your pool and other random facts. In *USENIX HotOS*, May 2015.

[12] B. Cox. Infinite noise trng (true random number generator). https://github.com/waywardgeek/infnoise.

[13] M. N. f. Dr. Maysoon M. Aziz. Numerical and chaotic analysis of chua's circuit. *Journal of Emerging Trends in Computing and Information Sciences*, 3(5):783–791, may 2012.

[14] J. K. Elaine Barke. Recommendation for random bit generator (rbg) constructions. *DRAFT NIST Special Publication 800-90C*, aug 2012.

[15] J. K. Elaine Barke. Recommendation for the entropy sources used for random bit generation. *NIST Special Publication 800-90B*, jan 2016.

[16] R. W. Erickson and D. Maksimović. *Fundamentals of Power Electronics*, chapter 6. Springer, New York, 2001.

[17] FiReBuG. http://apa.hopto.org/firebug/.

[18] Firesheep. https://codebutler.github.io/firesheep/.

[19] P. R. Gray, P. J. Hurst, S. H. Lewis, and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits.* John Wiley and Sons, New York, 5th edition, 2009.

[20] M. Hamburg, P. Kocher, and M. E. Marson. Analysis of intel's ivy bridge digital random number generator. Technical report, Cryptography Research, Mar. 2012.

[21] Hardware random number generator. http://www.cryogenius.com/hardware/rng/.

[22] Hardware random bit generator. https://web.jfet.org/hw-rng.html.

[23] A hardware random number generator. http://iank.org/trng.html.

[24] P. Horowitz and W. Hill. *The Art of Electronics.* Cambridge University Press, 2015.

[25] ID Quantique. http://www.idquantique.com/.

[26] imix platform design. https://github.com/helena-project/imix.

[27] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *ACM SenSys*, Nov. 2004.

[28] Kidekin. http://kidekin.nimp.co.uk/.

[29] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, Aug. 2010.

[30] F. Lab. ENT: A pseudorandom number sequence test program. http://www.fourmilab.ch/random/.

[31] Lampert rng. https://github.com/lampertb/LampertCircuitRNG.

[32] LE Tech. http://www.letech.jpn.com/index_en.html.

[33] U. Leonhardt. Quantum physics of simple optical instruments. *Reports on Progress in Physics*, 66, June 2003.

[34] M. Majzoobi, F. Koushanfar, and S. Devadas. FPGA-based true random number generator using circuit metastability with adaptive feedback control. In *CHES*, Sept. 2011.

[35] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*, chapter 12. CRC Press, Boca Raton, 1996.

[36] Normal number. *Encyclopedia of Mathematics*. https://www.encyclopediaofmath.org/index.php/Normal_number.

[37] OneRNG hardware random number generator. http://onerng.info.

[38] QRBG121. http://qrbg.irb.hr/.

[39] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits.* Pearson, 2$^{\text{nd}}$ edition, 2003.

[40] Open RNG based on modular entropy multiplication. https://github.com/alwynallan/redoubler.

[41] RNG version 2. http://robseward.com/misc/RNG2/.

[42] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST, 2010.

[43] R. Santoro, O. Sentieys, and S. Roy. On-the-fly evaluation of FPGA-based true random number generator. In *IEEE ISVLSI*, May 2009.

[44] D. Schellekens, B. Preneel, and I. Verbauwhede. FPGA vendor agnostic true random number generator. In *IEEE FPL*, Aug. 2006.

[45] Secure hash standard (shs). *Federal Information Processing Standards Publication*, aug 2015.

[46] P. Selinger. The GLIBC pseudo-random number generator. http://www.mathstat.dal.ca/~selinger/random/.

[47] R. A. Serway and J. W. Jewett. *Physics for Scientists and Engineers*. Brooks Cole, Boston, 9$^{\text{th}}$ edition, 2013.

[48] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen. How low energy is Bluetooth Low Energy? comparative measurements with ZigBee/802.15.4. In *IEEE WCNC*, Apr. 2012.

[49] Simtec Entropy Key. http://www.entropykey.co.uk/.

[50] P. Somlo. Zener-diode noise generators. *Electronics Letters*, 11(14), July 1975.

[51] B. Sunar, W. J. Martin, and D. R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Trans. Computers*, 56(1), Jan. 2007.

[52] TectroLabs. https://tectrolabs.com/.

[53] M. Tehranipoor and F. Koushanfar. A survey of hardware Trojan taxonomy and detection. 27(1):10–25, Jan. 2010.

[54] TPS61041. http://www.ti.com/product/TPS61041.

[55] L. Trevisan. Extractors and pseudorandom generators. In *ACM STOC*, May 1999.

[56] TRNG98. http://www.trng98.se/.

[57] True random numbers with RTL-entropy. http://www.rtl-sdr.com/true-random-numbers-rtl-entropy/.

[58] T. Ts'o. https://plus.google.com/+TheodoreTso/posts/SDcoemc9V3J, Sept. 2013.

[59] ubld.it TrueRNG. http://www.trng98.se/.

[60] G. Vazzana. Random sequence generator based on avalanche noise. http://holdenc.altervista.org/avalanche/.

[61] G. Vazzana. Random sequence generator based on chua circuit. http://holdenc.altervista.org/chua/.

[62] J. von Neumann. Various techniques used in connection with random digits. In *National Bureau of Standards Applied Mathematics Series*, pages 12:36–38. 1951.

[63] B. V. Zeghbroeck. *Principles of Semiconductor Devices.* Bart Van Zeghbroeck, 2011. Ch 4.5.