“The Collector”: A Gigabit True Random Number Generator Using Image Sensor Noise

James P Hughes and Yash Gupta

University of California Santa Cruz
{japhughe,ygupta}@ucsc.edu

***Abstract*—We describe a massively parallel, high performance and inexpensive method of capturing physical entropy from existing commodity CMOS imaging sensors. The covers the physics, how to measure its Shannon entropy [1], a simple high performance method to extract this entropy into cryptographic quality random numbers and finally the results of testing against the DieHarder [2] tests.**

The result is a recipe on how to create a True Random Number Generator (TRNG) using sensor noise that has a known amount of entropy and how to make sure that the TRNG is functioning properly.

I. Introduction

Cryptographic history is replete in examples of the effects of bad random number generators. Even given a constant stream of papers from 1995 [3] through today [4] there is still nothing to even suggest that this sorry state of affairs will ever end.

There have been many true random number generators but none have scaled with Moore’s law. Typically a TRNG generate a single bit at a time which results in slow (and typically a more expensive) solution compared to what our solution.

The current solution to achieve a large number of bits in a reasonable amount of time is to take a minimum amount of entropy and expand this with a PseudoRandom Number Generator (PRNG). PRNGs are also known as PseudoRandom Bit Generator (PRBG).

If we can easily create random numbers from an inexpensive source with far more entropy than we need, why bother with the complicated system of PRNGs that we have today?

A. Existing hardware random number generators

Most random number generators are focused on creating random bits, aggregate, whiten and then expanding with a PRNG. Samples include:

- rolling physical dice in a machine [5].
- measuring the noise from a single avalanche diode [6].
- taking hundreds of free running oscillators and mix them down to a single bit [7].
- taking a beam of photons to a beam splitter and counts which path the photons take [8].
- placing a Ceasium-137 radiation source in front of a radiation monitor [9].
- Intel CPUs have a TRNG that can generate over 75 Kbit/sec after the corrector which is then used to seed a PRNG [10].

There are only three known methods of getting random information from an CMOS image sensors. These include calculating the entropy from a light source [11] and from a more informal set of ideas “lavarand” and “LavaRnd” [12]–[14]. This paper differs in the methods of generating, gathering, measuring and distilling the entropy from each of these three other methods.

B. PRNG

Pseudorandom bit generators are typically used to expand s random bits (seed) to d bits $d \gg s$ [15, page 170]. The main failures of PRNGs are:

- The use of unreliable source of entropy [4], [16].
- The false assumption that the PRBG is secure [3], [17]–[23]
- The assumption that the algorithm is not backdoored [24], [25].

Compared to PRNGs, we focus eliminating the above failures by creating random bits directly raw entropy, bits that are never expanded.

II. CMOS Sensor to Random Bits

The process of taking data from a CMOS sensor to a collection of random bits is comprised of a series of steps. First we need to **gather raw data from the CMOS Sensor**, but raw data is not usable directly. The process to convert raw data with incomplete entropy to data with 100% entropy per bit is called **Whitening**. Whitening will compress the data stream such that the

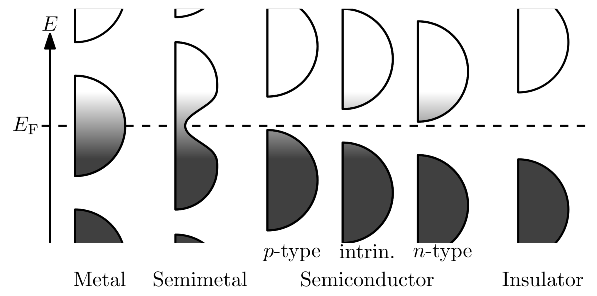

Figure 1: Bands of various materials [26]

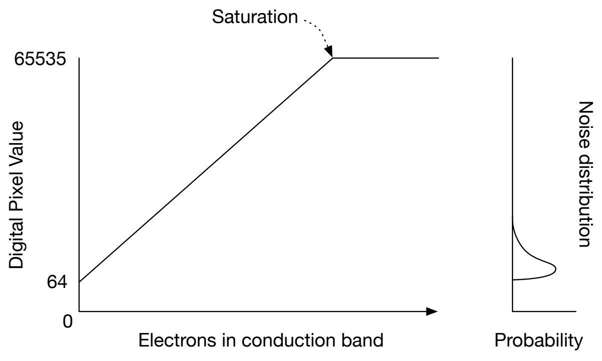

Figure 2: Sensor A2D

result will have 100% entropy per bit (every bit having a perfect 50/50 probability of being a 1 or 0). Before we can whiten the data, we need to determine the total amount of **Shannon Entropy** that at least 99.95% of the pixels have. Once the raw data has been whitened, it can be directly used as fully random bits.

To be sure that the device is functioning properly, we must first **calibrate** the system both before we use it and periodically after.

We will also discuss additional check desirable when a random number generator is being used in life-critical (medical, military, etc.) or safety-critical (Nuclear power, Aircraft) [27]. These include additional checks that the **operating environment** of the system is within norms, and **checks** that the calculations were performed correctly.

A. *CMOS Sensor as a Random Source*

Noise in CMOS sensors has been widely studied [28], [29] in an effort to understand and reduce it. These efforts to eliminate noise now concentrate on cooling, but nothing has been successful at eliminating noise at typical room temperature. We embrace the noise. We begin by using a CMOS imaging array, but these techniques can be applied equally to CCD array, any imaging array that can be operated in "raw mode". Raw mode is important because it gives us control over the analog to digital portion of the process and measure the low order bits.

CMOS image sensor noise is based on the principal that a phonon can excite an electron in a semiconductor and move that electron from the valence band to the conduction bands. These bands are described in Figure 1 and [30]. A phonon is a quantized mode of vibration occurring in rigid atomic lattices, such as those in crystalline solids. [31].

The electrons that move to the conduction band are described as pixel noise (also known as *shot noise*). Pixel noise of specific sensors are measured and characterized as μ_d electrons/second at some standard temperature [32]. Sensors used in photography where low noise in the image is desirable select a lower μ_d. As a random source a higher μ_d is better because it also produces more entropy. In any case the selection is not important to the discussion, we just need to know that it exists.

One device, the Sony ICX424AL image sensor has a μ_d of 12.86 e^{-}/s [33] or 2mV for a 1/60s black exposure [34]. We can assume this was measured at "room temperature" or 70°F.

Each pixel location has its own μ_d because of manufacturing non-uniformity of the array itself. Additionally, it is known that the accumulations of electrons in the conduction band over time is based on Poisson distribution. μ_d also increases exponentially with the temperature [35].

To measure the noise, the gain of the amplifiers needs to be set to ensure that the pixel noise is not at or below the "0" of the digitized result (in this case it was set to 64 out of 65535). The gain should not be set so high as to saturate. Figure 2 describes the process of converting the electrons in the conduction band to a digital pixel value as well as the expected probability distribution function for the noise.

B. *Calculating Shannon Entropy*

To determine the parameters of the whitening algorithm we need to calculate the Shannon Entropy of the bits from a "raw" image. This provides the average and distribution of the unpredictability in the CMOS sensor output. Calculating this correctly is imperative because the incorrect whitening can either result in a non-random output by assuming there is more entropy than there really is, or a lower number of random bits if we assume a lower entropy than there really is.

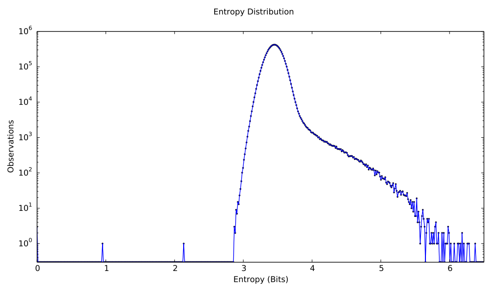

Figure 3: Distribution of sensor pixel entropy

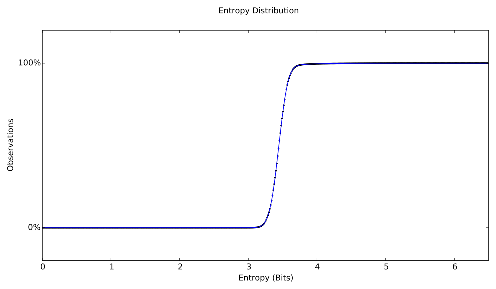

Figure 4: Cumulative distribution of sensor pixel entropy

The final number we will use for the whitening step will be the minimum entropy that 99.95% of the pixels have.

Other entropies, such as min-entropy [36], require knowledge of correlations that we do not have.

Shannon entropy is defined as:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i)\log_2 \mathrm{P}(x_i) \tag{1}$$

where X is the probability distribution of each of the values. For instance, if we assume values of $\{64, 64, 128, 192, 256\}$ would have a probability distribution of $X = \{\frac{2}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\}$ and then calculated the Shannon entropy of those values is $\mathrm{H}(X) = 1.9219$ bits.

For this paper we took 100 "black pictures" from a 9MP sensor. A black picture is the image generated by a CMOS sensor in the absence of light. Think of it as a camera with the lens cap still on. The data from each picture is stored as 16 bit words with some padding at the end of each row. To start with, we calculated the entropy in each pixel individually, that is, 100 entries for the pixel at (0,0) and then repeat this process for all 9MP. The distribution of entropy is shown in figure 3 that shows a peak at 3.4 bits. This means that, for every 16 bits of pixel information, we should be able to extract 3.4 bits of "random".

As a data point, a pixel with all 100 images having different values will have an entropy of $log_2 100 = 6.643$ bits. In these data, there are 3 pixels that have a 1 value that never changes and there is 1 pixel with 100 different values.

The cumulative distribution in figure 4 shows that more than 99% of the values between 3.1 and 3.9 bits and 99.9% between 3.0 and 4.7 bits. This is also stated as 0.05% of the pixels have less than 3.0 bits which is the same as saying that 99.95% of the pixels have at least 3.0 bits of entropy, which is the number we are looking for using with the whitening algorithm.

C. Whitening algorithm

There are many methods available for whitening. Our criteria for the whitening algorithm is that it is easy to show that it does not have a back door and that it can be easily evaluated to show that it does not destroy entropy. There are many other algorithms that can be used. Simply running SHA-2 or SHA-3 on the data is more than adequate [37]. There are additional methods of whitening described in [38].

Whitening is fundamentally different than typical PRNGs. Whitening compresses the data until it has 100% entropy. PRNGs take a smaller amount of entropy (also know as the seed) and uses a fixed algorithm to create a much larger amount of data ($d \gg s$). In this case, the data d will never have any more entropy than the seed s had. It is this expansion feature of PRNGs that we claim is both unnecessary and dangerous.

In this paper the algorithm starts with the average Shannon entropy s, the word size w in bits. We then calculate $p = \lceil w/s \rceil$ pixels to fill a w bit word. p must be larger than a natural boundary of the data (such as a pixel). Also, we need $p \geq w$ or there is a possibility that 2 or more bits may cancel each other out. If these two requirements can not be met adjusting the word size, you can run the process multiple times each time increasing the entropy per bit until it reaches 100%.

In algorithm 1, we use an array A indexed by all possible pixel values containing w bit random numbers. We then accumulate them and output the result after p pixels are added. The simple rotate is added to make sure that if the same pixel value is added to the same word, they do not cancel each other out.

For our case, we simply populated A with data from `/dev/random`. The result needs to be tested to make sure that there is adequate randomness in the result. Even

Data:
- an array A indexed by all possible pixel values containing w bit random numbers.
- p pixels per word (described above)
- Input pixels

Result: Output words containing fully random bits

```
while at least $p$ pixels left do
    $a \leftarrow 0$ ;
    for 1...$p$ do
        $a \leftarrow \text{Rotate}_1(a)$
        $x \leftarrow$ Get next pixel
        $a \leftarrow a \oplus A(x)$
    end
    Output $a$
end
```

Algorithm 1: Whitening algorithm

though there is PRNGs used in `/dev/random` for the constants in the algorithm, these data are not the source of randomness, the pixel values are.

In our case, the total number of pixel values is 65536 and $w = 32$ bits. With $s = 3.0$ we get a $p = 11$ pixels per 32 bit word.

D. Characterization and calibration

This paper takes an analog physical device and calibrates it to produce random numbers. In this case the amount of entropy that the CMOS sensor produces is an analog number, not a constant. Unlike other parts of computers like RAM, where there is constant error correction and detection, there are no hardware checking that there is randomness being produced. As a result, the CMOS sensor's Shannon Entropy needs to be calibrated before it is used and routinely while it is being used. The goal of calibration is both check that s is within expected bounds and then recalculate an appropriate p for this device. The goal is to guarantee the ultimate entropy is as expected.

For the calibration of the device, a test of a number of images roughly at least 10x the expected amount values per pixel so that the redundancy can be seen. In our case we have $s = 3.4$ bits or $2^{3.4}$ values per pixel. The 10x rule gives us about 100 images to get a good result. Certainly running more images does not hurt.

To make sure that the device is continuing to function, routine recalibration is needed. This would be similar to drug lot testing where a small portion of a lot is subjected to additional testing to make sure the lot is OK and to be able to adjust variance out of the system. This routine calculation needs to compare against the original expected distribution, not the most recent, to protect against gradual aging and reducing entropy.

E. Operational Environment

The analog nature of the CMOS sensor means that it is sensitive to variations in the environment and for life-critical and safety-critical applications this is important.

The largest environmental parameter is temperature. As the sensor is cooled, the amount of noise (entropy) goes down. If the temperature would be allowed to approach 0K, the entropy would approach 0 and the system would not produce random numbers. This is very similar to what NIST expects in their 140-4 at their Security Level 4 against "compromise due to environmental conditions or fluctuations outside of the module's normal operating ranges for voltage and temperature." [39].

If physical tampering is a reasonable threat, then tamper responsive behavior could be employed [40]. Additionally environmental sensors such as electomagnetic fields [41] could be added.

F. Checks on each image processed

For Life-Critical and Safety-Critical systems assume computers can make mistakes. It may be valuable to do a χ^2 test on the expected distribution of a single picture. Single image distribution are shown in figures 6 and 7.

The whitening algorithm can be run twice on the data to make sure that the calculation is correct. A final quick statistical test of the numbers can to be performed.

For the ultimate in protection several of these systems can be xor'd together.

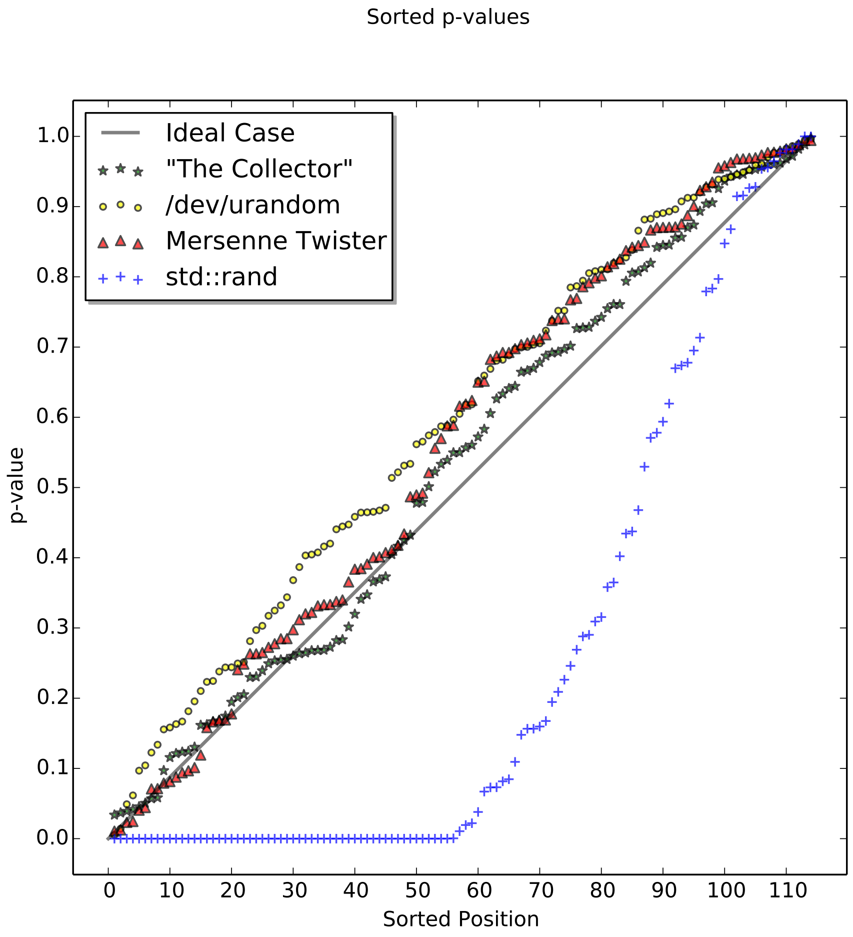

Figure 5: Dieharder test results

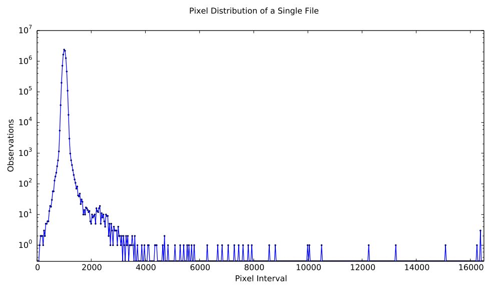

Figure 6: Distribution of pixel values in a single image

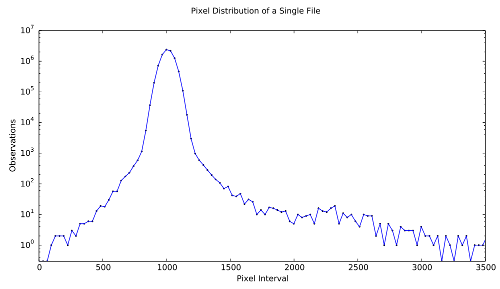

Figure 7: Distribution of pixel values in a single image zoomed in

III. Results

The benchmark for testing randomness is the Diehard test. "The primary point of dieharder (like diehard before it) is to make it easy to time and test (pseudo)random number generators, both software and hardware, for a variety of purposes in research and cryptography." [2]. These tests have been used as the "gold standard" to be able to compare and contrast TRNG and PRNGs.

The results of the testing is shown in figure 5. The Dieharder test [2] produces a a series of numbers that are should be uniformly distributed from $(0 \ldots 1)$. The graph are those numbers sorted and graphed. Closer to the diagonal is better.

In this graph, the LSFR `rand()` clearly fails. For this test we used a Linux CentOS system. On this system, `rand()` is a call to `random()` which "uses a nonlinear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to RAND_MAX". This result justifies the OWASP vulnerability that`random()` should never be used for cryptographic applications [42].

"The Collector" passes all tests and is closer to the diagonal than any other secure PRNG including `/dev/urandom`.

IV. Future work

Future work could be to characterize the sensors under different temperatures, characterize more failure modes and determine appropriate χ^2 values.

V. Conclusion

This paper concludes that it is possible to create true random numbers with at high performance and low cost thus eliminating the problems of insecure PRNGs.

References

[1] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[2] Robert G Brown, Dirk Eddelbuettel, and David Bauer. Dieharder: A random number test suite. version 3.31. 1, 2015.

[3] Ian Goldberg and David Wagner. Randomness and the netscape browser. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 21(1):66–71, 1996.

[4] Arjen K Lenstra, James P Hughes, Maxime Augier, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In *Advances in Cryptology–CRYPTO 2012*, pages 626–642. Springer, 2012.

[5] Dice-o-matic hopper and elevator - gamesbyemail.

[6] Jim Cheetham. OneRNG, Open Hardware Random Number Generator. http://onerng.info/. [Online; accessed October 7, 2015].

[7] D. Schellekens, B. Preneel, and I. Verbauwhede. Fpga vendor agnostic true random number generator. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, Aug 2006.

[8] Thomas Jennewein, Ulrich Achleitner, Gregor Weihs, Harald Weinfurter, and Anton Zeilinger. A fast and compact quantum random number generator. *Review of Scientific Instruments*, 71(4):1675–1680, 2000.

[9] John Walker. HotBits: Genuine random numbers, generated by radioactive decay. https://www.fourmilab.ch/hotbits/, 1996. [Online; accessed October 7, 2015].

[10] Benjamin Jun and Paul Kocher. The intel random number generator. *Cryptography Research Inc. white paper*, 1999.

[11] Bruno Sanguinetti, Anthony Martin, Hugo Zbinden, and Nicolas Gisin. Quantum random number generation on a mobile phone. *Phys. Rev. X*, 4:031056, Sep 2014.

[12] L.C. Noll, R.G. Mende, and S. Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system, March 24 1998. US Patent 5,732,138.

[13] Simon Cooper Landon Curt Noll. LavaRnd is a Random Number Generator. https://web.archive.org/web/20060925013542/http://www.lavarnd.org/what/how-it-works.html. [Online; accessed October 7, 2015].

[14] Tom McNichol. Totally random: How two math geeks with a lava lamp and a webcam are about to unleash chaos on the internet. *Wired*, 11(8), 2003.

[15] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[16] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 35–35, Berkeley, CA, USA, 2012. USENIX Association.

[17] Mike Bendel. Hackers describe ps3 security as epic fail, gain unrestricted access (2011), 2013.

[18] Daniel J Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko Van Someren. Factoring rsa keys from certified smart cards: Coppersmith in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 341–360. Springer, 2013.

[19] Adam Everspaugh, Yan Zhai, Robert Jellinek, Thomas Ristenpart, and Michael Swift. Not-so-random numbers in virtualized linux and the whirlwind rng. In *2014 IEEE Symposium on Security and Privacy*, pages 559–574. IEEE, 2014.

[20] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, 2012.

[21] Keaton Mowery, Michael Wei, David Kohlbrenner, Hovav Shacham, and Steven Swanson. Welcome to the entropics: Boot-time entropy in embedded devices. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 589–603. IEEE, 2013.

[22] Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *NDSS*, 2010.

[23] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 debian openssl vulnerability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 15–27. ACM, 2009.

[24] M Green. The many flaws of dual_ec_drbg. blog article (sept. 18, 2013).

[25] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. *A Formal Treatment of Backdoored Pseudorandom Generators*, pages 101–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[26] File:band filling diagram.svg.

[27] Ricky W. Butler and George B. Finelli. The infeasibility of experimental quantification of life-critical software reliability. *SIGSOFT Softw. Eng. Notes*, 16(5):66–76, September 1991.

[28] R. D. Gow, D. Renshaw, K. Findlater, L. Grant, S. J. McLeod, J. Hart, and R. L. Nicol. A comprehensive tool for modeling cmos image-sensor-noise performance. *IEEE Transactions on Electron Devices*, 54(6):1321–1329, June 2007.

[29] Eric R Fossum et al. Cmos image sensors: electronic camera-on-a-chip. *IEEE transactions on electron devices*, 44(10):1689–1698, 1997.

[30] Charles Kittel. *Introduction to Solid State Physics*. Wiley, 8 edition, 2004.

[31] Massoud Kaviany. *Heat transfer physics*. Cambridge University Press, 2014.

[32] EMVA Standard 1288: Standard for Characterization of Image Sensors and Cameras. http://www.emva.org/wp-content/uploads/EMVA1288-3.0.pdf. [Online; accessed Aug, 2016].

[33] Point Grey Blackfly PGE Imaging Performance Specification. http://www.ptgrey.com/support/downloads/10107. [Online; accessed October 9, 2015].

[34] Sony Diagonal 6mm (Type 1/3) Progressive Scan CCD Solid-state Image Sensor with Square Pixel for B/W Cameras. http://alignment.hep.brandeis.edu/electronics/Data/ICX424AL.pdf. [Online; accessed October 9, 2015].

[35] Helmuth Spieler. *Semiconductor Detector Systems*. Oxford University Press, 1 edition, 2005.

[36] Petr Jizba and Toshihico Arimitsu. The world according to rényi: thermodynamics of multifractal systems. *Annals of Physics*, 312(1):17 – 59, 2004.

[37] NIST DRAFT FIPS PUB. 202. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2014.

[38] Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. In *Cryptographic hardware and embedded systems-CHES 2003*, pages 166–180. Springer, 2003.

[39] Fips pub 140-2, security requirements for cryptographic modules, 2002. U.S.Department of Commerce/National Institute of Standards and Technology.

[40] Steve H. Weingart. *Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses*, pages 302–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[41] Mario Stipčević and Çetin Kaya Koç. True random number generators. In *Open Problems in Mathematics and Computational Science*, pages 275–315. Springer, 2014.

[42] Insecure Randomness. https://www.owasp.org/index.php/Insecure_Randomness. [Online; accessed November 14, 2016].