Entropy Sources for Seeding Pseudo-Random Number Generators

Project documentation

Field of Studies:	BSc in Computer Science
Specialization:	Computer perception and virtual reality
Author:	Matthias Thomann
Supervisor:	Prof. Dr. Rolf Haenni
Date:	January 12, 2021
Version:	1.0

Abstract

Random number generation has been an ongoing research topic for a long time now and is still being expanded today. There are many different aspects of what allows for reliable generation of random numbers. One of the core elements in the concept of randomness in computer science is the *entropy source*.
This document establishes fundamental knowledge about entropy sources and how random numbers are generated through the use of randomness extractors and pseudo-random number generators. Not only reliability of random number generation, but also the security of the algorithms play a major role. How can one achieve a random number generation system that is equally ideal in both randomness and security? Ultimately, no one should be able to interfere in the process, as it would render the idea randomness invalid.

Contents

1 Introduction

This document is written with the intent to explore how entropy sources are chosen and evaluated, hence a fundamental knowledge about computer science is required.

Randomness and random numbers play a major role in the fields of computer science, especially in security and cryptography. However, it proved to be rather problematic to find a number generation system that is both "reliably" random and secure. To find out how and if a truly random number can even be generated, one has to understand the concept of *information entropy* and *entropy sources* first.
From there, one can expand to *randomness extractors*, algorithms created with the intend to generate a random number from a strong entropy source with the aid of a so-called *random seed*. Of course, for the extractor to be ideal, it needs to be able to extract a random number that no one should be able to predict.

2 Entropy Sources

The concept of information theory is that the *informational value* of any given message is defined by how much content of it is deemed as surprising. Therefore, the entropy source is described as to where the message originated from.
In a more technical manner, one has to understand that algorithms and protocols that work with randomness are assuming that the computer has access to a sequence of truly random bits. Shaltiel describes those as a sequence of independent and unbiased coin tosses[1]. Of course, in actual implementations this sequence is generated by taking a sample from some "source of randomness", also known as an *entropy source*.

2.1 Quantifying Unpredictability

First, in order to be able to determine how weak or strong an entropy source is, one has to define how unpredictable the source is. But as described by Kelsey[2], there are some issues when it comes to quantifying unpredictability. For example, a series of coin flips is not reliable enough as there is insufficient information. Another example might be that the intended source is too complex to model.
However, there is a way to determine how reliable a source really is, which is calculating the entropy. There are several approaches to this issue.

2.1.1 Shannon-Entropy

Conceptually, information can be thought of as being stored in variables that can take on different values. The entropy of a variable is the amount of information contained in the variable. It is not quantified by the number of different values it can take on or its length, but rather how much of the information is new. This quantification of information is called the *Shannon entropy*. It is formally defined as:

$$E = -\sum_{i=1}^{k} p(i) \times \log_2 p(i) \tag{1}$$

To put the equation in an example, it is best to think of a fair coin that is flipped twice in a row. The chances for each result are as follows:

- $p_1 = p(\text{heads once, tails once}) = 0.25 + 0.25 = 0.5$
- $p_2 = p(\text{heads twice}) = 0.5 \times 0.5 = 0.25$
- $p_3 = p(\text{tails twice}) = 0.5 \times 0.5 = 0.25$

So for p_1, the entropy would be $0.5 \times \log_2 0.5 = 0.5 \times -1 = -0.5$. For both p_2 and p_3, it is $0.25 \times \log_2 0.25 = 0.25 \times -2 = -0.5$.
So in total, $E = -(p_1 + p_2 + p_3) = 1.5$. The Shannon entropy E of the example above is therefore denoted as 1.5 *bits* or *shannons*.

2.1.2 Min-Entropy

Next to the famous Shannon entropy, other well known formulas have been developed. The so-called *min-entropy* is the most conservative way of measuring the unpredictability of a set of outcomes. It is defined as follows, where H is the entropy and p is the probability of each event:

$$H = -\log_2 \underset{1 \leq i \leq k}{max} (p_i) \tag{2}$$

Generally, it is recommended to work with the min-entropy as it represents a very conservative measure of entropy. It is based solely on the biggest probability. Therefore, it is measuring the effectiveness of the strategy of guessing the most likely output of the entropy source[3].
For the example in subsubsection 2.1.1, the min-entropy H would be calculated as follows, as p_1 is the biggest probability:

$$H = -\log_2 (p_1) = -\log_2 0.5 = 1\, bit \tag{3}$$

Note that the maximum possible value for the min-entropy of a random variable with k distinct values is $\log_2 k$, which is the case when the random variable X has a uniform probability distribution, like $p_1 = p_2 = ... = p_k = 1/k$.

2.2 Real-Life Uses

Naturally, there are different kind of sources, each with individual strength in terms of the quality of randomness. This model shows the entropy source in detail. Each component will now be explained further.

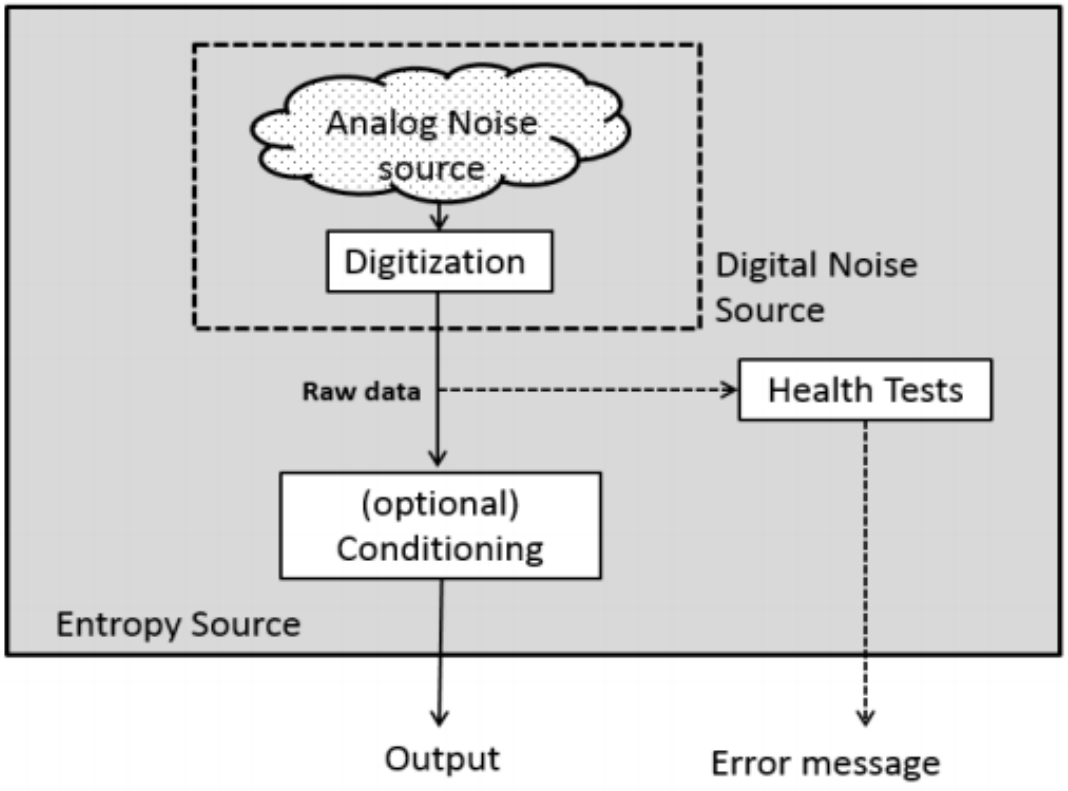

Figure 1: Depiction of an entropy source model [4].

2.2.1 Practical examples

Some practical examples of entropy sources are

- Measuring user dependent behavior, for example mouse movement or keyboard stroke timings.
- Measuring timing of past events, for example how much time the last disk operation took.
- Measuring thermal or radioactive noise.
- Measuring voltage levels.

More interestingly, here are the sources that are typically able to be access in software, repeating some of the above:

- Timings between key presses
- Timings and positions of mouse movements
- Timings between interrupts and related events, such as packets arriving to a network card, data arriving from a disk controller...
- Timings at which successive threads are scheduled on to the processor
- The amount of free memory
- Details of files that are liable to change frequently, for example the contents of the "temp" directory
- System temperature monitor readings
- Noise like a microphone input or other peripherals, for example a video camera

2.3 Measuring and Digitizing

It is important to note that the entropy source has the be measured and digitized in order for the randomness extractor to use it. This, of course, can prove to be difficult in some cases where one does not have access to any kind of proper measurement tools.

2.3.1 Quality

As mentioned above, sampling the entropy source can introduce errors. It also provokes the question whether or not the user is measuring what he/she thinks that he/she is actually measuring. One factor could be the noise or unreliability of readings produced by the measuring unit. Another question to solve is how much o sample variability is entropy and how much is just the complexity of the model.
To counter at least some model problems, the sampling rate can be adjusted. For example, measuring voltage level or thermal noise with frequency that is too high, correlated outputs are produced. This can be fixed by decreasing the rate at which the source is measured.

2.3.2 Performance

Equally important as the quality of the measurement of an entropy source is the performance of the measurement. Understandably, electronic systems tend to be the fastest. An example of slower events are loading executions of the operating system and system level events. Even slower are simple physical events, like the interactions between humans and computers.

2.3.3 Additional Noise Sources

As it may happen, other additional sources of noise may be available to the system of the entropy source's origin. Understandably, outputs of those additional sources may be combined to form a joint entropy of the outputs across all sources. However, it may prove to be hard to estimate the entropy of the combined sources. Also, dependencies between those sources may further complicate the entropy estimation.
Furthermore, it is assumed that the entropy sources have a unique primary noise source that is responsible to generate randomness. The National Institute of Standards and Technology (NIST) describes the issue as follows: "It should be noted that multiple copies of the same physical noise source are considered as a single noise source (e.g., a source with eight ring oscillators, where the sampled bits are concatenated to get an eight-bit output, or where the samples bits are XORed together)" [4].

2.4 Validation

Entropy source validation is a necessary step in the process in order to obtain assurance that all relevant requirements are met. There are established programs to validate any given entropy source officially. The National Voluntary Laboratory Accreditation Program (NVLAP) is a program by the NIST specifically for testing, calibrating and accrediting laboratories. The Cryptographic Algorithm Validation Program (CAVP) and Cryptographic Module Validation Program (CMVP) both provide reviews of the results[5].
Validation provides additional assurance that adequate entropy is provided by the source and may be necessary to satisfy some legal restrictions, policies, and/or directives of various organizations.
As described by Kelsey et al. [4], the validation consists of testing by an NVLAP-accredited laboratory against the requirements of SP 800-90B, followed by a review of the results by CAVP and CMVP. The SP 800-90B is a recommendation by NIST for the entropy sources used for random bit generation[6].

2.4.1 Validation Process

First of all, there are restrictions and requirements for the submitted data, which is to be validated. A sequential dataset of at 1'000'000 sample values obtained directly from the noise source is necessary. Should it not be possible to obtain that many samples, it is also allowed to use a concatenation of several smaller sets of consecutive samples.
The validation process defined by NIST also speaks of a *conditioning component* (see subsection 2.7). If such a component was used, another 1'000'000 consecutive conditioning component outputs must be submitted, concatenated in order they were generated and treated as a binary string for testing purposes.

For the validation process, an entropy estimate has to by submitted next to the sample data. The entropy estimate of a noise source, which is calculated from a single, long-

output sequence, might provide an overestimate if the noise source generates correlated sequences after restarts. This is why the source is restarted during validation exactly 1'000 times. Each time, a set of 1'000 consecutive samples from the source are collected and an output is generated.
Usually, a restart of the entropy source could be as simple as powering off the system, letting it cool down, wait for ten seconds, then proceed.
The idea behind those restart tests is to re-evaluate the entropy estimate that was provided for the entropy source, by using different outputs from many restarts. Additionally, it ensures that the outputs, generated after a restart, are drawn from the same distribution as every other output. It is important to note that the tests will also reveal whether or not the distribution of samples in a restart sequence is independent of its position in the restart sequence.

2.5 Health Tests

The validation is an official process that is required to check if the entropy source does provide enough entropy. But what if the entropy source is erroneous? This is where *health tests* are introduced, a vital and required part of any entropy source.
They aim to detect deviations from the intended behavior of the noise source in a quick and efficient manner. Also, they should be able to provide a high chance of correct assessments. A health test works by taking the output of the entropy source as input and characterizing the expected behavior based on those values.

Health tests are necessary as entropy sources may be fragile. Examples of how an entropy source could be influenced are mainly physical conditions of the device in use, such as:

- Temperature
- Humidity
- Electric field

2.5.1 Deviation Detection

Naturally, the questions arises as how to detect if an entropy source is actually healthy and stable. For that, there are multiple ways of doing so. In the publication of the SP 800-90B recommendation by NIST[4], common occurrences are:

1. When there is a significant decrease in the entropy of the outputs
2. When noise source failures occur
3. When hardware fails, and implementations do not work correctly

2.5.2 Types of Health Tests

An important detail to note is that health tests are performed before any conditioning (see subsection 2.7) is done. Also, there are different types of health tests. In a similar fashion to the restart tests, *start-up health tests* are applied after the system has been started or restarted, before the first use of the entropy source. They specifically validate that nothing has failed since the last use of the entropy source .
Then there are *continuous health tests*. These are run, as the name suggests, continuously and indefinitely on the outputs of the entropy source. Their goal is to detect failures during the use of the source, as it produces outputs. The continuous health tests use the digitized output of the underlying source and since they are executed constantly, it is important for them to have an extremely low chance of raising a false alarm. Since one can assume that the device has a low false positive probability itself, health tests are generally designed to mainly detect bigger failures.
In some cases, in order for the health test to be able to detect failures, it has to accumulate data over time. Should a failure be detected during usage of an entropy source, one has to assume that some or most of the past outputs may be subject to error due to an erroneous source or failure in the source.
Lastly, NIST defined *on-demand health tests* as tests that can be called at any time. They are usually not required to be executed during operation. However, the entropy source must be capable of performing such tests during runtime.

Should an error arise or should the health test detect a failure, it is job of the entropy source to notify the consuming application.

2.6 Entropy Estimation

It is already established that a strong entropy source is one that has the ability to reliably create random outputs. In subsection 2.1, a way of how to calculate the entropy of any given entropy source was introduced. In practice, there are more methods than just using the min-entropy.

2.6.1 Independent and Identically Distributed

The samples from a noise source are *independent and identically distributed* (IID) if each sample has the same probability distribution as every other sample, and all samples are mutually independent, as it is described by NIST[4]. Since a reliable entropy source should not be identically distributed, this is not the desired case. Also, a viable entropy source fails to produce IID outputs.

2.6.2 Recommended Estimators

The method used to estimate the entropy of an entropy source is called an *estimator*. Usually, it is recommended to use an estimated named "The Most Common Value Estimator", better known as the min-entropy estimator.

If the output of an entropy source is IID, is it recommended to only use the min-entropy estimator to calculate the entropy, as the entropy source is weak and therefore, only a very conservative way of measuring entropy should be used.

For non-IID data, many different estimators can be applied to the outputs in order to calculate the entropy. Only to name a few, there is the collision estimator, the Markov estimator, the compression estimator and many more. However, this document will not go deeper into the details of each different estimator.

2.7 Conditioning Component

In the process of random number generation, a so-called *conditioning component* may be used. As previously described, the higher the entropy of a noise source, the better, as it reflects the unpredictability of the source. Therefore, if a source has too many outputs, a conditioning component may be used to reduce the bias[7].

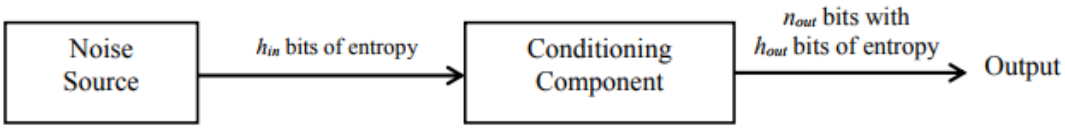

Figure 2: Conditioning component used on an entropy source[4].

3 Randomness Extraction

Now that there is an established way of how to choose an entropy source, how to measure its entropy and how to validate it, the next logical step is to create a system that can extract random numbers from the given entropy source. In other words, the task of the so-called *randomness extractor* now is to take an entropy source, and produce a sequence of truly random bits.
A randomness extractor (k, ϵ)-*extractor* is formally defined as:

$$Ext : \{0,1\}^n \rightarrow \{0,1\}^m \tag{4}$$

Ideally, there would be deterministic function Ext that converts an imperfect source of randomness X into a (nearly) uniformly random output $Ext(X)$. As long as the source of randomness has a sufficiently large amount of entropy, such an extractor should work for all sources.
An extractor is called a *seeded* extractor, if the extractor is given a public uniformly random seed S, which is independent of the source X, and the extracted output $Ext(X;S)$ is required to be close to uniform. For example, the seed S could be a part of a system random number generator (RNG) that extracts randomness from physical sources of entropy, such as the timing of interrupts. If the sources are truly independent of the seed S, then standard (strong) seeded extractors suffice to guarantee that the extracted outputs are nearly uniform[8].

To generate a random number from an entropy source, one has to use the mentioned *seed* as the input, a relatively small amount of random bits obtained from the entropy source. An example of how a randomness extractor could work was described by John von Neumann and has since been extensively documented and worked on. However, practical applications of randomness extractors is still a scientific topic that is being researched right now.

Random number generation is deterministic, often also called *pseudo-random*, which means that for the same input, one will always receive a uniform output. The basic concept of that is, as mentioned, described by Neumann[9]:

For example, given a truly random seed x is obtained by an entropy source X. Neumann's idea of quickly generating a pseudo-random sequence of numbers is to multiply the seed by itself. From there, the middle part of the resulting number is used as the new seed and the method is repeated.
Therefore, M is assumed to be the function that takes the middle part of a number and outputs it. For example, $M(12321) = 232$.

$$RNG(x) = M\left(x \times x\right)$$

In this example, for the seed x, the value $x = 121$ will be used. Of course, the seed could be anything obtained from the entropy source. The number generation process would

then call the function RNG with $RNG(121)$.

$$RNG(121) = M\,(121 \times 121) = M(14641) = 464$$

This 464 is now outputted as part of the generated random number sequence. It is also used as the next seed and the function is called anew. Therefore, for a next step, this would result in:

$$RNG(464) = M\,(464 \times 464) = M(215296) = 529$$

The output of the generated random number sequence is now 464 and 529 concatenated, which is 464529. In a third step, the result of the random number sequence would look like this:

$$RNG(529) = M\,(529 \times 529) = M(279841) = 984$$

And the resulting sequence is now 464529984. This process can now be repeated as many times as needed. This method is known as the *middle-square method*. However, there are some hot-topic issues with the concept of this kind of number generation. First of all, the randomness of the generator is solely dependent on the randomness on the seed. This presupposes that the entropy source is reliably enough and outputs a good entropy value.

3.1 Period

One of the main concerns in pseudo-random number generators is the so-called *period* of the generator. It describes how long it takes for generator to repeat itself and generate an identical set of random numbers as it has before.
In the example of the middle-square method above, the period would be very short. The issue is, that when starting with any given seed, the algorithm would eventually end up with a run-through where M of a certain number would output the same as the original seed, therefore forcing the generator to repeat itself. This, of course, results in the same sequence being generated again, and that length before the cycle repeats itself is the period.

In the particular case of the middle-square method, the period is actually at most the length of the seed. For example, given the seed is still $x = 121$, the algorithm will loop through every whole three-digit number there is between 000 and 999. This means that the algorithm will eventually hit 121 again and that at the latest of 999 cycles. Therefore, the period of $RNG(x)$ when x is a three-digit number is at the most 1000 and cannot expand beyond that.
However, if a large enough seed is chosen, the algorithm can expand into millions and millions of digits before repeating itself.

3.2 Seed Space

Another problem is the so-called *seed space*. For example, if one A could generate a truly random sequence of 20 digits, it would be equivalent to a uniform selection from the stack of all possible sequences of digits. This stack would contain 10^20 records, an astronomically large number. In comparison, generating a pseudo-random number sequence using a four-digit seed would be equivalent to a uniform selection from $10'000$ possible initial seeds, meaning that one could only ever generate $10'000$ different sequences. This describes the shrinking from the large *key space* to the actual, much smaller *seed space*. But for a pseudo-random sequence to be indistinguishable from a randomly generated sequence, it must be impractical for a computer to try all seeds and look for a match. In short, with pseudo-random generators, the security increases as the length of the seed increases.
This leaves with an important decision on what is possible and what is possible in reasonable time. The distinction is made between perfectly secure and practically secure. This concept has been adapted for a long time. An analogy for this is locking a bicycle with a padlock or number lock. It is practically secure, since it only stays unguarded for a short period of time. However, it is not perfectly secure.

References

[1] R. Shaltiel, *An introduction to randomness extractors*, https://cs.haifa.ac.il/~ronen/online_papers/ICALPinvited.pdf, [Online; accessed December 24, 2020].

[2] J. Kelsey, *Entropy and entropy sources in x9.82*, https://csrc.nist.gov/CSRC/media/Events/Random-Number-Generation-Workshop-2004/documents/EntropySources.pdf, [Online; accessed December 24, 2020], 2004.

[3] S. Vajapeyam, *Understanding shannon's entropy metric for information*, https://arxiv.org/ftp/arxiv/papers/1405/1405.2061.pdf, [Online; accessed April 3, 2020], 2014.

[4] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, Mike Boyle, *Recommendation for the entropy sources used for random bit generation*, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf, [Online; accessed April 3, 2020], 2016.

[5] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry McKay, Mary Baish, Michael Boyle, *Implementation guidance for fips 140-2 and the cryptographic module validation program*, https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf, [Online; accessed January 11, 2020], 2003.

[6] ——, *Recommendation for the entropy sources used for random bit generation*, https://csrc.nist.gov/publications/detail/sp/800-90b/final, [Online; accessed December 24, 2020], 2018.

[7] Naoya Torii, Hirotaka Kokubo, Dai Yamamoto, Kouichi Itoh, Masahiko Takenaka, Tsutomu Matsumoto, *Asic implementation of random numbergenerators using sr latches and its evaluation*, https://www.researchgate.net/publication/303291520_ASIC_implementation_of_random_number_generators_using_SR_latches_and_its_evaluation, [Online; accessed January 11, 2021], 2016.

[8] Yevgeniy Dodi, Vinod Vaikuntanathan, *Extracting randomness from extractor-dependent sources*, https://eprint.iacr.org/2019/1339.pdf, [Online; accessed December 24, 2020], 2020.

[9] *Pseudorandom number generators*, https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/random-vs-pseudorandom-number-generators, [Online; accessed December 24, 2020], 2013.

[10] *Github: Code repository.* [Online]. Available: https://github.com/mth0348/info-seminar.

[11] *Randomness extractors*, https://people.seas.harvard.edu/~salil/pseudorandomness/extractors.pdf, [Online; accessed November 20, 2020].