Short communication: An interpretation of the Linux entropy estimator

Benjamin Pousse
benjamin(dot)pousse(at)gmail(dot)com

Institute: None

Abstract. The Linux [1] random number generator (LRNG) aims to produce random numbers with all the limitations due to a deterministic machine. Two recent analysis exist for this generator [2,5]. These analysis provide strong cryptographic details about LRNG. However both fail to give a mathematical explanation of the entropy estimator embedded. In this paper we propose an interpretation using Newton polynomial interpolation.
Keywords: PRNG, entropy estimator.

1 Introduction

Many cryptographic primitives need random data, e.g. asymmetric cryptosystems for padding data and authentication protocols for generating challenge. A solution is to consider PRNG with entropy inputs [6]. This kind of PRNG outputs bits using its internal state which is frequently updated using various entropy sources. The entropy of the inputs must be estimated by these generator for ensuring security.

In this paper, we focus on the Linux [1] Random Number Generator which is a PRNG with entropy inputs. This generator has been studied in [2,5]. More precisely we focus on the entropy estimator integrated in the LRNG. This estimator is efficient in memory and computation, but has no mathematical justification. As far as we know descriptions exist for this estimator, but no interpretation. We propose here an explanation of this estimator using polynomial interpolation.

2 Entropy

2.1 Theory side

The entropy aims to compute the amount of information of an object. The main two theories are from Shannon [7] and Kolmogorov [4,3].

The Shannon theory is called information theory and is based upon the probability theory. More precisely the object concerned by this theory is called source, *i.e.* an object outputting messages according to a probabilistic distribution, and may be mathematically represented by random variable. The entropy of a source is defined according to its distribution.

In the other hand the Kolmogorov theory is called algorithmic information theory and is based upon the theory of computation. The object of the theory are messages themselves. The complexity of a message relies to the minimum length of a program producing this message. The main drawback of this theory is that the complexity of a message is not computable in practice.

Both theories have drawbacks:

- The Shannon theory needs to know the probability distribution of a source *a priori*. In other words the source may be well-known before the computation of its entropy. For example to achieve the compression of a message with the Shannon theory, the first step is to read entirely the message to compute statistical distribution of each character.
- The Kolmogorov complexity of a message is not computable. In practice approximations are computed and may have significant interest.

2.2 Practice side

Various entropy sources exist. They can be divided in two types: hardware source and software source.

Hardware sources are based upon a physical process such as electronic noise, cosmic ray and quantum phenomena. These sources are well studied and each agrees with a well defined probability distribution.

Software sources are based upon "unpredictable" data generated by the execution of software. These sources acts according to the actions of the user, the operating system, etc. As a consequence these sources can not be modeled using probability distribution *a priori*, *i.e.* these entropy sources have to be analyzed while they are used.

3 The Linux entropy estimator

3.1 The code

The LRNG is implemented in the file `drivers/char/random.c` in the Linux kernel[1]. The following code is part of the function `add_timer_randomness` and implements the entropy estimator.

CODE

```
1  delta = sample.jiffies - state->last_time;
2  state->last_time = sample.jiffies;
3
4  delta2 = delta - state->last_delta;
5  state->last_delta = delta;
6
7  delta3 = delta2 - state->last_delta2;
8  state->last_delta2 = delta2;
9
10 if (delta < 0)
```

```
        delta = -delta;
if (delta2 < 0)
        delta2 = -delta2;
if (delta3 < 0)
        delta3 = -delta3;
if (delta > delta2)
        delta = delta2;
if (delta > delta3)
        delta = delta3;

/*
 * delta is now minimum absolute delta.
 * Round down by 1 bit on general principles,
 * and limit entropy estimate to 12 bits.
 */
credit_entropy_bits(&input_pool,
                    min_t(int, fls(delta>>1), 11));
```

We explain variables and functions :

- `sample.jiffies` is the timing of the event being proceeded.
- `state` is a structure containing the previous value of `sample.jiffies`, `delta` and `delta2`.
- `min_t` (defined in `include/linux/kernel.h`) first cast its second and third arguments to the type presents in first argument, and then computes the minimum of these two values.
- `fls` (defined in `include/asm_generic/bitops/fls.h`) means "Find Last (most-significant) bit Set". The less-significant bit is numbered 1. In other word

$$fls(x) = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor \log_2(x) \rfloor + 1 & \text{otherwise} \end{cases}$$

This estimator is efficient as it requires only subtractions and to stock three variables.

3.2 Description

We describe the estimator as in [2] including the last modification as presented in [5]. The LRNG estimates the amount of entropy of an event as a function of its timing only. Let t_n denote the timing of event n. Define

$$\begin{cases} \delta_n^1 = t_n - t_{n-1} \\ \delta_n^2 = \delta_n^1 - \delta_{n-1}^1 \\ \delta_n^3 = \delta_n^2 - \delta_{n-1}^2 \end{cases}$$

Also define $MIN_n = \min(|\delta_n^1|, |\delta_n^2|, |\delta_n^3|)$. The entropy of event n, denoted ent_n, is estimated by

$$ent_n = \begin{cases} 0 & \text{if } MIN_n \leq 1 \\ \left\lfloor \log_2 \left(MIN_n \right) \right\rfloor & \text{if } 2 \leq MIN_n \leq 2^{12} \\ 11 & \text{otherwise} \end{cases}$$

In other words ent_n is the logarithm in base 2 of MIN_n bounded by 0 and 11.

4 Polynomial interpolation

We recall some background on polynomial interpolation. We don't give references on this topic as this theory is well-known and is presented in many mathematical books for student.

4.1 Problematic

Interpolation is part of numerical analysis. It aims to construct new data points from a discrete set of known data points. Polynomial interpolation is a specific method based upon polynomial to construct the new data points.

Definition 1. *Consider $a < b$ two integers and $(x_a, y_a), (x_{a+1}, y_{a+1}), \ldots, (x_b, y_b)$ $b - a + 1$ points where x_i and y_i are real numbers for $i = a \ldots, b$. A polynomial P interpolates the points $(x_a, y_a), \ldots, (x_b, y_b)$ if $P(x_i) = y_i$ for $i = a, \ldots, b$.*

P may be constructed using linear algebra. In particular the following theorem is fundamental :

Theorem 1. *With the previous notations, there exists an unique polynomial P with degree at most $b - a$ which interpolates the points $(x_a, y_a), \ldots, (x_b, y_b)$. This polynomial is called the interpolating polynomial of these points.*

4.2 Newton polynomial interpolation

Polynomial interpolation may be solved using various methods. We recall briefly the Newton's method. Usually this method is exposed considering the first point (x_a, y_a) as a privileged point. For simplicity in the document we choose to privilege the last point (x_b, y_b). This alternative is sometimes called backward Newton polynomial interpolation.

First we define the divided differences:

Definition 2. *With the previous notations, the backward divided differences are defined as*

$$\begin{cases} [y_i] := y_i & \text{for } i = a, \ldots, b \\ [y_{i+j}, \ldots, y_i] := \dfrac{[y_{i+j}, \ldots, y_{i+1}] - [y_{i+j-1}, \ldots, y_i]}{x_{i+j} - x_i} & \text{for } j = 1, \ldots, b-a, \ i = a, \ldots, b-j \end{cases}$$

Then the interpolating polynomial may be expressed using divided differences:

Proposition 1. *With the previous notations, the interpolating polynomial of the points* $(x_a, y_a), \ldots, (x_b, y_b)$ *is*

$$N_a^b(x) := \sum_{i=0}^{b-a} [y_b, \ldots, y_{b-i}] \prod_{j=0}^{i-1} (x - x_{b-j})$$

4.3 Example

Consider the points $P_0 = (1, 5), P_1 = (4, 7), P_2 = (6, 4), P_3 = (9, 2)$. The divided differences are:

$$\begin{array}{l|l|l|l|l}
x_0 = 1 & [y_0] = 5 & & & \\
 & & [y_1, y_0] = \frac{2}{3} & & \\
x_1 = 4 & [y_1] = 7 & & [y_2, y_1, y_0] = -\frac{13}{30} & \\
 & & [y_2, y_1] = -\frac{3}{2} & & [y_3, y_2, y_1, y_0] = \frac{3}{40} \\
x_2 = 6 & [y_2] = 4 & & [y_3, y_2, y_1] = \frac{1}{6} & \\
 & & [y_3, y_2] = -\frac{2}{3} & & \\
x_3 = 9 & [y_3] = 2 & & &
\end{array}$$

The corresponding interpolating polynomial are:

– when considering points P_2 and P_3:

$$N_2^3(x) = 2 - 2/3(x - 9)$$

– when considering points P_1, P_2 and P_3:

$$N_1^3(x) = 2 - 2/3(x - 9) + 1/6(x - 9)(x - 6)$$

– when considering points P_0, P_1, P_2 and P_3:

$$N_0^3(x) = 2 - 2/3(x - 9) + 1/6(x - 9)(x - 6) + 3/40(x - 9)(x - 6)(x - 4)$$

These polynomial are presented in Figure 1.

4.4 When data points are equidistributed

We consider equidistributed points. More precisely, there exists $h > 0$ such that $x_i - x_{i-1} = h$ for $i = a + 1, \ldots, b$. In this case the interpolating polynomial has a simpler expression.

First we introduce the notation Δ:

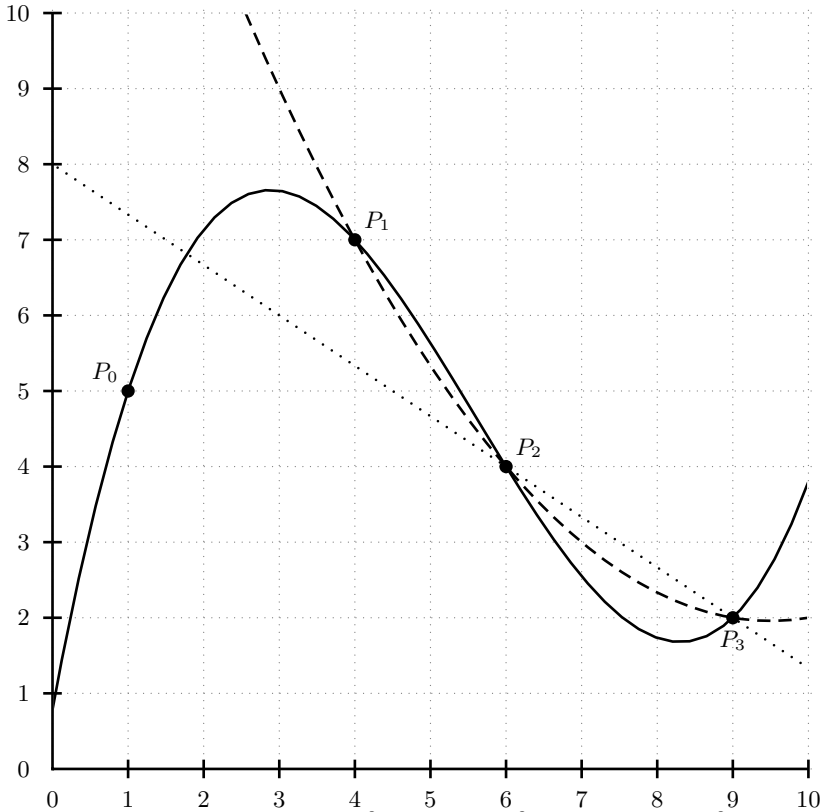

Fig. 1. Interpolating polynomials N_2^3 (dotted), N_1^3 (dashed) and N_0^3 (solid) for the points $P_0 = (1, 5), P_1 = (4, 7), P_2 = (6, 4), P_3 = (9, 2)$

Definition 3. *Let $(u_n)_{n\in\mathbb{N}}$ be a numerical sequence. The following notations are defined recursively:*

- $\Delta^0(u_n) := u_n$ *for* $n \in \mathbb{N}$.
- $\Delta^i(u_n) := \Delta^{i-1}(u_n) - \Delta^{i-1}(u_{n-1})$ *for* $i \in \mathbb{N}^*,\ n \geq i$.

Remark that in the literature the notation ∇ is sometimes used for these differences as they are defined backward.

Using these differences the following lemma allows to simplify the divided differences:

Lemma 1. *With the previous notations and considering the data points equidistributed, for $k = a, \ldots, b$ and $i = 0, \ldots, k - a$:*

$$[y_k, \ldots, y_{k-i}] = \frac{\Delta^i(y_k)}{h^i i!}$$

Proof: Let $a \leq k \leq b$. The relation is proven recursively on i.

- The case $i = 0$ is easy by definition of the divided differences and the notation Δ and using the convention $h^0 = 1$ and $0! = 1$.

– Suppose the relation verified for $i < k - a$. Then:

$$\begin{aligned}
[y_k, \dots, y_{k-(i+1)}] &= \frac{[y_k, \dots, y_{k-i}] - [y_{k-1}, \dots, y_{k-(i+1)}]}{x_k - x_{k-(i+1)}} \\
&\overset{\text{rec.}}{=} \frac{\frac{\Delta^i(y_k)}{h^i i!} - \frac{\Delta^i(y_{k-1})}{h^i i!}}{h(i+1)} = \frac{\Delta^i(y_k) - \Delta^i(y_{k-1})}{h^{i+1}(i+1)!} \\
&\overset{\text{def. } \Delta}{=} \frac{\Delta^{i+1}(y_k)}{h^{i+1}(i+1)!}
\end{aligned}$$

■

As a consequence the interpolating polynomial has a simpler expression:

Proposition 2 (*Newton Backward Divided Difference Formula*). *Using the previous notations and considering the data points equidistributed, the interpolating polynomial N_a^b evaluated at the point $x_b + sh$ with $s \in \mathbb{R}$ is:*

$$N_a^b(x_b + sh) = \sum_{i=0}^{b-a} \frac{s(s+1)\dots(s+i-1)}{i!} \Delta^i(y_b)$$

Finally the following corollary stands:

Corollary 1. *With the previous notations and considering the data points equidistributed, the interpolating polynomial N_a^b evaluated at the point $x_b + h$ is:*

$$N_a^b(x_b + h) = \sum_{i=0}^{b-a} \Delta^i(y_b)$$

4.5 Interpolation error

Interpolation theory leads to consider the following question: if the known data points are constructed using a given function f, *i.e.* $y_i = f(x_i)$ for $i = a, \dots, b$, how good is approximated f by the interpolating polynomial? Interpolation error is introduced to answer:

Definition 4. *With the previous notations and assuming the existence of a function f such that $f(x_i) = y_i$ for $i = a, \dots, b$, the interpolation error is defined as the distance between the evaluation of the function f and the evaluation of the interpolating polynomial. More precisely for x in the domain of f:*

$$e_a^b(x) := \left| f(x) - N_a^b(x) \right|$$

5 Link between the Linux entropy estimator and interpolation polynomial

5.1 Discussion on entropy

As presented in Section 2 the Linux entropy estimator has to evaluate the entropy of the source on-the-fly. According to this constraint the Kolmogorov complexity seems the best approach.

Informally the entropy can be interpreted as the unpredictability (randomness) of a sequence. Even if this don't rely to the formal definition of entropy, this idea may be an acceptable simplification.

The unpredictability of a sequence can be evaluated by answering the following question: given a finite set of data, is it possible to guess the next value? This question leads to consider interpolation as a solution. This idea is implemented in the Linux entropy estimator.

5.2 The link

As presented in Section 3.2 the Linux entropy estimator computes the amount of entropy of an event as a function of its timing. We consider each event and its timing as a couple (n, t_n). Then an estimation of entropy can be computed considering the interpolation error: when a new event (n, t_n) occurs, does it agree with the pattern of previous values? In other words, how large is $e^{n-1}_{n-1-i}(n)$ for various value of i? Let evaluate these values. As the data points are equidistributed, we have using Corollary 1:

$$\begin{aligned} e^{n-1}_{n-1-i}(x_n) &= |y_n - N^{n-1}_{n-1-i}(x_{n-1}+h)| \\ &= \Big| \underbrace{\underbrace{\underbrace{\underbrace{y_n}_{\Delta^0(y_n)} - \Delta^0(y_{n-1})}_{\Delta^1(y_n)} - \Delta^1(y_{n-1})}_{\Delta^2(y_n)} - \ldots}_{\ddots} - \Delta^i(y_{n-1}) \Big| \\ &= |\Delta^{i+1}(y_n)| \end{aligned}$$

Using this relation the variable $MIN_n = \min(|\delta^1_n|, |\delta^2_n|, |\delta^3_n|)$ defined in Section 3.2 can be interpreted as

$$\begin{aligned} MIN_n &= \min\left(|\Delta^1(t_n)|, |\Delta^2(t_n)|, |\Delta^3(t_n)|\right) \\ &= \min\left(e^{n-1}_{n-1}(n), e^{n-1}_{n-2}(n), e^{n-1}_{n-2}(n)\right) \end{aligned}$$

In other words the Linux entropy estimator evaluates the entropy of an event by this process:

- Consider the three interpolating polynomials based upon the last three events.
- Compute the three interpolation errors according to the new event.
- Take the minimum of these errors.
- Compute the logarithm in base 2 of this minimum (bounded by 0 and 11).

6 Conclusion

This short paper presents the first (as far as we know) interpretation of the Linux entropy estimator. Polynomial interpolation allows to understand the estimator.

Moreover the underlying idea seems to be based upon Kolmogorov complexity rather Shannon entropy. This choice was driven by the constraint that the estimator as to perform on-the-fly, and then it can not use statistical analysis of the source. Furtherwork may be done on the optimal number of points for the best interpolation (avoiding Runge's phenomenon).

Acknowledgements

The author would like to thank Marion Videau and Andrea Roeck for their comments.

References

1. Linux kernel, http://kernel.org 1, 1, 3.1
2. Gutterman, Z., Pinkas, B., Reinman, T.: Analysis of the linux random number generator. IACR Cryptology ePrint Archive 2006, 86 (2006) 1, 1, 3.2
3. Kolmogorov, A.: Logical basis for information theory and probability theory. Information Theory, IEEE Transactions on 14(5), 662–664 (1968) 2.1
4. Kolmogorov, A.: Three approaches to the quantitative definition ofinformation'. Problems of information transmission 1, 1–7 (1965) 2.1
5. Lacharme, P., Röck, A., Strubel, V., Videau, M.: The linux pseudorandom number generator revisited. IACR Cryptology ePrint Archive 2012, 251 (2012) 1, 1, 3.2
6. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996) 1
7. Shannon, C.: A mathematical theory of communication (1948) 2.1