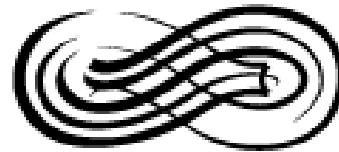


Pseudorandom Number Generation, Entropy Harvesting, and Provable Security in Linux

Seth Hardy



tsumego
FOUNDATION

`shardy@tsumego.com`

`http://www.tsumego.com`

Black Hat Europe 2004

May 20, 2004

Let's start with a few questions...

- What is a *pseudorandom number generator (PRNG)*?
- Why do we even need a new PRNG?
- What is entropy, and how is it used by a PRNG?
- What do *random* and *pseudorandom* mean? How about *quasirandom*?
- How can I tell whether a PRNG is good? What does “good” mean?
- What does “provably secure” mean in this context?

Why reinvent the wheel?

- Additional features... from a PRNG?
- Provable security... but what does that mean?
- Speed and efficiency: different platforms and uses have different needs.
- Simplicity of overall design: ease of use, understanding.
- Modularization: ease of extension, customization.

Section I: Background

PRNGs

What is a “pseudorandom number generator”?

$$G : \{0, 1\}^k \rightarrow \{0, 1\}^n$$

However, we'd like to see G have a few specific properties for it to be *useful*:

- n larger than k
- $G(x)$ computationally indistinguishable from random
- Hard to predict output even with some knowledge of the system.

So what would a good measure of evaluating these properties be?

Random? Sequence 1

Let's start by looking at a simple sequence:

1, 1, 1, 1, 1, . . .

Is this random?

Random? Sequence 1

Let's start by looking at a simple sequence:

1, 1, 1, 1, 1, . . .

Is this random?

We need to define the set we're picking from. What if $S = \{1\}$?

Yes! This is a random sequence, if we are picking from the above set.

Random? Sequence 2

Now, let's assume that $S = \{1, 2, 3, 4\}$, and look at the same sequence:

1, 1, 1, 1, 1, . . .

Is this still random?

Random? Sequence 2

Now, let's assume that $S = \{1, 2, 3, 4\}$, and look at the same sequence:

1, 1, 1, 1, 1, . . .

Is this still random?

We still haven't defined the probability of picking each $x \in S$. What happens if we pick according to these probabilities:

$$\Pr[x = 1] = 1 \quad \Pr[x = 2] = 0 \quad \Pr[x = 3] = 0 \quad \Pr[x = 4] = 0$$

Yes! This is also a random sequence, according to the above set and probability distribution.

Uniform Distribution

We can pick randomly according to any “probability distribution”:

$$\mathcal{D} : S \rightarrow \mathbb{R}$$

The probability distribution \mathcal{D} assigns a nonnegative probability to each $x \in S$, such that

$$\sum_{x \in S} \mathcal{D}(x) = 1$$

What if we want to pick something “*at random*”?

When most people say “at random”, what’s usually meant is “*uniformly at random*”. The *uniform distribution* \mathcal{U} is the probability distribution where everything is picked equally often:

$$\text{If } |S| = n, \text{ then } \Pr[x = s] = \frac{1}{n} \text{ for each } s \in_{\mathcal{U}} S.$$

Statistical Distance

How can we tell how “far apart” distributions are?

The *statistical distance* (also known as the L_1 metric) between two probability distributions \mathcal{D} and \mathcal{E} is:

$$d(\mathcal{D}, \mathcal{E}) = \frac{1}{2} \left| \sum_{x \in \mathcal{S}} \mathcal{D}(x) - \mathcal{E}(x) \right|$$

Often we want to see how close a distribution is to the uniform distribution \mathcal{U} . If $d(\mathcal{D}, \mathcal{U}) \leq \epsilon$, then we say \mathcal{D} is ϵ -close to uniform.

Alternatively, we could say \mathcal{D} is *quasirandom within ϵ* .

Entropy

Entropy is a common term used when looking at randomness.

But what exactly is entropy?

- A measure of information?
- A measure of randomness?
- A measure of redundancy?

What about different types of entropy? Shannon entropy, Renyi entropy, min entropy...

Shannon Entropy

The *Shannon entropy* H (often just called *entropy*) is the basic measure of information:

$$H(\mathcal{D}) = - \sum_{x \in S} \mathcal{D}(x) \log_2 \mathcal{D}(x)$$

Shannon entropy is measured in bits per “symbol” (each element in S).

For example, $H(\text{English}) = 2.62$. (We are looking here at the probability distribution over the set $S = \{A, B, C, \dots, Z\}$.)

However, $\log_2 26 \approx 4.70$, showing that there are appx. two bits of redundant information in each English character.

Min Entropy

Min entropy H_∞ can be thought of as a measure of the worst possible case of a probability distribution:

$$H_\infty(\mathcal{D}) = \min\{-\log_2 \mathcal{D}(x) : x \in S\} = -\log_2 \max\{\mathcal{D}(x) : x \in S\}$$

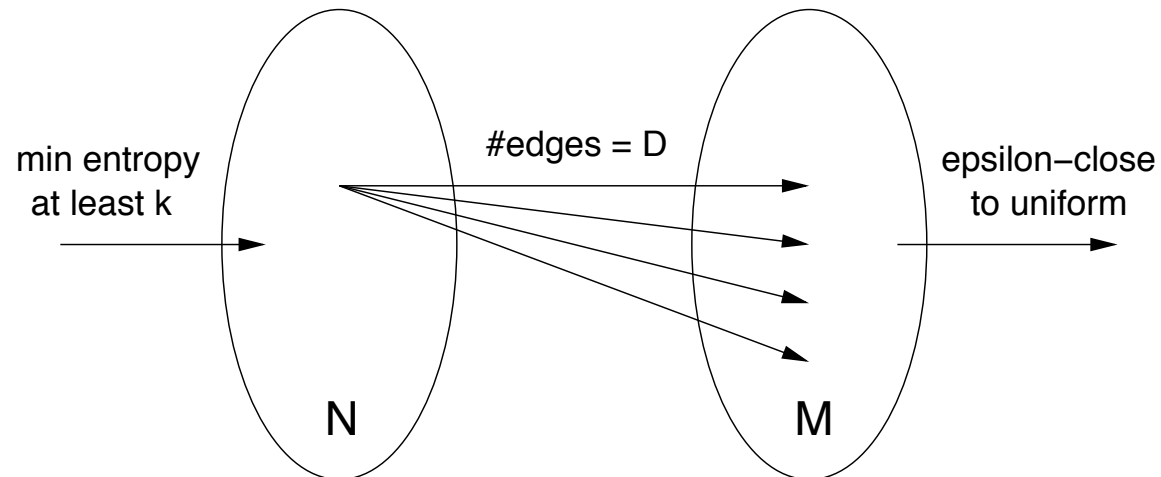
It is possible for a distribution to have a fairly high Shannon entropy, but a small min entropy.

For example, let $\mathcal{D}(x) = \frac{1}{2}$ for some $x \in S$ and some very small probability for all other $x' \in S$.

Section II:
Extractors as Provably Secure
PRNGs

Extractors

Definition. Let $[N], [M]$ be sets of vertices with sizes N and M , and E be a set of edges going from $[N]$ to $[M]$. The graph $G = ([N], [M], E)$ is a (k, ϵ) -extractor if, for any probability distribution \mathcal{D} on $[N]$ with $H_\infty(\mathcal{D}) \geq k$, $\Gamma(\mathcal{D})$ is ϵ -close to uniform on $[M]$.



How “bad” is the input distribution \mathcal{D} ? – What is the min entropy of \mathcal{D} ?

How “good” is the output distribution \mathcal{E} ? – How close to uniform is \mathcal{E} ?

Provable Security

Extractors take a “bad” distribution on $[N]$ and random bits, and use the additional randomness to “smooth” out the distribution into a “good” one over $[M]$.

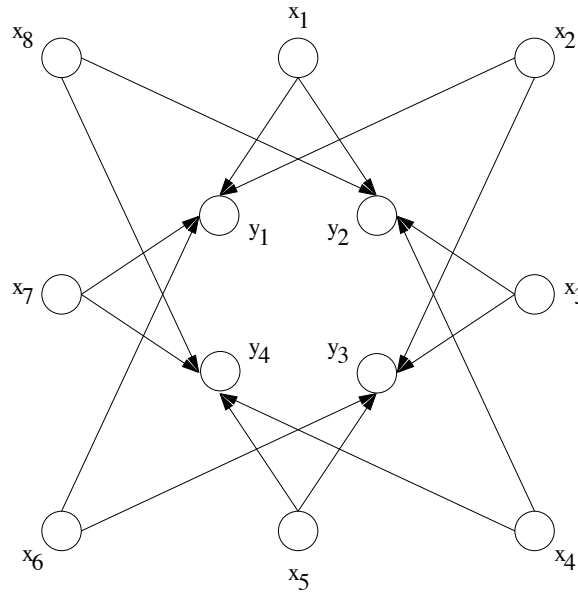
This is a provable level of security: we know how close to uniform the output will be, as long as the input meets the entropy requirement.

Provable security in this sense does not mean that it is unbreakable!

(this last line was in a box because of how important it is; read it again)

- Provable bound on computation needed to distinguish output from uniform; does not cover implementation, etc.

An Example Extractor

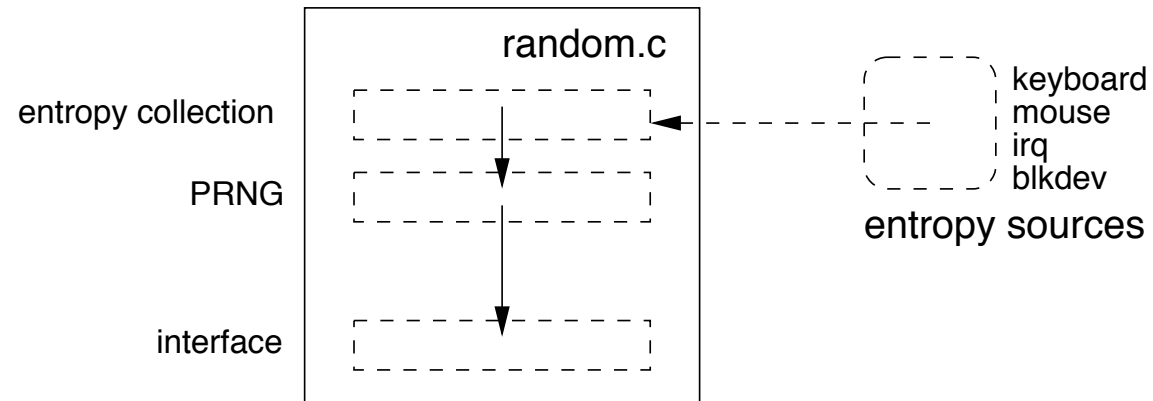


- x_i are the “bad” bits; edges from x_i to y_j are the “good” bits.
- y_j are the output bits.

Is this a good extractor?

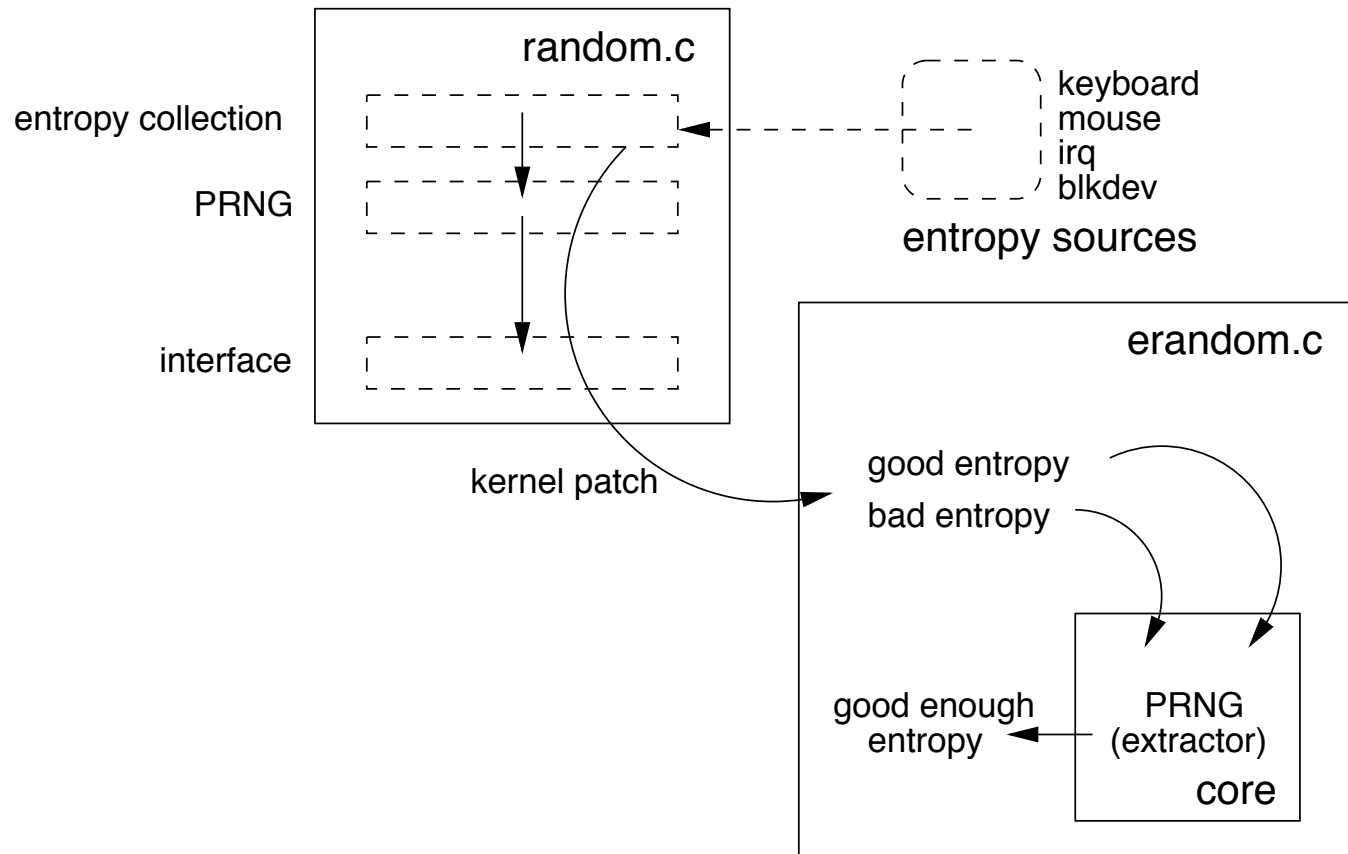
Section III: erandom Version 1

/dev/{,u}random Overview



- Exports functions to get timings from various sources (keyboard, mouse, etc.).
- Gathers entropy, uses as input to PRNG.
- Maintains internal “entropy pool” and number of good bits in pool.
- Hashes pool and provides bits as output when requested.
- `random` blocks when good bit count hits 0; `urandom` does not.

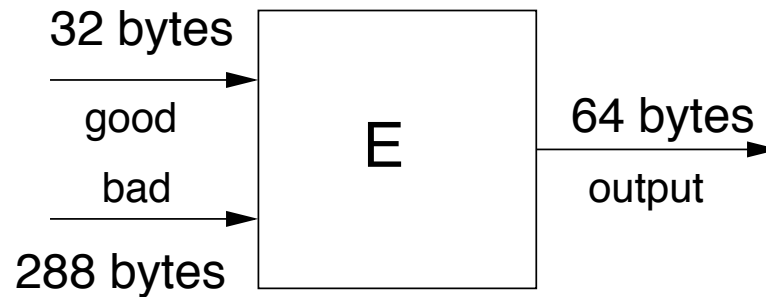
erandom v1 Overview



erandom v1 Overview, Continued

- A kernel patch is required to intercept the entropy from `random.c`.
- `erandom` runs as a module, and requires an extractor core.
- Extractor cores are also loaded as modules. Support for multiple cores possible.
- `erandom` uses the lowest order bit as “good” and the rest as “bad.” Core independent.
- The core implements an extractor, and registers the function with `erandom`.
- `erandom` provides an interface to the core as a character device, just like `/dev/random`.

Example Core: $(k - 1)$ -U Hash Family Extractor



For $a \in \mathbb{F}_{2^{256}}$, let

$$f_a(x_0, \dots, x_9) = x_0 + \sum_{i=1}^9 x_i a^i$$

As a graph: view x_0, \dots, x_9 as the vertex on the left, a as an edge, and f_a as the vertex on the right.

The output of the extractor is $a \circ f_a(x_0, \dots, x_9)$.

Problems With The Kernel Patch Method

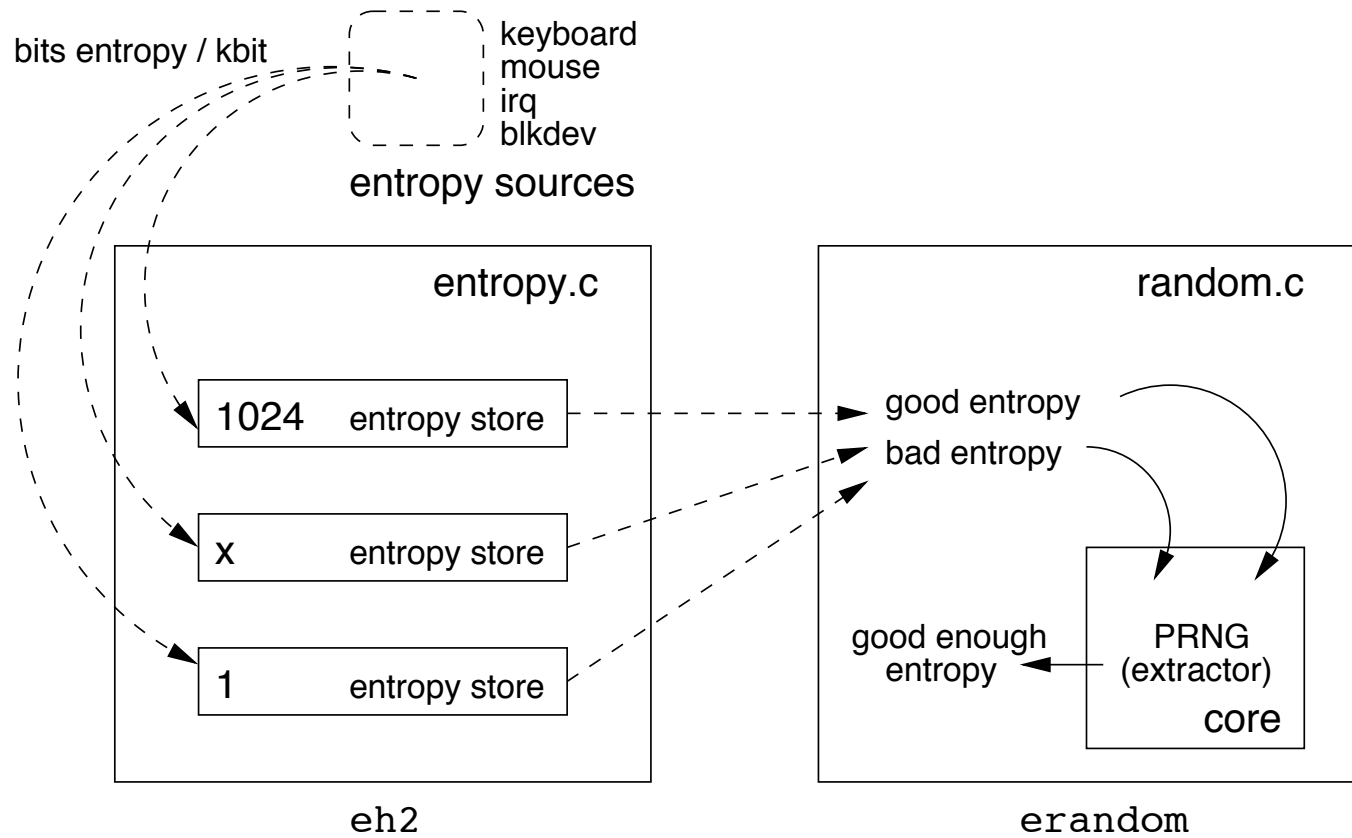
- The same entropy is used in both `{,u}random` and `erandom`.
- Not possible to have multiple entropy sources.
- No concept of different entropy quality levels.

The problem? The entropy harvester is inseparable from the PRNG.

The solution? Rewrite the entropy harvester with support for multiple sources, in the same way that the original `erandom` framework supported multiple PRNG cores.

Section IV: Entropy Harvesting

eh2 Overview



eh2 Overview, Continued

- One entropy store by default: “good” entropy.
- New entropy sources can be registered with the eh2 framework.
- Entropy stores can be created, deleted, managed, listed.
- Entropy sources contribute to the best possible entropy store.
- PRNGs can request entropy of a certain quality, and eh2 will return entropy at least as good as requested.

erandom Modifications

- `erandom` now replaces `{,u}random`.
- `erandom` is implemented as part of the kernel, not as a module.
- The new entropy sources provided by `eh2` are used.
- `eh2` and `erandom` use the same scale of entropy measurement.
- With support for multiple cores, the actual device names (i.e. `/dev/erandom`) are unimportant.

Section V: Conclusions

erandom Improvements

These are the features that `erandom` (hopefully!) gives us:

- Less assumptions about the quality of entropy gathered.
- Provable level of security; assuming input bits are good enough, output bits maintain a particular level of security.
- Speed of internal operations (extractors can be fast and simple).
- Simplicity of design.
- Modularity: support for multiple cores, letting users write their own to suit particular needs.

eh2 Improvements

These are the features that eh2 gives us:

- No assumptions of the quality of entropy gathered.
- Entropy can be of any quality...
- ...which means that entropy present in low-entropy sources is used, not ignored.
- Modularity: it is easy to create and work with new entropy sources.

Future Work

Some goals being worked towards are:

- Implement the `eh2` / `erandom` framework on other OSes, while keeping cores portable.
- Create tutorials to make it easy for users to design their own cores.
- Provide more accurate estimations of entropy in currently used entropy sources.

Questions?