True random number generator [closed]

Asked 16 years, 3 months ago Modified 9 years, 2 months ago

Viewed 26k times

25

Closed. This question is seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. It does not meet Stack Overflow guidelines. It is not currently accepting answers.

We don't allow questions seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. You can edit the question so it can be answered with facts and citations.

Closed 9 years ago.

Improve this question

Sorry for this not being a "real" question, but Sometime back i remember seeing a post here about randomizing a randomizer randomly to generate truly random numbers, not just pseudo random. I dont see it if i search for it.

Does anybody know about that article?

algorithm language-agnostic random prng

Share

Improve this question

Follow

edited Mar 7, 2009 at 0:17

Jonathan Leffler

752k ● 145 ● 946 ● 1.3k

asked Sep 1, 2008 at 10:25

goldenmean

18.9k ● 57 ● 157 ● 216

1 Well, to begin with, you need some true entropy to create a random number. Pseudo-random + Pseudo-random does not true random make. :) I did see an article about using the noise from a webcam in a dark box as entropy, but that is a bit outside the subject. – Lars Mæhlum Sep 1, 2008 at 10:30

1 You could seed a psuedo random number generator with a truly random seed from a service like <random.org>. However, if you know the seed you can still predict the psuedo random number generator. – Chris de Vries Sep 1, 2008 at 10:31

1 Not here, but here is some information and truly random numbers.
 Random.org – HadleyHope Sep 1, 2008 at 10:33

There is this article on SO, it might be the one you were looking for: http://stackoverflow.com/questions/137340/true-random-number-generation-using-ping – Darryl Hein Oct 5, 2008 at 16:50

I recommend hooking the logic circuits of a Bambelweeny 57 Sub-Meson Brain to an atomic vector plotter suspended in a strong Browian Motion producer (say a nice hot cup of tea). <hitchhikers.wikia.com/wiki/Infinite_Improbability_Drive> – Andy Brice Oct 5, 2008 at 17:12

27

I have to disagree with a lot of the answers to this question.

It is possible to collect random data on a computer. SSL, SSH and VPNs would not be secure if you couldn't.

The way software random number generator work is there is a *pool* of random data that is gathered from many different places, such as clock drift, interrupt timings, etc.

The trick to these schemes is in correctly estimating the *entropy* (the posh name for the randomness). It doesn't matter whether the source is bias, as long as you estimate the entropy correctly.

To illustrate this, the chance of me hitting the letter *e* in this comment is much higher than that of *z* , so if I were to use key interrupts as a source of entropy it would be bias - but there is still some randomness to be had in that input. You can't predict exactly which sequence of letters will come next in this paragraph. You can extract entropy from this uncertainty and use it part of a random byte.

Good quality real-random generators like Yarrow have quite sophisticated entropy estimation built in to them and will only emit as many bytes as it can reliably say it has in its "randomness pool."

Share Improve this answer answered Sep 1, 2008 at 13:20

Follow

Simon Johnson
7,902 ● 5 ● 38 ● 50

14 -1 Cryptographically secure pseudorandom != True random – NullUserException Aug 5, 2010 at 15:25

@NullUserException: Yes, but that's just the point. The data in the entropy pool *is* truly random, that's why it's collected so carefully. If you use this data directly, it's truly random; if you use it to seed a PRNG, you get (hopefully good) pseudorandom values. – sleske Jan 19, 2011 at 15:06

Sort of like Shannon-whatzama-callz-it? `random ^ keyboard = random`, `bunch of pseudo random inputs xor-ed together = random`? – Mateen Ulhaq Jun 17, 2011 at 5:17

@NullUserException: Your "true random" numbers may not even exist - what it the whole universe is a simulation being run off a PRNG? The point is, for all practical uses, cryptographically secure pseudorandom is equivalent to true random. – mikera May 22, 2012 at 1:47

17

I believe that was on thedailywtf.com - ie. not something that you want to do.

It is not possible to get a truly random number from pseudorandom numbers, no matter how many times you call randomize().

You *can* get "true" random numbers from special hardware. You could also collect entropy from mouse movements and things like that.

Share Improve this answer
Follow

answered Sep 1, 2008 at 10:30

Blorgbeard
103k ● 50 ● 235 ● 276

10

At the end of the post, I will answer your question of why you might want to use multiple random number generators for "more randomness".

There are philosophical debates about what randomness means. Here, I will mean "indistinguishable in every respect from a uniform(0,1) iid distribution over the samples drawn" I am totally ignoring philosophical questions of what random is.

Knuth volume 2 has an analysis where he attempts to create a random number generator as you suggest, and then analyzes why it fails, and what true random processes are. Volume 2 examines RNGs in detail.

The others recommend you using random physical processes to generate random numbers. However, as we can see in the Espo/vt interaction, these processes can have subtle periodic elements and other non-random elements, in part due to outside factors with deterministic behavior. In general, it is best never to assume randomness, but always to test for it, and you usually can correct for such artifacts if you are aware of them.

It is possible to create an "infinite" stream of bits that appears completely random, deterministically. Unfortunately, such approaches grow in memory with the

number of bits asked for (as they would have to, to avoid repeating cycles), so their scope is limited.

In practice, you are almost always better off using a pseudo-random number generator with known properties. The key numbers to look for is the phase-space dimension (which is roughly offset between samples you can still count on being uniformally distributed) and the bit-width (the number of bits in each sample which are uniformally random with respect to each other), and the cycle size (the number of samples you can take before the distribution starts repeating).

However, since random numbers from a given generator are deterministically in a known sequence, your procedure might be exposed by someone searching through the generator and finding an aligning sequence. Therefore, you can likely avoid your distribution being immediately recognized as coming from a particular random number generator if you maintain two generators. From the first, you sample i, and then map this uniformally over one to n, where n is at most the phase dimension. Then, in the second you sample i times, and return the ith result. This will reduce your cycle size to (orginal cycle size/n) in the worst case, but for that cycle will still generate uniform random numbers, and do so in a way that makes the search for alignment exponential in n. It will also reduce the independent phase length. Don't use this method unless you understand what reduced cycle and independent phase lengths mean to your application.

Share Improve this answer Follow

edited Jan 26, 2010 at 15:24

answered Sep 1, 2008 at 12:21

John with waffle
4,141 ● 23 ● 32

1 Er, what? Avoiding repeats would *reduce* the randomness. Random sequences repeat. And nobody is going to crack a PRNG by "searching through the generator and finding an aligning sequence" They'll either search all possible seeds, or if you use a poor PRNG, just directly figure out the sequence. – Nick Johnson Oct 5, 2008 at 9:28

Also, your proposal for making it "more random" is security through obscurity. Don't invent your own cryptosystems, you're not smart enough (and nor am I). Perfectly satisfactory cryptographically secure PRNGs are already readily available. – Nick Johnson Oct 5, 2008 at 9:29

You are confusing repeats of a sample, rounded to fit your desired range, and repeating the entire state of the system. Since deterministic RNGs are deterministic, if the state of the whole system is the same, the behavior of the generator will repeat(pumping). Please read Knuth vol 2 or other lit – John with waffle Oct 5, 2008 at 13:57

To other readers:, don't feel like you aren't smart enough to a) test whatever built in generator you are using (there are lot of bad ones, unsuitable for simulation work) b) learn enough about the existing generators to know which will suit your needs c) learn to best use results – John with waffle Oct 5, 2008 at 14:15

You said "such approaches grow in memory with the number of bits asked for (as they would have to, to avoid repeats), so their scope is limited." if you mean only that the whole state

can repeat (obviously this is the case), that's not much of a limitation: n bits of state allows 2^n states. – Nick Johnson Oct 5, 2008 at 16:23

10

An algorithm for truly random numbers cannot exist as the **definition** of random numbers is:

> Having unpredictable outcomes and, in the ideal case, all outcomes equally probable; resulting from such selection; lacking statistical correlation.

There are better or worse pseudorandom number generators (PRNGs), i.e. completely predictable sequences of numbers that are difficult to predict without knowing a piece of information, called the **seed**.

Now, PRNGs for which it is extremely hard to infer the seed are **cryptographically secure**. You might want to look them up in Google if that is what you seek.

Another way (whether this is truly random or not is a philosophical question) is to use random sources of data. For example, unpredictable physical quantities, such as noise, or measuring radioactive decay.

These are still subject to attacks because they can be independently measured, have biases, and so on. So it's really tricky. This is done with custom hardware, which is usually quite expensive. I have no idea how good

`/dev/random` is, but I would bet it is not good enough for cryptography (most cryptography programs come with their own RNG and Linux also looks for a hardware RNG at start-up).

Share Improve this answer

Follow

edited Sep 30, 2010 at 19:45

Dave Jarvis

31.2k ● 43 ● 181 ● 323

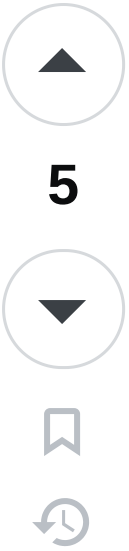
answered Oct 5, 2008 at 16:52

Sklivvz

31.1k ● 24 ● 118 ● 174

1 +1 - simple terms and good explanation. also, /dev/random is indeed good enough for crypto (check en.wikipedia.org/wiki//dev/random) – J.C. Inacio Sep 24, 2009 at 23:07

An algorithm is defined as a step-by-step procedure which terminates in a finite amount of time. A PRNG is a specific type of algorithm, but there is no reason that I could not actually have a random number generation algorithm based on physical phenomenon like one based on capturing photons and measuring their polarity. The definition of algorithm and of random are both important, and the fact that a PRNG is a specific subset of 'algorithm'. They all are algorithms. – Brian Stinar Sep 30, 2010 at 19:55



5

According to Wikipedia `/dev/random`, in Unix-like operating systems, is a special file that serves as a true random number generator.

The /dev/random driver gathers environmental noise from various non-deterministic sources including, but not limited to, inter-keyboard timings and inter-interrupt timings that occur within the operating system environment. The noise data is sampled and combined with a CRC-like mixing function into a continuously updating ``entropy-pool''. Random bit strings are obtained by taking a MD5 hash of the contents of this pool. The one-way hash function distills the true random bits from pool data and hides the state of the pool from adversaries.

The /dev/random routine maintains an estimate of true randomness in the pool and decreases it every time random strings are requested for use. When the estimate goes down to zero, the routine locks and waits for the occurrence of non-deterministic events to refresh the pool.

The /dev/random kernel module also provides another interface, /dev/urandom, that does not wait for the entropy-pool to re-charge and returns as many bytes as requested. As a result /dev/urandom is considerably faster at generation compared to /dev/random which is used only when very high quality randomness is desired.

Share Improve this answer

Follow

answered Sep 1, 2008 at 10:35

Espo

41.9k ● 21 ● 136 ● 161

No! Just because `/dev/random` seeds itself from more sources other than the current time doesn't mean it's truly random. Keyboard timings and kernel jitter are not truly random. However, I think that it can use a hardware RNG is there is one installed on the system. Then it would, in fact, be truly random. That said, in the vast majority of cases you don't need true random numbers. I think the only true RNG we use in my company is in a secure cryptoprocessor in a machine that basically just generates keys. – vt. Sep 1, 2008 at 10:44

@vt: Thank you for your information. The info in my first paragraph was from wikipedia. I will update that wikipedia-article later tonight with your answer if you get upvoted. – Espo Sep 1, 2008 at 10:54

Keyboard timings etc do have a certain (if low) level of information entropy. If put into decent algorithms you can generate true randomness. /dev/random is one implementation. The yarrow algorithm is another. The yarrow paper explains it will (google for it). – Hamish Downer Oct 10, 2008 at 18:21

4

John von Neumann once said something to the effect of "anyone attempting to generate random numbers via algorithmic means is, of course, living in sin."

Not even /dev/random is random, in a mathematician's or a physicist's sense of the word. Not even radioisotope decay measurement is random. (The decay rate is. The

measurement isn't. Geiger counters have a small reset time after each detected event, during which time they are unable to detect new events. This leads to subtle biases. There are ways to substantially mitigate this, but not completely eliminate it.)

Stop looking for true randomness. A good pseudorandom number generator is really what you're looking for.

Share Improve this answer

Follow

answered Oct 5, 2008 at 16:37

Rob Hansen

109 ● 1

Hardware radio-decay RNGs usually compare the size of the two intervals between three (detected) decays. This kills the dead-time issue. 'Course it leave DNL, and other electronics issues, but what you need is not "really, *really* random", but "random enough". – dmckee --- ex-moderator kitten Apr 3, 2009 at 18:05

As long as I'm riffing on physics today: you generally wouldn't actually choose a Geiger-Muller tube for this any more. Silicon detectors are smaller, cheaper, and require less power. – dmckee --- ex-moderator kitten Apr 3, 2009 at 19:46

@rjh: Very nice to quote Neumann ! – Arkapravo Apr 11, 2010 at 9:07

3

If you believe in a deterministic universe, true randomness doesn't exist. :-) For example, someone has suggested that radioactive decay is *truly* random, but IMHO, just because scientists haven't yet worked out the pattern, doesn't mean that there isn't a pattern there to be

worked out. Usually, when you want "random" numbers, what you need are numbers for encryption that no one else will be able to guess.

The closest you can get to random is to measure something natural that no enemy would also be able to measure. Usually you throw away the most significant bits, from your measurement, leaving numbers with are more likely to be evenly spread. Hard core random number users get special hardware that measures radioactive events, but you can get some randomness from the human using the computer from things like keypress intervals and mouse movements, and if the computer doesn't have direct users, from CPU temperature sensors, and from network traffic. You could also use things like web cams and microphones connected to sound cards, but I don't know if anyone does.

Share Improve this answer

Follow

answered Sep 1, 2008 at 16:22

rjmunro

28k ● 21 ● 112 ● 134

2 Actually, it has been mathematically proven that there is no underlying order behind the "randomness" inherent in quantum mechanics (Bell's Inequality, en.wikipedia.org/wiki/Bell%27s_theorem). Unless you're willing to accept nonlocality, which makes the universe really weird :-P – Dan Lenski Oct 5, 2008 at 17:19

I would like to emphasize Dan's comment here. This was a bone of contention throughout most of the 20th century, but the issue is well settled at this point. Moreover Bell's theorem

is surprisingly accessible to the non-specialist.
– dmckee --- ex-moderator kitten Apr 3, 2009 at 19:50

@Dan - Bell's theorem doesn't rule out the possibility of there being absolutely no randomness at all (see Superdeterminism - en.wikipedia.org/wiki/Superdeterminism). It's possible that the entire universe runs off one big PRNG.
– mikera May 22, 2012 at 2:28

And such a universe would be inherently non-local. – nomen Nov 20, 2012 at 0:49

3

To summarize some of what has been said, our working definition of what a secure source of randomness is is similar to our definition of cryptographically secure: it appears random if smart folks have looked at it and weren't able to show that it isn't completely unpredictable.

There is **no** system for generating random numbers which couldn't conceivably be predicted, just as there is no cryptographic cipher that couldn't conceivably be cracked. The trusted solutions used for important work are merely those which have proven to be difficult to defeat so far. If anyone tells you otherwise, they're selling you something.

Cleverness is rarely rewarded in cryptography. Go with tried and true solutions.

Share Improve this answer

Follow

answered Oct 26, 2008 at 23:44

Peter Burns

45.3k ● 7 ● 40 ● 56

3

A computer usually has many readily available physical sources of random noise:

- Microphone (hopefully in a noisy place)
- Compressed video from a webcam (pointed to something variable, like a lava lamp or a street)
- Keyboard & mouse timing
- Network packet content and timing (the whole world contributes)

And sometimes

- Clock drift based hardware
- Geiger counters and other detectors of rare events
- All sorts of sensors attached to A/D converters

What's difficult is estimating the entropy of these sources, which is in most cases low despite the high data rates and very variable; but entropy can be estimated with conservative assumptions, or at least not wasted, to feed systems like Yarrow or Fortuna.

Share Improve this answer

Follow

answered Aug 5, 2010 at 15:19

user412090

356 ● 4 ● 7

It's not possible to obtain 'true' random numbers, a computer is a logical construct that can't possibly create 'truly' random anything, only pseudo-random. There are

2

better and worse pseudo-random algorithms out there, however.

In order to obtain a 'truly' random number you need a physical random source, some gambling machines actually have these built in - often it's a radioactive source, the radioactive decay (which as far as I know is *truly* random) is used to generate the numbers.

Share Improve this answer

Follow

answered Sep 1, 2008 at 10:30

ljs

37.8k ● 36 ● 109 ● 124

@kronoz: Some processors (such as the AMD Geode) have a hardware RNG, based on noisy transistors, which exploit quantum randomness for a true random source. Radioactive decay is unnecessary, there are many other equally random effects. – rmmh Sep 1, 2008 at 16:33

There are many sources that contain enough entropy to be suitable for random generators, thermal drift and white noise are such example. These in turn can affect the different components in the computer. MEasuring this effect *can help* with randomness. – Newtopian Oct 27, 2008 at 2:19

so ... yes, strictly speaking a computer is incapable of generating random sequence computationally, but since this system is based ultimately on analogue components it is possible to still get a glimpse of the "real" world and extract some if it inherent randomness. – Newtopian Oct 27, 2008 at 2:21

radioactive decay isn't necessarily 'truly' random, we simply haven't found a determinant for it yet. – DevinB Mar 18, 2009 at 15:18

1 @devinb: While your claim cannot be *absolutely* ruled out, it would require throwing out a century of experimentally verified physics. Quantum randomness is either *really* random or non-local. Either way it is beyond human prediction. – dmckee --- ex-moderator kitten Apr 3, 2009 at 19:47

1

One of the best method to generate a random number is through Clock Drift. This primarily works with two oscillators.

An analogy of how this works is imagine a race car on a simple oval circuit with a while line at the start of the lap and also a while line on one of the tyres. When the car completes a lap, a number will be generated based on the difference between the position of the white line on the road and on the tyre.

Very easy to generate and impossible to predict.

Share Improve this answer

Follow

answered Sep 1, 2008 at 11:40

GateKiller

75.8k ● 75 ● 175 ● 204