## **COMPUTING / HARDWARE**

# COVER

# Behind Intel's New Random-Number Generator

The random-number generator uses digital circuits to stump the smartest hackers By GREG TAYLOR, GEORGE COX / SEPTEMBER 2011



Imagine that it's 1995 and about to make your very f online purchase. You ope Netscape browser, sippin as the home page slowly You then navigate to Amazon.com, a new onlin bookstore your friend told about. As you proceed to your purchase and enter y payment information, the your browser points to ch from one starting with "htt that begins with "https." T signals that your compute

Image: Carl DeTorres

established an encrypted connection with Amazon's server. This allows you to send credit card information to the s without worrying that an identity thief will intercept the transmission.

Unfortunately, your first online transaction was doomed from the start: It will soon be discovered that the supposec secure transfer protocol your browser just followed wasn't very secure after all.

The problem was that <u>the secret keys Netscape was using</u> weren't random enough. They were strings of only 40 b meaning there were around a trillion possible number combinations. That may seem like a lot, but hackers were at break these codes, even with mid-1990s computer speeds, in about 30 hours. The nominally random number Netsused to form a secret key was based on just three values—time of day, process identification number, and parent-identification number—all of them predictable. This allowed the attackers to reduce the number of keys that they netry, and to find the right one much sooner than Netscape had anticipated.

Netscape's programmers would have loved to use a completely random number to form the encryption key, b

hard time figuring out how to come up with one. That's because digital computers are always in welldefined states, which change only when the programs they are running tell them to change. The best you can often do with a computer is to simulate randomness, generating what are called pseudorandom numbers by using some sort of mathematical procedure. A set of such numbers may at first glance look perfectly random, but somebody else using the same procedure could easily generate exactly the same set of numbers, which often makes them a poor choice for encryption.

Researchers have managed to devise pseudorandom-number generators that are considered cryptographically secure. But you must still start them off using a special seed value; otherwise, they'll always generate the same list of numbers. And for that seed, you really want something that's



Photo: iStor

impossible to predict.

Transistor 1

Fortunately, it's not hard to harvest truly unpredictable randomness by tapping the chaotic universe that surrounds a computer's orderly, deterministic world of 1s and 0s. But how exactly do you do that?

For several years, you could find an online source of random numbers, called <u>Lavarand</u>. It got its numbers from the pictures a computer took of the waxy blobs churning away inside lava lamps. More sophisticated hardware-based systems use quantum-mechanical phenomena, such as photons striking a half-silvered mirror, as a basis for generating random numbers. You can even get an ordinary unassisted computer to produce random numbers based on erratic events taking place within

its own mundane hardware—the precise timing of keystrokes, for example. But to get many of these numbers, you to hammer away at a lot of keys.

We and our colleagues at Intel think this should be easier. That's why for more than a decade now, many of our co chip sets have included an analog, hardware-based random-number generator. The problem is that its analog circu squanders power. Also, it's hard to keep that analog circuitry working properly as we improve our fabrication proce That's why we have now developed a new and entirely digital system that allows a microprocessor to produce a cc stream of random values without those difficulties. Soon it will be coming to a processor near you.

1.2



Transistor 2

Intel's first attempt to

A WHOLE

developec

to generat randomne

lava lamp: million per

grabbed n

from the La

website.

LAVA Lavarand

average computer prod better random numbers 1999, when the compar introduced the Firmwar chip-set component. <u>The random-number genera</u> <u>that chip (PDF)</u> is a rin oscillator-based analog that works by taking so the thermal noise that's in all resistors, amplifyi then using the resulting signal to change the pe

**UNCERTAIN CIRCUITS:** When transistor 1 and transistor 2 are switched on, a coupled pair of inverters force Node A and Node B into the same state [left]. When the clock pulse rises [yellow, right], these transistors are turned off. Initially the output of both inverters falls into an indeterminate state, but random thermal noise within the inverters soon jostles one node into the logical 1 state and the other goes to logical 0. *Click on image for enlargement.* 

relatively slow-running clock. On each of the erratic ticks of that slow clock, the chip then samples the output of a fast-running clock, which cycles back and forth quite regularly between its two possible binary states: 0 and 1. The sampling then produces a random sequence of 1s and 0s.

The rub here is that the analog circuitry needed to amplify thermal noise consumes lots of power—and that circuitr operates whether or not the computer actually needs random numbers for what it's doing at the time. Those analog are also a nuisance when it comes time to improve the manufacturing technology used to make the processor. Eve years, chipmakers modify their fabrication lines to produce integrated circuits at a finer scale, allowing them to pac transistors into the same area. Making these shifts is pretty straightforward for CMOS digital circuitry, but each ne generation of analog circuitry requires careful reevaluation and testing—a major headache.

That's why in 2008 Intel set out to make a random-number generator that uses only digital hardware. Intel research based in Hillsboro, Ore., and at the company's Bangalore Design Lab in India started by tackling a key part of the -how to make a random source of bits without using analog circuitry.

Ironically, the solution this team came up with breaks a cardinal rule of digital design: that a circuit should be in a v defined state, outputting only logical 0s or logical 1s. A digital circuit element can, of course, spend short periods s

between these two possible states. But it should never remain teetering, uncertain of which way to go. That would delays and could even produce system failures.

But in this random-bit generator, <u>teetering is a feature</u>, not a bug</u>. The design includes a pair of inverters, circuit ele whose single output is the opposite of its single input. We connect the output of one inverter to the other inverter's the first inverter's output is 0, so is the input to the second inverter, whose output is then 1. Or if the first inverter's is 1, the second inverter's output will be 0.

This random-bit source adds two rather oddly placed transistors to this two-inverter circuit. Switching those transis forces the inputs and outputs of both inverters to the logical 1 state. The inverters have to be modified slightly to ta sort of abuse, but that's easy enough to do.

Interesting things can now happen when the added transistors are turned off. The two-inverter circuit doesn't like h its inputs and outputs in the same state: It wants the output of one inverter to be 0 and the other 1. But which inver should output which?

There are two possibilities, and for the briefest of moments, the circuit hovers between them. In a perfect world, it linger like that forever. But in reality, even a small amount of thermal noise—random atomic vibrations—within the will send it racing toward one of its two stable states. It's the physically random properties of the thermal noise tha determine the outcome of this otherwise indecisive circuit.

In this way, our simple digital circuit can easily harvest some of the ubiquitous randomness of nature. All we need to connect those two extra transistors to a clock that regularly turns both of them on and off. Every time the clock the circuit generates one random bit.

This digitized approach to random-bit generation would work fine if all inverter circuits were absolutely identical. Bu messiness of the physical world never really allows that. In reality, no two inverters are exactly the same. Having differences in the speed or strength of their responses might seem like a mild offense, but in this application, such differences could easily compromise the randomness we were trying to extract from the circuit.

To keep the inverters in balance, we built a feedback loop into the new hardware. The circuitry in that loop perform targeted fiddling until the two possible output values, 0 and 1, each occur roughly half the time. This helps our desi satisfy one of the rules for statistical randomness: In a long stream of numbers, there should be roughly the same of all possible digits. By adjusting the internal workings of each inverter on the fly, we can defend against the predi that cryptologists so dread.



In parallel with resear efforts to refine a cons digital source of randon other Intel engineers be developing the additiona needed to deliver those effectively in a product. might think that a proce could simply collect the bits from basic random generator hardware, fee

**THREE-LAYER NUMBERS:** Intel's Bull Mountain random-number generator prevents bias and correlation with a three-step process. First, a digital circuit generates a stream of raw random bits. Next, a conditioner generates healthy random seeds. Third, a pseudorandom-number generator readies the numbers for use in software. *Click on image for enlargement.* 

to an application, and call it a day. But the truth is that those bits aren't as random as you'd want them to be. The restream coming out of basic hardware, no matter how good, can have bias and correlation.

Our goal was to build a system that delivers random numbers compliant with and certified to meet common cryptostandards, such as those set by the <u>National Institute of Standards and Technology</u>. To guarantee the quality of ou numbers, we designed a three-stage process—which uses digital circuitry, a conditioner, and a pseudorandom-nur generator—now known by the code name <u>Bull Mountain</u>.

Our previous analog random-number generator was able to produce only a few hundred kilobits of random numbers second, whereas our new generator spins them out at a rate of around 3 gigabits per second. It starts by collecting mostly random output of the two inverters 512 bits at a time. Further circuitry then breaks each package of 512 bits pair of 256-bit numbers. Of course, if the original 512 bits aren't completely random, those 256-bit numbers won't b completely random either. But they can be mathematically combined or conditioned in such a way as to produce a number that's closer to that ideal.

You can see this better with a simple illustration. Suppose for a moment that the random-bit generator spits out jus at a time, which is to say that it provides a stream of binary numbers that range in value from 0 to 255. Suppose fu that those 8-bit numbers aren't completely random. Imagine, for example, that some subtle flaw in the circuitry ten suppress values at the high end of the range. On casual examination, the random-number stream looks good, but i plot millions of values, you'll notice that the high numbers come up a little less frequently than the low numbers.

One possible fix for that is simple: Always take two of these 8-bit values, multiply them together, and then throw or upper 8 bits of the resultant 16-bit number. Such a procedure would largely eliminate the bias.

Bull Mountain doesn't work with 8-bit numbers: It works, as we said, with 256-bit numbers. And it doesn't just multi together—it does a more sophisticated cryptographic combination. But the basic idea is the same. You can think o conditioning step as concentrating whatever degree of randomness the two-inverter circuit can provide.

We really like to sleep well at night, so we've built in additional circuitry that tests to make sure that the concentrat machinery is always working with bit streams that aren't too biased. If it detects a biased output, we tag it as unher refine it to meet our standards. This way, only healthy pairs of bit streams are combined.

Guaranteed randomness isn't all that useful if the random values aren't produced and vetted quickly enough to mee demand. While the hardware's circuitry generates random numbers from thermal noise much more quickly than its predecessors, it's still not fast enough for some of today's computing requirements. To allow Bull Mountain to spev random numbers as quickly as software pseudorandom-number generators, and also maintain the high quality of th random numbers, we add yet another level of circuitry. It uses the 256-bit random numbers to seed a cryptographic secure pseudorandom-number generator that creates 128-bit numbers. From one 256-bit seed, the pseudorandomgenerator can spit out many pseudorandom numbers. With those seeds coming at a rate of 3 gigahertz, a healthy ± these secret codes quickly builds up.

A new instruction, called RdRand, provides a way for software that needs random numbers to request them from tl hardware that's producing them. Built for Intel's 64-bit microprocessors, RdRand is the key to using Bull Mountain. retrieves a 16-, 32-, or 64-bit random value and makes it available in a software-accessible register. The RdRand instruction has been public for about a year, and the first Intel processor to provide it is known as Ivy Bridge. The I set performs 37 percent faster than its predecessor, and its smallest features have been reduced from roughly 32 nanometers to about 22. The overall increase in efficiency easily accommodates the demands of our random-numl generator.

While lava lamps might look cool, they don't go with every decor. We think our approach to random-number gen on the other hand, will be the most versatile of its kind, able to work across the range of Intel silicon products.

As we mentioned, the timing of keystrokes has provided a convenient source of randomness for seeding random-r generators in the past. So have mouse movements and even the time it takes a disk drive to find some of the infor

#### Behind Intel's New Random-Number Generator - IEEE Spectrum

stored on it. But such events don't always give you enough random bits, and to some extent the bits you get are predictable. Worse, because we now live in a world of servers with <u>solid-state drives</u> and virtualized workloads, th sources of physical randomness just aren't available to a lot of computers. Those machines need to be able to get numbers from something other than random events taking place on the periphery. Bull Mountain provides a solution

So if you're a programmer, get ready for a prolific source of randomness to be put at your fingertips. And even if you want to part with a pseudorandom-number generator you've grown to love—whether for cryptography, scientific co even gaming—you'll now have Bull Mountain to produce the seeds for it. We expect to see those seeds scattered with all sorts of wonderful software growing and blossoming as a result.

This article originally appeared in print as "Digital Randomness."

### About the Author

Greg Taylor and George Cox have over half a century of combined experience as Intel architects. Helping to devel hardware random-number generator was their first collaboration. Cox believes this work could make a fundamenta computing problem—the need for a dependable source of truly random numbers—"just go away." Taylor has helpe 10 generations of microprocessors and traces his interest in digital logic to his father, who worked on the first com computer produced in the United States.

Recommend

519 people