The Value of Natural Randomness (version 1)

Don Davis (May 25, 2017)

Natural Randomness now is the gold standard for random-number generation. Before the Internet came along, the computing industry used randomness mainly for numerical simulation problems, and for this purpose, a predictable PRNG was not only acceptable, but was preferable. With the Internet came e-commerce and networked gaming, both of which require an unpredictable randomness source.

There are two ways to meet the Internet's need for unpredictable random numbers:

- Secure pseudo-random number generators (PRNGs), and
- True random number generators (TRNGs).

These two types of RNG differ in cost and in how reliable are their security properties. Further, a secure PRNG needs to be initialized with a truly-random seed, and ideally, the secure PRNG should be reseeded repeatedly. For this reason, many customers deploy a mix of software and hardware RNGs, using TRNG hardware chiefly just to reseed software PRNGs.

A *secure P*RNG is a software program that will generate unpredictable random numbers on request. Most secure PRNGs can produce random numbers at very high data rates, nearly 1 gigabit per second, and a few PRNGs can go even faster. But, this speed comes at the cost of ad-hoc security guarantees: the PRNGs' algorithms are *designed* so that it's intractable to compute the PRNG's next random numbers, even if you know a lot of other numbers the PRNG has produced. But, *designed* to be secure and *known* to be secure are very different.

Most of the fast secure PRNGs are built around cryptographic hash algorithms, like SHA-1 or SHA-256. These hash functions have tended to have short lifetimes of 6-10 years, from initial publication to the utter ruin of cryptanalysis. Other fast PRNGs have been built around ciphers, like NIST's standard AES, but again, these ciphers were only *designed* to be hard to break; they're not *proven* to be secure. Only two PRNGs have been *proven* to be secure. Both are appallingly slow (by 2000x or more), and even so, their security proofs must be qualified: these algorithms were "proven" to be hard to break, *assuming* that factoring larger numbers is hard, or else *assuming* that the discrete-logarithm problem is hard. There's no proof, though, that those problems in turn, are actually intractable.

So in sum, secure PRNGs have value in many applications, but each PRNG's security is clouded by the residual threat that the PRNG's old output bits will become vulnerable to exposure, if some researcher discovers a weakness in the

algorithm. This is not a theoretical worry: several hash functions and ciphers have been cryptanalysed and thus abandoned, even though earlier, they were officially-standardized, respected, and relied-upon as secure.

In contrast, a TRNG's unpredictability is inherent in some well-understood physical process, so its security is more reliable. True random number generators rely on a variety of physical processes to get unpredictable results, for example:

- Radioactive decay (Random.org),
- Thermal noise (Intel's on-chip Secure Key RNG),
- Quantum fluctuations (qrng.anu.edu.au)
- Air turbulence inside hard-drives (Unix' /dev/random).

This physical unpredictability is why a TRNG is the gold standard for secure networking.

The advantage of these naturally-random sources is that even if an attacker were able to know all of the system's internal state variables with high precision, this still wouldn't help him to predict the TRNG's next random number. This unpredictability is guaranteed for some TRNGs by the mathematics underlying quantum mechanics; for others, the mathematics of nonlinear dynamics is equally airtight. In both areas of physics, the unpredictability is not only empirically verified, but is also well-understood mathematically.

So in the end, the question persists: Why would I pay for natural randomness, when I can instead get my application's random numbers from a secure PRNG algorithm, for free? The answer is: It depends on what's at risk. If your application's traffic has low dollar-value, and if you don't store your end-users' personal information, and if your industry isn't subject to data-security regulations, then natural randomness probably isn't necessary.

In fact, you can probably get away with using a PRNG based on a NIST-approved hash or cipher, even if you do have high-value transactions, or even if you do store your customers' credit-card numbers, and even if you are subject to regulatory requirements and penalties. If someday a research team in China or Bulgaria announces that they've just cracked the NIST-approved algorithm on which your PRNG relies, then no regulator will blame you.

BUT: The market's opinion is another story. Your end-users' identities and assets still may be at risk, and your customers and competitors may not be as forgiving as the regulators. If your customers' trust in you is important, if your brand's integrity is important, then it's not a good long-term plan to trust in unproven & unprovable PRNG algorithms. That's when natural randomness is worthwhile.