



[<< Back to Article](#)COMMENTARY 

Security Matters

Commentary by Bruce Schneier  

# Did NSA Put a Secret Backdoor in New Encryption Standard?

Bruce Schneier 11.15.07 | 12:00 AM

Random numbers are critical for cryptography: for encryption keys, random authentication challenges, initialization vectors, nonces, key-agreement schemes, generating prime numbers and so on. Break the random-number generator, and most of the time you break the entire security system. Which is why you should worry about a new random-number standard that includes an algorithm that is slow, badly designed and just might contain a backdoor for the National Security Agency.

Generating random numbers isn't easy, and researchers have discovered lots of [problems and attacks](#) over the years. A recent [paper](#) found a flaw in the Windows 2000 random-number generator. Another [paper](#) found flaws in the Linux random-number generator. Back in 1996, an early version of SSL was [broken](#) because of flaws in its random-number generator. With John Kelsey and Niels Ferguson in 1999, I co-authored [Yarrow](#), a random-number generator based on [our own cryptanalysis work](#). I improved this design four years later -- and renamed it Fortuna -- in the book *Practical Cryptography*, which I co-authored with Ferguson.

The U.S. government released a new official standard for random-number generators this year, and it will likely be followed by software and hardware developers around the world. Called [NIST Special Publication 800-90](#)

(.pdf), the 130-page document contains four different approved techniques, called DRBGs, or "Deterministic Random Bit Generators." All four are based on existing cryptographic primitives. One is based on hash functions, one on [HMAC](#), one on block ciphers and one on elliptic curves. It's smart cryptographic design to use only a few well-trusted cryptographic primitives, so building a random-number generator out of existing parts is a good thing.

But one of those generators -- the one based on elliptic curves -- is not like the others. Called `Dual_EC_DRBG`, not only is it a mouthful to say, it's also three orders of magnitude slower than its peers. It's in the standard only because it's been championed by the NSA, which first proposed it years ago in a related standardization project at the American National Standards Institute.

The NSA has always been intimately involved in U.S. cryptography standards -- it is, after all, expert in making and breaking secret codes. So the agency's participation in the NIST (the U.S. Commerce Department's National Institute of Standards and Technology) standard is not sinister in itself. It's only when you look under the hood at the NSA's contribution that questions arise.

Problems with `Dual_EC_DRBG` were first [described](#) in early 2006. The math is complicated, but the general point is that the random numbers it produces have a small bias. The problem isn't large enough to make the algorithm unusable -- and Appendix E of the NIST standard describes an optional work-around to avoid the issue -- but it's cause for concern. Cryptographers are a conservative bunch: We don't like to use algorithms that have even a whiff of a problem.

But today there's an even bigger stink brewing around `Dual_EC_DRBG`. In an [informal presentation](#) (.pdf) at the CRYPTO 2007 conference in August, Dan Shumow and Niels Ferguson showed that the algorithm contains a weakness that can only be described a backdoor.

This is how it works: There are a bunch of constants -- fixed numbers -- in the standard used to define the

algorithm's elliptic curve. These constants are listed in Appendix A of the NIST publication, but nowhere is it explained where they came from.

What Shumow and Ferguson showed is that these numbers have a relationship with a second, secret set of numbers that can act as a kind of skeleton key. If you know the secret numbers, you can predict the output of the random-number generator after collecting just 32 bytes of its output. To put that in real terms, you only need to monitor one [TLS](#) internet encryption connection in order to crack the security of that protocol. If you know the secret numbers, you can completely break any instantiation of Dual\_EC\_DRBG.

The researchers don't know what the secret numbers are. But because of the way the algorithm works, the person who produced the constants might know; he had the mathematical opportunity to produce the constants and the secret numbers in tandem.

Of course, we have no way of knowing whether the NSA knows the secret numbers that break Dual\_EC-DRBG. We have no way of knowing whether an NSA employee working on his own came up with the constants -- and has the secret numbers. We don't know if someone from NIST, or someone in the ANSI working group, has them. Maybe nobody does.

We don't know where the constants came from in the first place. We only know that whoever came up with them could have the key to this backdoor. And we know there's no way for NIST -- or anyone else -- to prove otherwise.

This is scary stuff indeed.

Even if no one knows the secret numbers, the fact that the backdoor is present makes Dual\_EC\_DRBG very fragile. If someone were to solve just one instance of the algorithm's elliptic-curve problem, he would effectively have the keys to the kingdom. He could then use it for whatever nefarious purpose he wanted. Or he could publish his result, and render every implementation of the random-number generator completely insecure.

It's possible to implement Dual\_EC\_DRBG in such a way as to protect it against this backdoor, by generating new constants with another secure random-number generator and then publishing the seed. This method is even in the NIST document, in Appendix A. But the procedure is optional, and my guess is that most implementations of the Dual\_EC\_DRBG won't bother.

If this story leaves you confused, join the club. I don't understand why the NSA was so insistent about including Dual\_EC\_DRBG in the standard. It makes no sense as a trap door: It's public, and rather obvious. It makes no sense from an engineering perspective: It's too slow for anyone to willingly use it. And it makes no sense from a backwards-compatibility perspective: Swapping one random-number generator for another is easy.

My recommendation, if you're in need of a random-number generator, is not to use Dual\_EC\_DRBG under any circumstances. If you have to use something in SP 800-90, use CTR\_DRBG or Hash\_DRBG.

In the meantime, both NIST and the NSA have some explaining to do.

- - -

*Bruce Schneier is CTO of BT Counterpane and author of [Beyond Fear: Thinking Sensibly About Security in an Uncertain World](#).*