The cryptographic security of Dual_EC_DRBG

The Dual_EC_DRBG is the only randomizer in FIPS SP800-90 and X9.82 whose security is based on the difficulty of solving a known "hard" problem. That is, determining the internal state of Dual_EC from observed output is equivalent to solving the Elliptic Curve discrete log problem, for which no subexponential algorithms have been found, despite decades of effort. Even though the use of Dual_EC is considerably more expensive computationally than other DRBGs which appear in those standards, its higher level of assurance is worth the cost when secure random is needed for key establishment, authentication and the assurance of data integrity which a keyed hash can provide.

Assuring that the internal state of a DRBG cannot be determined from seeing some of its output is clearly the most important test a deterministic algorithm must pass. Failure to do this would completely compromise the security of cryptographic protocols which rely on the unpredictability of the unseen outputs. It is this feature which sets Dual_EC apart from the other DRBGs: no other DRBG in SP800-90 or X9.82 can relate the difficulty of determining its internal state from known outputs to a mathematically difficult problem.

However, there are other notions of "secure random" in the literature. In 1984 Blum and Micali introduced the notion of a *cryptographically secure random number generator*, defining it as one which passes the *next-bit test*: There exists no polynomial time algorithm which, given n bits of output from the generator, can predict the (n+1)st bit with probability **significantly greater**[1] than 1/2. The only deterministic RBG in SP800-90 and X9.82 which attempts to quantify its score on the *next-bit test* is the Dual_EC_DRBG.

The Dual_EC_DRBG in X9.82 and SP800-90 produces bits in blocks: an (unknowable, initially non-deterministic) x-coordinate 's' on an elliptic curve over GF(p) is used as a scalar multiplier to jump to another point sP on the curve. Its x-coordinate in turn jumps to another point $(sP)_x * Q$. That point's x-coordinate is truncated by removing the high-order 'd' bits, and the remaining bits are output as a block of random. The process then iterates using $s = (sP)_x$, effecting a random walk on the curve. The value 'd' defaults to 16 (17 for P521, to get a multiple of 8) but can be varied by the implementation as it sees fit. The default was chosen as a compromise between maximum efficiency (d=0) and maximum entropy (d=curvesize-1).

The next-bit issue is addressed in Appendix C of X9.82 (Appendix E of SP800-90). The discussion there focuses on what is by far the worst case of the *next-bit test*: One gets to observe all but one bit in a truncated x-coordinate; predict the missing bit. A formula for the entropy in a truncated x-coordinate is derived, from which one could compute the probability of predicting that 'next bit'.

[Note that if the 'next bit' in a *next-bit test* occurs in the **next block** of output, it's part of a different x-coordinate on the curve. That there is no information about this bit from the previous

[1] The definition does not attempt to assign a numerical value to this term.

x-coordinate can be inferred from the difficulty of the EC discrete log problem. So the only concern is about a 'next bit' in the same x-coordinate. ]

For the curve P256 the appendix gives a table of computed values of the entropy formula for 'd' between 0 and 16, the other P curves having essentially identical tables. The calculated entropy in each 240-bit block of Dual_EC output using P256 is 239.9999890 . This could be interpreted to say that, given 239 bits of a block, there are .0000110 bits of information about the 240-th bit. Said differently, more than 90,000 full blocks, or nearly 22 million bits, would have to be seen in order to get *1 bit* of information to distinguish Dual_EC output from a true random source. If that's not sufficient for an application the definition of Dual_EC allows 'd' to be increased, and the formula can be used to set the truncation parameter to whatever level of indistinguishability might deemed to be needed. No other DRBG in SP800-90 or X9.82 ever addresses this issue.

Before resetting 'd' one might ask: "Should I be concerned about the .0000110 'missing' bits of entropy in blocks of Dual_EC output?" Firstly, most applications choosing to use Dual_EC for key generation, key establishment and other cryptographic functions requiring secure random will **not** be hiding that fact. The USG in particular has expressed its intent to use Dual_EC for such applications and will do so **overtly**. Thus there will be no need to "distinguish" that Dual_EC is being used rather than some other DRBG.

More importantly, 1 'missing' bit of entropy in 22,000,000 bits does not give a cryptanalyst any meaningful advantage in guessing a secret key comprised of such bits. Certainly not a key of any reasonable size. Further, there have not been any monobit or polybit biases found in Dual_EC_DRBG output. It is this type of bias which "Bleichenbacher" type attacks make use of. [Granted, there *is* a tiny bias remaining in the truncated output blocks due to the modular arithmetic. The NIST primes used by Dual_EC reduce the modular bias to a negligible size, and the truncation only reduces that further. This too is addressed in the Dual_EC appendix. ]

For generating the secret keys, both ephemeral and static, needed for signing and key establishment; for symmetric-cipher session keys; to produce nonces that ensure liveliness and prevent derived-key collisions; for making random challenges in authentication protocols: Dual_EC_DRBG provides the high-security random needed for these and similar cryptographic functions.