ANNEX G: (Informative) DRBG Mechanism Security Properties

Comment [EBB1]: This doesn't seem to "tie in" to Section 8 of Part 1 as well as it should.\

G.1 Overview

The security properties of the three DRBG mechanisms specified in this Standard may be used to specify the assumptions about security of these DRBG mechanisms. A maximum number of output bits per Generate call, and a maximum number of Generate calls have been specified for each DRBG mechanism. These numbers allow a specification of the expected level of resistance to attacks that involve collecting large numbers of outputs and searching for internal collisions or the lack of collisions in the output.

Comment [EBB2]: Neither this sentence, nor the original seem to be right.

G.2 HMAC_DRBG

Claimed Security Properties. HMAC_DRBG, with an n-bit MAC, a maximum of 2^{48} Generate requests , and a maximum of $2^{16} \times n$ bits of output for each request, cannot be distinguished from random substantially more easily than guessing an n-bit secret key.

Using an Approved n-bit hash function in the HMAC construction, HMAC_DRBG provides n bits of security. The DRBG mechanism generates up to 2^{48} sequences of outputs, each of up to 2^{16} n-bit HMAC outputs. This provides a pool of 2^{64} output values. Distinguishing this set of 2^{64} output values from an ideal random sequence is no easier than guessing an n-bit HMAC key.

The best known distinguisher for this DRBG mechanism relies on the fact that each Generate request uses HMAC with a single key in the OFB-mode to generate up to 2^{16} blocks of output. In each 2^{16}-block output sequence, the probability of an internal collision (a repeated output value) is approximately 2^{31-n}. After 2^{48} such output sequences, the probability that of an internal collision is about 2^{79-n}. For SHA-1, n = 160, so there is about a 2^{-81} probability of such a collision. If a collision occurs, the outputs would repeat after the collision, making distinguishing the outputs from random would be very easy. Internal collisions between different Generate requests are not generally an issue, since each Generate request uses a new HMAC key. The probability of any two keys colliding when n = 160 is about 2^{-65}.

G.3 CTR_DRBG

Claimed Security Properties. Two block cipher algorithms are currently Approved for use with CTR_DRBG: AES and three-key TDEA.

- CTR_DRBG using AES, with a k-bit key, a maximum of 2^{48} generate requests and a maximum of 2^{19} bits of output for each request, cannot be distinguished from random substantially more easily than guessing a k-bit AES key.
- CTR_DRBG using three-key TDEA, with a maximum of 2^{32} generate requests and a maximum of 2^{13} bits of output for each request, cannot be distinguished from random substantially more easily than guessing a 112-bit random key[1].

CTR_DRBG, using an n-bit block cipher with a k-bit key and an s-bit security strength, provides s bits of security.

- Using AES, CTR_DRBG has a 128-bit block and a key length (equivalently, a maximum security strength) of 128, 192, or 256 bits. The DRBG mechanism generates up to 2^{48} sequences of outputs of at most 2^{12} 128-bit blocks per sequence. This provides a total of 2^{64} 128-bit outputs that may be generated from a single DRBG instantiation. Distinguishing the full sequence of outputs from an ideal random sequence is not significantly easier than guessing the k-bit AES key.

Comment [EBB3]: Note that there is not an equivalent statement for TDEA in the next bullet item.

[1] Three-key TDEA uses a 192-bit key; this is equivalent to a 112-bit random key because of shortcut attacks on three-key TDEA.

- Using three-key TDEA, CTR_DRBG has a 64-bit block, a key length of 168 bits, and a security strength of 112 bits. The DRBG mechanism generates up to 2^{32} output sequences of at most 2^{7} 64-bit blocks per sequence.

The best known distinguishing attacks that do not use any properties of the block ciphers are based on the fact that in counter mode, an *n*-bit block never repeats within the same Generate call. This provides a property by which an ideal random sequence might be distinguished from a CTR_DRBG sequence.

- AES: Consider a random sequence of 2^{12} 128-bit values (i.e., 2^{19} bits of output for the sequence). The probability of a collision (i.e., a pair of 128-bit blocks that is repeated) is about $2^{19-128} = 2^{-109}$. After 2^{48} such values, the probability is about 2^{-61} that a sequence of 2^{48} blocks, each consisting of 2^{12} 128-bit values, would contain at least one collision. This provides a distinguisher with an advantage of 2^{-68} against CTR_DRBG with AES.
- Three-key TDEA: Consider a random sequence of 2^{7} 64-bit blocks (i.e., 2^{13} bits of output for the sequence). The probability of a collision (i.e., a pair of 64-bit blocks that is repeated) is about $2^{13-64} = 2^{-51}$. After 2^{32} such outputs, the probability is about 2^{-19} that a random sequence of 2^{32} blocks of 2^{7} 64-bit values each would include at least one such collision. This provides a distinguisher with an advantage of 2^{-20} against CTR_DRBG with TDEA.

In both cases, consider the possibility of cycling. Each generate call uses a new key. Assuming that each key is random, the probability of a collision on the key for CTR_DRBG with AES is expected to be about 2^{-33} for ~~AES with~~ 128-bit keys, 2^{-97} for 192-bit keys, and 2^{-129} for 256-bit keys. For three-key TDEA, the probability of a key collision in any of the 2^{32} Generate calls allowed for a single DRBG instance is about 2^{-114}.

If two Generate calls have the same key, this leads to a potentially detectable situation if and only if the counter values used in the two Generate calls overlap. For AES, this occurs with a probability of 2^{-116} for a 128-bit key; for three-key TDEA, the probability is 2^{-57}. Thus, a total probability of detectable key collisions for the maximum size and number of Generate requests per DRBG instantiation using three-key TDEA is 2^{-171}, and for AES is 2^{-149} with 128-bit keys.

Comment [EBB4]: This was 23, but I think it needs to be 19. If not, I'll change it back, but where does the 23 come from?

Comment [EBB5]: This was 57; I changed it to 61.

Comment [EBB6]: This was 32, but I think it's supposed to be 48.

Comment [EBB7]: What should this be, and how do you compute it?

Comment [EBB8]: Against what? distinguishing the output from random?

Comment [EBB9]: How do you get the 20?

Comment [EBB10]: See above comment.

Comment [EBB11]: How are these numbers arrived at?

Comment [EBB12]: How do you get these numbers?

```
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**[2] 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 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

---

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

.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.