X9.82, RBG, Issues for the Workshop

1. **DRBGs:**
   a) How many DRBGs of each type should we include?
   b) In these documents, we have adopted a security parameter of 2^{64} for three related ideas:
      1) We assume a maximum of 2^{64} of any operation (generating output bits, reseeding, instantiating) by a single entity or application that is not actively participating in an attack. Thus, we can assume that no consuming application will ever be attacked based on requesting more than 2^{64} bits from an RBG, or reseeding more than 2^{64} times.
      2) We assume that the largest set of bits we care about for distinguishing attacks is 2^{64} bits in length, regardless of the application or the source.
      3). We assume that concerns about internal state collisions between DRBG instances never include more than 2^{64} of these instances.

      Why Do We Need a Bound? Some very general attacks on RBGs require knowledge of the maximum number of operations or outputs to be considered.

      Is 2^{64} the right boundary? A natural value for a bound would seem to be the security level--if we claim a 128-bit security level, we expect up to 2^{128} outputs from the DRBG to resist attack. This doesn't work so well in practice, however, for two reasons:

      1) This disallows obviously-correct ways of building DRBGs using block cipher algorithms.
      2) More generally, the number of computations an attacker is willing and able to carry out has no particular connection with the number a legitimate user is willing and able to carry out. So setting the parameter this way doesn't track with the real nature of the situation.

      Similar comments apply to the maximum number of potentially colliding RBG instances we care about, or the maximum number of RBG outputs we are about distinguishing from random at one time. These parameters have no natural relationship to the amount of work an attacker might do, and setting them to the security level leads to unreasonable results.
   c) For the MS_DRBG: Should sample moduli be provided in the Standard (e.g., for testing)? Should the factorization information be included with the test vectors?
2. **NRBGs:** What lies between a very Basic NRBG and an Enhanced NRBG? X9.82 is allowing NRBGs that don't default to an approved DRBG (which we refer to as Basic NRBGs). Obviously, there is a wide range of possibilities here, from a conditioned entropy source with health tests to an NRBG that defaults to a DRBG-like thing that is not an approved DRBG from Part 3. This flexibility may cause problems for validation.

a) What should we insist on for any NRBG? Is it reasonable, for instance, to insist that every NRBG maintain an internal state to guard against fluctuations in entropy production (or in undetected fluctuations)?

b) Should a really Basic NRBG (one with no state, or a state that is not protected cryptographically) have to provide more stringent, continuous health tests?

c) How can validation make up for the perceived gap in assurance between Basic and Enhanced NRBGs?

3. **Testing (general):**

a) What does power-up mean for a smart card? Is power up when the smart card is programmed by the manufacturer? When any initial values are inserted? When the smart card is used in a vendor's machine? When the owner uses it to access a facility?

b) What should be done when a catastrophic failure occurs (e.g., the health tests fail)? When should a retry be allowed automatically? When should a complete shut down occur in order to service the problem?

4. **DRBG Testing:**

a) Is the approach in Section9.9 reasonable for known answer testing?

b) How many strings should be requested and of what lengths?

c) Should the error handling be tested? Alternatively, should a signature or MAC on the code be checked?

5. **NRBG Testing:** An NRBG is responsible for assessing the amount of entropy that it is providing and, in many cases, for providing full entropy output. Unlike a DRBG, it must have knowledge of the entropy source so it can determine whether the source is operating as expected. This will probably involve a great deal of work (at design time) modeling the source, followed by constant health testing at runtime. However, we must assume that runtime testing resources are limited, so any tests performed must be as "lean and mean" as possible. With this in mind, here are some issues to consider:

a) Is it reasonable (especially in the Enhanced case) to omit the standard statistical tests on output (perhaps running a few simple tests) and concentrate resources closer to the entropy source?

b) Where should testing be performed? On digitized entropy input? On conditioned entropy?

c) What tests are appropriate for a low rate entropy source? What is the most efficient way to estimate min-entropy (presumably we just need to get the best estimate of the most likely output)? Should we be testing for independence as well?

d) Given a good statistical model for the entropy source (and perhaps a good understanding of failure modes and their statistical model), what can we do to improve the power of our health tests?

e) For a Basic NRBG, how much more work is needed to assure that the entropy source is working properly?

f) What entropy assessment methods are appropriate? It seems appropriate to require the validator to be able to instrument the source (i.e. slowly vary the parameters, such as temperature, voltage, whatever) so they can see how performance is affected? Also, the validator should exercise the health tests by hooking up an artificial entropy source that can be finely controlled by the validator to see what conditions the health tests can detect.