Selected text from Section 9:

9.2 DRBG Boundary

A DRBG **shall** be implemented within a DRBG boundary; the state (see Section 9.3) and any other inputs **shall not** be output, but **shall** exist solely within the DRBG boundary. When implemented within a FIPS 140-2 cryptographic module, the DRBG boundary **shall** either be identical to the cryptographic boundary or the DRBG **shall** be fully contained within the cryptographic boundary.

When the DRBG boundary does not coincide with the cryptographic boundary, a basic DRBG component within the cryptographic boundary (e.g., a hash function or block cipher algorithm) **shall not** be accessible for other purposes by a function outside the DRBG boundary. For example, an encryption function that is within the cryptographic boundary, but not within the DRBG boundary **shall not** use an algorithm that resides within the DRBG boundary for its encryption operation.

A basic cryptographic function within a DRBG boundary **may** be used by other functions within the DRBG boundary. However, the internal state of the DRBG **shall not** be used by functions other than the DRBG. For example, a hash function used by the DRBG may also be used for digital signature generation within the DRBG boundary as long as any information associated with the DRBG is not used or affected by the digital signature process.

Comment [ebb1]: Page: 39
Do we want to allow this ?

9.3 States

An initial state of a DRBG **shall** be generated prior to the generation of output by the DRBG. The state of a DRBG includes information that is acted upon, the strength of the generator and, optionally, keys used by the generator. The initial state includes:

1. One or more values that are derived from the seed(s); at least one of these derived values is updated during the operation of the DRBG (e.g., at least one component of the state is updated during each call to the DRBG),
2. Any keys required by the DRBG,
3. A transformation of the initial state; this information remains static until replaced by new values during reseeding,
4. Other information that is particular to a specific DRBG; this information may remain static or may be updated during the operation of the DRBG, and
5. The security strength provided by the DRBG if an implementation is designed to handle multiple levels of security.

At any given time after a DRBG has been initialized, a DRBG exists in a state that is defined by all prior input information. No portion of the DRBG state **shall** be accessible from outside the DRBG (i.e., all portions of the state **shall** be retained within the DRBG boundary) and **shall** be protected at least as well as the intended use of the output bits by the consuming application.

A DRBG **shall** transition between states on demand (i.e., when the generator is requested to provide new pseudorandom bits). A DRBG may also be implemented to transition in response to external events (e.g., system interrupts) or to transition continuously (e.g., whenever time is available to run the generator). Additional unpredictability is introduced when the generator transitions between states continuously or in response to external events. However, when the DRBG transitions from one state to another between requests, reseeding and rekeying may need to be performed more frequently.

Re Seeds (selection from Section 9.4):

1. Seed use: DRBGs may be used to generate both secret and public information. In either case, the seed **shall** be kept secret. A single instantiation of a DRBG **should not** be used to generate both secret and public values. Cost and risk factors must be taken into account when determining whether different instantiations for secret and public values can be accommodated.

   A seed that is used to initialize one instantiation of a DRBG **shall not** be intentially used as a seed for another instantiation of the DRBG.

2. Seed entropy: A seed **shall** have entropy that is equal to or greater than twice the required security strength *s* for the consuming application; if the application requires a security strength of *s*, then the required entropy is at least 2*s*. Table 1 identifies the five security strengths provided by Approved DRBGs, along with the associated entropy requirements. If a selected DRBG and the seed are not able to provide the required strength, then a different DRBG **shall** be used. If multiple seeds are required, then each individual seed **shall** have entropy equal to or greater than the required strength of the consuming application.

   Comment [ebb2]: Page: 41
   Need to check that this statement is true - see Section 10

   Table 1: Minimum Entropy and Seed Size

Bits of Security Strength	80	112	128	192	256
Minimum Entropy and Seed Size	160	224	256	384	512

3. Seed size: The minimum size of the seed depends on the selected DRBG, the security strengh required by the consuming application and the entropy source. The seed size **shall** be at least equal to the required entropy and may be larger, depending on the entropy source (see above discussion). For example, if 160 bits of entropy are required, the quality of the entropy source may necessitate a seed size of 240 bits or more to achieve the 160 bits of entropy. Table 1 lists the minimum seed size for a security strength and entropy requirement.

9.5 Keys

Some DRBGs require the use of one or more keys. When not explicitly prohibited, these keys may be provided from an external source (i.e., from a source outside the DRBG boundary), or the DRBG may be designed to generate keys from seed material. The use of externally provided keys may be appropriate, for example, in low risk applications with memory constraints (e.g., smart cards), when the generation of sufficient seed material for keys is impractical (e.g., the source of sufficient entropy is too costly), or the quality of the DRBG's entropy source is questionable, but high-quality keys can be obtained outside the DRBG boundary. A key and its use in a DRBG **shall** conform to the following:

1. Key use: Keys **shall** be used as specified in a specific DRBG. A DRBG requiring a key(s) **shall not** provide output until the key(s) is available.
2. Key entropy: The entropy for the combination of keys **shall** be at least twice the required security strength of the consuming application. For example, when 112 bits of security are required by an application, the key(s) shall have at least 224 bits of entropy.
3. Key size: Key sizes **shall** be selected to support the desired security strength of the consuming application (see SP 800-57).
4. Keys determined from a seed: A key determined from a seed **shall** be independent of the rest of the initial input determined by that seed. If multiple keys are used by a DRBG, as opposed to the same key used in multiple places, then each key **shall** be independent of all other keys.

   For DRBGs that determine a key from the same seed as an initial value and any other keys (i.e., a single seed is used to determine all initial inputs for the DRBG, including keys), the seed **shall** have entropy that is equal to or greater than twice the required strength of the consuming application.

   For DRBGs that use multiple seeds to determine a DRBG instance, each seed **shall** be used to determine a different part of the initial input (e.g., the initial value for the DRBG and each distinct key **shall** be determined from different seeds). The combination of all seeds**shall** have entropy that is equal to or greater than twice the required strength of the consuming application.

Comment [eb3]: Page: 43
I'm not sure that this has been accomplished by the 3BlockCipherDRBG.

5. Keys provided from an external source: Keys generated externally **shall** have full entropy (i.e., each bit of a key **shall** be independent of every other bit of the key) and **shall** be generated using an Approved NRBG or an Approved DRBG (or chain of DRBGs) that is seeded by an Approved NRBG. The keys **shall** be protected in accordance with [SP 800-57].

Comment [ebb4]: Page: 43
Do we want to require this ?

6. Rekeying: Rekeying (i.e., replacement of one key with a new key) is a means of recovering the secrecy of the output of the DRBG if a key becomes known. Periodic rekeying is a good countermeasure to the potential threat that the keys and DRBG output become compromised. However, the result from rekeying is only as good as the NRBG (or chain of DRBGs that is initiated by a NRBG) used to provide the new key. In some implementations (e.g., smartcards), an adequate rekeying process may not be possible, and rekeying may actually reduce security. In these cases, the best policy might be to replace the DRBG, obtaining a new key in the process (e.g., obtain a new smart card).

   Generating too many outputs using a given key may provide sufficient information for successfully predicting future outputs. Periodic rekeying will reduce security risks, reducing the likelihood of a compromise of the target data that is protected by cryptographic mechanisms that use the DRBG.

Keys **shall** have a specified finite keylife (i.e. a cryptoperiod). Keys **shall** be updated (i.e., replaced) periodically. Expired keys or keys that have been updated **shall** be destroyed (see Sp 800-57). If keys become known (e.g., the seeds are compromised), unauthorized entities may be able to determine the DRBG output.

7. Key separation: A key used within a DRBG **shall not** be used for any purpose other than random bit generation. Different instances of a DRBG **should** use different keys.

Comment [ebb5]: Page: 44
Do we want to allow this ? If the seed is supplied externally, the seed would still be different

9.7 Forward and Backward Secrecy

Comment [ebb6]: Page: 44
This section could be empty for now. However, the current text indicates a possible discussion (that may not be correct right now).

Each of the DRBGs in this Standard has been designed to provide forward and backward secrecy when observed from outside the DRBG boundary, given that the observer does not know the seed or any state values.

When observed from within the DRBG boundary, each of the DRBGs provides backward secrecy.

Comment [ebb7]: Page: 44
Is this true ?

Forward secrecy may be provided by the addition of user input. However, the degree of forward secrecy depends on the amount of entropy introduced by the user input. If the user input introduces entropy that is at least equal to the strength of the DRBG for every request for pseudorandom bits, then "full" forward secrecy is provided. This may be impractical for many applications. However, user input with even a small amount of entropy provides some degree of forward secrecy. It may be appropriate for an application to require user input with high entropy for critical applications (e.g., the generation of digital signature keys).