X9.82 Part 2 – Non-Deterministic Random Bit Generators

Foreword

Introduction

1. Scope

This part of ANS X9.82 defines techniques for the generation of random bits using non-deterministic generators. The main difference between an NRBG and the deterministic random bit generators from Part 3, is that an NRBG is responsible for accumulating and testing the entropy it samples from its environment, whereas the DRBG principally relies on the seed handed to it for its entropy. This part includes:

Comment [ebb1]: Page: 1
Actually, the DRBG is getting "entropy input", according to our model, possibly concatenating a personalization string, and then deriving the seed from that. Maybe change "seed" in the text to "bits"?

1. A model for a non-deterministic random bit generator,
2. Criteria and requirements for approving non-deterministic random bit generators,
3. Implementation issues, and
4. Tests and test guidance.

The precise structure, design and development of a non-deterministic random bit generator are outside the scope of this standard.

2. Conformance

An implementation of a non-deterministic random bit generator may claim conformance with ANS X9.82 if it implements the mandatory requirements of this part of the standard.

3. Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. Nevertheless, parties to agreements based on this document are encouraged to consider applying the most recent edition of the referenced documents indicated below. For undated references, the latest edition of the referenced document (including any amendments) applies.

ANS X9.82, Part 1-200x, Overview and Basic Principles

ANS X9.82, Part 3-200x, Deterministic Random Bit Generators

4. Terms and Definitions

For the purposes of Part 2, the following terms and definitions apply.

4.x

Algorithm

A clearly specified mathematical process for computation; a set of rules that, if followed, will give a prescribed result. [TR-1]

4.x

Approved

Approved in an ANSI X9 standard or the X9 registry or by a process specified in an ANSI X9 standard or technical guideline.

4.x

Backtracking Resistance

The assurance that previous values cannot be determined from the current value or subsequent values.

Biased

A bit string (or number) chosen from a sample space is said to be biased if one bit string (or number) is more likely than another bit string (or number) to be chosen. Contrast with unbiased.

4.

Birthday Phenomenon

Sometimes referred to as the birthday paradox, the birthday phenomenon is the (perhaps surprising) result that it takes about 23 people in a group to have over a 50% chance of 2 people having matching birthdays (disregarding the year). More generally, when elements are randomly selected from a set, once a subset about the size of the square root of the original set is selected, it becomes probable that two elements in the subset will have matching values.

4.x

Bit String

A bit string is an ordered sequence of 0's and 1's. The leftmost bit is the most significant bit in the string and is the newest bit generated. The rightmost bit is the least significant bit of the string. [TR-1]

4.x

Black Box

An idealized machine that accepts inputs and produces outputs but is designed such that an observer cannot see inside or determine exactly what is going on inside. Contrast with a glass box.

4.x

Consuming Application

An application that uses a random number generator.

4.x

Cryptographic Key (Key)

A parameter that determines the operation of a cryptographic function such as :

1. the transformation from plain text to cipher text and vice versa,
2. the synchronized generation of keying material,
3. a digital signature computation or validation. [TR-1, except for the highlighted word]

4.x

Cryptography

The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification, prevent its unauthorized use or a combination thereof.

4.

Cryptographically-strong Random Number Generator

A random number generator is said to be cryptographically strong when it has an assessed strength against an attack by an adversary that provides an appropriate level of security. At the time this standard was written this was at least 80 bits of security, but this is expected to increase over time.

4.x

Cycle

A single complete execution of a periodically repeated phenomenon; a periodically repeated sequence of events. [American Heritage Dictionary]

4.x

Deterministic Algorithm

An algorithm that given the same inputs always produces the same outputs.

4.x

Deterministic Random Bit Generator (Deterministic RBG) (DRBG) (alternate definition for consideration)

A deterministic algorithm for producing a random-appearing sequence of bits from an initial value called a *seed.*

4.x

Digital Signature

A cryptographic transformation of data, which, when associated with a data unit, may provide the services of :

(a) Origin authentication,

(b) Data integrity, and

(c) Signer non-repudiation. [TR-1]

4.x

Entropy

A measure of the disorder, randomness, or variability in a closed system. The entropy of X is a mathematical measure of the amount of information provided by an observation of X. Entropy is the uncertainty about the outcome before an observation of X [HAC]. Entropy is the total amount of information yielded by a set of bits. It is representative of the work effort required for a computationally unbounded adversary to be able to reproduce the same set of bits.

4.x

Entropy Source

A source of unpredictable (that is, random) information, such as thermal noise or hard drive seek times.

4.x

Glass Box

An idealized machine that accepts inputs and produces outputs and is designed such than an observer can see inside and determine exactly what is going on inside. Contrast with a black box.

4.x

Hash Function

A (mathematical) function that maps values from a large (possibly very large) domain into a smaller range. For the purposes of this standard, a hash function satisfies the following properties:

1. (One-way) It is computationally infeasible to find any input that maps to any pre-specified output;

2. (Collision free) It is computationally infeasible to find any two distinct inputs that map to the same output. [TR-1: definition of cryptographic hash function, rather than hash function.]

4.x Indistinguishable from Random (TBD)

4.x

Implementation Validation Testing

Testing by an independent party to ensure that an implementation of a standard conforms to the specifications of that standard.

4.x

Initialization Vector (IV)

A number used as a starting point for the encryption of a data sequence in order to increase security by introducing additional cryptographic variance and to synchronize cryptographic equipment. [TR-1]

4.x

Instance of a Deterministic RBG

An instance of a deterministic RBG is an individual chain of internal states defined by the specific deterministic RBG technique that is used, the algorithm(s) on which it is based, the seed, and any other algorithm inputs.

Internal State

The collection of stored information inside an instantiation of an RBG. This can include both secret and non-secret information.

4.x

Key

See Cryptographic Key. [TR-1]

4.x

Key Establishment

A procedure that results in shared keying material among different parties.

4.x

Keying Material

The data (e.g., keys, certificates, and initialization vectors) necessary to establish and maintain cryptographic keying relationships. [TR-1]

4.x

Min-entropy

A bitstring *X* has min-entropy *k* if for every x, $\Pr[X = x] \le 2^{-k}$, where **Pr** is the probability function. That is, *X* contains *k* bits of min-entropy or randomness. Informally, min-entropy is the "best" kind of entropy and measures the entropy in the worst case.

4.

Negligible Probability

A probability of something happening that can be ignored for all practical purposes as the chance is so small. For example, the chance of an adversary making an outright correct guess of a 128-bit AES key has a chance of success of $1/2^{128}$ and so can be ignored.

4.x

Non-Deterministic Random Bit Generator (Non-deterministic RBG) (NRBG)

Produces output that is dependent on some unpredictable physical source that produces entropy. Other names for non-deterministic RBGs are True Random Number (or Bit) Generators and, simply, Random Number (or Bit) Generators.

4.x

One-Time Pad

A key that is produced by a (true) random process and distributed by hand. The key is exclusive-OR'ed with the plain text to produce the ciphertext.

4.x

Operational Testing

Testing within an implementation immediately prior to or during normal operation to determine that the implementation continues to perform as implemented and optionally validated.

4.x

Prediction Resistance

The assurance that subsequent (future) values cannot be determined from the current or previous values.

4.x

Pseudorandom

A sequence of bits or a number that appears to be selected at random even though the selection process is done by a deterministic algorithm. [TR-1, except for the alteration highlighted and the term is not hyphenated.]

4.x

Public Key

In an asymmetric (public) key cryptosystem, that key of an entity's key pair which is publicly known. [TR-1]

4.x

Random

A value in a set that has an equal probability of being selected from the total population of possibilities and hence is unpredictable. [TR-1 except for "is"]

4.x

Random Bit Generator (RBG)

A device or algorithm that outputs a sequence of binary bits that appears to be statistically independent and unbiased.

4.x

Random Number Generator (RNG)

A device or algorithm that uses a random bit generator and a conversion mechanism to produce a random number.

4.

Security Level

A specific security level x (in bits) is associated with a cryptographic function or key if it is expected to take about 2^x operations to break the function or key using the known best attack.

4.x

Seed

A string of bits that is used as input to a Deterministic Random Bit Generator (DRBG). It will determine a portion of the internal state of the DRBG and its entropy needs to be sufficient to support the security strength of the DRBG. (Note : Would like to include connection to NRBG here, such as 'The preferred source of the seed is an approved NRBG' or some such wording)

4.x

Sequence

An ordered set of quantities. [American Heritage Dictionary - needs work]

4.x

Statistically Unique

Two definitions are given, the first is the most intuitive and the second is the most detailed and precise.

1. When a random value is required to be statistically unique, it may be selected either with or without replacement from the sample space of possibilities. That is, it is allowed (but not required) to exclude previously selected values.

Note: To say a random value is statistically unique means that it is not a requirement that a series of such values with a size of about the square root of the size of the set of possibilities are expected to have a repeated value; that is, it is not considered an attack concern for an adversary to be able to distinguish from random selection in this case.

If selected at random with replacement and it is important for the value to be unique in a set of such selected values (that is, there is at most a negligible chance of a repeat among the set), then the set of values will need to have a size much less than the square root of the size of the sample space, due to the birthday phenomenon.

2. The precise calculation for the probability of two *n*-bit quantities repeating that were randomly selected with replacement which is equal to: [1-0/(2^n)) (1 - 1/(2^n)) (1-2/(2^n)) (1-3/(2^n)) ... (1-(L-1)/(2^n)] where *L* is the number of elements selected. When L is much smaller than 2^n, an approximate formula can be used which is Pr[at least one collision | L, n] ~= 1 - e^{- ((L)(L-1))/(2^{n+1}). The probability of two *n*-bit quantities repeating that were randomly selected without replacement is equal to 0. Therefore, when a value is required to be statistically unique, it may be selected either with or without replacement. [New]

4.x

Stationary Entropy Source

An entropy source that consistently obeys a single statistical model over time. For example, a biased coin for which the probability of a head is a constant *p* for each toss, where $0<p<1$, could serve as a stationary entropy source.

4.x

String

See Bit String.

4.x

Target Data

The data that is to be protected using the random bits generated by the RBG. Target data includes plaintext data that is to be encrypted using a randomly generated key and information that is to be signed using a public key pair that is generated using the random bits.

4.x

Type 1 Error

The conclusion that a statistical hypothesis being tested is false, when in fact it is true.

4.

Unbiased

A bit string (or number) chosen from a sample space is said to be unbiased if all potential bit strings (or numbers) have the same possibility of being chosen. Contrast with biased.

4.x

Unpredictable

In the context of random bit generation, an output bit is unpredictable if an adversary has only a negligible advantage (that is, essentially not much better than chance) in predicting it correctly.

5. Symbols and Abbreviated terms

Symbols and Abbreviations	Meaning
ANS	American National Standard
ANSI	American National Standards Institute
DRBG	Deterministic Random Bit Generator
FIPS	Federal Information Processing Standard
NRBG	Non-deterministic Random Bit Generator
RBG	Random Bit Generator
RNG	Random Number Generator

6. General Discussion

6.1 Overview

Part 1 of this standard described several cryptographic applications for random values, the characteristics that these random values should have, and some mathematical and cryptographic background information on the concept of randomness. It introduced a general model for random bit generation, which could be implemented either as a deterministic or non-deterministic random bit generator (referred to in this standard as DRBG and NRBG, respectively). Part 1 described the properties that a random bit stream must possess and derived from them the properties that a random bit generator must have and requirements that it must meet to produce such a stream, regardless of the actual mechanisms used.

While an NRBG is capable of generating random bits to be used directly by a cryptographic application, it will generally not produce bits as quickly as a DRBG based on similar technology would. In many cases it will be more efficient to use the NRBG to produce random bits for a seed that will be used by a DRBG to generate bits for the application. Thus, one important use for an NRBG that satisfies this Standard is seeding one or more DRBGs. One way of distinguishing this part of the Standard from Part 3 is that this part of the standard is much more concerned with gathering entropy from the environment, whereas part 3 focuses on algorithms for taking a seed and producing pseudorandom values for cryptographic applications.

Comment [ebb2]: Page: 1
The outputs from Part 2 could also be used for cryptographic applications.

The remainder of this document will present the NRBG model and develop requirements in the following manner. Section 6.2 will take the model from Part 1 and particularize it for the case of an NRBG. It will describe the components that comprise an NRBG and briefly discuss what each component are meant to achieve. It will also motivate the properties which the output of an NRBG is meant to have. Section 7 will discuss requirements, first on the output of the NRBG and then, following naturally from the output requirements, on the components of the NRBG itself. Following the requirements, Section 8 will discuss two paths to achieving a secure NRBG under this standard. Section 8 will also lay out requirements for these two approaches. Section 9 will be a thorough discussion of entropy, including some examples of entropy sources. Section 10 will discuss validation and other issues that NRBG designers and evaluators must consider to achieve a higher assurance.

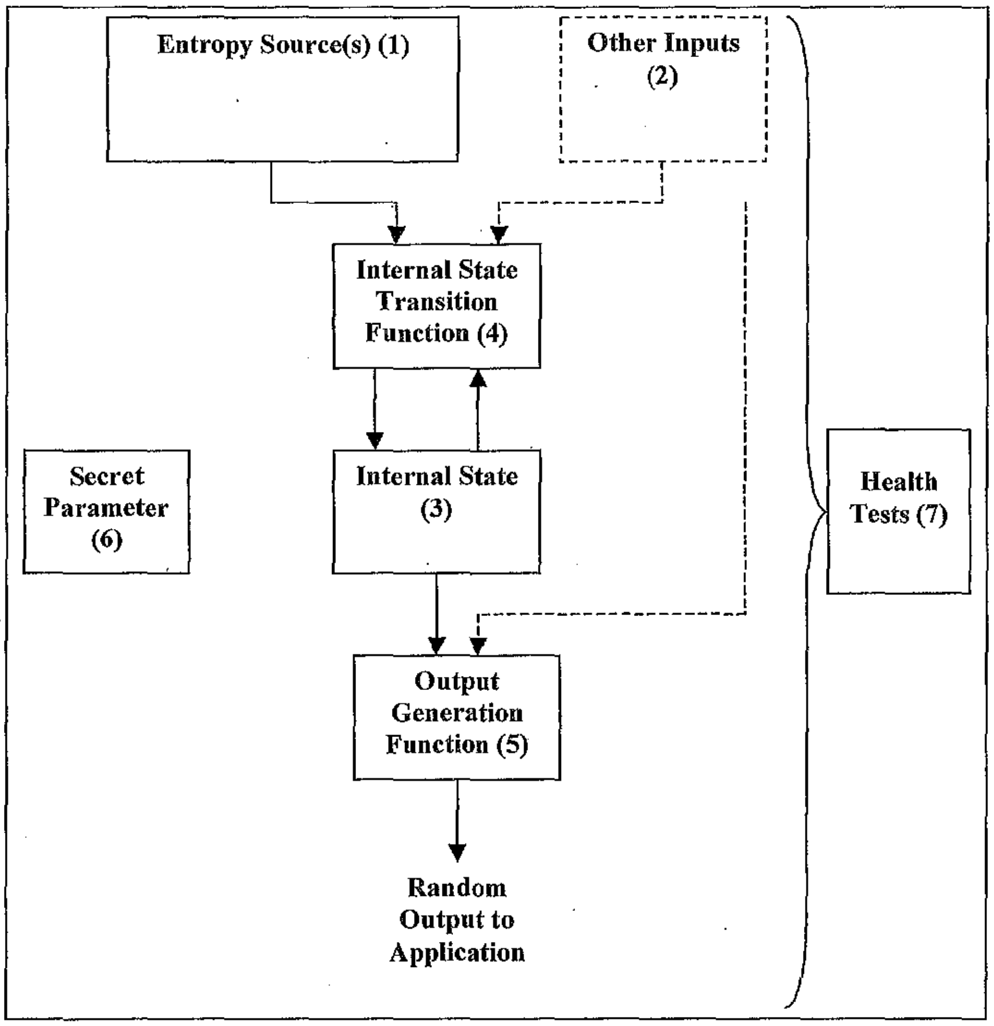
6.2 Functional Model for an NRBG

Figure 1 shows a functional block diagram illustrating a conceptual NRBG derived from the model in Part 1. In this diagram, dashed lines indicate a component that is optional, depending on various factors. These components (and their required properties) are intended to prevent various security weaknesses associated with random bit generation that have been known to occur in cryptographic applications and environments. Typically, each component will be required in an NRBG.

There may be applications where the threat that a component is designed to counter is either absent or is handled procedurally. If this can be adequately justified and documented, that component may be omitted from the NRBG. Note that this decision introduces an extra burden on the NRBG designer/implementer to determine the exposure of his product to various avenues of attack and to carefully justify the reasoning behind his decision (see in particular the discussion of the Primitive NRBG in Section 8.1). The following discussion assumes that each component is present and is providing the security functions it was intended to provide.

In brief, sampled noise from the entropy source is combined with the internal state using the state transition function. The NRBG monitors the accumulation of entropy, and when there is sufficient entropy (and a call is made for random bits) the output generation function maps the internal state to a random output. Generally, more entropy will need to be sampled before more output can be generated (this depends on the specifics of the design).

A secure NRBG will also include mechanisms designed to increase the likelihood of continued secure operation in the event of failures or compromises. Detectable failures are addressed through the inclusion of periodic health tests on the various components. Undetectable failures are addressed by the inclusion of a safety margin in the maintenance of entropy during NRBG operation, so that decreases in available input entropy due to unexpected events or statistical model inaccuracies are less likely to result in biased random output. The Secret Parameter offers further protection as additional entropy in the system.


Entropy Source(s) (1)
Other Inputs (2)
Internal State Transition Function (4)
Secret Parameter (6)
Internal State (3)
Health Tests (7)
Output Generation Function (5)
Random Output to Application

An objective for an NRBG meeting this Standard will be for the NRBG to continue to operate in a manner no less secure than an approved DRBG in the event that the entropy source completely fails. When certain components of the model are missing, additional assurances in the form of highly reliable entropy sources and more stringent health tests must be present.

6.2.1 Description of Individual Components

6.2.1.1 Entropy Source(s)

In an NRBG, unpredictability is based on the use of one or more sources of entropy. Such a source can be a physical component (electrical, electronic, or mechanical) or it could be a more nebulous source having no actual physical representation, but which produces usable information as a result of the interaction among two or more components or processes. An example of the a physical component is a noisy diode, which receives a constant input voltage level and outputs a continuous, normally distributed analog voltage level. An example of a more nebulous source is process timing interactions, such as the sampling of a high-speed counter whenever a human operator presses a key on a keyboard.

In general, samples of the entropy source will not be adequate for direct use as random output, because entropy sources exhibit statistical biases. This shortcoming is remedied in an NRBG by processing the entropy source output with an internal state transition function.

The use of multiple entropy sources, rather than a single entropy source, may be useful for a variety of reasons. Using multiple entropy sources can provide a layer of protection against degradation of the NRBG output due to the failure of one or more (but not all) of the entropy sources or one or more (but not all) of the entropy sources straying from the characterized statistical model. In situations where some of the entropy sources have some degree of external visibility, additional entropy sources that are less externally accessible will lessen the usefulness of knowledge of the more visible entropy sources to the adversary.

In some cases, the multiple entropy sources may be multiple simple entropy sources. This might be true in hardware (e.g. a bank of ring oscillators) where a simple source is replicated many times or in software (e.g. sampling various statistics/events in an operating system) where diverse variables, each with a small amount of entropy, are sampled. Reasons for going with multiple entropy sources such as these include increased rate of entropy accumulation and greater resistance to an entropy source failure.

Comment [ebb3]: Page: 1
Tried to reconcile multiple entropy sources and composite entropy sources.

Finally, having multiple entropy sources can provide the capability for split control, enabling applications where multiple users require access to the same NRBG output but distrust each other's potential influence over the individual entropy sources. For such applications, it is possible to design the NRBG so that a user's trust in a single entropy source is sufficient for trust in the final NRBG output. See the Assurance Section (#?) for further guidance (Ed Note: Should we move this? It seems different from the others (more of a niche application)).

6.2.1.2 Other Inputs

The operation of an NRBG may require various additional inputs other than the non-deterministic entropy source. Section 6.6.3 of Part 1 describes several such inputs, including commands and power, and optional time variant data such as counters, clocks, or user-supplied data.

6.2.1.3 Internal State

The internal state is the memory of the NRBG; it contains information about the bits produced by the entropy source(es), as well as other information used by the NRBG, such as counters and secret parameters. Essentially, the bits from the entropy source(es) accumulate in the internal state with whatever entropy is contained in those bits. For some entropy sources, this ~~function~~ is critical, as the entropy from the entropy source may be sparse. Even when the entropy source is outputting bits with high entropy, the internal state carries information between calls to the NRBG. By retaining this state information, the NRBG can produce random output as a function of not only the current input from the entropy source, but also several (or all) previous entropy source inputs. This provides a layer of protection against entropy source failure or degradation, and the compromise of the random output by an adversary who has knowledge of or influence on the entropy source. The current internal state is processed in combination with any new entropy source data by the internal state transition function to produce the next internal state. Thus, the internal state may consist of a "pool" of bits, in addition to any optional counters or other values.

Comment [ebb4]: Page: 1
The internal state is not a function.

Comment [ebb5]: Page: 1
define

6.2.1.4 Internal State Transition Function

This component replaces the value of the internal state by processing the current value of the internal state, any new inputs from the entropy sources, and optional additional inputs. This allows the NRBG to distill random output from entropy sources, while carrying forward the influence of previous entropy source inputs. As the part of the internal state is meant to be a pool of unbiased random bits, it is the responsibility of the state transition function to produce unbiased bits even though some inputs, such as the entropy source, may have a serious bias.

6.2.1.5 Output Generation Function

This component provides random output to the requesting application by processing all or a subset of the bits in the current internal state, along with any optional additional inputs. The purpose of this component is to provide backtracking resistance (in the event that the entropy source fails) by preventing the random output from revealing information about the previous or current values of the internal state, entropy source inputs, or random outputs. Because of the output generation function, the current internal state can make a meaningful contribution to the entropy of the next internal state.

6.2.1.6 Secret Parameter

The secret parameter is an additional input to the NRBG at initialization. The secret parameter can be thought of as an additional entropy source for the NRBG. As such, it serves as an additional layer of protection against a degraded or compromised entropy source. Depending on the handling of the secret parameter, it could also protect against a compromised internal state. There are many possible uses for the secret parameter. It may be used to customize the internal state transition function (in which case, a particular secret parameter will cause the internal state to follow a different path, given a sequence of entropy inputs, than it would have under a different secret parameter). The secret parameter could also be used to customize the output generation function (in which case it would disguise the fact that two different instances of the NRBG were visiting the same states). Another use is for the secret parameter to initialize the internal state (perhaps along with some input from the entropy source). In any case, the purpose of the secret parameter is to provide insurance in case of an entropy source failure or compromise. Depending on how the secret parameter is used (e.g. the secret parameter might be a removable key), it may also protect the NRBG from an attacker who has gained knowledge of the internal state.

Comment [ebb6]: Page: 1
What does this mean? I sounds scary.

6.2.1.7 Health Tests

This component ensures that the overall NRBG process continues to operate correctly and that the NRBG output continues to be random. These tests should detect failures in the NRBG; the NRBG shall not provide output until the health tests can be successfully passed. These tests are an integral part of the NRBG design; they are performed automatically at power-up or initialization, without intervention by other applications, processes, or users, and may also be requested ~~by the user~~ at any time.

Health tests can be divided into three types: statistical tests on the entropy source, known answer tests on deterministic components and simple randomness tests on the output bits. This final category has been the focus of health testing in the past. Given the use of cryptographic algorithms in the output function, only the most devastating system failures would be detectable in the output bits via the traditional randomness tests. One goal of this Standard is to move the health testing emphasis from output bits to the entropy source.

6.3 NRBG Types

6.3.1 Primitive NRBGs

The most basic NRBG consists of an exceptionally robust entropy source and a conditioning function on the bits obtained from the entropy source. All the security in the basic NRBG rests in the reliability of the entropy source and the health tests on that source, if any. The internal state is completely independent of previous calls to the NRBG for a random number. Thus, if the entropy source degrades, only a health test can prevent the NRBG from producing a number with insufficient entropy.

[Provide a picture and a discussion of conditioning functions (description, purpose, etc.). Is the conditioning function an internal state transition function?]

There are many possible designs for a Primitive NRBG. It could be simply an entropy source with a conditioning routine to ensure that the bits that are output are unbiased. Or it might have some internal state that is carried forward from one infusion of entropy to the next. It could even have Internal State Transition and Output Generation functions to protect the Internal State, but which fail some of the normal requirements on those functions. (At a certain threshold, it's hard to justify not just configuring it as a Standard

NRBG). Whatever the case, some security assurances that are inherent in a Standard NRBG are not available in a Primitive NRBG.

6.3.2 Standard NRBGs

Comment [ebb7]: Page: 1
This is just a modification of the text that was in 8.2, so probably needs more work.

A Standard NRBG is, basically, a primitive NRBG combined with an Approved DRBG that provides a fail-safe mechanism. This type of NRBG has mathematical guarantees of some level of secure operation, assuming that the NRBG is initially working correctly. A Standard NRBG retains information between samplings of the entropy source; the value in retaining this information is that a transient failure of the entropy source does not have devastating consequences.

The Standard NRBG is meant to address the residual vulnerabilities present in the Primitive NRBG. This Standard sets out 2 goals for a Standard NRBG.

1. It will provide an infinite security level if the entropy source is functioning properly.
2. It will continue to operate in a manner that is no less secure than an Approved DRBG in the event that the entropy source completely fails. (This goal is accomplished in the designs described in the following sections by having the NRBG default to a secure, Approved DRBG if the entropy source fails- assuming, of course, that the DRBG has previously obtained sufficient entropy).

This section discusses the meaning of the two goals; Section XX presents two examples that are designed to meet them.

By definition, an NRBG is meant to provide outputs that have full entropy. An adversary with no knowledge or control of the entropy source should not be able to predict an *n*-bit output with any advantage over exhausting the 2^n possible values. However, if care is not taken in integrating the DRBG into the design, a narrow pipe might be introduced that would throttle the entropy provided by the entropy source (see Appendix A for more detail).

A Standard NRBG is meant to provide the assurance of a secure DRBG if the entropy source fails. The following figure demonstrates one way to achieve this goal.

NEED PICTURE

Comment [ebb8]: Page: 1
Should probably depict the two types of Standard NRBGs that we've discussed: the parallel design and the "serial" design.

Figure X shows a "parallel" design, where the DRBG provides a pseudorandom stream of bits that are used to mask the output of a Basic NRBG. As indicated in the figure, output from the Standard NRBG will be used to periodically to reseed the DRBG (these outputs will not be used for other purposes). The initial seed for the DRBG may be provided by a separate entropy source, so that the security of the DRBG will not depend on the NRBG entropy source. This initial seed is considered as a secret parameter for the NRBG.

There are other ways of achieving goal 2. See Appendix A for another example.

(TBD)

A Standard NRBG will provide increased assurance of secure operation over a Primitive NRBG. Typically, the DRBG component will encapsulate the Internal State, the Internal State Transition Function and the Output Generation Function (although these components might also show up in other parts of the NRBG; for instance, there might be other parts of the internal state external to the DRBG). The DRBG will have health tests that it must make available to the NRBG. It should also provide a means for using a secret parameter (one way of doing this is to use the secret parameter as a seed for the DRBG; another is to use it as a key where this is allowed in the DRBG design). Given that an approved DRBG from Part 2 of this standard is being used, additional requirements for an Enhanced NRBG can be viewed as issues in integrating the DRBG into the overall design

7. NRBG Requirements

Comment [ebb9]: Page: 1
Section 7.1 was removed, since it discussed NRBG properties, which I don't think we need to include here.

Part 1 of the standard described a set of requirements on RBGs. This section addresses these requirements in the context of an NRBG.

For the purposes of this Standard, there are requirements, features that are highly desirable, but not required, and optional features. A requirement will always be stated using "**shall**". A feature that is highly desirable but not required will always be stated using '**should**' (some of these recommendations will include mandatory requirements that apply if the recommended feature is included). An optional feature ~~is something that~~ may be implemented to meet application requirements, but is not required by this Standard; an optional feature will be stated using '**may**' terminology.

[Do the general properties in Part 1 (Section 10.1 that y need to be addressed here?].

7.1 Entropy Source

Comment [ebb10]: Page: 1
An attempt was made to unify the requirements as originally specified for the primary and secondary entropy sources. I'm sure that this still needs work.

7.1.1 General Entropy Source Requirements

The entropy source is the foundation of the non-deterministic behavior of the NRBG. By definition, an NRBG **shall** include this component.

The functional requirements for the ~~primary~~ entropy source are as follows:

1. Although the entropy source is not required to produce unbiased and independent outputs, it **shall** exhibit probabilistic behavior; i.e., it **shall not** be definable by any known algorithmic rule.

~~2. The entropy source may be composed from multiple sources to improve resiliency from degradation or misbehavior.~~

Comment [ebb11]: Page: 1
This is covered below.

2. The designer **shall** document the operation of the entropy source. This documentation **shall** include a thorough description of the source.

3. The designer **shall** assess the rate of entropy contribution from the entropy source as accurately as possible (see Section 6.3.2). This requires that the operation of the entropy source be based upon well-established physical principles or

extensively characterized behavior so that an appropriate statistical model for the entropy source can be identified.

4. The statistical model of the entropy source and justifications for its appropriateness **shall** be thoroughly documented.

5. The documentation of the entropy source **shall** reference existing research and literature that is relevant to the entropy source. This information will aid in any validation process.

6. The entropy rate computed using the statistical model **shall** be used in the determination of the rate at which the internal state transition function processes additional entropy source data in order to produce random output at the desired rate.

7. The entropy source **shall** be amenable to testing to ensure proper operation. In particular, it **shall** be possible to collect a data sample from the entropy source during the validation process, to allow an independent verification of the claimed statistical model, the entropy rate, and the appropriateness of the health tests on the entropy source.

8. Failure or severe degradation of the entropy source **shall** be detectable. This aspect of the design **shall** be documented.

9. The entropy source **shall**, to the greatest extent possible, be protected from adversarial knowledge or influence. Knowledge of or influence on the entropy source output by an adversary would effectively reduce the NRBG to a deterministic RBG.

10. The documentation of the entropy source **shall** describe conditions under which the entropy source is known to malfunction or become inconsistent with the nominal statistical model.

11. The documentation of the entropy source **shall** provide supporting evidence that the entropy source is actually non-deterministic. [Note: I feel that this is covered by 4,5,6,7 above and should be deleted.]

12. Multiple entropy sources **shall** be included in the NRBG design if a single entropy source is insufficiently reliable from a failure perspective. In this case, all entropy sources **shall** satisfy these requirements.

13. Multiple entropy sources **shall** be included in the NRBG design if a single entropy source produces entropy at a rate that is insufficient for the desired rate of random bit generation. In this case, all entropy sources **shall** satisfy the same requirements.

Optional, recommended features of the entropy source are as follows:

14. The entropy source **should** be stationary in a mathematical sense. The probabilistic behavior of such a source will not change significantly over time. For example, if the entropy source produces outputs from a certain alphabet with a certain statistical distribution, it **should** be consistent in this bias over time. An entropy source that is not stationary will greatly complicate the process of estimating the rate of entropy contribution and increase the difficulty of validating the resulting NRBG design (unless the design includes additional entropy sources that do satisfy this requirement).

15. If not required by requirements 12 and 13, multiple entropy sourceses **may** be used to improve resiliency to possible degradation or misbehavior. This can help meeting the requirement that the possibility of misbehavior is sufficiently small.

16. Appropriate health tests that are tailored to the known statistical model of the source **should** place special emphasis on the detection of misbehavior near the boundary between the nominal operating environment and abnormal conditions. This requires a thorough understanding of the operation of the entropy source.

17. When multiple entropy sources are used; each entropy source **should** operate independently to ensure that the combined entropy sources will not lose entropy due to statistical dependence. Independence of entropy sources also facilitates the design and evaluation processes by allowing the primary and secondary entropy sources to be analyzed separately, and also reduces the likelihood that a failure in the one entropy source would increase the likelihood of a failure in the other entropy sources.

18. Multiple entropy sources **should** be included in the NRBG design if either of the following is true: (1) a single entropy source is somewhat non-stationary (i.e., inconsistent) in its statistical behavior, making the estimation of input entropy more difficult, or (2) there is concern that a single entropy source may not be free of adversary knowledge or influence. All entropy sources **should** satisfy the same requirements, although it may be acceptable for some (but not all) of the entropy sources to be somewhat more deterministic than others. That is, actions by the user or factors from the system environment could influence (although not completely determine) the output from this source in a perceptible way.

7.1.2 Additional Requirements for Primitive NRBGs

In additional to the requirements in Section 7.1.1, the following requirement is applicable to Primitive NRBGs:

1. The chance that the entropy source can fail must be sufficiently small (10^{-6} ?). Evidence of this shall be documented. One way to achieve this requirement is to use multiple entropy sources.

7.2 Other Inputs

The functional requirements for other inputs are as follows:

1. The NRBG **shall**, to the greatest extent possible, include protections against the manipulation of any inputs (commands, clock, timers, power, etc.) by an adversary. This can best be accomplished by limiting the influence that these inputs have over the overall control of the NRBG. Power input is, of course, a special case; the disruption of power will obviously result in a complete denial of service. If this is a concern, then the operating environment of the NRBG must provide uninterruptible power (this is a system issue beyond the scope of this standard).
2. Any others?

7.3 Internal State

7.3.1 General Requirements for the Internal State

The internal state of the NRBG consists of information that is carried over between calls to the NRBG. This component carries influence from some or all of the previous entropy source data.

1. The state elements that accumulate or carry entropy for the NRBG **shall** have at least max(128, x) bits of entropy, where x is the desired cryptographic strength expressed in bits of security. (x bits of security means that it takes about 2^x operations to attack the cryptographic system.)

Comment [ebb12]: Page: 1
Not sure where this goes, perhaps in 7.3.3?. It seems to be more DRBG related, but maybe needs to be reworded.

2. The internal state **shall** be protected in a manner that is consistent with the use and sensitivity of the NRBG output. A possible means of accomplishing this include assigning the internal state to a memory region that is accessible only to the NRBG, hosting the NRBG on a standalone computer or device, or through security policies that physically protect the system and its environment.
3. The internal state **shall** be functionally maintained properly across power failures, reboots, etc. or regain a secure condition quickly (i.e., either the integrity of the internal state **shall** be assured, or the internal state **shall** be re-initialized).
4. A specific internal state **shall not** be deliberately reused, although this might occur by chance.

Since the NRBG is supposed to provide information-theoretic security (i.e. full entropy is sampled before each output), the optional property from Part 1 (which calls for states used to produce secret data to be fully independent from states which produce public data) is automatically fulfilled with an NRBG.

There is one further optional feature on the internal state.

1. The NRBG **should** be designed so that the internal state continues to accumulate influence from the entropy sources even when the output generation function does not require new working state data (this could be done as a background process when processor or system resources are available). This is especially recommended if there are long periods of time between application requests for random output, during which the internal state might be more vulnerable to

observation due to the increased length of time that it would otherwise remain unchanged. (The example in Section 8.1 will include this feature.)

7.3.2 Additional Requirements for Primitive NRBGs

7.3.3 Additional Requirements for Standard NRBGs

1. The size of the internal state in bits **shall** be sufficient to enable the NRBG to continue to act as an Approved DRBG satisfying the requirements of Part 3 of this Standard if the entropy source fails or becomes completely known or controlled by an adversary. If the entropy source data becomes constant, the maximum possible cycle length is bounded by the number of possible internal states of the DRBG, and this places an upper bound on the work an adversary would need to recover the internal state (through exhaustion). The number of bits in the working state portion of the internal state **shall** be at least the number of bits in each random output block produced by the NRBG.

7.4 Internal State Transition Function

7.4.1 General Requirements on the Internal State Transition Function

The internal state transition function is responsible for combining data from the entropy source, the internal state, and optional additional inputs, to produce a new internal state that the output generation function can use to produce random output for the consuming application. The internal state transition function consists of one or more deterministic functions ~~parameterized by the current secret parameter value~~.

Comment [ebb13]: Page: 1
Is this characteristic of both Primitive and Standard NRBGs, or of just Standard NRBGs?

The functional requirements for the internal state transition function are as follows:

1. All of the bits in the internal state and the entropy source input shall potentially influence any of the output bits from the internal state transition function.

2. For each replacement of the working state portion of the internal state, the entropy source data processed by the internal state transition function **shall** be of sufficient quantity to contain at least as many bits of entropy as the number of bits in the internal state. An accurate assessment of the entropy rate of the input source is critical for this determination.

~~Part 1 includes an optional property for the internal state transition function, that it may recover from compromise through periodic incorporation of new entropy. This is inherent in the design of an NRBG.~~

Comment [ebb14]: Page: 1
We shouldn't need to address Part 1 properties, just requirements. If this is not one of teh "features" in the requirements section, it should be.

7.4.2 Additional Requirements for Primitive NRBGs

1. The internal state transition function **shall** be a one-way function based on an accepted ASC X9 approved cryptographic function, such as a hash function, block cipher algorithm, stream cipher algorithm, or a combination of such functions. Examples of cryptographic functions that are appropriate as a basis for an internal

state transition function include SHA-1, SHA-256, SHA-512, or various modes of AES.

2. If the conditioning function is mathematical, documentation where it may fail, including bounds on the behavior of the entropy source inside which everything should work okay.

Comment [ebb15]: Page: 1
Does this requirement belong under the transition function or the entropy source? I.e., where does the conditioning function fall?

7.4.3 Additional Requirements for Standard NRBGs

1. The NRBG **shall** use an approved DRBG from Part 3 of this Standard sothat the NRBG fails to an Approved DRBG if the entropy source fails.
2. The NRBG design **shall** not use "narrow pipes" that will constrict the flow of entropy between the source and the output bits.
3. The DRBG **shall** be chosen to provide at least the desired level of "fallback security" to the NRBG.

7.5 Output Generation Function

7.5.1 General Requirements for the Output Generation Function

The output generation function is a function that produces random output for the requesting application based on the current value of the internal state. The output generation function must provide random output to the requesting application in a way that does not reveal any internal state or entropy source input information. It is not an objective of the output generation function to provide additional smoothing to the entropy source input, since this is the responsibility of the internal state transition function.

The functional requirements for the output generation function are as follows:

1. The output generation function **shall not** introduce detectable statistical biases into the random output.

2. The output generation function **shall not** reuse data from the working state portion of the internal state when providing random data to the requesting application. That is, either the internal state transition function **shall** replace the working state between applications of the output generation function, or the output generation function **shall** use a previously unused portion of the working state.

3. The output generation function **shall** process at least as many bits from the internal state as the number of bits in each random output block produced by the output generation function. Depending on the type of output generation function used, it may be necessary to process significantly more than this number of bits from the internal state, in order for the output generation function to be non-invertible (see #4 below). For example, an acceptable output generation function could consist of XORing two *m*-bit sequences from the current internal state to form an *m*-bit random output. Another alternative would be an ASC X9 approved

hash function that hashes at least *m* bits from the internal state to produce an *m*-bit random output

4. The output generation function **shall** be non-invertible. That is, knowledge of the random output produced by the output generation function **shall not** reveal any information about the input to the function.

7.5.2 Additional Requirements for Primitive NRBGs

7.5.3 Additional Requirements for Standard NRBGs

7.6 Secret Parameter

Standard NRBGs **shall** use one or more secret parameters; at a minimum, the NRBG requires a secret seed as the secret parameter. Primitive NRBGs **may** use secret parameters. The following requirements apply to secret parameters:

1. The secret parameter **shall** have a specified finite cryptoperiod, after which the NRBG **shall** be reinitialized with a new secret parameter and sufficient additional entropy, as specified by the security level desired. ~~(Editor's note: Should the secret parameter be optional?)~~
2. The length of the secret parameter **shall** be sufficient to prevent an exhaustive attack on the NRBG output in the event that the entropy sources fail or become known to the adversary, and the working state portion of the internal state is compromised.
3. The secret parameter **shall** be protected from adversarial knowledge. In a properly designed internal state transition function and secret parameter usage scheme, this secrecy will help to maintain the confidentiality of the random output even if the adversary gains control or knowledge of the input source data.
4. The secret parameter **shall** be tested before use to ensure that its storage location has not failed to a trivial pattern. Patterns tested shall include constant zeroes, constant ones, and alternating zero and one sequences.
5. The initial value of the secret parameter **shall** contain sufficient entropy for the security level to be provided by the NRBG. This initial secret parameter **shall** either be generated by this NRBG or by another approved NRBG. If the secret parameter is to be generated by this NRBG, a reasonable approach is to run the NRBG in its usual mode, taking the secret parameter from the resulting random output, taking care not to reveal that particular random output or provide it to any applications. If there is any concern that the entropy source data is known or influenced by the adversary during the production of this value, then the key generation process **shall** obtain additional entropy data either from another system component or through interaction with the user (e.g., key presses, timings between key presses, or mouse movements) to be used in conjunction with the entropy source data (i.e., the source of this additional data serves as a temporary secondary entropy source).

Comment [ebb16]: Page: 1
How do we deal with an infinite security level? For Standard NRBGs, do we mean the fall-back security level?

A recommended feature for the secret parameter is the following:

6. The secret parameter **should** be preserved between operational sessions in order to provide the NRBG with a unique state that has sufficient entropy at each power-up initialization without having to immediately create a new secret parameter value. If this is done, it **shall** be preserved in a way that protects it from adversary access. This protection could take the form of storage in a memory area accessible only to the NRBG process, storage in encrypted form, or storage in a removable token.

7.7 Health Tests

7.7.1 General Requirements for Health Tests

The health described in this section **shall** be integrated into the design of the NRBG for operational testing (validation tests described in Section 9 will include additional tests). The health tests presented in this section include three sets of tests – tests on the deterministic components of the NRBG, tests on the entropy sources, and tests on the random output produced by the NRBG. The tests on deterministic components **shall** apply to all NRBG designs. There may be cases where the rate of input entropy or random output is too low to feasibly implement all of the specific tests in Sections 7.7.1.2 and 7.7.1.3 on either the entropy sources or the NRBG output. In such cases, the designer may attempt to modify the tests or test thresholds as appropriate, to permit smaller sample sizes, while keeping the Type 1 error probability approximately the same. This standard will assume that the health tests in Sections 7.7.1.2 and 7.7.1.3 can feasibly be applied, given the input entropy rate and random output rate of the NRBG.

General requirements and requirements that apply to each of these sets of tests are presented in the following four sections. In some cases, the NRBG may have additional features or functionality not addressed by this Standard; additional specialized tests not addressed in the following sections may be included. In these cases, the designer **shall** thoroughly document the objectives of these additional features and the basis for the additional tests.

7.7.1.1 General Health Test Requirements

The ~~testable~~ functional requirements for all three categories of health tests introduced above are as follows:

1. The NRBG **shall** automatically perform thorough health tests at each power-up or initialization.

2. The NRBG **shall** ~~also~~ allow the user or consuming application to request the health tests (on the entropy sources, deterministic components, and random output) at any time. (Is there a smart card issue here (poor interface for requesting anything)? I think the NRBG would not be on the smart card).

3. All data output from the NRBG **shall** be inhibited while the health tests are being performed, in order to conceal information about the operation of the NRBG and to prevent the release of any information about possible failures. ~~Data that has~~

~~successfully passed the tests on random output (as opposed to the known answer tests) can be used as random output following the completion of all health tests. It would be a grave error to use the output from known answer tests as random output, as it is completely predictable.~~

4. If the NRBG is implemented as software or firmware, the health tests performed at initialization **shall** include an integrity check on the device that hosts the implementation code (RAM, ROM, or programmable logic device). Examples of ways to do this include a digital signature or message authentication code applied to the software or firmware.

The recommended health tests are as follows:

5. The NRBG **should** perform the health tests at various times during an operational session, in addition to during power-up initialization (a reasonable interval is once per day). If the length of time between power-ups is not great, this additional testing may not be necessary.

7.7.1.2 Health Tests on Deterministic Components

The objective of these tests is to ensure that the deterministic components of the NRBG continue to correctly process any possible set of inputs. Since, by definition, there is no unpredictability in these components, the best method of testing them is to use known answer tests. Such tests initialize the component or function to a fixed initial state, input a fixed input to the function, then compare the resulting output with the correct output that was computed previously by another implementation of the function (e.g., a verified computer simulation used during NRBG development) and stored with the NRBG implementation.

The functional requirements for the health tests on the deterministic components are as follows:

1. The known answer tests **shall** be included in the overall health tests performed at each power-up and/or re-initialization, at periodic intervals during use, and when requested by the user or consuming application.

2. The comparison sequence (the result to be compared with the known answer) that is roduced for any known answer test **shall** be sufficiently long so that the probability of passing the test with failed or degraded components is acceptably low. Since 32-bit checksums are used in many information assurance applications, 32 bits is a suggested as a minimum length for known answer test values.

3. The set of known answer tests **shall** include an all encompassing known answer test on the entire NRBG, in order to not only test each component of the NRBG individually, but also to test the overall NRBG control and the interaction among all the components. This can be done by setting the internal state to a fixed pattern, overriding the entropy source data with a fixed sequence, and running the

NRBG process in its operational mode, to produce an output sequence of at least the length determined according to #2 above. The known answer test then compares this output with the result that was previously computed using the same inputs and an independent implementation or simulation of the NRBG.

4. The bits produced during a known answer test (including any function of these bits) **shall not** be output from the NRBG.

5. If any of the health tests on the deterministic components return a failure result, the NRBG **shall** enter an error state and indicate a failure condition to the application or user. The NRBG **shall not** perform any random output generation while in the error state. The NRBG **shall** require user intervention (e.g., power cycling or re-initialization), followed by successfully passing the health tests, in order to exit the error state. Note that it is likely that the NRBG will require maintenance for this to occur.

6. The known answer tests **shall** thoroughly exercise each aspect of the function being tested in order to maximize the probability that failures will be detected. This generally requires that the fixed input pattern be long enough to provide a representative sample of possible inputs to each major functional component of the function being tested. For example, if a function contains several accesses into an eight-bit lookup table as part of its operation, a known answer test **shall** require that a significant proportion (all?) of the 256 possible table addresses occur during the known answer test.

Additional recommendations regarding the health tests on the deterministic components are as follows:

7. The known answer tests on the deterministic components of the NRBG **may** be eliminated in favor of implementing the NRBG as two redundant and independent processes (other than the entropy sources) whose outputs are continuously compared. In this case, a mismatch **shall** be handled as a health test failure, with entry to the error state.

7.7.1.3 Health Tests on Entropy Sources

The objective health tests on the entropy source is to detect variation in the behavior of the entropy source from the intended behavior. Since the entropy source will not produce unbiased, independent binary data in the vast majority of cases, traditional randomizer tests (e.g., monobit, chi-square, and runs tests that test the hypothesis of unbiased, independent bits) will virtually always fail and, thus, not be useful. In general, tests on the entropy sources will have to be tailored carefully to the entropy source, taking into account the non-uniform statistical behavior of the correctly operating entropy source. Therefore, the health tests on entropy sources may be less sophisticated than some of the statistical tests typically applied to RBG output. For an example of a health test tailored to a specific hypothetical entropy source, see Section X.

For very complicated entropy sources, it may not be feasible to develop statistical tests that correspond precisely to a statistical model of the entropy source. In these cases it may be more appropriate to a) apply a simple entropy estimate to the output of the source and b) determine whether the data sample contains any occurrences of values known to be associated with failures of that entropy source. The selection of the patterns used for such tests should take into account the entropy source's likely failure behavior. For an example of such a health test, see Section X.

The functional requirements for the health tests on the entropy sources are as follows:

1. The tests on the entropy sources **shall** be included in the overall health tests that are performed at each power-up and/or re-initialization, at periodic intervals during use, and when requested by the user or consuming application.

2. At a minimum, each entropy source **shall** be tested for activity. That is, the test **shall** collect a quantity of data from each source and confirm that it does not consist solely of a constant output. (Constant outputs are those consisting only of a single value of the digitized entropy source output. For example, if the noisy diode in Section 8.1.3.1 produced the value 0110 at each sampling, it would fail an activity test.) The size of the data sample collected will depend on the characteristics of the entropy source, and **shall** be chosen such that when the entropy source is operating correctly, the probability of no activity within a sample of that size is acceptably low (10^{-4} is a recommended value for this Type 1 error rate).

3. The tests applied to each of the entropy sources~~,~~ and the rationale for their appropriateness~~,~~ **shall** be thoroughly documented. The rationale **shall** indicate why the tests are believed to be appropriate for detecting failures in the entropy sources.

4. If any of the health tests on the deterministic components return a failure result (see #6 below), the NRBG **shall** enter an error state and indicate a failure condition to the application or user. The NRBG **shall not** perform any random output generation while in the error state. The NRBG **shall** require user intervention (e.g., power cycling or re-initialization), followed by successfully passing the health tests, in order to exit the error state. Note that it is likely that the NRBG will require maintenance for this to occur.

The optional, recommended features of the entropy source health tests are as follows:

5. The tests on each entropy source **should** include tests that take the known characteristics of the entropy source into account.

6. If a health test on an entropy source returns a failure result, the NRBG **should** repeat the test a moderate number of times before declaring an error condition and entering the error state. The total number of attempts **shall not** exceed three. If the entropy source passes the test during this set of attempts, the NRBG can resume normal operation. Otherwise, the NRBG **shall** enter an error state.

7. Entropy bits that have successfully passed the entropy source health tests **may** be used to produce NRBG output.

7.7.1.4 Health Tests on Random Output

The objective of these tests is to provide a final check on the randomness of the output from the NRBG. In general, the inclusion of the internal state transition function and output generation function results in the health tests on the random output from an NRBG playing a smaller role than they would if the tests were applied directly to output from a non-deterministic entropy source. These functions will typically do such thorough mixing that even a complete failure of the entropy sources would not cause detectable statistical irregularities in the random NRBG output. This is of course a consequence of the requirement that the NRBG continue to operate as an approved DRBG if the entropy sources fail or come under the influence of an adversary.

A major goal of this Standard is to move the emphasis on statistical testing away from the output bits (where it has traditionally been) and back to the entropy source. However, statistical tests on random output are still useful, and are addressed in the requirements below.

The implementation of health tests on the NRBG output is recommended, but not required. If health tests on the random output are implemented, the following are recommended:

1. The periodic tests on the random output **should** be included in the overall health tests performed at each power-up and/or re-initialization, at periodic intervals during use, and when requested by the user or consuming application.

2. The NRBG **should** perform at least a simple test on each block of random output produced by the output generation function. This test **should**, at a minimum, verify that the random output does not consist entirely of constants or alternating zeroes or ones.

3. (Ed Note: Need to discuss this. Is it still worthwhile? Should either move to an appendix or refer to Part 1)At a minimum, the periodic NRBG health tests **should** include the following set of tests on a sequence of 20,000 bits of random output from the NRBG. The overall set of tests is considered to have passed if all four individual tests are passed. The indicated thresholds correspond to a Type 1 error probability of 10^{-4}.

    a. Monobit test: Let X be the number of ones in the sample. The test is passed if $9725 < X < 10275$.

    b. Poker test: Divide the sequence into 5,000 consecutive four-bit segments. Count the number of occurrences of each of the sixteen possible four-bit values. Let $f(i)$ be the number of occurrences of the four-bit value i, where $0 \le i \le 15$. Evaluate the following: $X = \frac{16}{5000}\sum_{i=0}^{15} f(i)^2 - 5000$.

The test is passed if $2.16 \le X \le 46.17$.

c. Runs test: A run is defined as a maximal sequence of consecutive bits of either all ones or all zeroes. The occurrences of runs for both consecutive zeroes and consecutive ones of all lengths from one to six should be counted and stored. The test is passed if these counts are each within the corresponding interval specified in the table below. This must hold for both the zeroes and ones. For the purposes of this test, runs of length greater than six are considered to be of length six.

Run Length	Required Interval
1	2,343 – 2,657
2	1,135 – 1,365
3	542 – 708
4	251 – 373
5	111 – 201
6+	111 - 201

Table 1: Runs Test Intervals

d. Long runs test: A long run is defined to be a run of length 27 or more of either zeroes or ones. The test is passed if there are no long runs.

4. If any of the health tests on the random output return a failure result (see #6 below), the NRBG **shall** enter an error state and indicate a failure condition to the application or user. The NRBG **shall not** perform any random output generation while in the error state. The NRBG **shall** require user intervention (e.g., power cycling or re-initialization), followed by successful passing of the health tests, in order to exit the error state. Note that it is likely that the NRBG will require maintenance for this to occur.

5. If a health test on random output returns a failure result, the NRBG **should** repeat the test a moderate number of times. The total number of attempts **should not**

exceed three. If the random output passes the test during this set of attempts, the NRBG can resume normal operation.

6. Data that has successfully passed the tests **may** be used as random output after completion of the health tests.

Ed Note: I took out the Requirements on Component Interaction. Two of them were inherent in component requirements, one was the definition of an Enhanced NRBG and the final one (about constantly folding in new entropy to the internal state) I added as an optional requirement on the internal state.

7.7.2 Additional Requirements for Primitive NRBGs

1. Health Tests **shall** be sufficiently robust to detect deviations in the entropy source with (?) accuracy and Type 1 (or is it 2) Error of (?). Evidence of this **shall** be documented.

7.7.3 Additional Requirements for Standard NRBGs

1. The DRBG **shall** provide an interface that allows the NRBG to exercise the DRBG's health tests.

8. Entropy

8.1 Entropy Measurement

The estimation of the quantity of entropy produced by an entropy source is an important aspect of the design of an NRBG meeting this standard. The beginning point for this calculation is the determination of a statistical model describing the behavior of the entropy source. Given this model, the amount of entropy produced by the entropy source can be estimated using various formulas. For example, suppose that the entropy source produces one of n possible outputs or sequences of outputs at each time interval, with the i^{th} possible outcome having probability p_i. The most common definition of entropy is the Shannon definition, $H = -\sum_{i=1}^{n} p_i \log_2 p_i$, which is useful in various information theory contexts. However, cryptographic researchers have studied an alternate family of entropy measurements known as Rényi entropy, parameterized by a value α, where $0 \le \alpha < \infty$. The Rényi entropy of order α for the above distribution is defined as $H_\alpha = \frac{1}{1-\alpha} \log_2 \sum_{i=1}^{n} p_i^{\alpha}$. This family actually includes Shannon entropy, since it can be shown using l'Hôpital's Rule that although H_1 is undefined, the limit of H_α as α approaches one is H. Also, it is easy to show that the limit of H_α as α approaches ∞ is $-\log_2(\max\{p_i\})$, referred to as min-entropy.

The measurements H_2 and min-entropy have particular advantages for RBG analysis. It can be shown that for a fixed distribution $\{p_i\}$, H_α is a decreasing function of α. Thus, min-entropy is the most conservative measurement of entropy, and is useful in determining a worst-case estimate of the sampled entropy. Min-entropy also has a very useful interpretation relating it to the probability of success for an adversary who is trying to guess the value of a string. Suppose a string is generated with min-entropy of K bits. This means that the probability of the most likely value of the string is 2^{-K}. So, if an adversary follows the optimum guessing strategy, his probability of success is only 2^{-K}. Furthermore, if such an adversary follows the optimum strategy for making 2^{w} guesses (i.e. he guesses the most likely 2^{w} values) his probability of success will be at most 2^{w-K}.

Min-entropy also enjoys a practical advantage. Given a distribution with a limited number of values and a limited number of samples of that distribution, the easiest probability to estimate will be that of the most likely value, and this is the only probability needed to compute min-entropy. Contrast this with other entropy measures which depend on the whole probability distribution.

On the other hand, H_2 has an intuitive relationship to the repeat rate of a sequence of outputs having a given probability distribution. Also, H_2 can be efficiently computed using an iterative matrix manipulation procedure, given a Markov or Hidden Markov model of the entropy source. [Give a reference for this - our two recently released NSA documents or some form thereof. I still intend to say more about Markov modeling here.] This standard will emphasize the use of min-entropy, because it gives a conservative estimate, it's simple to compute (given the probability distribution) and it fits the natural model of the adversary making optimal guesses. The example in Section 8.1 will use both min-entropy and H_2, to demonstrate the reasoning and calculations involved in modeling an entropy source.

8.2 Managing Entropy(?)

Probably should add more here.....

8.3 Detailed NRBG Example

This section presents a hypothetical example of an NRBG satisfying the requirements of this standard. This description is intended only to give a general idea of a sample NRBG, and does not include all the detail that would be necessary to fully define and implement an NRBG satisfying this standard.

8.3.1 General Structure

Note : I didn't really look at this example during this pass, since I think you plan on changing it.

Note: Next time around I'll have one of John's new hash-based DRBGs built into this. This needs a lot of work right now....

This NRBG is designed to produce random outputs in 160-bit blocks by processing non-deterministic outputs from a noisy diode. The design satisfies the requirement of being at least as strong as an approved DRBG in the event the entropy source completely fails (this is addressed Section 8.1.4 below). The overall structure of the NRBG is illustrated in Figure 3. In this illustration, "SHA-1" indicates the Secure Hash Algorithm.

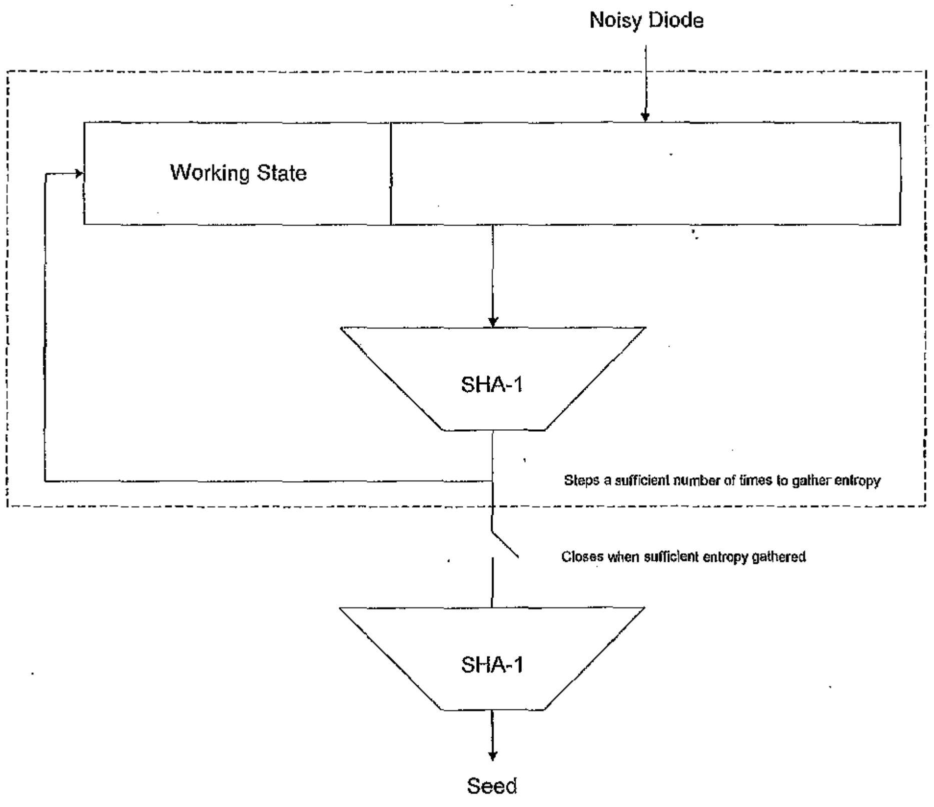

Noisy Diode
Working State
SHA-1
Steps a sufficient number of times to gather entropy
Closes when sufficient entropy gathered
SHA-1
Seed

Figure 1: Hypothetical Example NRBG

For this sample NRBG, the realizations of the functional components defined in Sections 6 and 7 are as follows:

For this sample NRBG, the realizations of the functional components defined in Part 1 and addressed more specifically in Part 2, are as follows:

1. Primary entropy source: A single noisy diode
2. Optional secondary entropy source and other inputs: Control inputs only; no additional entropy source
3. Internal state: 160-bit register for working state and current 160-bit SHA-1 initialization vector as secret parameter
4. State transition function: SHA-1 standard hash function of current working state and additional entropy source input, using secret parameter as initialization vector
5. Output generation function: SHA-1 standard hash function of working state producing 160-bit output.
6. Health tests: Known answer test on the combined operation of deterministic components after replacing variable values with known values, tailored statistical tests on the entropy source, and general statistical tests on random output.

8.3.2 Details of Operation

8.3.2.1 Entropy Source

The entropy source is a noisy diode that generates random voltages according to a normal distribution having mean 6.0 and standard deviation 1.0 when the ambient temperature is within the nominal range. At each sampling interval, the diode output voltage is digitized and converted to a four-bit value. Table 2 shows the mapping of voltages to digital values as well as the probability of each output.

Sampled Voltage	Probability p_i	Digitized Output
$-\infty < Z < 2.5$	0.000233	0000
$2.5 \le Z < 3$	0.001117	0001
$3 \le Z < 3.5$	0.004860	0010
$3.5 \le Z < 4$	0.016540	0011
$4 \le Z < 4.5$	0.044057	0100
$4.5 \le Z < 5$	0.091848	0101
$5 \le Z < 5.5$	0.149882	0110
$5.5 \le Z < 6$	0.191462	0111
$6 \le Z < 6.5$	0.191462	1000
$6.5 \le Z < 7$	0.149882	1001
$7 \le Z < 7.5$	0.091848	1010
$7.5 \le Z < 8$	0.044057	1011
$8 \le Z < 8.5$	0.016540	1100
$8.5 \le Z < 9$	0.004860	1101
$9 \le Z < 9.5$	0.001117	1110
$9.5 \le Z < \infty$	0.000233	1111

Table 2: Digitization Voltage Ranges and Probabilities

Table 3: Noisy Diode Output Probabilities

The entropy source input data used for each replacement of the working state is required to contain at least 128 bits of entropy. The H_2 entropy of the noisy diode is $-\log_2[\sum_{i=0}^{15} p_i^2] \cong 2.84067$. If we base our entropy estimate on the H_2 entropy then, since each noisy diode output contains 2.84067 bits of entropy, 128 bits of entropy requires a sequence of at least $\left\lceil \frac{128}{2.84067} \right\rceil = 46$ four-bit outputs from the noisy diode.

However, the min-entropy of the diode is $-\log_2(\max\{p_i\}) = -\log_2(0.191462) =$ 2.38487 (less than the H_2 entropy, as expected). Since we are using min-entropy as our measure, we will require at least $\left\lceil \frac{128}{2.38487} \right\rceil = 54$ samples from the diode. This input will be digitized, and padded consistent with the SHA-1 specification.

8.3.2.2 Primary Tasks

This NRBG consists of two processes – a background process called "CHURN" that collects entropy source data and replaces the internal state, and a process called "GENERATE" that produces an 80-bit random output when requested. "CHURN" runs whenever processor resources are available. "GENERATE" runs only when called by a requesting application. This structure allows the NRBG to continue to accumulate additional influence from the entropy source over time, rather than relying on just the minimum amount of entropy influence required by this standard. The operation of "CHURN" and "GENERATE" are illustrated by the flowcharts in Figure 3.

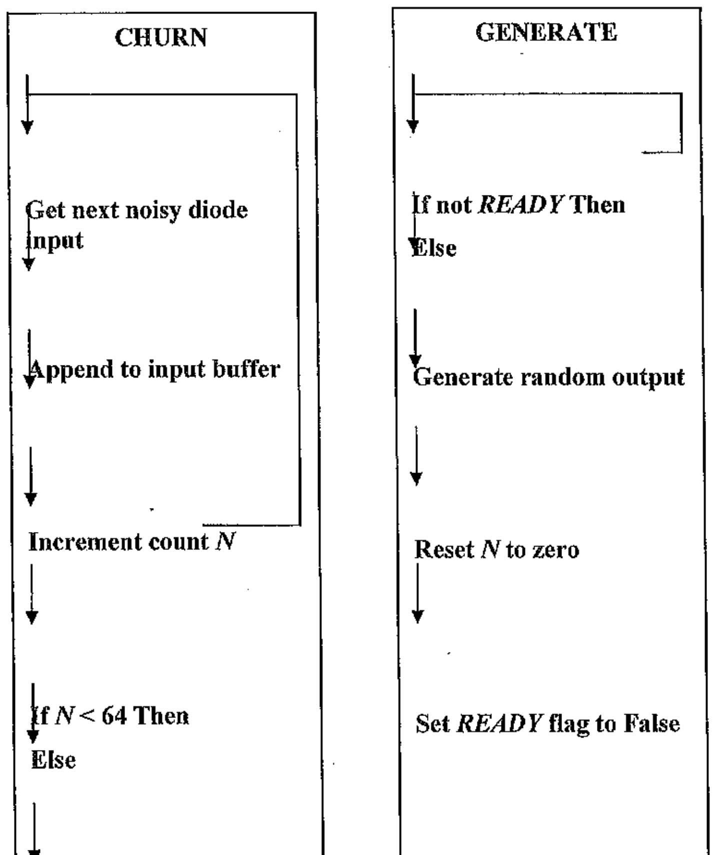


Figure 2: Logic flow for NRBG tasks CHURN and GENERATE

As illustrated in Figure 3, CHURN operates continuously as a background process (as allowed to by the overall system). It is essentially a loop that collects four-bit outputs from the noisy diodes, stores them in a 64-output (256-bit) input buffer, and replaces the working state whenever this buffer is full. As soon as the first working state replacement has been completed, a "Ready" flag is set to indicate that the NRBG is ready to produce random output, although CHURN continues to collect additional entropy source outputs and replace the working state. The "Ready" flag is reset during NRBG initialization. Note that due to the scheduling of other system tasks, this data collection, buffering, and updating may not occur continuously.

As illustrated in Figure 3, the GENERATE operation receives requests for random output from the application. When requested by the application, GENERATE first checks the "Ready" flag to ensure that the working state contains sufficient entropy. If the "Ready" flag is not set, GENERATE waits until CHURN sets the flag to indicate sufficient entropy in the working state. If the "Ready" flag is set, GENERATE uses the current internal state to produce an 80-bit random output. GENERATE accomplishes this by using the output generation function (SHA-1) to hash the contents of the working state and extracting the 80 high-order bits of the resulting 160-bit hash output as the 80-bit random output.

8.3.2.3 Support Tasks

In addition to the two primary tasks CHURN and GENERATE, this NRBG also has two additional processes to perform necessary support functions. The first is a process called "START" that runs once at the beginning of each power-on session, initializing the NRBG by setting necessary variables to their initial states, and if necessary, creating an initial secret parameter. The second is a process called "SELFTEST" that runs during power-up initialization, at 24-hour intervals within an operational session, and whenever requested by the user, performing a set of health tests on the NRBG.

The START task begins by calling SELFTEST to perform health tests. Then START sets the working state to either the secret parameter (for the initial startup) or to the stored working state if the NRBG is resuming operation.

The SELFTEST task performs a set of health tests during power-up initialization, at 24-hour intervals within an operational session, and whenever requested by the user. The details of the tests are given in Section 8.1.3.4, and include known answer tests on the deterministic components, statistical tests on the entropy source, and statistical tests on the random output of the NRBG.

8.3.2.4 Health Test Details

The NRBG performs a set of health tests during power-up initialization and at 24-hour intervals within a power-up session. This set of tests consists of the following:

- The health test on the deterministic components consists of an overall known answer test on the entire NRBG. This requires setting the SHA-1 initialization value to the 160-bit value TBD and the working state to the 160-bit value TBD, and replacing 64 four-bit values from the noisy diode with the 256-bit value TBD. The NRBG then produces an 80-bit output using the same steps used in normal operation. The resulting output is compared with the value TBD and a failure is declared if there is a mismatch.

- There are two health tests on this entropy source, each designed to have a Type 1 error probability of approximately 10^{-4}. The first consists of sampling $N = 1000$ outputs, counting the number O_i of occurrences of each of the sixteen possible outcomes, and performing a χ^2 goodness-of-fit test on the results. Because of the general rule requiring an expected count of at least five in each category, we combine the first three and last three counts into combined categories, giving a total of twelve categories. The output probabilities and expected counts E_i are shown in Table 4. The statistic $\chi^2_{(11)} = \sum_{i=1}^{12} \frac{(O_i - E)^2}{E_i}$ has a χ^2 distribution with eleven degrees of freedom when the entropy source is operating correctly. The entropy source is declared to fail the health test if this statistic exceeds 37.4. The second test looks for behavior known to have increased likelihood when the temperature is outside the nominal range. It is known that this situation results in frequent periods of voltages in the low end of the range. This test uses the same sample of 1000 outputs and searches for occurrences of two or more consecutive 0000 outputs. If such a sequence occurs in the sample, an error condition would be declared. Note that this test is complementary to the first test; the first test looks for general inconsistencies of the output with the nominal statistical model, while the second test looks for a specific type of outcome known to be associated with a known failure condition.

Category	Digitized Output	Expected Count
1	0000-0010	6.21
2	0011	16.54
3	0100	44.06
4	0101	91.85
5	0110	149.88
6	0111	191.46
7	1000	191.46
8	1001	149.88
9	1010	91.85
10	1011	44.06
11	1100	16.54
12	1101-1111	6.21

Table 4: Category Expected Values in 1000 Samples

- The health test on the random output consists of collecting and concatenating 250 consecutive 80-bit values (a total of 20,000 bits) from the NRBG and performing the monobit, poker, runs, and long runs tests specified in Section 7.1.6.4. If any of the tests fail, an error condition is declared.

In addition, each 80-bit random output produced during normal operation is examined; if it matches any of the four sequences consisting of all ones, all zeroes, or alternating ones and zeroes, an error condition is declared.

8.3.3 Failsafe Design Consequences

This NRBG is designed to satisfy the design requirement of being no less secure than an approved DRBG in the event that the entropy source completely fails in an undetectable way or comes under control by an adversary.

8.4 Entropy Source Examples

This section discusses several entropy sources in detail. These discussions, while not exhaustive, are meant to indicate the sort of analysis that is needed to model an entropy source and achieve validation. Note: Thanks to John Kelsey for lots of help (and more to follow?) on this section.

8.4.1 Coin Flipping

Although clearly impractical in most situations, coin flipping provides a good model for an entropy source. Since coin tossing is commonly considered to be a standard intuitive model for random bit generation, and since, under fairly simple assumptions, the addition of some mathematically based post-processing of the coin toss outcomes will result in perfectly independent and unbiased binary outputs, this standard permits the generation of random bits using a coin-tossing procedure. The coin flipping procedure described here (and in Part 1) may also be seen as a Primitive NRBG, suitable for providing entropy input for DRBGs in certain extreme cases. One thing that it has going for it is a well-understood model and a mathematically provable conditioning routine. The drawbacks (which include very poor performance and an absurdly porous security boundary) are certainly substantial.

The basic coin tossing NRBG permitted by this standard consists of a person repeatedly flipping a coin and assigning one side of the coin to the outcome "zero" and the other side of the coin to the outcome "one". This raw sequence may be biased (e.g. the coin might not be fair). Assuming that the coin toss outcomes are independent and identically distributed (but not necessarily unbiased), the Peres Unbiasing procedure can be used to assess the amount of entropy in the sequence. The output of Peres Unbiasing is an unbiased sequence, and the length of this sequence is approximately the Shannon entropy of the original sequence. At this point, if the entropy is sufficient for the application, the original sequence may be used. Alternatively, if the requesting application requires an entropy input of length equal to the amount of entropy (i.e., the application requires full entropy), the reduced sequence may also be used. See ### in Part 1 for more details on Peres Unbiasing. (Hey, I thought we were using min-entropy in this standard!)

Depending on the availability of computational resources, it may be possible to extend the basic coin tossing NRBG to provide trust among several parties. For example, if several participants have a stake in the random output produced, it may not be acceptable for one participant to have sole responsibility for the coin tossing, especially if the participants are in different locations and cannot observe the participant tossing the coin. The following procedure results in a random binary sequence of N bits (N can be any positive integer) with the property that no participant or coalition of participants can unfairly influence the final outcome toward a favorable outcome, and any participant who desires a random sequence can trust the randomness of the final outcome. This procedure requires that each participant have access to an implementation of a hash function with a suitable security level. For concreteness, SHA-1 will be used in this example.

1. Each participant P_i forms an N-bit sequence x_i and an additional M-bit sequence y_i using the basic coin tossing procedure, including the Peres Unbiasing process. (See below for a discussion of the value of M.)

2. Each participant P_i computes $z_i = H[x_i \mid y_i]$, where H is the SHA-1 hash function and "|" represents concatenation, and sends z_i to each of the other participants. This commits participant P_i to the random sequence x_i.

3. Each participant P_i sends x_i and y_i to each of the other participants.

4. Each participant verifies the SHA-1 hash of the x_i and y_i sequences of each of the other participants, ensuring that no participant's random sequence x_i has been changed in response to any other participant's x_i.

5. Each participant computes the XOR of all the x_i sequences. The first N bits of the result are the cooperatively generated random bits.

The purpose of the M additional bits is to prevent participants from determining other participants' x_i sequences using pre-computed lookup tables. Without these values, such an attack would be possible if N is sufficiently small. Thus, M should be large enough to make recovery of a SHA-1 pre-image of length $M+N$ bits computationally infeasible. Note that any participant can ensure the randomness of the final output without trusting any other participants.

Comment [ebb17]: Provide an example?

8.4.2 Mouse Movements

One entropy source that is present on a PC is themovement of the mouse. Movements of the mouse could be sampled fairly frequently (say one hundred times per second) and some property (absolute position, velocity or some other) could be digitized and accumulated as an entropy input to the NRBG (or as ~~additional~~ entropy input to a DRBG). To do this properly, a model is required for the probability distribution of the mouse measurements, which will hopefully provide a good idea of the rate of entropy that can be expected, as well as a means to measure the proper functioning of the source. Note that this source depends on activity from the user. Thus, mouse measurements may vary over time as the user's activity varies (e.g., greater entropy while web surfing than during coffee breaks). This could be mitigated by sampling over long periods and accumulating or by using an initialization routine that prompts the user to move the mouse a great deal during a short period of time. At any rate, this entropy source will not be useful in all situations; in particular, a server running with no human being present at the console will have much better choices of entropy than this one!

The software that allows the operating system to communicate with the mouse is the mouse driver. Its job is to look~~, to software,~~ like a generic mouse driver to the software, also communicating with the mouse. The mouse driver hides the details of communicating with the mouse. Assuming that the NRBG talks to the mouse through the driver, it's not surprising that knowledge of the driver could be important in determining the entropy rate of the source.

The basic unit of detected mouse motion is called the mickey. Typically, one mickey is about 1/200 of an inch, but some mouse hardware is more sensitive than this. If the mouse is moved less than one mickey, it doesn't register as motion.

There are essentially three kinds of physical arrangements for hooking a mouse to a system:

a. A bus mouse hooks to some connector on the system bus, which can read the current signal from the mouse. This requires no processor in the mouse itself. In some designs, the mouse driver polls the mouse 30+ times a second. In other designs, the connector waits until the signal from the mouse changes, and then causes an interrupt so that the mouse driver can read from the connector.

b. A serial mouse communicates over a serial port. In the past, this was likely a RS232 port, but USB mice are now becoming more common. Basically, the mouse's processor detects motion, and generates a packet, typically of 3-5 bytes, specifying how many mickeys have changed in the X and Y directions since the last packet, and its button state. The three-button mice, and the ones with wheels on them for faster scrolling, include extra bytes to send that extra state.

c. A PS/2 mouse works is similar to a serial mouse; it has its own processor, and communicates with the keyboard/mouse controller on the PC by sending packets asynchronously. Documentation has shown that the mouse sample rate could be as low as 10 samples per second.

The mouse driver keeps track of the mouse cursor position, button status, and whatever other events are available. It is free to do whatever makes sense to it to interpret the mouse signals, so that the driver can (for example) give the mouse a certain amount of inertia. Some drivers make an adjustment for skew in the signals they receive. For instance, certain drivers for pencil-tip mice on laptops will respond to the user holding the tip to the right for a few seconds by causing the cursor to drift to the left when the user releases the mouse. Other drivers update the position so frequently that there is almost certainly some software acceleration taking place (i.e. the driver is interpolating the signals it receives from the mouse). If NRBG designers are relying on the mouse driver to pass mouse movements on for use as entropy, they need to be aware that the data has likely been conditioned in some way, and they need to build their entropy model accordingly.

8.4.3 System Loading

John sent me some stuff on this, which I will incorporate.

8.4.4 Ring Oscillator

Need to add some text here.

9 Validation

9.1 Validation for NRBG in General

Security-relevant branches in the code that govern behavior in exceptional conditions (e.g., initialization, failed health tests, etc.) **shall** be validated by forcing all error conditions to occur during validation testing.

9.2 Validation for Basic NRBG

The validation requirements for a Primitive NRBG are much more stringent than those for a Standard NRBG. This reflects the fact that the Primitive NRBG is relying much more heavily on the entropy source to work properly (and on the health tests to catch it when it doesn't). When this is contrasted with the mathematical guarantees provided by a Standard NRBG (which only depends on a sporadically functional entropy source and properly functioning deterministic components to provide a high level of fallback security), it's reasonable to expect that a Primary NRBG will require a more thorough validation process.

Comment [ebb18]: I don't think this is what we want to say.

9.3 Validation for Enhanced NRBG

Use Approved NRBG. Testing of Entropy Sources……

Appendix 1 Design Considerations for Standard NRBGs

This appendix describes some of the issues that an NRBG designer must consider when integrating a DRBG into the NRBG.

Section 8.2 provided two goals for an NRBG, namely that it should offer infinite security when the entropy source is operating correctly, and that it should default to a secure DRBG when the entropy source fails. ~~We explore these ideas further here.~~

Comment [ebb19]: Fix the reference.

Comment [ebb20]: This depends on the Standard NRBG type.

An NRBG is meant to provide full entropy outputs. It might seem sufficient to process *n* bits of entropy from the source for every *n* bits of output. However, if care is not taken in integrating the DRBG into the design, a narrow pipe might be introduced that would throttle the entropy provided by the source

NEED PICTURE.

In this example, the entropy source is sampled until tests determine that it has provided 352 bits of entropy to the internal state. Then the NRBG will output as many as 352 bits. Since the hash function only has a 160-bit chaining value, there is a narrow pipe that prevents such an output from having more than 160 bits of entropy. Care must be taken at design time to prevent this from happening. (Note: Needs work)

Discussion of alternate NRBG configuration.

Minimizing Reliance on Crytpo Algorithm Properties.