DRAFT NIST Special Publication 800-90C

Recommendation for Random Bit Generator (RBG) Constructions

Elaine Barker

John Kelsey

Computer Security Division

Information Technology Laboratory

C O M P U T E R   S E C U R I T Y

August 2012

U.S. Department of Commerce
Rebecca Blank, Acting Secretary

National Institute of Standards and Technology
Patrick D. Gallagher
Under Secretary for Standards and Technology and Director

NIST **National Institute of Standards and Technology** • Technology Administration • U.S. Department of Commerce

Abstract

This Recommendation specifies constructions for the implementation of random bit generators (RBGs). An RBG may be a deterministic random bit generator (DRBG) or a non-deterministic random bit generator (NRBG). The constructed RBGs consist of DRBG mechanisms as specified SP 800-90A and entropy sources as specified in SP 800-90B

KEY WORDS: deterministic random bit generator (DRBG), entropy, entropy source, non-deterministic random bit generator (NRBG), random number generator, source of entropy input.

Acknowledgements

The National Institute of Standards and Technology (NIST) gratefully acknowledges and appreciates contributions by Mary Baish and Mike Boyle from the National Security Agency for assistance in the development of this Recommendation. NIST also thanks the many contributions by the public and private sectors.

Table of Contents

Authority

Conformance testing for implementations of this Recommendation will be conducted within the framework of the Cryptographic Algortihm Validation Program (CAVP) and the Cryptographic Module Validation Program (CMVP). The requirements of this Recommendation are indicated by the word "**shall**." Some of these requirements may be out-of-scope for CAVP or CMVP validation testing, and thus are the responsibility of entities using, implementing, installing or configuring applications that incorporate this Recommendation.

Random Bit Generator (RBG) Constructions

1 Scope

Cryptography and security applications make extensive use of random numbers and random bits. However, the generation of random bits is problematic in many practical applications of cryptography. The purpose of this Recommendation is to specify **approved** random bit generators (RBGs). By matching the security requirements of the application using the random bits with the security claims of the RBG generating those bits, an application can safely use the random bits produced by an RBG conforming to this Recommendation.

NIST Special Publications (SPs) 800-90A and SP 800-90B have addressed the components of RBGs:

- SP 800-90A, *Random Number Generation Using Deterministic Random Bit Generator Mechanisms*, specifies several Deterministic Random Bit Generator (DRBG) mechanisms containing **approved** cryptographic algorithms.
- SP 800-90B, *Entropy Sources*, provides guidance for the development and validation of entropy sources – mechanisms that provide truly unpredictable bits from some nondeterministic process.

SP 800-90C specifies the construction of **approved** RBGs using the DRBG mechanisms and entropy sources from SP 800-90A and SP 800-90B. SP 800-90C is essentially a partial extraction from American National Standard (ANS) X9.82, Part 4, which specifies constructions for an RBG, and constructions for building components that are used within those RBG constructions; for more detailed information, see Part 4 of ANS X9.82.

Throughout this document, the term “this Recommendation” refers to the aggregate of SP 800-90A, SP 800-90B and SP 800-90C.

The information in SP 800-90C is intended to be combined with the information in SP 800-90A and SP 800-90B in order to:

- Construct an RBG with the required security properties, and
- Verify that an RBG has been constructed in compliance with this Recommendation.

The precise structure, design and development of an RBG is outside the scope of this Recommendation.

2 Terms and Definitions

Algorithm
A clearly specified mathematical process for computation; a set of rules that, if followed, will give a prescribed result.

Approved
FIPS-approved or NIST-recommended.

Approved DRBG
A DRBG implementation that is compliant with SP 800-90A and SP 800-90C.

Approved entropy source
An *entropy source* implementation that is compliant with SP 800-90B.

Backtracking resistance
An RBG provides *backtracking resistance* relative to time *T* if it provides assurance that an adversary that has knowledge of the state of the RBG at some time(s) subsequent to time *T* (but incapable of performing work that matches the claimed security strength of the RBG) would be unable to distinguish between observations of *ideal random bitstrings* and (previously unseen) bitstrings that are output by the RBG at or prior to time *T*. In particular, an RBG whose design allows the adversary to "backtrack" from the initially-compromised RBG state(s) to obtain knowledge of prior RBG states and the corresponding outputs (including the RBG state and output at time *T*) would not provide backtracking resistance relative to time *T*. (Contrast with *prediction resistance*.)

Behavior test
A test performed on parts of an implementation to ensure that the implementation is behaving as expected, usually by testing to determine that outputs lie within an acceptable range. For this Recommendation, behavior tests are performed on the noise source within the entropy source; see SP 800-90B.

Big-endian format
The most significant bytes (the bytes containing the high order or leftmost bits) are stored in the lowest address, with the following bytes in sequentially higher addresses.

Bits of security
See *security strength*.

Bitstring
An ordered sequence of 0's and 1's. The leftmost bit is the most significant bit in the string. The rightmost bit is the least significant bit of the string (see *big-endian format*).

Chain of RBGs (or DRBGs)

A succession of "nested" RBGs whereby the source of entropy input (SEI) for one DRBG is another DRBG, NRBG or entropy source. The initial SEI is either an NRBG or an entropy source.

Condition
To process input bitstrings in such a way that any bias observed in the bits of the resulting output will be less than some specified amount. Full-entropy output may or may not be provided, depending upon the amount of bias that remains in the output. (See *full entropy*.)

Conditioning Component
An optional component of an *entropy source* that is intended to reduce the bias of the outputs.

Construction
A specific method of designing an RBG or some component of an RBG to accomplish a stated goal.

Consuming application
An application that uses random bits (or numbers) obtained from an **approved** random bit (or number) generator.

Derivation function
A function that is used to either derive internal state values or to distribute entropy throughout a bitstring.

Deterministic Random Bit Generator (DRBG)
An RBG that includes an **approved** DRBG mechanism and (at least initially) has access to a source of entropy input. The DRBG produces a sequence of bits from a secret initial value called a seed, along with other possible inputs. A DRBG is often called a Pseudorandom Number (or Bit) Generator. (Contrast with a *Non-deterministic random bit generator* (NRBG)).

DRBG mechanism
The portion of an RBG that includes the functions necessary to instantiate and uninstantiate a DRBG, generate pseudorandom bits, (optionally) reseed the DRBG and test the health of the DRBG mechanism. **Approved** DRBG mechanisms are specified in SP 800-90A.

Enhanced NRBG
An RBG that relies on the entropy source and an **approved** DRBG mechanism to provide full-entropy output. (Contrast with a *DRBG*.)

Entropy
See min-entropy.

Entropy input
An input bitstring that provides an assessed minimum amount of unpredictability for a DRBG mechanism. (See *min-entropy*.)

Entropy source
A process or mechanism that produces unpredictable digital data. In particular, the data output by the entropy source are considered to be observations of a random variable. There is no assumption that the unpredictable data has a uniform distribution. ***Approved*** *entropy sources* are described in SP 800-90B. (Contrast with the *source of entropy input*.)

Externally-conditioned entropy source
The combination of an entropy source and a conditioning mechanism that is external to that entropy source. The entropy source itself may also have a conditioning component (i.e., the entropy source may be an *internally-conditioned entropy source*).

Fresh entropy
A bitstring output from an entropy source for which there is a negligible probability that it has been previously output by the entropy source and a negligible probability that the bitstring has been previously used by the RBG.

Full entropy
For the purposes of this Recommendation, an n-bit string is said to have full entropy if that bitstring is estimated to contain at least $(1-\varepsilon)n$ bits of entropy, where $0 \le \varepsilon \le 2^{-64}$. A source of full-entropy bitstrings serves as a practical approximation to a source of ideal random bitstrings of the same length. (See *ideal random sequence*.)

Full Entropy Source (FES)
An entropy source that has been internally or externally conditioned to provide *full entropy* output. (See *externally-conditioned entropy source* and *internally-conditioned entropy source*.)

Generate

To produce random or pseudorandom bits from a properly instantiated RBG.

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

Ideal random bitstring

See *ideal random sequence*.

Ideal random sequence

Each bit of an ideal random sequence is unpredictable and unbiased, with a value that is independent of the values of the other bits in the sequence. Prior to the observation of the sequence, the value of each bit is equally likely to be 0 or 1, and the probability that a particular bit will have a particular value is unaffected by knowledge of the values of any or all of the other bits. An ideal random sequence of *n* bits contains *n* bits of entropy.

Independent SEIs
Multiple *sources of entropy input* (SEIs) are independent if the probability of correctly predicting the output of any one of the SEI is unaffected by knowledge of the output of any or all other SEIs.

Instantiate
The process of initializing a DRBG with sufficient entropy to generate pseudorandom bits at the desired *security strength*.

Internally-conditioned entropy source
An *entropy source* that contains a *conditioning component* that reduces the bias (if any) in the bits of the digitized data obtained from its *noise source*. (Also see *externally-conditioned entropy source* and *full entropy source*.)

Keying material
The data (e.g., keys, certificates, and initialization vectors) necessary to establish and maintain cryptographic keying relationships.

Known-answer test
A test that uses a fixed input/output pair (where the output is the correct output from the component for that input) to detect whether a component was implemented correctly or to detect whether it continues to operate correctly.

Live Entropy Source
An ***approved*** *entropy source* (see SP 800-90B) that can provide an RBG with bits having a specified amount of entropy immediately upon request or within an acceptable amount of time, as determined by the user or application relying upon that RBG.

Min-entropy
The min-entropy (in bits) of a random variable *X* is the largest value *m* having the property that each observation of *X* provides at least *m* bits of information (i.e., the min-entropy of *X* is the greatest lower bound for the information content of potential observations of *X*). The min-entropy of a random variable is a lower bound on its entropy. Min-entropy is often used as a worst-case measure of the unpredictability of a random variable.

Nonce
A time-varying value that has at most a negligible chance of repeating; for example, a random value that is generated anew for each use, a time stamp, a sequence number, or some combination of these.

Noise Source
That component of an *entropy source* that contains the non-deterministic, entropy-producing activity.

Non-deterministic Random Bit Generator (NRBG)
An RBG that has access to an *entropy source* and (when working properly) produces output bitstrings that have *full entropy*. Often called a True Random Number (or Bit) Generator. (Contrast with a *deterministic random bit generator* (DRBG)).

Null
The empty bitstring.

Prediction resistance
An RBG provides prediction resistance relative to time *T* if it provides assurance that an adversary with knowledge of the state of the RBG at some time(s) prior to *T* (but incapable of performing work that matches the claimed *security strength* of the RBG) would be unable to distinguish between observations of ideal random bitstrings and (previously unseen) bitstrings output by the RBG at or subsequent to time *T*. In particular, an RBG whose design allows the adversary to step forward from the initially compromised RBG state(s) to obtain knowledge of subsequent RBG states and the corresponding outputs (including the RBG state and output at time *T*) would not provide prediction resistance relative to time *T*. (Contrast with *backtracking resistance*.)

Pseudocode
Text that resembles program code, but does not conform to any specific programming language.

Random Bit Generator (RBG)
A device or *algorithm* whose output bits appear to be statistically independent and unbiased (given the assumed computational capabilities of the observer).

Reseed
To acquire additional bits with sufficient entropy for the desired *security strength*.

Secure channel
A path for transferring data between two entities or components that ensures confidentiality, integrity and replay protection, as well as mutual authentication between the entities or components. The secure channel may be provided using cryptographic, physical, logical or procedural methods, or a combination thereof.

Security strength
A number associated with the amount of work (that is, the number of basic operations of some sort) that is required to "break" a cryptographic algorithm or system in some way. In this Recommendation, the security strength is specified in bits and is a specific value from the set {112, 128, 192, and 256}. If the security strength associated with an algorithm or system is *S* bits, then it is expected that (roughly) 2^S basic operations are required to break it

Seed period

The period of time between *instantiating* or *reseeding* a DRBG with one seed and reseeding that DRBG with another seed.

Source of entropy input (SEI)

A component of an RBG that outputs bitstrings that can be used as *entropy input* by a *DRBG mechanism*. An SEI may be an ***approved*** *entropy source* (see SP 800-90B), an **approved** RBG employing an **approved** entropy source to obtain entropy input for its DRBG mechanism, or a nested chain of **approved** RBGs whose initial member employs an **approved** entropy source to obtain entropy input for its DRBG mechanism.

Source RBG
An RBG that is used directly as a *source of entropy input*.

Threat model
A description of a set of security aspects that need to be considered; a threat model can be defined by defining a set of possible attacks that can be used to assess the probability of the attack, the potential harm from the attack, etc.

3 Symbols and Abbreviated Terms

The following abbreviations are used in SP 800-90C.

Symbols and Abbreviations	**Meaning**
ANS	American National Standard
CTR_DRBG	A DRBG specified in SP 800-90A that is based on block cipher algorithms.
DRBG	Deterministic Random Bit Generator
FES	Full Entropy Source
FIPS	Federal Information Processing Standard
HMAC_DRBG	A DRBG specified in SP 800-90A that is based on HMAC.
NIST	National Institute of Standards and Technology
NRBG	Non-deterministic Random Bit Generator
RBG	Random Bit Generator
RNG	Random Number Generator
SEI	Source of Entropy Input
SP	Special Publication
XOR	Boolean Bitwise Exclusive-Or

The following symbols and function calls are used in SP 800-90C.

Symbol	Meaning
+	Addition.
$X \oplus Y$	Boolean bitwise exclusive-or (also bitwise addition modulo 2) of two bitstrings X and Y of the same length.
$X \,\|\|\, Y$	Concatenation of two bitstrings X and Y.
XOR	Boolean Bitwise Exclusive-Or.
0^x	A string of x zero bits.
df	A derivation function specified in SP 800-90A.
FullEntropySource_Get	A request made to an internally or externally-conditioned entropy source in order to obtain a bitstring having full entropy.
Generate_function	A request made to a DRBG in order to obtain pseudorandom bits.
GetEntropy	A request made to an entropy source in order to obtain a bitstring containing entropy.
Get_entropy_input	A request made to an SEI in order to obtain input.
Instantiate_function	The function used to create a DRBG instantiation.
length_in_bits	The length (in bits) of a bitstring.
MakeNextNonce	A request for a nonce to be provided.
NRBG_Generate	A request made to an NRBG to generate random bits.
NRBG_Instantiate	A request made to an NRBG in order to instantiate any DRBGs used by the NRBG.
Reseed_function	A request made to a DRBG in order to reseed a DRBG instantiation.

4 General Discussion

An RBG that conforms to this Recommendation produces random bits for a consuming application. The security of the RBG depends on:

- A deterministic process; the RBGs currently specified in this SP 800-90C include DRBG mechanisms as discussed and specified in SP 800-90A, and
- A source of non-deterministic outputs, i.e., an entropy source as specified in SP 800-90B.

There are two broad classes of RBGs specified in SP 800-90C: Non-deterministic Random Bit Generators (NRBGs) and Deterministic Random Bit Generators (DRBGs). DRBGs are divided into two types: those that can provide prediction resistance, and those that cannot. The choice of using an NRBG or DRBG may be based on the following:

- NRBGs provide full entropy output. See Section 5.2 for a discussion of full entropy. See Sections 5.7 and 9 for discussions of NRBGs.
- DRBGs provide output that cannot be distinguished from an ideal random sequence without an infeasible amount of computational effort. When designed and used as specified in this Recommendation, DRBGs have a fixed (finite) security strength, which is a measure of the amount of work required to defeat the security of the DRBG. A DRBG with an *s*-bit security strength is suitable for providing keying material and other random values to any cryptographic mechanism that claims no higher than the *s*-bit security strength. See Sections 5.6 and 8 for discussions of DRBGs. DRBG mechanisms are discussed in SP 800-90A.
- Applications using DRBGs may require that the DRBG be capable of periodically reseeding itself in order to thwart a possible compromise of the DRBG or to recover from an actual compromise. Reseeding is accomplished by the insertion of fresh entropy into the DRBG using an entropy source. When the entropy source is available on-demand (see discussions of Live Entropy Sources in Section 5.4), a DRBG is said to support prediction resistance. See Section 5.5 for further discussion of prediction resistance.
- A DRBG without prediction resistance cannot recover from a compromise until it has access to an entropy source, and fresh entropy can be obtained. However, the DRBG can still be used for many applications where an on-demand entropy source and immediate resetting are not required.

SP 800-90C provides constructions for designing and implementing DRBGs and NRBGs from components specified in SP 800-90A and SP 800-90B. A construction is a method of designing an RBG or some component of an RBG to accomplish a specific goal. One or more of the constructions provided herein **shall** be used in the design of an RBG that conforms to this Recommendation.

4.1 RBG Security

Any failure of the components of an RBG could affect its security. Any RBG designed to comply with this Recommendation will function at the designed security strength only if the following requirements are satisfied.

1. Entropy sources **shall** comply with SP 800-90B.
2. Entropy sources **shall** provide the requested amount of entropy whenever the entropy source is functioning correctly; if the requested amount of entropy cannot be provided, the RBG **shall** fail (see Section 11.1.3). In particular, if an entropy source is a Full Entropy Source, the output has full entropy.
3. DRBG mechanisms **shall** comply with SP 800-90A.
4. Every DRBG **shall** be instantiated using an appropriate source of entropy input (see Section 7).
5. RBG boundaries **shall** include mechanisms that either detect or prevent access to RBG components from outside the boundary with respect to a specific threat model (see Section 5.1).
6. Bitstrings containing entropy **shall** only be used once.

4.2 Assumptions

The RBG constructions in SP 800-90C are based on the following assumptions:

1. An entropy source output string containing $2n$ bits of entropy can be conditioned into a string of n bits that contains full entropy output[1] using an **approved** conditioning function, where n is the length of the output block of the **approved** derivation function (see SP 800-90B). Note that in SP 800-90C, a derivation function[2] is used as a conditioning function.
2. A DRBG that has been instantiated at a given *security_strength* can be used to produce *security_strength*/2 bits of full entropy output if constructed as specified herein. Specifically,
    - A DRBG can be used to externally condition output from an entropy source (see the Oversampling construction in Section 9.3).
    - When a DRBG is reseeded at a security strength of s bits, the first $s/2$ bits generated from that DRBG will contain full entropy.
3. The DRBG mechanisms specified in SP 800-90A meet their explicit security claims (e.g., backtracking resistance, claimed security strength, etc.).
4. The derivation functions in SP 800-90A distribute the entropy in the input string across the corresponding output string. If the entropy provided by the input string is n bits (so

[1] Under rare circumstances, it is possible that a bit of entropy could be lost, producing an output with slightly less than full entropy.

[2] **Approved** derivation functions for RBG constructions are provided in SP 800-90A.

that the length of the input string is r bits for some $r \geq n$), and the length of the output string is t bits, then the following is true:

- If $t \leq n/2$, then the output string has full entropy output (i.e., the output string has $(1-\varepsilon)t$ bits of entropy, for some $\varepsilon \leq 2^{-64}$).
- If $n/2 < t \leq n$, then the output string has at most t bits of entropy, but is not guaranteed to have full entropy.
- If $t > n$, then the output string has at most n bits of entropy.

4.3 Document Organization

The remainder of SP 800-90C describes how to construct an RBG from the components described in SP 800-90A and SP 800-90B.

Section 5 provides RBG concepts, such as RBG boundaries and distributed RBGs; full entropy, full entropy sources and live entropy sources; backtracking and prediction resistance; and introductory discussions on DRBGs and NRBGs.

Section 6 describes the conceptual interface calls used in SP 800-90C.

Section 7 provides an overview of the sources of entropy input (SEI) to be used by a DRBG.

Sections 8 and 9 provide guidance for constructing DRBGs and NRBGs, respectively.

Section 10 provides constructions for implementing a DRBG's **Get_entropy_input** call using various sources of entropy input (SEIs) and for external conditioning of entropy source output.

Section 11 discusses testing, including both health testing and implementation-validation testing.

Appendix A contains diagrams of basic RBG configurations.

Appendix B identifies the requirements in this Recommendation that are the responsibility of entities using, installing or configuring applications or protocols that incorporate RBGs.

Appendix C contains discussions on post-processing of RBG output.

Appendix D contains a list of references.

Additional material is addressed in American National Standard X9.82, Part 4, including expanded explanations and:

- A step-by step description for constructing an RBG,
- Obtaining entropy from entropy sources that are only available intermittently,
- Guidance on combining RBGs,
- Detailed RBG examples, and
- Security and implementation considerations.

5 Random Bit Generator Concepts

5.1 RBG Boundaries and Distributed RBGs

An RBG **shall** exist within a conceptual RBG security boundary that is defined with respect to one or more threat models, which include an assessment of the probability of an attack and the potential harm caused by the attack. The RBG boundary **shall** be designed to assist in the mitigation of these threats using either physical or logical mechanisms or both.

An RBG boundary **shall** contain all components required for the RBG. Data **shall** enter an RBG only via the RBG's public input interface(s) (if any) and **shall** exit only via its public output interface(s). Figure 1 illustrates the primary components of an RBG: an entropy source, a DRBG mechanism and health tests for the RBG. The boundary for a DRBG mechanism is discussed in SP 800-90A. Boundaries for the noise source within an entropy source, and the entropy source boundary itself are discussed in SP 800-90B. Both the entropy source and the DRBG mechanism contain their own health tests within their respective boundaries; the health tests shown in Figure 1 pertain to those tests required for testing the health of the RBG as a whole.

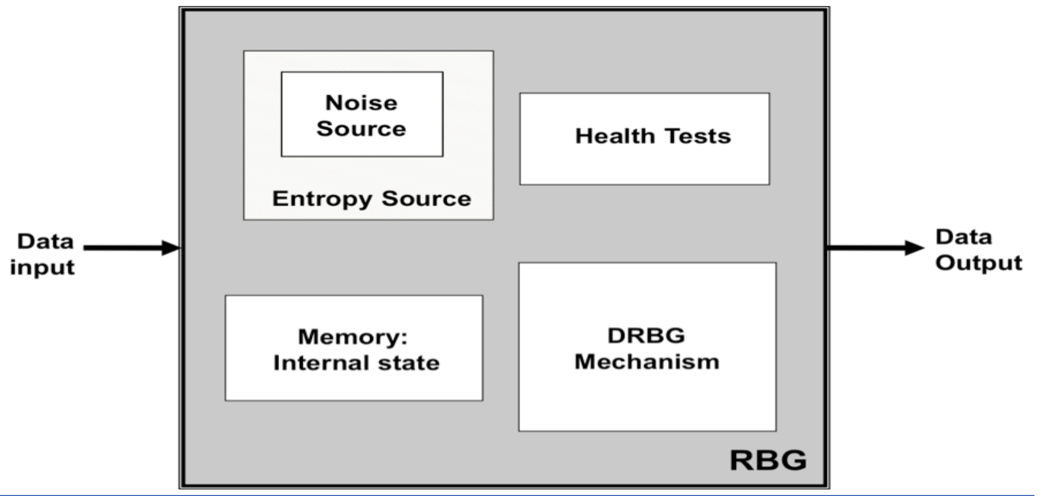

Figure 1: Security Boundary for a Single Device

An RBG may be contained within a single device (see Figure 1) or may be distributed across multiple devices (see an example in Figure 2). In the latter case, each device **shall** have a sub-boundary that contains the RBG component(s) within that device. The RBG component(s) within each sub-boundary **shall** be protected using either physical or logical mechanisms (or both). Test functions **shall** be provided within each sub-boundary to test the health of the RBG component(s) within that sub-boundary. Communications between the sub-boundaries **shall** use reliable secure channels that protect the confidentiality and integrity of the data transferred between the sub-boundaries. The boundary for a distributed RBG encapsulates the contents of the sub-boundaries, as well as the secure channels. The security provided by a distributed RBG is no more than the security provided by the secure channel(s).

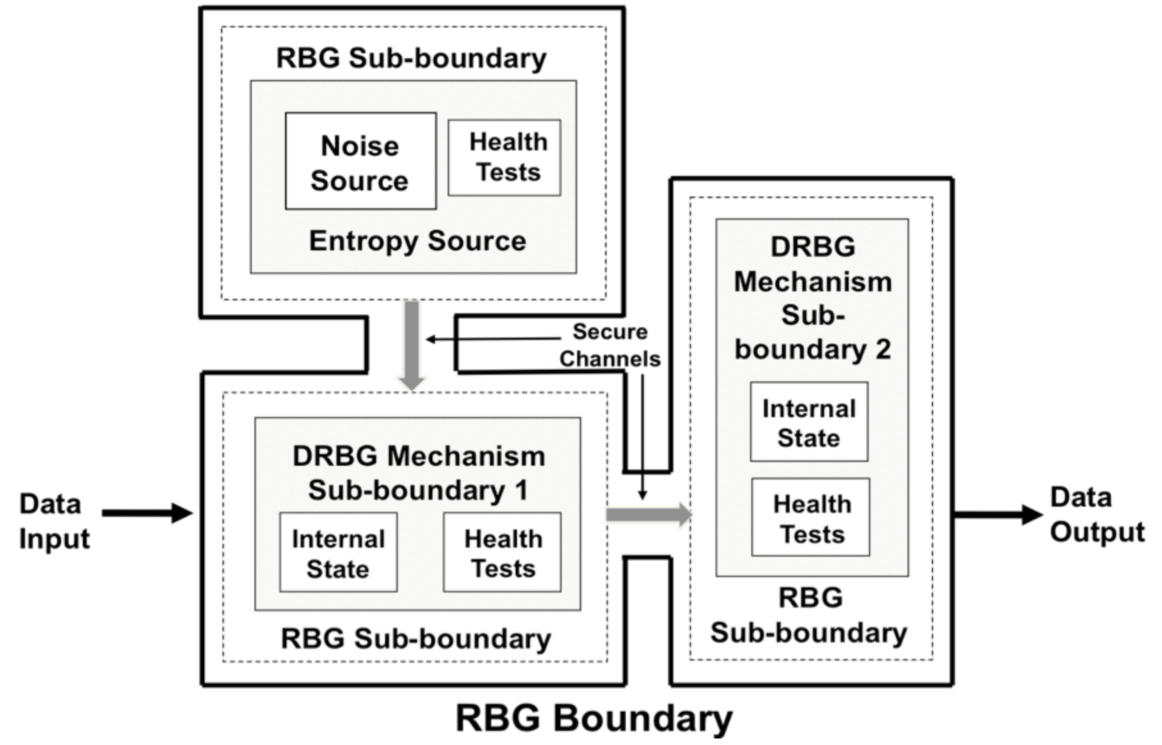

Figure 2: Distributed RBG

In the example in Figure 2, the entropy source and its noise source are contained within a single RBG sub-boundary, while the DRBG mechanism is distributed across other sub-boundaries (see SP 800-90A for further discussion of a distributed DRBG mechnism boundary).

When an RBG uses cryptographic primitives (e.g., an **approved** hash function), other applications either inside or outside of the RBG's boundary or sub-boundary may use the same implementation of that cryptographic primitive as long as the RBG's output and internal state are not modified or revealed by this use.

5.2 Full Entropy

Each bit of a bitstring with full entropy is unpredictable (with a uniform distribution) and independent of every other bit of that bitstring. For the purposes of this Recommendation, an *n*-bit string is said to have full entropy if the string is estimated to contain at least $(1-\varepsilon)n$ bits of entropy, where $\varepsilon \leq 2^{-64}$. Informally, this means that a bitstring has full entropy if the amount of entropy is essentially the same as its length.

5.3 Entropy Sources and Full Entropy Sources

SP 800-90B discusses entropy sources. An entropy source may optionally contain a conditioning component, which is responsible for reducing bias in the noise source output bits (i.e., the conditioning component may not provide full entropy output).

An internally-conditioned entropy source is an entropy source with a conditioning component (i.e., the conditioning of the noise source output is performed within the entropy source).

The output of an entropy source may be conditioned externally, either instead of or in addition to any conditioning performed within the entropy source itself. If external conditioning is performed, it **shall** be accomplished using the construction in ANS X9.82, Part 4 or using a DRBG mechanism as specified for the Oversampling Construction in Section 9.3.

An entropy source that provides full entropy output is called a Full Entropy Source (FES); the entropy source may require internal or external conditioning to achieve this property.

Output returned from an entropy source or FES will normally be used by its encompassing RBG (i.e., the RBG defined by the RBG boundary containing the entropy source or FES). However, the output may be delivered outside the RBG (e.g., when providing entropy input for a separate RBG). Such output **shall not** itself be used as random output. In addition, this output **shall not** also be used by the encompassing RBG.

5.4 Live Entropy Source

A Live Entropy Source is an **approved** entropy source that can provide the requested amount of entropy immediately or within an acceptable amount of time as determined by the user or application requesting random bits from an RBG. A Live Entropy Source provides fresh entropy, which is required for an RBG to provide full entropy outputs or outputs with prediction resistance, as well as to instantiate the initial DRBG in a DRBG chain. A DRBG chain is a succession of "nested" RBGs whereby the source of entropy input (SEI) for one DRBG is another DRBG, NRBG or entropy source. The initial SEI is either an NRBG or an entropy source. See Section 7 for further discussion.

A Live Entropy Source can be used to support any security strength using an appropriate construction as specified or referenced in SP 800-90C.

An NRBG always has a Live Entropy Source, but this may not be the case for a DRBG. A DRBG may use an entropy source for instantiation, but the entropy source may not be available thereafter; in this case, the entropy source is not considered to be Live.

The Live Entropy Source could be directly accessible (e.g., a DRBG has a Live Entropy Source that is always available), or it could be indirectly accessible via an RBG that has a Live Entropy Source (e.g., an NRBG, or a DRBG with a Live Entropy Source). See Sections 5.6, 5.7 and 8.1 for further discussion.

An entropy source, instead of being Live, may be available only during the instantiation of a DRBG. In this case, the DRBG cannot provide full entropy output or outputs with prediction resistance (see Sections 5.5 and 8.2).

Note that entropy sources may only be available intermittently; they are considered Live only when actually available. The use of intermittent entropy sources is addressed in ANS X9.82, Part 4.

5.5 Backtracking and Prediction Resistance

All **approved** RBGs in this Recommendation are designed to provide backtracking resistance, which means that a compromise at a given point in time will not compromise prior RBG outputs.

An RBG may also support prediction resistance, which means that a compromise in the past or present will not compromise future RBG outputs. Prediction resistance requires the availability of a properly functioning Live Entropy Source to provide fresh entropy bits; if the entropy source fails, prediction resistance cannot be provided.

All NRBGs compliant with SP 800-90C support prediction resistance. Each call to the NRBG results in fresh entropy bits, because the entropy source is always available and used (see Section 5.7).

DRBGs with access to a Live Entropy Source are capable of supporting prediction resistance when the DRBG is invoked with prediction resistance requested. In order for the DRBG to provide prediction resistance when a request is made for this service, the DRBG is reseeded (see SP 800-90A).

Full entropy output can be provided by a DRBG that supports prediction resistance by requesting *security_strength*/2 bits of output for each DRBG generate request (see assumption 1 in Section 4.2).

For a more complete discussion of backtracking and prediction resistance, see SP 800-90A.

5.6 Deterministic Random Bit Generators (DRBGs)

An RBG may be a DRBG. A DRBG consists of a DRBG mechanism and a source of entropy input (SEI). An SEI may be an entropy source tht conforms to SP 800-90B, or an RBG that is ultimately based on an entropy source that conforms to SP 800-90B. Section 7 of SP 800-90C discusses sources of entropy input (SEIs). Section 8 discusses the construction of a DRBG from these components. DRBG mechanisms are specified in SP 800-90A.

A DRBG **shall** be instantiated before it can provide pseudorandom bits using an SEI that is available at that time. If the DRBG has access to a Live Entropy Source during normal operation (either via direct access to an entropy source, or via access to the Live Entropy Source of another RBG), then the DRBG can provide prediction resistance (see Section 5.5).

5.7 Non-deterministic Random Bit Generators (NRBGs)

An RBG may be an NRBG. An NRBG provides output bits that are indistinguishable (in practice) from an ideal random sequence to any observer; that is, an NRBG can provide full entropy – a request for *n* bits of output will result in a bitstring of *n* bits, with each bit essentially providing one bit of entropy. See Section 5.2 for a more precise definition of full entropy and Section 9 for futher discussions about NRBGs.

An NRBG has access to a Live Entropy Source. Because an entropy source is always available, a properly functioning NRBG always provides fresh entropy, backtracking resistance, and prediction resistance.

In addition to a Live Entropy Source, the NRBGs specified in this Recommendation – referred to as Enhanced NRBGs – include a DRBG mechanism. If the entropy source fails without detection, the security provided by the NRBG is reduced to the security strength of the **approved** DRBG used in the NRBG construction. This assumes that the DRBG has been properly instantiated with sufficient entropy to support that security strength.

SP 800-90B discusses the handling of errors during the health testing that is performed prior to and during operation; SP 800-90B allows the handling of these errors either completely within the entropy source, or allows the provision of an error indicator to the calling application (e.g., the NRBG routine calling the entropy source). If such an error indication is provided by the entropy source, the NRBG relying on that entropy source **shall** enter an error state and discontinue operations.

5.8 Post-processing of RBG Output

The output of an RBG may be modified by an **approved** post-processing method (see Appendix C) before the output is used by a consuming application. When post-processing is performed, the output of the post-processing operation **shall** be used in place of any use of the original RBG output.

6 RBG Interfaces

6.1 General Pseudocode Conventions

All algorithms in SP 800-90C are described in pseudocode that is intended to explain the algorithm's function. These pseudocode conventions are not intended to constrain real-world implementations, but to provide a consistent notation to describe the constructions herein. By convention, unless otherwise specified, integers are 32-bit unsigned, and when used as bitstrings, they are represented in big-endian format.

6.2 RBG Function Calls

Calls can be made to an RBG without knowledge about whether the RBG is a DRBG or an NRBG. These calls consist of an **RBG_Startup** command that is used to request that the RBG activate itself and become ready to generate bits, and an **RBG_Generate** command that generates random bits upon request. Since the constructions included in SP 800-90C do not include the use of these commands, one should refer to ANS X9.82, Part 4 for more information, including constructions and examples that use these RBG calls.

6.3 DRBG Function Calls

A DRBG contains a DRBG mechanism and a source of entropy input. See SP 800-90A for more information about these functions. Note that the definitions for the input parameters and output values are provided as footnotes. The DRBG supports the following interfaces:

1. (*status*[3], *state_handle*[4]) =
   Instantiate_function (*requested_instantiation_security_strength*[5], *prediction_resistance_flag*[6], *personalization_string*[7]).
2. (*status*, *returned_bits*[8]) = **Generate_function** (*state_handle*, *requested_number_of_bits*[9], *requested_security_strength*[10], *prediction_resistance_request*[11], *additional_input*[12]).

---

[3] *status*: the status information returned from a function call.

[4] *state_handle*: the pointer to the DRBG instantiation's internal state values.

[5] *requested_instantiation_security_strength*: the security strength requested for the instantiation of the DRBG.

[6] *prediction_resistance_flag*: indicates that the DRBG mechanism may be requested in subsequent calls to provide prediction resistance.

[7] *personalization_string*: an optional string to be used for the personalization of the DRBG mechanism during its instantiation.

[8] *returned_bits*: the bitstring that is returned from the function call.

[9] *requested_number_of_bits*: the length of the bitstring to be returned from the function call.

[10] *requested_security_strength*: the minimum security strength to be provided in the RBG output.

[11] *prediction_resistance_request*: requests that the DRBG mechanism provide prediction resistance for this function.

A DRBG may optionally support the following interface:

3. *status* = **Reseed_function**(*state_handle*, *prediction_resistance_request*, *additional_input*)

Note that the use of the **Uninstantiate_function** specified in SP 800-90A is not explicitly discussed in SP 800-90C.

Internally, a DRBG mechanism makes a function call to obtain entropy input:

4. (*status*, *entropy_input*[13]) = **Get_entropy_input(***min_entropy*[14], *min_length*[15], *max_length*[16], *prediction_resistance_request*).

Two derivation functions (**df**s) are defined in SP 800-90A for use within some DRBG mechanisms. When a derivation function is required in an RBG construction in SP 800-90C, one of the derivation functions from SP 800-90A **shall** be used. They are invoked using the following call:

5. (*status*, *requested_bits*[17]) = **df**(*input_string*[18], *no_of_bits_to_return*[19]).

6.4 NRBG Function Calls

A non-deterministic random bit generator (NRBG) supports the following two interfaces. The defintion of the parameters used as input and output are the same as those used for the DRBG function calls in Section 6.3.

1. (*status*, *state_handle*) = **NRBG_Instantiate**(*personalization_string*).
2. (*status*, *returned_bits*) = **NRBG_Generate**(*state_handle*, *requested_number_of_bits, additional_input*).

An NRBG may optionally support the following interface in order to reseed the DRBG in an Enhanced NRBG construction:

3. *status* = **NRBG_Reseed**(*state_handle*, *additional_input*).

call.

12 *additional_input*: optional input provided to the DRBG mechanism during generation or reseeding.

13 *entropy_input*: a bitstring returned from the **Get_entropy-input** call.

14 *min_entropy*: the minimum amount of entropy to be provided by the **Get_entropy_input** call.

15 *min_length*: the minimum length of the output string (i.e., *entropy_input*) to be returned from the **Get_entropy_input** call.

16 *max_length*: the maximum length of the output string (i.e., *entropy_input*) to be returned from the **Get_entropy_input** call.

17 *requested_bits*:the bitstring returned from a **df** function call.

18 *input_string*: the input bitstring that is processed by the **df** function call.

19 *no_of_bits_to_return*: the number of bits to be returned from the **df** function call.

Note that some of the parameters in the above calls are optional (e.g., *additonal_input*), depending on the design.

6.5 Entropy Source Calls

An entropy source, as discussed in SP 800-90B, is a mechanism for producing bitstrings that cannot be completely predicted, and whose unpredictability can be quantified in terms of min-entropy. Direct access to the output of an entropy source may be accomplished using one or both of the following two calls, as appropriate. Note that the definitions of the input and output values is provided in footnotes.

1. Access an entropy source using the following call:

   (*status*, *entropy_bitstring*[20], *assessed_entropy*[21]) = **GetEntropy**().

   The assessment may be implicit and known to the part of the RBG that makes use of the entropy source outputs; i.e., the *assessed_entropy* need not be explicitly provided as an output.

2. Access a Full Entropy Source (FES) using the following call:

   (*status*, *entropy_bitstring*) = **FullEntropySource_Get**().

   The entropy source may be internally conditioned as specified in SP 800-90B, or may be externally conditioned using a construction in ANS X9.82, Part 4 or using the Oversampling construction specified in Section 9.3.2. The length of the *entropy_bitstring* is dependent on the design of the entropy source.

   Note that an NRBG always provides full entropy output, so it may be used in place of a full entropy source.

[20] *entropy_bitstring*: the bitstring containing entropy that is returned from the entropy source call.

[21] *assessed_entropy*: the amount of entropy returned in the bitstring (i.e., entropy_bitstring) returned from the **GetEntropy** function call.

7 Sources of Entropy Input (SEI)

In order to construct a DRBG or an Enhanced NRBG that contains a DRBG mechanism, the RBG designer **shall** construct a source of secret, random or pseudorandom input for the DRBG mechanism, i.e., a source of entropy input (SEI). An SEI is used by a DRBG mechanism to construct seed material for instantiation. The SEI may also be used to construct seed material for reseeding on demand or automatically at the end of the seed period of the DRBG mechanism and for meeting requests for prediction resistance and full entropy.

There are two primary components that may be used to construct an SEI: **approved** RBGs and **approved** entropy sources; see the constructions in Sections 10.1 and 10.2, respectively. An SEI can, in fact, be a nested chain of RBGs (see Figure 3), ultimately originating from an entropy source that may not be available after the instantiation of the initial RBG in the chain (shown in Figure 3 as the innermost RBG); i.e., there may be no entropy source available except during instantiation.

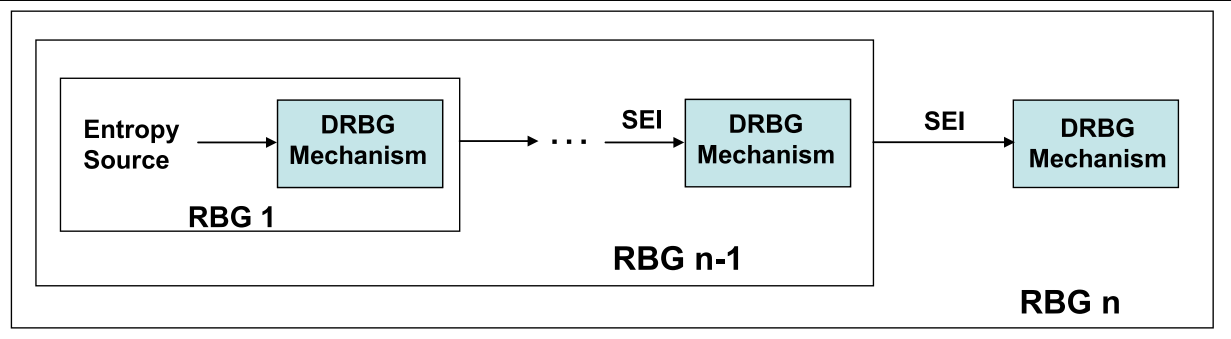

Figure 3: RBG Chain

To avoid possible confusion, a DRBG mechanism using an SEI is called the target DRBG mechanism; an SEI that is an RBG, DRBG or NRBG is referred to as the source RBG, source DRBG or source NRBG, respectively. Note that the source RBG could be either a DRBG or an NRBG. The target DRBG mechanism invokes a **Get_entropy_input** call, which is, in effect, translated to the appropriate call for the selected SEI by the nominal "interface routine" (e.g., the **Get_entropy_input** call is translated to a **Generate_function** call if a DRBG is used as the SEI).

The requirements for an SEI are determined by the needs of the consuming application and the target DRBG mechanism that is used to fulfill requests from those applications. The requirements for the SEI(s) are:

1. During instantiation, the SEI(s) **shall** support at least the security strength that is intended for the target DRBG mechanism that is using it.
    a. An RBG used as the SEI can support the security strength to be provided by the constructed target DRBG as follows:

    - If the (source) RBG is either 1) a DRBG with access to a Live Entropy Source or 2) an NRBG, then the target DRBG can be instantiated at any security strength. For example, if the desired security strength for the target DRBG is 256 bits, then a DRBG with a security strength of 128 bits can be used as the SEI when it has access to a Live Entropy Source, and the appropriate constructions are used (see the example in ANS X9.82, Part 4).
    - If the (source) RBG is a DRBG without a Live Entropy Source, then the target DRBG can be instantiated at a security strength that is less than or equal to the security strength of the source DRBG. For example, if the desired security strength for the target DRBG is 192 bits, then the (source) DRBG must be instantiated at a security strength of at least 192 bits.

   b. An **approved** entropy source can support any desired security strength when used as an SEI.

2. If the target DRBG is intended to allow reseeding, either on-demand or at the end of the DRBG's seed period, then the SEI **shall** be available on-demand.

   a. A (source) RBG that is available on-demand and is either 1) a DRBG with access to a Live Entropy Source or 2) an NRBG can be used to reseed the target DRBG at any security strength.

   b. A (source) RBG that is available on-demand and is a DRBG without a Live Entropy Source can be used to reseed the target DRBG at a security strength that is less than or equal to the security strength of the source DRBG.

   c. An **approved** entropy source that is available on-demand can be used to reseed the target DRBG at any security strength.

3. If the target DRBG is intended to support requests for prediction resistance, then an SEI that has access to a Live Entropy Source **shall** be used.

   a. An RBG that is used as the SEI can support prediction resistance if the RBG is available on-demand and is either a DRBG with access to a Live Entropy Source or an NRBG.

   b. An **approved** entropy source that is available on-demand (i.e., the entropy source is a Live Entropy Source) can be used to support prediction resistance when used as an SEI.

4. If the target DRBG is not required to be reseeded or to support prediction resistance, than the SEI is not required to be available after instantiation.

5. If the SEI is not within the same sub-boundary as the target DRBG, then a secure channel **shall** be used to transfer data from the SEI to the target DRBG.

6. If the **CTR_DRBG** is used as the target DRBG mechanism (see SP 800-90A), and a derivation function will not be used, then the SEI used by the **CTR_DRBG shall** be 1) an FES, 2) an NRBG or 3) a DRBG that can support the security strength intended for the **CTR_DRBG**.

8 DRBG Construction

A DRBG is constructed from a DRBG mechanism and an SEI. DRBG mechanisms are specified in SP 800-90A, and examples of DRBGs are provided in Appendix B of that document. Section 10 of this document (i.e., SP 800-90C) provides constructions to access the appropriate SEI from the DRBG's **Get_entropy_input** call.

As shown in Figure 4 (below), the DRBG's source of entropy input (SEI) could be a DRBG, an NRBG or an entropy source. Note that a source DRBG could be a chain of DRBGs (see Section 7). A chain of nested DRBGs consists of a target DRBG and one or more higher-level DRBGs that serve as the source for the target DRBG. If fresh entropy is provided during normal operation of the DRBG chain, the fresh entropy **should** be provided to the initial DRBG in the chain so that all subsequent DRBGs in the chain benefit from the new entropy.

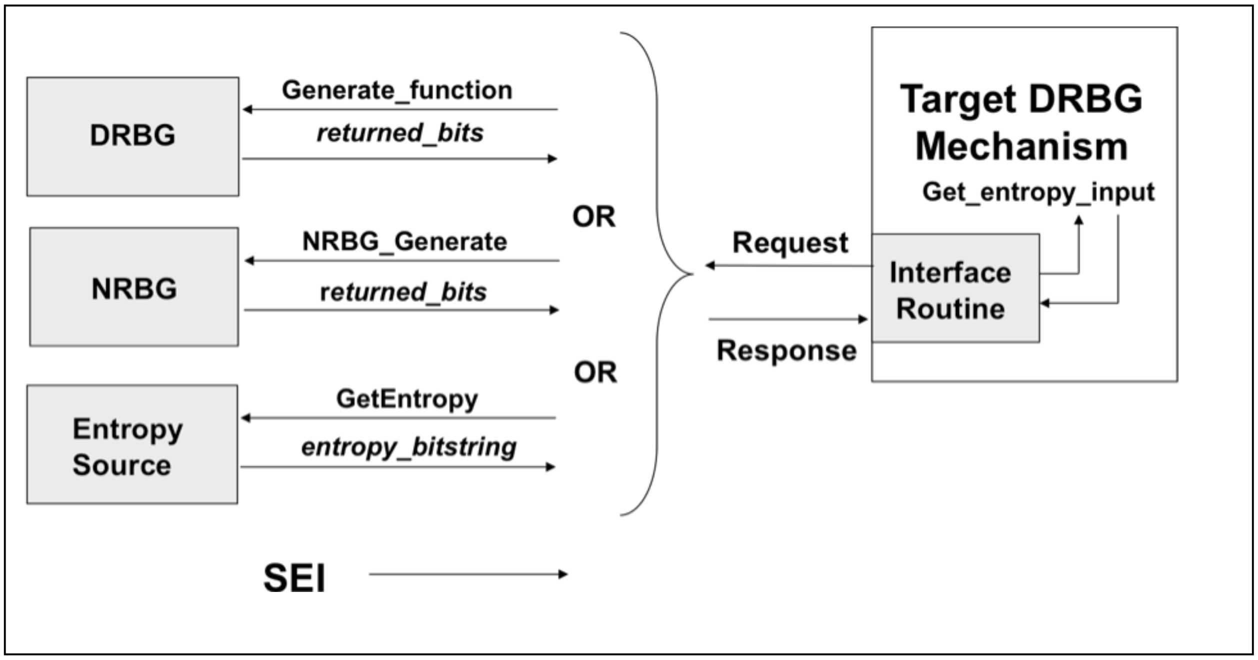

Figure 4: SEIs for a DRBG

SP 800-90C uses the concept of a DRBG "supporting a security strength". A DRBG directly or indirectly <u>supports</u> a given security strength *s* if either:

- The DRBG has been instantiated at a security strength that is equal to or greater than *s*, or
- The DRBG has access to a Live Entropy Source (i.e., either the DRBG's SEI is a Live Entropy Source, or
- The SEI has access to a Live Entropy Source through a chain of RBGs (see Section 7)).

Note that the security strength for a DRBG is set during the instantiation process, and is recorded in the internal state for that instantiation.

8.1 DRBGs with Live Entropy Sources

A DRBG with a Live Entropy Source (see Section 5.4) can be instantiated and reseeded, and provide prediction resistance whenever required. The initial DRBG in any chain of DRBGs **shall** be instantiated using either an **approved** entropy source, or an **approved** NRBG. If the entropy source or NRBG is no longer available at some time after the instantiation of a DRBG or chain of DRBGs, then the discussions in Section 8.2 apply, since the entropy source or NRBG is no longer Live.

As shown in Figure 5 (below), prediction resistance may be requested by a target DRBG mechanism (e.g., the DRBG called by a consuming application or another DRBG mechanism) in the following ways:

1. By an entropy source that may (or may not) have a conditioning component,
2. By an entropy source that is known to provide full entopy output,
3. By an entropy source whose output is passed through an external conditioning function,
4. By an NRBG, and
5. By a DRBG with access to a Live Entropy Source. Note that the (source) DRBG is not the (target) DRBG; i.e., the target DRBG **shall not** be self seeding[22].

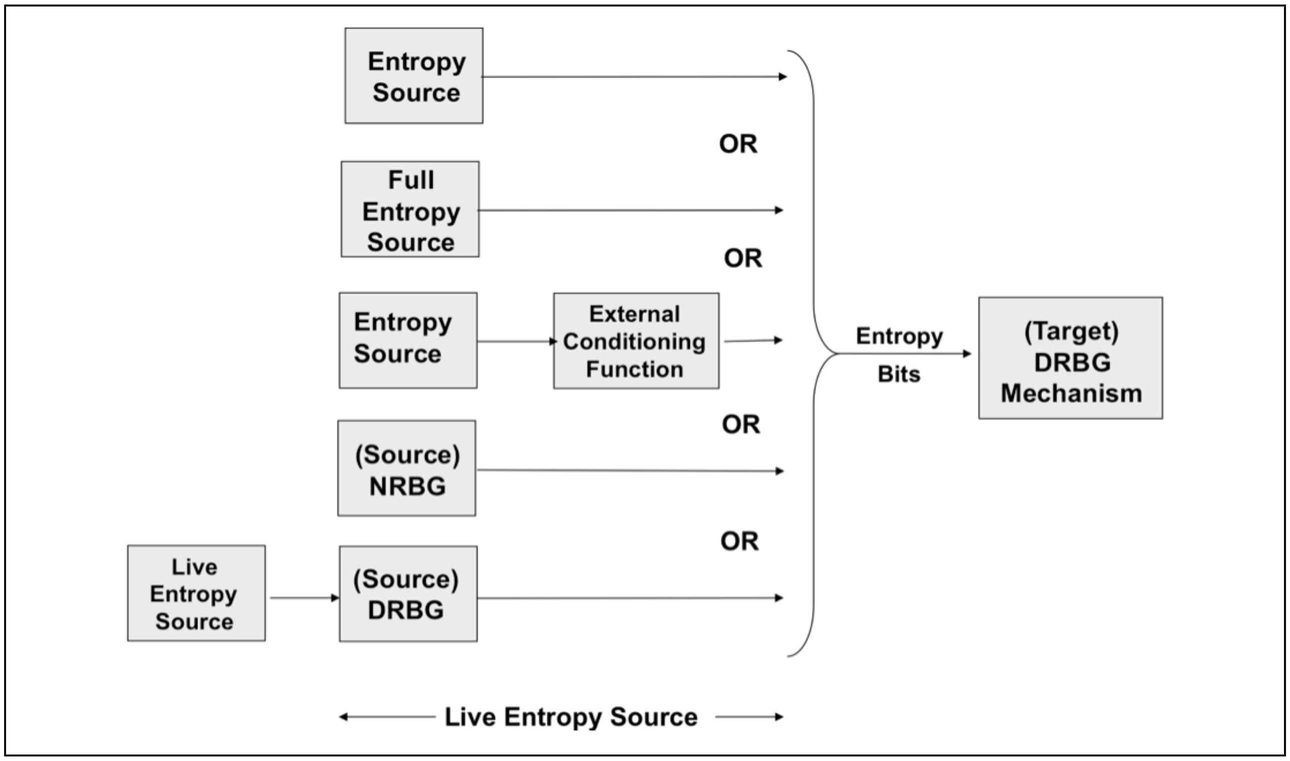

Figure 5: DRBG with a Live Entropy Source

A high-level example of a DRBG with a Live Entropy Source is provided in Apendix A.4.

[22] Prediction resistance is provided by reseeding a DRBG instantiation at the security strength set for that instantiation.

8.2 DRBGs without Live Entropy Sources

A (target) DRBG without access to an entropy source after instantiation (i.e., the DRBG does not have a Live Entropy Source) cannot provide prediction resistance. However, the DRBG can be reseeded if:

- The (target) DRBG has access to a source DRBG with a security strength that meets or exceeds the security strength of the target DRBG; note that since the target DRBG does not have access to a Live Entropy Source, the source DRBG does not have such access either. The target DRBG **shall not** be the same instantiation as the source DRBG; i.e., the target DRBG instantiation **shall not** reseed itself. However, the target DRBG and source DRBG may be different instantiations of the same DRBG implementation (e.g., the two instantiations use the same software, but have different internal states).
- An entropy source or NRBG is made available to the target DRBG expressly for the purpose of reseeding.

A high-level example of a DRBG without a Live Entropy Source is provided in Appendix A.3.

8.3 Sources of Other DRBG Inputs

Fully implementing a DRBG requires a decision about the inclusion of nonces, personalization strings, and additional input, as well as how this information will be obtained.

Nonces:

In the case of the nonces specified in SP 800-90A, if the nonce is not provided by the implementation environment, then it **shall** be provided by the SEI. See SP 800-90A for further discussion.

Personalization strings:

Personalization strings are optional input parameters during DRBG instantiation that may be used to differentiate between instantiations. If possible, the DRBG implementation **should** allow the use of a personalization string. Details on personalization strings are provided in SP 800-90A.

Additional input:

SP 800-90A allows *additional_input* to be provided by a consuming application during the **Generate_function** and **Reseed_function** requests. RBG designers **should** include this option in the selected DRBG mechanism.

9 NRBG Construction

An NRBG is a mechanism for producing bits with full entropy. These bits are expected to be indistinguishable (in practice) from an ideal random sequence to any adversary. As stated in Section 5.7, this document provides constructions for Enhanced NRBGs. The following two constructions are provided:

- XOR Construction—This NRBG is based upon combining the output of a Full Entropy Source (see Section 5.3) with the output of an instantiated DRBG using an exclusive-or (XOR) operation (see Section 9.2).
- Oversampling Construction – This NRBG is based upon using an entropy source that provides entropy input for a constructed DRBG so that full entropy is provided (see Section 9.3).

The advantages of using Enhanced NRBGs include the following:

- If the underlying DRBG mechanism in the Enhanced NRBG has been instantiated securely, and the entropy source fails in an undetected manner, the NRBG will automatically downgrade to providing outputs at the security strength of the DRBG instantiation (the "fall-back" security strength), rather than providing outputs with full entropy, i.e., the NRBG will then act as a DRBG.
- Small deviations in the behavior of the entropy source in an Enhanced NRBG will be masked by the DRBG output.

In all DRBG and Enhanced NRBG constructions in SP 800-90C, an entropy source that deviates just slightly from correct behavior leads to a very small security impact; the DRBG mechanisms mask any misbehavior, and the practical impact is a small decrease in the expected work to guess the DRBG's internal state and the NRBG's output.

Examples of Enhanced NRBGs are provided in ANS X9.82, Part 4.

9.1 The DRBG Mechanism within the NRBG

In the NRBG constructions specified in SP 800-90C, the DRBG mechanism used by the NRBG **shall** be instantiated at the highest possible security strength that is consistent with its cryptographic components and the security strengths supported by this Recommendation (i.e., 112, 128, 192, 256 bits). For example, if the **HMAC_DRBG** specified in SP 800-90A is used as the DRBG mechanism, the highest possible security strength cannot exceed the length of the output block of the hash function used. If SHA-1 is used as the hash function for the **HMAC_DRBG**, the output block is 160 bits in length, and the highest security strength possible is 128 bits (note that a security strength of 160 is not used for the Federal government, so a security strength of 128 bits is the highest that can be supported by SHA-1). However, if SHA-256 is used by the **HMAC_DRBG**, the DRBG can be instantiated at 256 bits, since the output length of SHA-256 is 256 bits, and 256 bits is a supported security strength.

The DRBG mechanism included in the NRBG may be implemented to be directly accessed by a consuming application. Direct requests to the DRBG mechanism may use either the same instantiation used by the NRBG, or a separate instantiation may be used. The DRBG instantiation(s) **shall** be used as discussed in Section 8.

If a separate instantiation is used for direct DRBG access, the separate instantiation may be instantiated at any security strength supported by the cryptographic components and this Recommendation, rather than at the highest security strength, as required by the NRBG constructions. For example, a DRBG based on SHA-1 could be instantiated at 128 bits for the instantiation used for the NRBG, and at 112 bits for the instantiation used for direct access. When a separate instantiation is used, the SEI may be any SEI discussed in Section 7, including the entropy source of the NRBG.

9.2 Construction: Enhanced NRBG – XOR Construction

The XOR Construction is shown in Figure 6; a high-level example is provided in Appendix A.1.

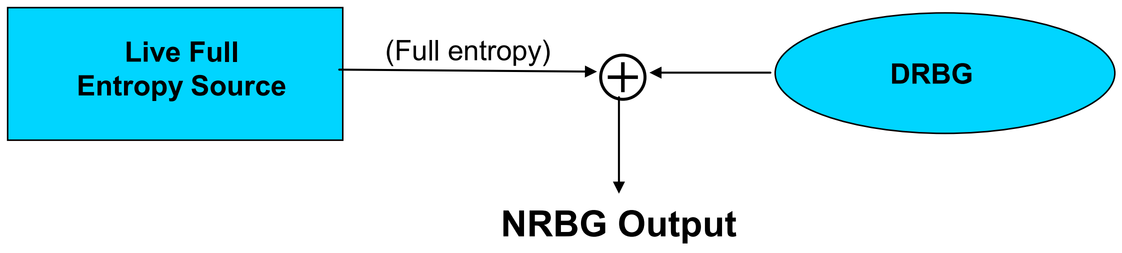

Figure 6: Enhanced NRBG – XOR Construction

For the XOR Construction:

- A Live Entropy Source that provides full entropy output (i.e., a Full Entropy Source) **shall** be used. This requires one of the following:
  - An entropy source that provides full entropy output (e.g., after internal conditioning), or
  - An entropy source as specified in SP 800-90B (with or without internal conditioning) that is externally conditioned as specified in ANS X9.82, Part 4, or
  - An NRBG designed as specified for the Oversampling Construction in Section 9.3.
- A DRBG that accesses an SEI for instantiation and reseeding **shall** be used (see Section 7). The DRBG mechanism **shall** be subject to the normal reseeding requirements of a DRBG using an available Live Entropy Source (e.g., the NRBG's entropy source).
- The SEI bits that are used as input to the DRBG **shall not** be used for any other purpose (e.g., as bits within the NRBG construction that are XORed with the output of the DRBG to produce the NRBG output for a consuming application)[23].

[23] This follows the general rule that bits conaining entropy must only be used once (see Section 4.1). Thus, entropy bits used to seed or reseed the DRBG, and entropy source output to be XORed into the DRBG outputs for this

9.2.1 NRBG Instantiation

The DRBG instantiation used in the NRBG **shall** be instantiated at its highest security strength. Let *highest_DRBG_security_strength* be the highest security strength that the DRBG mechanism can assume.

NRBG_Instantiate:

```
**Input:** string *personalization_string*.

**Output:** string *status*, integer *state_handle*.

**Process:**

    1. (status, state_handle) = Instantiate_function(highest_DRBG_security_strength,
       personalization_string).

    2. Return (status, state_handle).
```

The *personalization_string* is an optional parameter to the **NRBG_Instantiate** call. If provided in the **NRBG_Instantiate** call, it **shall** be passed to the **Instantiate_function** call, if the DRBG mechanism can handle it.

The returned *state_handle* is used for subsequent access to the DRBG instantiation.

9.2.2 NRBG Generation

Let *highest_DRBG_security_strength* be the highest security strength that the DRBG mechanism can assume, let *n* be the requested number of bits, and let the *state_handle* be the value returned from the **NRBG_Instantiate function** (see Section 9.2.1).

NRBG_Generate:

```
Input: integer (state_handle, n), string additional_input.

Output: string status, bitstring returned_bits.

Process:

    1. tmp =Null.

    2. sum = 0.

    3. While (sum < n)

        3.1 (status, entropy_bitstring) = FullEntropySource_Get ().

        3.2 If status indicates an error, then return (status, Null).

        3.3 tmp = tmp || entropy_bitstring.

        3.4 sum = sum + length_in_bits(entropy_bitstring).

    4. tmp = the leftmost n bits of tmp.

    5. (status, returned_bits) = Generate_function(state_handle, n,
       highest_DRBG_security_strength, additional_input).
```

construction must not be reused.

6. If *status* indicates an error, then return (*status*, *Null*).
7. *tmp* = *tmp* ⊕ *returned_bits*.
8. Return (SUCCESS, *tmp*).

If additional input is provided in the **NRBG_Generate** call of step 5, it **shall** be included in the **Generate_function** call.

9.2.3 Direct DRBG Access

When accessed directly, the XOR circuitry of the NRBG construction will not be used. The SEI may be any of those discussed in Section 7, including the entropy source or Full Entropy Source (FES) of the XOR Construction.

9.3 Construction: Enhanced NRBG – Oversampling Construction

The Oversampling Construction is shown in Figure 7; a high-level example is provided in Appendix A.2. In this construction, the entropy source need not be a Full Entropy Source. Conceptually, the DRBG mechanism accesses a Live Entropy Source to obtain prediction resistance (i.e., reseeding the DRBG from the entropy source with sufficient entropy bits for the instantiated security strength of the DRBG mechanism), and returns a number of bits equal to half the security strength of the DRBG instantiation during a single NRBG request. This results in full-entropy outputs and prediction resistance from the DRBG itself (see Assumption 1 of Section 4.3).

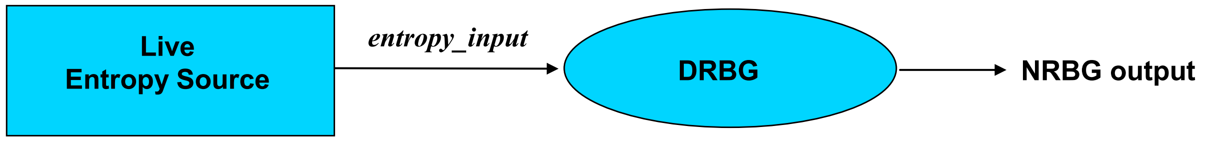

Figure 7: Enhanced NRBG - Oversampling Construction

For the Oversampling Construction:

- A Live Entropy Source **shall** be used, and
- A DRBG mechanism with a prediction resistance capability **shall** be used that results in one or more reseeds of the DRBG for each request for bits from the NRBG.

9.3.1 NRBG Instantiation

The DRBG mechanism of the NRBG **shall** be instantiated at its highest security strength. Let *highest_DRBG_security_strength* be the highest security strength that the DRBG mechanism can assume.

NRBG_Instantiate:

Input: string *personalization_string*.

Output: string *status*, integer *state_handle*.

Process:

```
1. (status, state_handle) = Instantiate_function(highest_DRBG_security_strength,
   prediction_resistance _flag = TRUE, personalization_string).
2. If status indicates an error, then return (status, Null)
   Else return (status, state_handle).
```

Prediction_resistance **shall** be requested in the **Instantiate_function** call in step 1.

The *personalization_string* is an optional input parameter in the **NRBG_Instantiate** call; however, if a *personalization_string* is provided, it **shall** be included as the *personalization_string* parameter in the **Instantiate_function** call.

The returned *state_handle* is used for subsequent access to the DRBG instantiation.

9.3.2 NRBG Generation

Let *n* be the requested number of bits, and let *state_handle* be the value returned from the **NRBG_Instantiate function** (see Section 9.3.1).

NRBG_Generate:

Input: integer (*state_handle*, *n*), string *additional_input*.

Output: string *status*, bitstring *returned_bits*.

Process:

```
1. tmp =Null.
2. sum = 0.
3. While (sum < n)
   3.1 (status, returned_bits) = Generate_function(state_handle, s/2, s,
       prediction_resistance_request = TRUE, additional_input).
   3.2 If status indicates an error, then return (status, Null).
   3.3 tmp = tmp || returned_bits.
   3.4 sum = sum + s/2.
4. Return (SUCCESS, the leftmost n bits of tmp).
```

Prediction_resistance **shall** be requested in the **Generate_function** call in step 3.1.

The *additional_input* is an optional input parameter in the **NRBG_Generate** call; however, if *additional_input* is provided, it **shall** be included as *additional_input* in the **Generate_function** call.

Note that in step 3.1, *s*/2 bits of output are requested from a DRBG that provides *s* bits of security strength. The *returned_bits* will have full entropy, as stated in assumption 1 of Section 4.2.

9.3.3 Direct DRBG Access

The DRBG mechanism may be directly accessed as a normal DRBG using the same or a different instantiation than that used when the DRBG mechanism is performing as part of the

NRBG. If the directly accessed DRBG instantiation is the same as the instantiation used for the Oversampling Construction, then prediction resistance **shall** be requested; if a separate instantiation is used for direct access to the DRBG, then a request for prediction resistance is optional. The SEI for direct DRBG access may be any of those discussed in Section 7, including the entropy source of the Oversampling Construction.

10 Additional Constructions

The constructions in this section are used to customize an RBG to accommodate various SEI decisions.

- Section 10.1 contains constructions that can be used by a DRBG to access its SEI when the SEI is an RBG, and
- Section 10.2 contains constructions that can be used by a DRBG to access an SEI when the SEI is an entropy source.

ANS X9.82, Part 4 contains additional constructions that can be used to provide external conditioning of entropy source output, and for obtaining entropy when an entropy source is available intermittently.

The constructions in Sections 10.1 and 10.2 are used within the target DRBG to satisfy the **Get_entropy_input** call made by that DRBG. These constructions describe the interface routine between the DRBG functions and the SEI. Figure 5 in Section 8 depicts the use of an SEI by a target DRBG mechanism. The target DRBG mechanism invokes a **Get_entropy_input** call, which is, in effect, translated to the appropriate call for the selected SEI by the interface routine.

10.1 Constructions Using an RBG as an SEI

The following apply when using an RBG as an SEI:

- If a Live Entropy Source is not available to a source RBG during normal operation (e.g., the entropy source was available only during instantiation of the source RBG): the source RBG is a DRBG; the target DRBG can support a security strength that is less than or equal to the security strength of the source DRBG.
- The target DRBG mechanism cannot support prediction resistance unless the source RBG supports prediction resistance. The source RBG may be either a DRBG or an NRBG.
- The source RBG **shall** generate at least the minimum number of bits to fulfill the **Get_entropy_input** request from the requesting DRBG (the target DRBG).
- The source RBG may be used to provide a nonce to construct the DRBG seed according to the guidance provided in SP 800-90A. If a nonce is not provided by the application or environment, and a nonce is required by the target DRBG mechanism, an SEI **shall** be used to obtain the nonce.
- The presence of a Live Entropy Source makes it possible for a source RBG to serve as an SEI for a target DRBG mechanism with any security strength, since all security strengths can be supported (see Section 5.4). For example, a source DRBG with 128 bits of security strength and prediction resistance (i.e., access to a Live Entropy Source) can be used to provide entropy input for a target DRBG mechanism with 256 bits of security strength.

The following construction uses a DRBG as an SEI. Additional constructions are provided in ANS X9.82, Part 4 for the case where the SEI is an NRBG and where the RBG type is unknown. In addition, a construction in ANS X9.82, Part 4 is provided whereby a DRBG with prediction resistance can be used to provide full entropy output

A target DRBG can use another DRBG as an SEI during a **Get_entropy_input** call by constructing the **Get_entropy_input** routine as described below. If prediction resistance is to be requested in the call, then the SEI **shall** have access to a Live Entropy Source.

The **Get_entropy_input** call in the target DRBG accesses the source DRBG using the following construction:

```
Get_entropy_input:
    Input: integer (min_entropy, min_length, max_length, prediction_resistance_request).
    Output: string status, bitstring returned_bits.
    Process:
        1. If (min_entropy > min_length), then min_length = min_entropy.
        2. If (min_length > max_length), then return (FAILURE, Null).
        3. (status, returned_bits) = Generate_function (state_handle, min_length, min_entropy,
               prediction_resistance_request).
        4. If status indicates an error, then return (status, Null).
        5. Return (SUCCESS, returned_bits).
```

The **Generate_function** needs to know the *state_handle* (as determined during the DRBG instantiation process); this is available in the DRBG's internal state (see SP 800-90A), the number of bits to be returned, and the minimum security strength that needs to be provided. This information is provided in the *min_length* and *min_entropy* parameters of the **Get_entropy_input** call. Since the **Generate_function** call will return the exact number of bits requested, the *max_length* parameter is not required in the **Get_entropy_input** call to this construction and may be omitted.

If prediction resistance is to be requested from the source DRBG, the *prediction_resistance_request* parameter needs to be present in the **Get_entropy_input** call, and its value **shall** be passed as a **Generate_function** parameter.

10.2 Constructions Using an Entropy Source as an SEI

An entropy source may be used as an SEI by a DRBG. By contrast with using an RBG as an SEI, an entropy source can support *any* security strength, and can always support prediction resistance and full entropy requests when the entropy source is Live (see Section 5.4). However, there are often constraints in the target DRBG mechanism or its implementation on the length of the entropy input that can be processed.

This section provides a construction that can be used with most entropy sources. Additional constructions are provided in ANS X9.82, Part 4 for entropy sources that can provide full entropy output, and for accumulating and condensing entropy source output when an output with sparse entropy is provided.

An entropy source can be used as an SEI by the (target) DRBG mechanism's **Get_entropy_input** call using the following construction:

Get_entropy_input:

Input: integer (*min_entropy*, *min_length*, *max_length*, *prediction_resistance_request*).

Output: string *status*, bitstring *returned_bits*.

Process:

```
1. If ($min\_length > max\_length$), then return (FAILURE, $Null$).
2. $tmp = Null$.
3. $entropy\_total = 0$.
4. While ($entropy\_total < min\_entropy$)
    4.1 ($status, entropy\_bitstring, assessed\_entropy$) = GetEntropy().
    4.2 If $status$ indicates an error, then return ($status$, $Null$).
    4.3 $tmp = tmp \| entropy\_bitstring$.
    4.4 $entropy\_total = entropy\_total + assessed\_entropy$.
5. $n$ = length_in_bits($tmp$).
6. If ($n < min\_length$), then $tmp = tmp \| 0^{min\_length - n}$.
7. If ($n > max\_length$), then $tmp$ = df($tmp$, $max\_length$).
8. Return SUCCESS, $tmp$.
```

Note that the *prediction_resistance_request* parameter in the **Get_entropy_input** call is not needed and can be ignored if present.

If *min_length* is greater than the allowed *max_length*, an error indication is returned (see step 1).

Steps 2 and 3 are used to set the initial values for the output string (*tmp*) and the counter for the entropy to be collected.

In step 4 of this construction, the **GetEntropy** function is used to collect a string of entropy bits (*tmp*) that meets or slightly exceeds the amount of entropy required (*min_entropy*) using as many calls as necessary. It is acceptable to collect more bits of entropy than required (i.e, *entropy_total* may exceed *min_entropy*), as long as the implementation of the DRBG mechanism can accommodate this without an unacceptable performance penalty.

If the bitstring containing the collected bits (*tmp*) is shorter than *min_length*, *tmp* is padded with sufficient zeros to increase its length to *min_length* bits (see step 6). If the bitstring is longer than *max_length*, a derivation function specified in SP 800-90A is used to reduce the length of *tmp* and to distribute the entropy over the resulting *max_length*-bit string (see step 7).

11 Testing

Two types of testing are specified in this Recommendation that may be performed on an RBG: health testing and implementation-validation testing. Health testing **shall** be performed on all RBGs that claim conformance with this Recommendation (see Section 11.1). Section 11.2 provides information on implementation validation.

11.1 Health Testing

Health testing is the testing of an implementation prior to and during normal operation (e.g., periodically) to determine that the implementation continues to perform as expected and as validated (if implementation validation was performed). Health testing is performed by the RBG itself; i.e., the tests are designed into the RBG implementation. Two types of tests **shall** be performed: behavior tests and known-answer tests.

- Behavior tests are performed on the parts of an implementation for which an exact response cannot be predicted. Such tests are specified in SP 800-90B for entropy sources.
- Known-answer tests are performed on the deterministic parts of an implementation (e.g., on an encoded algorithm) and are appropriate for the DRBG mechanims in SP 800-90A, on the RBG constructions in SP 800-90C, and may be appropriate for deterministic components within SP 800-90B.

The deterministic components of an RBG are normally less likely to fail than the components for which behavior testing is required. Therefore, known-answer tests may be performed less frequently than behavior tests.

An RBG **shall** support the health tests specified in SP 800-90A and SP 800-90B, as well as performing health tests on the components of SP 800-90C and the RBG as a whole. SP 800-90A specifies the use of known-answer tests, and SP 800-90B specifies the use of both known-answer tests and behavior tests.

The strategy for testing the RBG as a whole is to test the layers of components recursively, using known-answer tests, where appropriate, in order to verify the correct operation of the parts of the RBG that are not simply components from SP 800-90A or SP 800-90B.

11.1.1 Testing Components Recursively

Whenever an RBG (the target RBG) receives a request to startup, or receives a specific request to perform health testing, a request for health testing **shall** be issued to any DRBG component or SEI component within the device receiving the request (e.g., within the sub-boundary receiving the testing request).

When the SEI consists of a chain of RBGs within a single device:

- If the previous RBGs in the chain are not tested separately, then the health test request **shall** recurse up the chain, triggering health tests of all the accessible RBGs that constitute the SEI[24].
- Any previous RBGs in the chain that are tested separately from this recursive test **should** provide an indication of testing success or failure to subsequent RBGs in the chain.
- The entropy source for the target RBG (or the initial RBG in the chain of RBGs) **shall** also be given a health test request as soon as it is available.

The results of the tests **should** propagate down to the target RBG. If any component of the RBG (or chain of RBGs) fails a health test, then the target RBG fails the health test.

11.1.2 Known-Answer Testing for SP 800-90C and additional ANS X9.82, Part 4 Components

Known-answer tests **shall** be performed on constructions used by an implementation prior to the first use of the RBG after startup and prior to fulfilling the first request. A known-answer test **shall** be performed on each implemented construction, or on logical sets of constructions. When a construction is grouped with different subsets of other constructions, each such group **shall** be tested. For example, if construction A is used with construction B to execute one process, and with constructions B and C to execute a different process, then all components of each set of constructions **shall** be tested.

11.1.3 Handling Failure

When a failure is detected in an RBG component and reported to the RBG-as-a whole, the RBG **shall** enter an error state. For example, if the entropy source reports that an unrecoverable error has occurred in the noise source, the RBG needs to enter an error state.

SP 800-90A and SP 800-90B discuss the error handling of DRBG mechanisms and entropy sources, respectively. Operator intervention **shall** be required to recover from the error state when the RBG-as-a-whole is notified of an error.

11.2 Implementation Validation

Implementation validation is the process of verifying that an RBG and its components fulfill the requirements of this Recommendation. An RBG is validated by:

- Validating the components from SP 800-90A and SP 800-90B.
- Validating the use of the constructions in SP 800-90C and ANS X9.82, Part 4 via code inspection or known-answer tests or both, as appropriate.
- Validating the integer/bit conversion routines in SP 800-90A that are used via known-answer tests.
- Validating that the appropriate documentation as specified in SP 800-90C has been provided. (see below)

[24] When the RBG boundaries for the chain of RBGs are distributed, it may not be feasible to test all RBGs in the chain.

Documentation **shall** be developed that will provide assurance to users and testers that an RBG that claims conformance to this Recommendation has been implemented correctly. This documentation **shall** include the following as a minimum:

- An identification of the construction(s) and components used for the RBG, including a diagram of the interaction of these construction(s) and components.
- Appropriate documentation as specified in SP 800-90A and SP 800-90B; if either the DRBG mechanism or the entropy source has been validated for conformance to SP 800-90A or SP 800-90B, respectively, the appropriate validation certificate **shall** also be provided.
- An identification of the features supported by the RBG (e.g., access to the underlying DRBG mechanism, etc.).
- A description of the health tests performed, including an identification of the periodic intervals for performing the tests.
- A description of any support functions other than health testing.
- A discussion about how the integrity of the health tests will be determined subsequent to implementation validation.
- A discussion about the grouping of constructions for health testing (see Section 11.1.2).
- A description of the RBG components within the RBG boundary (see Section 5.1).
- If the RBG is distributed, a description about how the RBG is distributed, how each distributed portion is constructed, and the secure channel that is used to transfer information between the sub-boundaries (see Section 5.1).

Appendix A: Diagrams of Basic RBG Configurations

RBGs may be implemented in a variety of ways. Several common configurations are provided below. Additional details about these basic configuration, as well as more complicated configurations are provided in Appendix B of ANS X9.82, Part 4.

A.1 The XOR Construction

The XOR Construction is specified in Section 9.2, and requires a Full Entropy Source and a DRBG mechanism. For this example, assume that the entropy source itself provides full entropy output (i.e., an external conditioning function is not required). A DRBG mechanism specified in SP 800-90A is used that will obtain its entropy input directly from the NRBG's entropy source as shown in Figure A-1. The DRBG mechanism is designed to be able to access the entropy source whenever required (e.g., if the DRBG needs to be reseeded); that is, the DRBG uses the NRBG's entropy source as a Live Entropy Source.

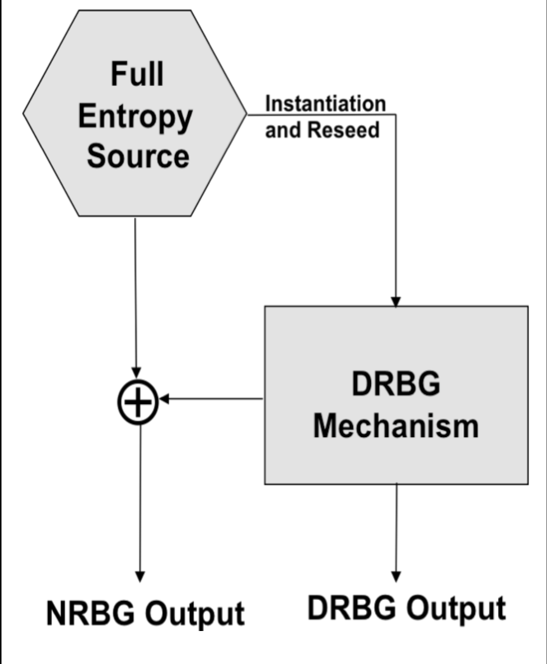

Figure A-1: XOR Construction Example

As specified in Section 9.1, the DRBG must be instantiated (and reseeded) at the highest security strength possible for the implemented DRBG mechanism.

Calls are made to the NRBG using the NRBG calls specified in Section 6.4. For this example, all components are contained within a single RBG boundary.

The DRBG mechanism itself can be accessed directly using the DRBG calls specified in Section 6.3. Since the NRBG's Live Entropy Source is always available, the DRBG can support prediction resistance. In this example, calls made directly to the DRBG use the same instantiation as is used for NRBG calls.

If the entropy source produces output at a slow rate, a consuming application might call the NRBG only when full entropy bits are required, obtaining all other output directly from the DRBG.

A.2 The Oversampling Construction

The Oversampling Construction is specified in Section 9.3, and requires an entropy source and a DRBG mechanism (see Figure A-2); the entropy source need not provide full entropy output.

As specified in Section 9.1, a DRBG used as part of the NRBG must be instantiated (and reseeded) at the highest security strength possible for the implemented DRBG mechanism.

Calls are made to the NRBG using the NRBG calls specified in Section 6.4.

The DRBG mechanism can also be accessed directly as a DRBG using the DRBG calls specified in Section 6.3. For this example, direct access to the DRBG will use a separate instantiation from that used by its role as part of the NRBG, i.e., two DRBG instantiations (i.e., two DRBGs) will be used: one for the NRBG, and another that can be directly accessed.

The NRBG's DRBG supports prediction resistance by design (see Section 9.3). For this example, since a Live Entropy Source is always available, the (directly accessed) DRBG will also support prediction resistance.

As in the case of the XOR example in Appendix A.1, if the entropy source produces output at a slow rate, a consuming application might call the NRBG only when full entropy bits are required, obtaining all other output directly from the (directly accessed) DRBG.

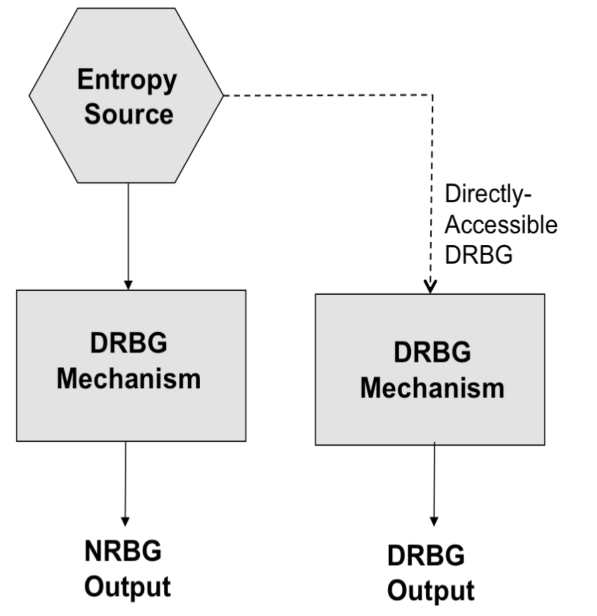

Figure A-2: Oversampling Construction Example

A.3 A DRBG Without a Live Entropy Source

A DRBG may have access to an SEI only during instantiation (e.g., the DRBG will not have access to a Live Entropy Source or a source RBG during normal operation). For example, this will often be the case for smart card applications. In this case, the DRBG is seeded only once (i.e., reseeding is not possible).

Since an SEI (e.g., a Live Entropy Source) is not available during normal operation, prediction resistance and reseeding cannot be provided.

Calls are made to the DRBG using the DRBG calls specified in Section 6.3.

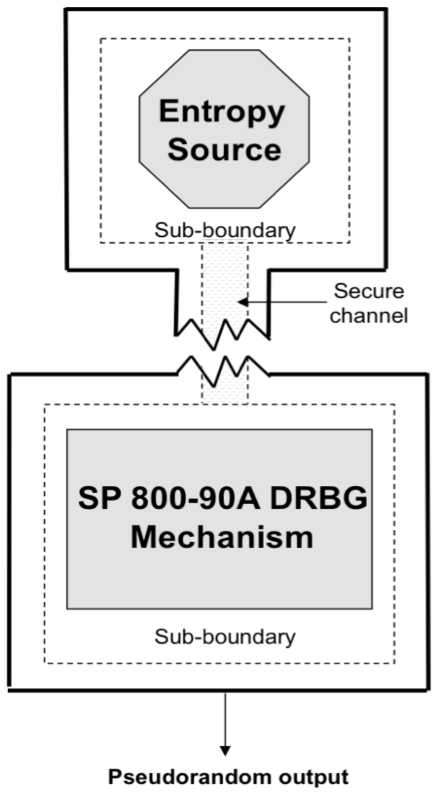

Figure A-3: DRBG Seeded Once

A.4 A DRBG with a Live Entropy Source

A DRBG with a Live Entropy Source can provide prediction resistance on request. The entropy source could reside in the same device as the DRBG, or could reside outside the device, with a secure channel available to transfer the requested entropy bits to the DRBG mechanism (i.e., the DRBG is distributed).

For this example, assume that everything is the same as the example in Appendix A.3, except that a Live Entropy Source is available within the same boundary as the DRBG mechanism. That is, the SEI is an **approved** entropy source, no secure channel is required, and only a single DRBG instantiation will be used. Since a Live Entropy Source is available during normal operation, prediction resistance and reseeding are supported. Figure A-4 depicts this example.

Calls are made to the DRBG using the DRBG calls specified in Section 6.3.

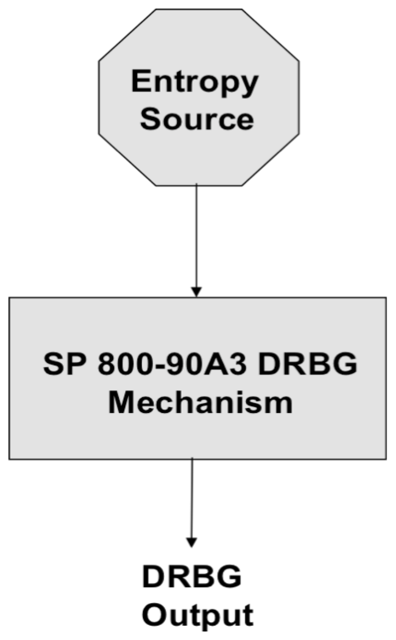

Figure A-4: DRBG with a Live Entropy Source

Appendix B: **Conformance to SP 800-90C Requirements**

Conformance to many of the requirements in this Recommendation are the responsibility of entities using, installing or configuring applications or protocols that incorporate the RBGs specified in SP 800-90C, i.e., a given implementation may not have the means of fulfilling these requirements. These requirements include the following:

Section	**Shall** statement
4.1	4. Every DRBG **shall** be instantiated using an appropriate source of entropy input (see Section 7).
4.1	6. Bitstrings containing entropy **shall** only be used once.
5.1	An RBG **shall** exist within a conceptual RBG security boundary that is defined with respect to one or more threat models, which include an assessment of the probability of an attack and the potential harm caused by the attack.
5.1	The RBG boundary **shall** be designed to assist in the mitigation of these threats using either physical or logical mechanisms or both.
5.1	Communications between the sub-boundaries **shall** use reliable secure channels that protect the confidentiality and integrity of the data transferred between the sub-boundaries.
5.3	Such output **shall not** itself be used as random output.
7	In order to construct a DRBG or an Enhanced NRBG that contains a DRBG mechanism, the RBG designer **shall** construct a source of secret, random or pseudorandom input for the DRBG mechanism, i.e., a source of entropy input (SEI).
7	During instantiation, the SEI(s) **shall** support at least the security strength that is intended for the target DRBG mechanism that is using it.
7	If the target DRBG is intended to allow reseeding, either on-demand or at the end of the DRBG's seed period, then the SEI **shall** be available on-demand.
7	If the target DRBG is intended to support requests for prediction resistance, then an SEI that has access to a Live Entropy Source **shall** be used.
7	If the SEI is not within the same sub-boundary as the target DRBG, then a secure channel **shall** be used to transfer data from the SEI to the target DRBG.
7	If the **CTR_DRBG** is used as the target DRBG mechanism (see SP 800-90A), and a derivation function will not be used, then the SEI used by the **CTR_DRBG shall** be 1) an FES, 2) an NRBG or 3) a DRBG that can support the security strength intended for the **CTR_DRBG**.
8.1	The initial DRBG in any chain of DRBGs **shall** be instantiated using either an **approved** entropy source, or an **approved** NRBG.

8.2	The target DRBG **shall not** be the same instantiation as the source DRBG; i.e., the target DRBG instantiation **shall not** reseed itself.
8.3	In the case of the nonces specified in SP 800-90A, if the nonce is not provided by the implementation environment, then it **shall** be provided by the SEI.
9.2	A Live Entropy Source that provides full entropy output (i.e., a Full Entropy Source) **shall** be used.
9.2	The SEI bits that are used as input to the DRBG **shall not** be used for any other purpose (e.g., as bits within the NRBG construction that are XORed with the output of the DRBG to produce the NRBG output for a consuming application).
10.1	The source RBG **shall** generate at least the minimum number of bits to fulfill the **Get_entropy_input** request from the requesting DRBG (the target DRBG).
10.1	If prediction resistance is to be requested in the call, then the SEI shall have access to a Live Entropy Source
11.1.1	Whenever an RBG (the target RBG) receives a request to startup, or receives a specific request to perform health testing, a request for health testing **shall** be issued to any DRBG component or SEI component within the device receiving the request (e.g., within the sub-boundary receiving the testing request).
11.1.1	If the previous RBGs in the chain are not tested separately, then the health test request **shall** recurse up the chain, triggering health tests of all the accessible RBGs that constitute the SEI
11.1.1	The entropy source for the target RBG (or the initial RBG in the chain of RBGs) **shall** also be given a health test request as soon as it is available.

Appendix C: Post-Processing of RBG Output

The output of an RBG may be modified by an **approved** post-processing method before it is used by a consuming application. Such post-processing methods **shall** be designed so that they do not significantly reduce the security strength provided in the RBG output. When post-processing is performed, the output of the post-processing operation **shall** be used in place of any use of the original RBG output.

The **approved** method for post-processing of RBG output requested by a consuming application is specified in this appendix using a permutation P of that RBG output (see Appendix C.2). This method is used in conjunction with a function F performed on another RBG output (see Appendix C.1), for example, to select the permutation algorithm to be used.

Let R_1 be the RBG output requested by the consuming application, and let m be the length of R_1 in bits. Let r_1 be the integer representation of R_1.

Let R_2 be another RBG output obtained for the purpose of post-processing using the function F, and let n be the length of R_2 in bits. Let r_2 be the integer representation of R_2.

Let k be the number of permutation algorithms.

Any bitstring to integer (BtoI_convert) conversion **shall** be performed as prescribed in SP 800-90A, e.g., r_1 = **BtoI_convert**(R_1).

C.1 The Function F

Let F(R_1) be a function on n-bit strings with integer output in the range 0 to k-1, where k is an arbitrary positive integer that corresponds to the number of permutation algorithms to be used during the post-processing.

Let k be an integer in the range 1 to $2^{n.}$.

Three definitions for F are provided[25]:

1. F(R_2) = r_2 mod k.
2. F(R_2) = **BtoI_convert**(H(R_2)) mod k, where H is an **approved** hash function specified in FIPS 180-4.
3. F(R_2) = **BtoI_convert**(HMAC(key, R_2)) mod k,

   where HMAC uses an **approved** hash function and a fixed key in the HMAC computation.

In methods 2 and 3, k could be as large as 2^{outlen}, or as small as 1, where *outlen* is the length of the hash function output in bits.

Note that using a single permutation algortihm (i.e., k = 1), while permitted, would not require the use of the function F to "choose" it. On the other hand k = 2 (or more) might make sense for

[25] Other definitions may be added at a later time.

some applications.

Note that in these cases, the k permutations are selected with (nearly) equal probability, but that is not a requirement imposed by this **approved** post-processing method.

C.2 The Permutations

Let $\{P_0, P_1, \ldots, P_{k-1}\}$ be a set of permutations (one-to-one functions) from an m-bit value to another m-bit value. The P_i's may be fixed, or they may be generated using a random or secret value.

The security of this post-processing method depends on the P_i being permutations.

C.2.1 Exclusive-OR with Fixed Masks

Let P_i be a bitwise exclusive-or operation with fixed masks A_i, i.e.,

$$P_i(R_1) = (R_1 \oplus A_i),$$

where R_1 and A_i are m-bit vectors, and $0 \le i < k$.

The function F is used to select the mask to be used.

For example, if there are four such masks (i.e., $k = 4$), the simple function $F(R_2) = r_2 \bmod 4$ might be used to choose among them (i.e., $F(R_2)$ is the two rightmost bits of r_2). Then, the post-processor's output $P_{F(R2)}(R_1)$ would be $R_1 \oplus A_{r2 \bmod 4}$. Note that in this method, $2 \le n \le m$, where n is the length of R_2 (in bits), and m is the length of R_1 (in bits).

C.2.2 Using a Symmetric-Key Block Cipher

Let P_i be a symmetric-key block cipher, e.g., AES or Triple-DEA (TDEA), as specified in FIPS 197 and SP 800-67, respectively. P_i uses key_i. When using this method, each permutation is performed on a portion of R_1 that is equal in length to the length of the block cipher output block (*blocklen*), where *blocklen* is 128 bits for AES, and 64-bits for TDEA.

Let a *blocklen* portion of R_1 be selected as R_3 and processed as described below. This method may be used on multiple *blocklen* portions of R_1. When multiple portions of R_1 are permuted, these portions **shall not** overlap.

When AES is used as the block cipher,

$$P_i(R_3) = \text{AES}(key_i, R_3),$$

which produces 128-bit permutations of R_3.

When Triple-DEA is used as the block cipher,

$$P_i(R_3) = \text{TDEA}(key_i, R_3),$$

which produces 64-bit permutations of R_3.

The function F is used to select the key to be used for the permutation.

For example, suppose that there are ten 256-bit AES keys ($k = 10$). Let $F(R_2) =$ **BtoI_convert**(SHA256(R_2)) mod 10. The post-processed output $P_{F(R_2)}(R_3)$ would be

AES($key_{F(R2)}$, r_3) . Note that in this case, $4 \le n \le m$, where n is the length of R_2, and m is the length of R_3. The minimum length of R_2 is determined by the modulus value 10, which is represented in binary as 4 bits.

A similar example, but one with a much larger value for k, (e.g., $k = 2^{128}$), might use key_i = SHA256(128-bit representation of i). Let F(R_2) = SHA256(R_2). The output $P_{F(R2)}(R_3)$ of the post-processor would be AES(SHA256($R2$), R_3). Note that in this case, $n = m = 128$.

C.2.3 Using SBOXes

Let P_i be a byte-permutation 'SBOX$_i$' from [FIPS 197] that is applied to each byte of input, with the final output being the concatenation of the individually permuted bytes:

$$P_i(B_1||B_2|| \ldots ||B_{m/8}) = SBOX_i(B_1)||SBOX_i(B_2)||\ldots||SBOX_i(B_{m/8}),$$

where m is the number of bits to be operated upon and output from the permutation.

For example, suppose that $m = 128$, and there are two byte permutations SBOXes from which to choose: SBOX$_0$ and SBOX$_1$. Suppose that F maps 8-bit strings to their parity:

F(R_2) = 0 if R_2 has an even number of 1's, and

F(R_2) = 1 if R_2 has an odd number of 1's.

The post-processor's output $P_{F(R2)}(R_1)$, on the input pair R_2 and $R_1 = B_1||B_2|| \ldots ||B_{16}$ would be $SBOX_{parity(R2)}(B_1)$ || $SBOX_{parity(R2)}(B_2)$ ||…|| $SBOX_{parity(R2)}(B_{16})$. To complete the example, suppose that the two byte permutations are specified as: SBOX$_0$ = the AES SBOX, and SBOX$_1$ is the inverse permutation to the AES SBOX.

Appendix D: References

[FIPS 197] Advanced Encryption Standard (AES), November 2001, available at http://csrc.nist.gov/publications/PubsFIPS.html.

[SP 800-90A] Recommendation for Random Number Generation Using Deterministic Random Bit Generators, January 2012, available at http://csrc.nist.gov/publications/PubsSPs.html.

[SP 800-90B] Recommendation for the Design and Validation of Entropy Sources for Random Bit Generation, [Insert date].

Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, Tal Rabin: “Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes”. CRYPTO 2004: 494-510.

P. Gutmann, “Software Generation of Random Numbers for Cryptographic Purposes,” Proceedings of the 1998 Usenix Security Symposium, 1998.

J. Kelsey, B. Schneier, D. Wagner, and C. Hall. “Cryptanalytic Attacks on Pseudorandom Number Generators,” Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, 1998, pp. 168-188.

Niels Ferguson and Bruce Schneier, Practical Cryptography, published by Wiley in 2003. ISBN 0-471-22357-3.

J. Kelsey, B. Schneier, and N. Ferguson, “Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator,” Sixth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1999.

Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography

Werner Schindler, Wolfgang Killmann: “Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications.” CHES 2002: 431-449

Ben Jun and Paul Kocher, “The Intel Random Number Generator,” Technical Report from Cryptography Research

“Evaluation of VIA C3 NEHEMIAH Random Number Generator,” Cryptography Research white paper

RFC 4086: Randomness Recommendations for Security, Eastlake, Schiller, and Crocker, http://www.ietf.org/rfc/rfc4086.txt