The NIST SP 800-90 Deterministic Random Bit Generator Validation System (DRBGVS)

Updated: September 2, 2011
Previous Update: July 6, 2011
Original: March 10, 2009

Timothy A. Hall

National Institute of Standards and Technology

Information Technology Laboratory

Computer Security Division

TABLE OF CONTENTS

Update Log

9/02/11

- Added Appendix B, “Requirements Identified By “Shall” Statements That Are Tested by the CAVP validation testing.”

7/6/11

- Section 6.3
    - Updated instructions for testing with prediction resistance enabled and disabled.

1 Introduction

This document, *The NIST SP 800-90 Deterministic Random Bit Generator Validation System (DRBGVS)*, specifies the procedures involved in validating implementations of the Deterministic Random Bit Generator mechanisms approved in NIST SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised) (March 2007)* [1].

The *DRBGVS* is designed to perform automated testing on Implementations Under Test (IUTs). This document provides the basic design and configuration of the *DRBGVS*. It defines the purpose, the design philosophy, and the high-level description of the validation process for DRBG. The requirements and procedures to be followed by those seeking formal validation of an implementation of DRBG are presented. The requirements described include the specification of the data communicated between the IUT and the DRBGVS, the details of the tests that the IUT must pass for formal validation, and general instruction for interfacing with the DRBGVS.

2 Scope

This document specifies the tests required to validate IUTs for conformance to the NIST SP 800-90 [1]. When applied to IUTs that implement DRBG, the DRBGVS provides testing to determine the correctness of the algorithm contained in the implementation. The DRBGVS consists of a single test that determines if the DRBG implementation produces the expected random bit output given a set of entropy and other inputs. The requirements of NIST SP 800-90 addressable at the algorithm level and indicated by **shall** statements that are tested by the validation suite are listed in Appendix B.

3 Conformance

The successful completion of the tests contained within the DRBGVS is required to be validated as conforming to the DRBG. Testing for the cryptographic module in which the DRBG is implemented is defined in FIPS PUB 140-2, *Security Requirements for Cryptographic Modules* [2].

4 Definitions and Abbreviations

4.1 Definitions

DEFINITION	**MEANING**
CST laboratory	Cryptographic Security Testing laboratory that operates the DRBGVS
Deterministic Random	The algorithms specified in NIST SP 800-90, *Recommendations for*

Bit Generator	*Random Number Generation Using Deterministic Random Bit Generators (DRBG)* for generating random bits.

4.2 Abbreviations

ABBREVIATION	MEANING
DRBG	Deterministic Random Bit Generator
DRBGVS	Deterministic Random Bit Generator Validation System
IUT	Implementation Under Test

5 Design Philosophy of the Deterministic Random Bit Generation Validation System

The DRBGVS is designed to test conformance to NIST SP 800-90 rather than provide a measure of a product's security. The validation tests are designed to assist in the detection of accidental implementation errors, and are not designed to detect intentional attempts to misrepresent conformance. Thus, validation should not be interpreted as an evaluation or endorsement of overall product security.

The DRBGVS has the following design philosophy:

1. The DRBGVS is designed to allow the testing of an IUT at locations remote to the DRBGVS. The DRBGVS and the IUT communicate data via *REQUEST (.req)* and *RESPONSE (.rsp)* files.

2. The testing performed within the DRBGVS uses statistical sampling (i.e., only a small number of the possible cases are tested); hence, the successful validation of a device does not imply 100% conformance with the standard.

6 DRBGVS Test

The DRBGVS for DRBG consists of a single test. The DRBGVS requires the vendor to select the mechanism or mechanisms, Hash_DRBG, HMAC_DRBG, CTR_DRBG, or Dual_EC_DRBG, and options (e.g., which SHA algorithm is used for hashing functions). Separate files will be generated for each mechanism.

6.1 Configuration Information

To initiate the validation process of the DRBGVS, a vendor submits an application to an accredited laboratory requesting the validation of its implementation of DRBG. The vendor's implementation is referred to as the Implementation Under Test (IUT). The request for validation includes background information describing the IUT along with information needed by the DRBGVS to perform the specific tests. More specifically, the request for validation includes:

1. Vendor Name;
2. Product Name;
3. Product Version;
4. Implementation in software, firmware, or hardware;
5. Processor and Operating System with which the IUT was tested if the IUT is implemented in software or firmware;
6. Brief description of the IUT or the product/product family in which the IUT is implemented by the vendor (2-3 sentences);
7. The DRBG mechanisms and options (e.g., SHA sizes, block cipher algorithms) supported by the IUT.

6.2 The Deterministic Random Bit Generator Test

The DRBGVS consists of a single test that exercises the DRBG instantiate, generate and reseed operations and is organized as follows. CAVS generates a separate request (.req) file for each type of DRBG mechanism supported by the implementation. The four NIST SP 800-90 DRBG mechanisms are Hash_DRBG, HMAC_DRBG, CTR_DRBG, and Dual_EC_DRBG. The file names are simply the mechanism name with a .req extension (e.g., CTR_DRBG.req).

Each file lists the supported options for that mechanism. For Hash_DRBG and HMAC_DRBG, this is simply what SHA sizes are used. For CTR_DRBG, this is the block cipher functions used (e.g., AES-128) and whether or not a derivation function (df) is used. For Dual_EC_DRBG, this is the curve names (e.g., P-521) and SHA sizes used. All options supported by the mechanism are listed on a single line following the file header. Examples:

```
# HMAC_DRBG options: SHA-1 :: SHA-224 :: SHA-384 :: SHA-512

# CTR_DRBG options: 3KeyTDEA use df :: AES-128 use df :: AES-256 no df

# Dual_EC_DRBG options: P-256 SHA-1 :: P-256 SHA-256 :: P-521 SHA-512
```

For each option, a series of test cases is specified. The test case is defined by six lines of bracketed parameters. The first identifies the option. The second indicates whether the

prediction resistance flag is on or off. The remaining four indicate bit lengths of input parameters to the Instantiate and Generate functions. For example, an HMAC_DRBG SHA-1 test case might be:

```
[SHA-1]
[PredictionResistance = True]
[EntropyInputLen = 128]
[NonceLen = 64]
[PersonalizationStringLen = 0]
[AdditionalInputLen = 0]
```

Each test case contains 15 trials. For cases with prediction resistance enabled, each trial consists of (1) instantiate drbg, (2) generate one block of random bits, do not print (3) generate one block of random bits, print out (4) uninstantiate. There are eight values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length as defined in NIST SP 800-90. Below is a sample trial for the HMAC_DRBG SHA-1 test case:

```
COUNT = 0
EntropyInput = 7a0f5bc462fd0d65156d8b1a6ba7387d
Nonce = 1ad732f0b703c3f5
PersonalizationString =
AdditionalInput =
EntropyInputPR = a1b85ba779582722098e7a6c002f5ebb
AdditionalInput =
EntropyInputPR = c335a123499584ec3188a52655294af4
ReturnedBits = ?
```

For test cases without prediction resistance, each trial consists of (1) instantiate drbg, (2) generate one block of random bits, do not print (3) reseed, (4) generate one block of random bits, print out (5) uninstantiate. There are eight values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call. A test case and one trial sample for CTR_DRBG is listed below.

```
[3KeyTDEA use df]
[PredictionResistance = False]
[EntropyInputLen = 112]
[NonceLen = 56]
[PersonalizationStringLen = 112]
[AdditionalInputLen = 0]

COUNT = 0
EntropyInput = 60afe5d671ffdfca5744cae6cdce
Nonce = 489f2692847ca1
PersonalizationString = b26ff8dfb1a920b7c064d423b23d
```

```
AdditionalInput =
EntropyInputReseed = 3e97714ae391b119a02ed85887f8
AdditionalInputReseed =
AdditionalInput =
ReturnedBits = ?
```

More information on the input values is in the section below.

6.3 Input values

Prediction resistance: if an implementation supports prediction resistance, the CST lab should check the "Prediction Resistance Enabled" box for each mechanism tested. If an implementation can be used without prediction resistance, the lab should check the "Prediction Resistance Not Enabled" box. Implementations that either have prediction resistance always on or always off will have one box checked; implementations that can be used either way will have both boxes checked.

Derivation function (df) for CTR_DRBG: counter-mode (CTR) block cipher mechanism DRBGs are defined in NIST SP 800-90 for use with a derivation function (df) and with no df. One or both of these options may be checked on the CTR_DRBG tab, whichever one(s) the implementation uses.

CAVS has default bit lengths for the inputs it provides. If the implementation can support these bit lengths, then do not change them. If an implementation does not support one of the defaults, the bit lengths can be edited by pushing the "Edit Input Lengths" button. The defaults and restrictions on each of the input lengths are as follows:

Entropy input: the default bit length is the maximum security strength supported by the mechanism/option. This is the minimum bit length CAVS will accept, as CAVS tests all DRBGs at their maximum supported security strength. Longer entropy inputs are permitted, with the following exception: for CTR_DRBG with no df, the bit length **must** equal the seed length.

Nonce: the default nonce bit length is one-half the maximum security strength supported by the mechanism/option. Longer nonces are permitted. CTR_DRBG with no df does not use a nonce.

Personalization string: CAVS has two default bit lengths for personalization string, 0 and maximum supported security strength, except in the case of CTR_DRBG with no df, where the second length must be <= seed length. If the implementation only supports one personalization string length, then set both numbers equal to each other. If the implementation does not use a personalization string, set both numbers to 0 (zero).

Additional input: the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

Appendix A References

[1] *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*, NIST SP 800-90, National Institute of Standards and Technology, March 2007.

[2] *Security Requirements for Cryptographic Modules*, FIPS Publication 140-2, National Institute of Standards and Technology, May 2001.

Appendix B Requirements Identified By "Shall" Statements That Are Tested by the CAVP validation testing

The "shall" statements in all special publications indicate requirements that must be fulfilled to claim conformance to this Recommendation. The "shall" statements in the Special Publications address requirements at the algorithm, module, product level, and/or a higher level.

This section identifies the "shall" statements tested at the algorithm level when performing the DRBG validation test suite.

8.6 Seeds

The seed and its use by a DRBG mechanism **shall** be generated and handled as specified in the following subsections.

See below.

8.6.1 Seed Construction for Instantiation

Entropy input **shall** always be used in the construction of a seed; requirements for the entropy input are discussed in Section 8.6.3.

CAVS input files provide entropy input for use in the instantiate function, where it is used to construct the initial seed.

Except for the case noted below, a nonce **shall** be used; requirements for the nonce are discussed in Section 8.6.7.

CAVS input files provide a valid nonce value for use the instantiate function.

8.6.8 Reseeding

Reseeding of the DRBG **shall** be performed in accordance with the specification for the given DRBG mechanism.

The CAVS DRBG tests validate the reseed function, when implemented.

9 DRBG Mechanism Functions

A function need not be implemented using such envelopes, but the function **shall** have equivalent functionality.

See below.

9.1 Instantiating a DRBG

The following or an equivalent process **shall** be used to instantiate a DRBG.

Instantiate_function (requested_instantiation_security_strength, prediction_resistance_flag, personalization_string):

…

The CAVS DRBG tests verify the correct operation of the Instantiate_function.

9.2 Reseeding a DRBG Instantiation

The following or an equivalent process **shall** be used to reseed the DRBG instantiation.

Reseed_function (state_handle, prediction_resistance_request, additional_input):

…

The CAVS DRBG tests verify the correct operation of the Reseed_function.

9.3.1 The Generate Function

The following or an equivalent process **shall** be used to generate pseudorandom bits.

Generate_function (state_handle, requested_number_of_bits, requested_security_strength**,**

prediction_resistance_request, additional_input):

…

The CAVS DRBG tests verify the correct operation of the Generate_function.

10.1 DRBG Mechanisms Based on Hash Functions

Table 2 specifies the values that **shall** be used for the function envelopes and DRBG algorithm for each **approved** hash function.

Tested in CAVS DRBG tests. IUT cannot pass validation tests unless Table 2 is followed.

10.1.1 Hash_DRBG

The **Hash_DRBG** requires the use of a hash function during the instantiate, reseed and generate functions; the same hash function **shall** be used throughout a **Hash_DRBG** instantiation.

Tested in CAVS Hash_DRBG Tests.

10.1.1.2 Instantiation of Hash_DRBG

The following process or its equivalent **shall** be used as the instantiate algorithm for this DRBG mechanism (see step 9 of the instantiate process in Section 9.1).

Tested in CAVS Hash_DRBG Tests.

10.1.1.3 Reseeding a Hash_DRBG Instantiation

The following process or its equivalent **shall** be used as the reseed algorithm for this DRBG mechanism (see step 6 of the reseed process in Section 9.2):

Hash_DRBG_Reseed_algorithm (working_state, entropy_input, additional_input**):**

Tested in CAVS Hash_DRBG tests.

10.1.1.4 Generating Pseudorandom Bits Using Hash_DRBG

The following process or its equivalent **shall** be used as the generate algorithm for this DRBG mechanism (see step 8 of the generate process in Section 9.3):

Hash_DRBG_Generate_algorithm (working_state, requested_number_of_bits, additional_input**):**

Tested in CAVS Hash_DRBG tests.

10.1.2 HMAC_DRBG

The same hash function **shall** be used throughout an **HMAC_DRBG** instantiation.

Tested in CAVS HMAC_DRBG tests. The result will not be correct unless the same specified hash function is used throughout the instantiation.

10.1.2.2 The HMAC_DRBG Update Function (Update)

The following or an equivalent process **shall** be used as the **HMAC_DRBG_Update** function.

Tested in CAVS DRBG tests. The HMAC_DRBG Update_function is called inside instantiate, reseed, and generate in order to update the internal state. Thus, its correct operation is reflected in a correct result returned from the second call to generate.

10.1.2.3 Instantiation of HMAC_DRBG

The following process or its equivalent **shall** be used as the instantiate algorithm for this DRBG mechanism (see step 9 of the instantiate process in Section 9.1):

Tested in the CAVS HMAC_DRBG tests.

10.1.2.4 Reseeding an HMAC_DRBG Instantiation

The following process or its equivalent **shall** be used as the reseed algorithm for this DRBG mechanism (see step 6 of the reseed process in Section 9.2):

Tested in the CAVS HMAC_DRBG tests.

10.1.2.5 Generating Pseudorandom Bits Using HMAC_DRBG

The following process or its equivalent **shall** be used as the generate algorithm for this DRBG mechanism (see step 8 of the generate process in Section 9.3):

Tested in the CAVS HMAC_DRBG tests.

If the implementation allows additional_input, but a given request does not provide any additional_input or additional_input is not supported, then a Null string **shall** be used as the additional_input in step 6 of the HMAC_DRBG generate process.

This is tested in CAVS for implementations that support both a Null string and at least one nonzero bit length additional input. For these implementations, CAVS will specific a Null string as the additional input for some of the trials.

10.2.1 CTR_DRBG

The same block cipher algorithm and key length **shall** be used for all block cipher operations of this DRBG.

Tested in CAVS CTR_DRBG tests. The result will not be correct unless the same block cipher algorithm and key length is used throughout the instantiation.

Table 3 specifies the values that **shall** be used for the function envelopes and the CTR_DRBG mechanism (algorithms).

Tested in CAVS CTR_DRBG tests. IUT cannot pass validation tests unless Table 3 is followed.

When a derivation function is not used by an implementation, the seed construction (see Section 8.6.1) **shall** not use a nonce.

CAVS CTR_DRBG with no derivation function (no df) tests check that a nonce is not used in seed construction. IUT cannot pass validation tests otherwise.

When using TDEA as the selected block cipher algorithm, the keys **shall** be handled as 64-bit blocks containing 56 bits of key and 8 bits of parity as specified for the TDEA engine specified in [SP 800-67].

A CTR_DRBG IUT using TDEA as the block cipher function cannot pass validation tests unless the CAVS-generated keys are interpreted in this way.

10.2.1.2 The Update Function (CTR_DRBG_Update)

The following or an equivalent process **shall** be used as the **CTR_DRBG_Update** function.

Tested in CAVS DRBG tests. The CTR_DRBG Update_function is called inside instantiate, reseed, and generate in order to update the internal state. Thus, its correct operation is reflected in a correct result returned from the second call to generate.

10.2.1.3.1 Instantiation When Full Entropy is Available for the Entropy Input, and a Derivation Function is Not Used

The following process or its equivalent **shall** be used as the instantiate algorithm for this DRBG mechanism:

Tested in the CAVS CTR_DRBG with no derivation function (no df) tests.

10.2.1.3.2 Instantiation When a Derivation Function is Used

Let Block_Cipher_df be the derivation function specified in Section 10.4.2 using the chosen block cipher algorithm and key size.

The following process or its equivalent **shall** be used as the instantiate algorithm for this DRBG mechanism:

Tested in the CAVS CTR_DRBG with derivation function (df) tests.

10.2.1.4.1 Reseeding When Full Entropy is Available for the Entropy Input, and a Derivation Function is Not Used

The following process or its equivalent **shall** be used as the reseed algorithm for this DRBG mechanism (see step 6 of the reseed process in Section 9.2):

Tested in the CAVS CTR_DRBG with no derivation function (no df) tests.

10.2.1.4.2 Reseeding When a Derivation Function is Used

Let **Block_Cipher_df** be the derivation function specified in Section 10.4.2 using the chosen block cipher algorithm and key size.

The following process or its equivalent **shall** be used as the reseed algorithm for this DRBG mechanism (see reseed process step 6 of Section 9.2):

Tested in the CAVS CTR_DRBG with derivation function (df) tests. IUT cannot pass validation tests otherwise.

10.2.1.5.1 Generating Pseudorandom Bits When a Derivation Function is Not Used for the DRBG Implementation

The following process or its equivalent **shall** be used as the generate algorithm for this DRBG mechanism (see step 8 of the generate process in Section 9.3.3):

Tested in the CAVS CTR_DRBG with no derivation function (no df) tests.

10.2.1.5.2 Generating Pseudorandom Bits When a Derivation Function is Used for the DRBG Implementation

The **Block_Cipher_df** is specified in Section 10.4.2 and **shall** be implemented using the chosen block cipher algorithm and key size.

Tested in the CAVS CTR_DRBG with derivation function (df) tests. IUT cannot pass validation tests otherwise.

The following process or its equivalent **shall** be used as generate algorithm for this DRBG mechanism (see step 8 of the generate process in Section 9.3.3):

Tested in the CAVS CTR_DRBG with derivation function (df) tests.

10.3.1 Dual Elliptic Curve Deterministic RBG (Dual_EC_DRBG)

Table 4 specifies the values that **shall** be used for the envelope and algorithm for each curve.

Tested in CAVS Dual_EC_DRBG tests. IUT cannot pass validation tests unless Table 4 is followed.

10.3.1.2 Instantiation of Dual_EC_DRBG

The following process or its equivalent **shall** be used as the instantiate algorithm for this DRBG mechanism (see step 9 of the instantiate process in Section 9.1):

Tested in CAVS Dual_EC_DRBG tests.

10.3.1.3 Reseeding of a Dual_EC_DRBG Instantiation

The following process or its equivalent **shall** be used to reseed the **Dual_EC_DRBG** process after it has been instantiated (see step 6 of the reseed process in Section 9.2):

Tested in CAVS Dual_EC_DRBG tests.

10.3.1.4 Generating Pseudorandom Bits Using Dual_EC_DRBG

c. x(A) is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point **shall** be translated back to affine coordinates before x() is applied.

Tested in CAVS Dual_EC_DRBG tests. IUT cannot pass validation tests unless the point is in affine coordinates before x() is applied.

The following process or its equivalent **shall** be used to generate pseudorandom bits (see step 8 of the generate process in Section 9.3):

Tested in CAVS Dual_EC_DRBG tests.

10.4.1 Derivation Function Using a Hash Function (Hash_df)

The following or an equivalent process **shall** be used to derive the requested number of bits.

Tested in the CAVS Hash_DRBG and Dual_EC_DRBG tests. IUT cannot pass validation tests unless Hash_df is implemented properly according to this requirement.

10.4.2 Derivation Function Using a Block Cipher Algorithm (Block_Cipher_df)

The following or an equivalent process **shall** be used to derive the requested number of bits.

1. *input_string* : …This string **shall** be a multiple of 8 bits.

Comment: L is the bitstring represention of the integer resulting from **len** (input_string)/8. L **shall** be represented as a 32-bit integer.

Comment : N is the bitstring represention of the integer resulting from number_of_bits_to_return/8. N **shall** be represented as a 32-bit integer.

Comment : i **shall** be represented as a 32-bit integer, i.e., **len** (i) = 32.

Tested in the CAVS CTR_DRBG with derivation function (df) tests. IUT cannot pass validation tests otherwise.

10.4.3 BCC Function

The following or an equivalent process **shall** be used to derive the requested number of bits.

Tested in the CAVS CTR_DRBG with derivation function (df) tests. IUT cannot pass validation tests unless BCC is implemented properly according to this requirement.

11.2 Implementation Validation Testing

A DRBG mechanism **shall** be tested for conformance to this Recommendation.

CAVS validation testing provides a method to achieve this requirement.

A DRBG mechanism **shall** be designed to be tested to ensure that the product is correctly implemented. A testing interface **shall** be available for this purpose in order to allow the insertion of input and the extraction of output for testing.

Tested by CAVS validation tests. IUT must have a testing interface in order to pass validation tests.

A.1 Constants for the Dual_EC_DRBG

The **Dual_EC_DRBG** requires the specifications of an elliptic curve and two points on the elliptic curve. One of the following NIST **approved** curves with associated points **shall** be used in applications requiring certification under [FIPS 140].

CAVS Dual_EC_DRBG tests use only the NIST Approved curves and associated points.

A.2.1 Generating Alternative P, Q

The curve **shall** be one of the NIST curves from [FIPS 186] that is specified in Appendix A.1 of this Recommendation, and **shall** be appropriate for the desired security_strength, as specified in Table 4, Section 10.3.1.

CAVS Dual_EC_DRBG tests use only the NIST Approved curves and associated points.