9.9 Self-Testing of the DRBG

9.9.1 Discussion

A DRBG **shall** perform self testing to obtain assurance that the implementation continues to operate as designed and implemented (operational testing). A DRBG may also be tested to validate that it has been implemented correctly. See Section 11 for a discussion of operational and implementation validation testing.

9.9.2 Specifications

9.9.2.1 Test Specification Variables

Abort_to_error_state (*status_message*)
: The abort routine for critical failures that is specified in Section 9.9.2.10.

additional_input_flag
: Indicates whether additional input should be provided for testing, where *additional_input_flag* = {*Additional_input_provided, No_additional_input_provided*}.

additional_input_text
: The text to be used as additional input during the testing of the pseudorandom bit generation and reseeding processes.

DRBG_specific_parameters
: DRBG-specific parameters to be included in the test function calls. Thee parameters are identified for each DRBG in Section 10, if required. Note that the presence of these parameters may require additional steps in the testing process. This will be addressed for each DRBG, when necessary.

*entropy_input*_1, *entropy_input*_2
: The entropy input returned from the **Get_entropy (...)** function.

ES_Selftest ( )
: The entopy input source testing function specified in Section 9.9.2.9.

expected_instantiated_state_with_personalization_string (*strength, prediction_resistance_flag*)
: An array of expected values of the state that is compared against the state generated during instantiation testing when a *personalization_string* is used.

expected_instantiated_state_with_no_personalization_string(*strength, prediction_resistance_flag*)
: An array of expected values of the state that is compared against the state generated during instantiation testing when no *personalization_string* is used.

expected_large_string_with_no_prediction_resistance (*strength, additional_input_flag*)
: An array of expected values for each strength when a large number of pseudorandom bits is requested from the generation process without prediction resistance.

Comment [ebb1]: Page: 65
Should there be more than one value ? Should there be different lengths ?

expected_large_string_with_prediction_resistance (*strength, additional_input_flag*)
: An array of expected values for each strength when a large number of pseudorandom bits is requested from the generation process with prediction resistance.

expected_reseeded_state_with_additonal_input (*strength*)
: An array of expected states when reseeding is performed and additional input is provided; a state is defined for each *strength* to be tested.

expected_reseeded_state_with_no additonal_input (*strength*)
: An array of expected states when reseeding is performed and no additional input is provided; a state is defined for each *strength* to be tested.

expected_small_string_with_no_prediction_resistance (*strength, additional_input_flag*)
: An array of expected values for each strength when a small number of pseudorandom bits is requested from the generation process without prediction resistance.

expected_small_string_with_prediction_resistanc e(*strength, additional_input_flag*)
: An array of expected values for each strength when a small number of pseudorandom bits is requested from the generation process with prediction resistance.

Get_entropy (*min_entropy, min_length, max_length, mode*)
: A function that acquires entropy input from an entropy source. See Section 9.5.2

large_no_of_bits
: The number of pseudorandom bits requested during testing of the pseudorandom bit generation process. This value is larger than a block of bits produced by the DRBG and is specific to the DRBG and its specification. See the DRBGs in Section 10 for an appropriate value for a given implementation.

max_length
: The maximum length for a string of bits.

max_strength
: The maximum security strength supported by a DRBG implementation (as opposed to a DRBG instantiation).

min_entropy
: The minimum amount of entropy required.

min_length
: The minimum length of a string of bits.

mode
: An indication of whether requests for entropy input are for normal operation or for testing. Possible values are *mode* = {0 = *Normal_operation*, 2 = *Failure*, 3 = *Fixed_entropy_input_1*, 4 = *Fixed_entropy_input_2*,.... }, where *Fixed_entropy_input_n* selects a fixed value as the entropy input.

Null
: A null (i.e., empty) string.

prediction_resistance_flag
: Indicates whether or not prediction resistance requests should be handled. Possible values are

prediction_resistance_flag = {*No_prediction_resistance*, *Allow_prediction_resistance*}.

pseudorandom_bits — The pseudorandom bits that are generated during a single call to the generation process.

requested_strength — The requested strength during a pseudorandom bit generation process.

reseed_counter — A count of the number of requests for pseudorandom bits since instantiation or reseeding.

reseed_interval — The maximum number of requests for the generation of pseudorandom bits before reseeding is required.

small_no_of_bits — The number of pseudorandom bits requested during testing of the pseudorandom bit generation process. This value is smaller than a block of bits produced by the DRBG and is specific to the DRBG and its specification. See the DRBGs in Section 10 for an appropriate value for a given implementation.

state (*state_pointer*) — An array of states for for different DRBG instantiations. A state is carried between DRBG calls. The *state* consists of multiple elements that are accessed as *state* (*state_pointer*).*element*. The state elements are specific to each DRBG. The *state* may be considered as *Empty*, *Test_not_empty* or contain the state for an instantiation. *Test_not_empty* **shall** be an illegal value (i.e., not *Empty* and not a recognized normal operational value for the state).

state_pointer — A pointer to the state space for a given DRBG instantiation. An invalid/incorrect state pointer is specified as *Invalid_state_pointer*.

status — The status returned from a function call, where *status* = "Success" or a failure message.

strength — The security strength to be provided by the DRBG instantiation.

temp — A temporary value.

Test_Generation (*strength*, *state_pointer*) — The pseudorandom bit generation testing function specified in Section 9.9.2.4.

Test_Generation_Error_Handling (***strength***, ***state_pointer***) — The testing function specified in Section 9.9.2.7 for error handling by the pseudorandom bit generation process.

Test_Instantiation (*strength*, *prediction_resistance_flag*) — The instantiation testing function specified in Section 9.9.2.3.

Test_Instantiation_Error_Handling (*strength*) — The testing function specified in Section 9.9.2.6 for error handling by the instantiation process.

Test_Reseeding (*strength, state_pointer*)
The reseeding test function specified in Section 9.9.2.5.

Test_Reseeding_Error_Handling (*state_pointer*)
The testing function specified in Section 9.9.2.8 for error handling by the reseeding process.

Test_personalization_string A personalization string to be used during testing.

Comment [ebb2]: Page: 68
Is a single string sufficient ? Should there be different lengths ?

Uninstantiate_DRBG (*state_pointer*)
The uninstantiate process discussed in Section 9.8 and specified for each DRBG in Section 10.

9.9.2.2 Test_DRBG (...)

Test_DRBG (...) shall test each DRBG process that resides in a DRBG boundary. As discussed in Section 8.3, the testing function is contained within the same DRBG boundary as the DRBG process being tested. Therefore, the internal state values are available for modification and examination by the testing function. When an error is detected during DRBG testing, the process **shall** enter an error state (see Section 9.9.2.10).
Each DRBG function within a DRBG boundary **shall** be tested in accordance with Section 11.4 (operational testing) using the following process.
The following **Test_DRBG (...)** process is the highest level routine of the tests. The steps used by an implementation depends on the DRBG processes that are available in the DRBG boundary.

- Steps 1 and 2 **shall** be present if a source of entropy input is available.
- Step 3 **shall** include all security strengths implemented.
- Steps 3.1, 3.2, 3.7, 4 and 5 **shall** be present if the instantation process is available and prediction resistance is not required.
- Steps 3.8, 3.9, 3.14, 4 and 5 **shall** be present if the instantation process is available and prediction resistance can be handled.
- Steps 3.3, 3.4, 6 and 7 **shall** be present if the generation process is available and prediction resistance is not required. Note that if the instantiation process is not available, the *state_pointer* **shall** be set to a state space that is not otherwise used (e.g., reserved for testing only).
- Steps 3.10, 3.11, 6 and 7 **shall** be present if the generation process is available and prediction resistance can be handled. Note that if the instantiation process is not available, the *state_pointer* **shall** be set to a state space that is not otherwise used (e.g., reserved for testing only).
- Steps 3.5, 3.6, 8 and 9 **shall** be present if the reseeding process is available and prediction resistance is not required. Note that if the instantiation process is not available, the *state_pointer* **shall** be set to a state space that is not otherwise used (e.g., reserved for testing only).
- Steps 3.12, 3.13, 8 and 9 **shall** be present if the reseeding process is available and prediction resistance can be handled. Note that if the instantiation process is not available, the *state_pointer* **shall** be set to a state space that is not otherwise used (e.g., reserved for testing only).
- Step 10 **shall** be present for all implementations.

The following process or its equivalent **shall** be used to test a DRBG implementation.
Test_DRBG ( ):

Input: None
Output: string *status*.
Process:

1. *status* = **ES_Selftest ( ).** Comment : Test the entropy input source. See Section 9.9.2.9.
2. If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure of the entropy input source").
   Comment : Test normal operation for each strength supported by a DRBG implementation.
3. For *strength* = 80, 112, 128, 192, 256
   Comment : Test the instantiation process with no prediction resistance. See Section 9.9.2.3.

   3.1 (*status, state_pointer*) = **Test_Instantiation** (*strength, No_prediction_resistance*).
   3.2 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during instantiation (no prediction resistance):" || *status*).
   Comment : Test the generation process. See Section 9.9.2.4.

   3.3 *status* = **Test_Generation** (*strength, state_pointer*).
   3.4 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during pseudorandom bit generation (no prediction resistance):" || *status*).
   Comment : Test the reseeding process. See Section 9.9.2.5.

   3.5 *status* = **Test_Reseeding** (*strength, state_pointer*).
   3.6 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during reseeding (no prediction resistance) :" || *status*).
   3.7 *status* = **Uninstantiate_DRBG** (*state_pointer*).
   Comment : Test the instantiation process with prediction resistance. See Section 9.9.2.3.

   3.8 (*status, state_pointer*) = **Test_Instantiation** (*strength, Allow_prediction_resistance*).
   3.9 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during instantiation (with prediction resistance):" || *status*).
   Comment : Test the generation process. See Section 9.9.2.4.

Comment [ebb3]: Page: 70
Have not yet included tests for in between sizes.

3.10 *status* = **Test_Generation** (*strength, state_pointer*).

3.11 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during pseudorandom bit generation (with prediction resistance):" || *status*).

Comment : Test the reseeding process. See Sectuion 9.9.2.5.

3.12 *status* = **Test_Reseeding** (*strength, state_pointer*).

3.13 If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during reseeding (with prediction resistance) :" || *status*).

3.14 *status* = **Uninstantiate_DRBG** (*state_pointer*).

Comment : Test error handling. Note that *strength* should now be the highest strength available in an implementation

Comment : Test error handling during instantiation. See Section 9.9.2.6.

4. *status* = **Test_Instantiation_Error_Handling** (*strength*).
5. If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during instantiation error handling test :" || *status*).

Comment : Test error handling during pseudorandom bit generation. See Section 9.9.2.7.

6. *status* = **Test_Generation_Error_Handling** (*strength, state_pointer*).
7. If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during pseudorandom bit generation error handling test :" || *status*).

Comment : Test error handling during reseeding. See Section 9.9.2.8.

8. *status* = **Test_Reseeding_Error_Handling** (*strength, state_pointer*).
9. If (*status* ≠ "Success"), then **Abort_to_error_state** ("Self testing failure during reseeding error handling test :" || *status*).
10. **Return** ("Success").

9.9.2.3 Test_DRBG_Instantiation (...)

The following **Test_Instantiation (...)** process **shall** be present when the DRBG boundary contains the instantiation process. Calls to **Instantiate_DRBG (...) shall** be considered as calls to the instantiation process for the appropriate DRBG (e.g., **Instantiate_Hash_DRBG (...)**).

- Steps 1-3 **shall** be present if an implementation can handle a personalization string.
- Step 4 **shall** be present if steps 5-7 are present.
- Steps 5-7 **shall** be present if an implementation can handle a Null personalization string and does not require prediction resistance.
- Step 8 **shall** be present for all implementations.

Note that steps 5-7 are not followed by a call for uninstantiation. This will allow the final instantiation to be used for subsequent testing (e.g., for pseudorandom bit generation). The

test sets may be reordered, but the final test set **shall** provide an instantiation that can be used for further testing.
The following process or its equivalent **shall** be used to test a DRBG instantiation process.
Test_ Instantiation ( ):

Input: integer *strength, prediction_resistance_flag.*
Output: string *status*, integer *state_pointer.*
Process:

Comment: Test with a personalization string. See Section 9.5.1.

1. (*status, state_pointer*) = **Instantiate_DRBG** (*strength, prediction_resistance_flag, Test_personalization_string, DRBG_specific_parameters, Fixed_entropy_input_1*).
2. If (*status* ≠ "Success"), then **Return** (*status*).
3. If (*state* (*state_pointer*) ≠ *expected_instantiated_state_with_personalization_string* (*strength, prediction_resistance_flag*), then **Return** ("Incorrect test state using a *personalization_string* ").

   Comment: Remove the state. See Section 9.8.

4. *status* = **Uninstantiate_DRBG** (*state_pointer*).

   Comment: Test with no personalization string. See Section 9.5.1.

5. (*status, state_pointer*) = **Instantiate_DRBG** (*strength, prediction_resistance_flag, Null, DRBG_specific_parameters, Fixed_entropy_input_*1).
6. If (*status* ≠ "Success"), then **Return** (*status*).
7. If (*state* (*state_pointer*) ≠ *expected_instantiated_state_with_ no_personalization_string* (*strength, prediction_resistance_flag*), then **Return** ("Incorrect test state with a null *personalization_string*").
8. **Return** ("Success", *state_pointer*).

9.9.2.4 Test_Generation (...)

The following **Test_Generation (...)** process **shall** be present when the DRBG boundary includes the generation process. Calls to **DRBG (...) shall** be considered as calls to the generation process for the appropriate DRBG (e.g., **Hash_DRBG (...)**).

- The appropriate steps of steps 1-12 **shall** be present if a generation process does not require prediction resistance.
  - Steps 1-3 and 7-9 **shall** be present when an implemenation is capable of handling *additional_input.*
  - Steps 4-6 and 10-12 **shall** be present when an implemenation can handle null *additional_input.*
- Step 13 **shall** be present if an implementation does not require prediction resistance at all times.

- The appropriate steps of steps 14-25 **shall** be present if a generation process can handle prediction resistance.
  - Steps 14-16 and 20-22 **shall** be present when an implemenation is capable of handling *additional_input.*
  - Steps 17-19 and 23-25 **shall** be present when an implemenation can handle null *additional_input.*
- Steps 26-28 **shall** be present if an implementation is unable to reseed from the generation process, but **shall** be omitted otherwise.
- Steps 29-32 **shall** be present when reseeding is available online, but **shall** be omitted otherwise.

The following process or its equivalent **shall** be used to test a pseudorandom bit generation process.

Test_Generation ( ):

Input: integer *requested_strength, state_pointer.*
Output: string *status.*
Process:

Comment : Request the generation of a small number of bits with an *additional_input* string and no prediction resistance. See Section 9.7.1.

1. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
2. If (*status* ≠ "Success"), then **Return** (*status*).
3. If (*pseudorandom_bits* ≠ *expected_small_string_with no prediction_resistance* (*requested_strength, Additional_input_provided*)), then **Return** ("Incorrect bits returned when *additional_input* but no prediction resistance is provided, and a small string is requested").

   Comment : Request the generation of a small number of bits with no *additional_input* string and no prediction resistance. See Section 9.7.1.

4. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *Null*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
5. If (*status* ≠ "Success"), then **Return** (*status*).
6. If (*pseudorandom_bits* ≠ *expected_small_string_with_no_prediction_resistance* (*rerquested_strength, No_additional_input_provided*)), then **Return** ("Incorrect bits returned when no *additional_input* and no prediction resistance is provided, and a small string is requested ").

   Comment : Request the generation of a larger number of bits with an

additional_input string. See Section 9.7.1.

7. (*status*, *pseudrandom_bits*) = **DRBG** (*state_pointer*, *large_no_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
8. If (*status* ≠ "Success"), then **Return** (*status*).
9. If (*pseudorandom_bits* ≠ *expected_ large_string_with no_prediction_resistance* (*requested_strength, Additional_input_provided*)), then **Return** ("Incorrect bits returned when *additional_input* but no prediction resistance is provided, and a large string is requested").

   Comment : Request the generation of a larger number of bits when no *additional_input* is provided. See Section 9.7.1.

10. (*status*, *pseudrandom_bits*) = **DRBG** (*state_pointer*, *large_no_of_bits*, *requested_strength*, *Null*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
11. If (*status* ≠ "Success"), then **Return** (*status*).
12. If (*pseudorandom_bits* ≠ *expected_large_string* (*requested_strength, No_additional_input*)), then **Return** ("Incorrect bits returned when no *additional_input* and no prediction resistance is provided, and a large string is requested").

    Comment : Return if there is no prediction resistance capability in the *state*. See Section 9.7.1.

13. If (*state* (*state_pointer*).*prediction_resistance_flag*) = *No_prediction_resistance*), then go to step 26.

    Comment : Test the prediction_resistance capability.

    Comment : Request the generation of a small number of bits with an *additional_input* string. See Section 9.7.1.

14. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *Provide_prediction_resistance*, *Fixed_entropy_input*_2).
15. If (*status* ≠ "Success"), then **Return** (*status*).
16. If (*pseudorandom_bits* ≠ *expected_small_string_with_prediction_resistance* (*requested_strength, Additional_input_provided*)), then **Return** ("Incorrect bits returned when *additional_input* and prediction resistance is provided, and a small string is requested").

    Comment : Request the generation of a small number of bits with no

additional_input string. See Section 9.7.1.

17. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *Null*, *Provide_prediction_resistance*, *Fixed_entropy_input_3*).
18. If (*status* ≠ "Success"), then **Return** (*status*).
19. If (*pseudorandom_bits* ≠ *expected_small_string_with prediction_resistance* (*requested_strength, No_additional_input_provided*)), then **Return** ("Incorrect bits returned when no *additional_input* is provided but prediction resistance is requested, and a small string is requested ").

    Comment : Request the generation of a larger number of bits with an *additional_input* string. See Section 9.7.1.

20. (*status*, *pseudrandom_bits*) = **DRBG** (*state_pointer*, *large_no_of_bits*, *requested_strength*, *additional_input_text*, *Provide_prediction_resistance*, *Fixed_entropy_input_4*).
21. If (*status* ≠ "Success"), then **Return** (*status*).
22. If (*pseudorandom_bits* ≠ *expected_ large_string_with prediction_resistance* (*requested_strength, Additional_input_provided*)), then **Return** ("Incorrect bits returned when *additional_input* is provided, but prediction resistance is requested, and a large string is requested").

    Comment : Request the generation of a larger number of bits when no *additional_input* is provided. See Section 9.7.1.

23. (*status*, *pseudrandom_bits*) = **DRBG** (*state_pointer*, *large_no_of_bits*,*requested_strength*, Null, *Provide_prediction_resistance*, *Fixed_entropy_input_5*).
24. If (*status* ≠ "Success"), then **Return** (*status*).
25. If (*pseudorandom_bits* ≠ *expected_large_string_with prediction_resistance* (*requested_strength, No_additional_input*)), then **Return** ("Incorrect bits returned when no *additional_input* is provided, but prediction resistance is requested, and a large string is requested").

    Comment : Test the end of the DRBG when reseeding and prediction resiatence is not available (i.e., step 3 of **Hash_DRBG (...)**). See Section 9.7.1

26. *state (state_pointer).reseed_counter = reseed_interval.*
27. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input_1*).

28. If (*status* ≠ "DRBG can no longer be used. Please re-instantiate or reseed"), then **Return** ("Incorrect result for *reseed_interval* test").

    Comment : Test the reseeding capability when *reseed_counter* ≥ *reseed_interval* and the reseeding process is available (i.e., step 12 of **Hash_DRBG (...)**).
29. *state*(*state_pointer*).*reseed_counter* = *reseed_interval - 1*.
30. (*status*, *pseudorandom_bits*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input*_6).
31. If (*status* ≠ "Success"), then **Return** (*status*).
32. If (*pseudorandom_bits* ≠ *string_after_reseeding* (*requested_strength*)), then **Return** ("Incorrect reseeding process").
33. **Return** ("Success").

9.9.2.5 Test_Reseeding (...)

The following **Test_ Reseeding (...)** process **shall** be available when an implementation has the reseeding process. Calls to **Reseed_DRBG_Instantiation (...) shall** be considered as calls to the reseeding process for the appropriate DRBG (e.g., **Reseed_Hash_DRBG_Instantiation (...)**).

- Steps 1-3 **shall** be present if an implementation can handle additional input during reseeding.
- Steps 4-6 **shall** be present if an implementation can handle a null additional input string during reseeding.

The following process or its equivalent **shall** be used to test a DRBG reseeding process.

Test_Reseeding ( ):

Input: integer *strength, state_pointer*.
Output: string *status*.
Process:

Comment: Test with additional input.

1. *status* = **Reseed_DRBG_Instantiation** (*state_pointer*, *additional_input_text*, *Fixed_entropy_input*_7).
2. If (*status* ≠ "Success"), then **Return** (*status*).
3. If (*state*(*state_pointer*) ≠ *expected_reseeded_state_with_additional_input* (*strength*)), then **Return** ("Incorrect reseed test state when additional input is provided").

   Comment: Test with no additional input.
4. *status* = **Reseed_DRBG_Instantiation** (*state_pointer*, *Null*, *Fixed_entropy_input*_8).
5. If (*status* ≠ "Success"), then **Return** (*status*).

6. If (*state*(*state_pointer*) ≠ *expected_reseeded_state_with_no_additional_input* (*strength*)), then **Return** ("Incorrect reseed test state when no additional input is provided").

7. **Return** ("Success").

9.9.2.6 Test_Instantiation_Error_Handling (...)

The following **Test_Instantiation_Error_Handling (...)** process **shall** be available when an implementation has the instantiation process. Calls to **Instantitate_DRBG (...) shall** be considered as calls to the instantiation process for the appropriate DRBG (e.g., **Instantiate_Hash_DRBG (...)**).

- Note that *strength* **shall** be the highest strength available in an implementation.
- If the *No_prediction_resistance* flag in steps 1, 3 and 6 cannot be handled by an implementation, the flag **shall** be changed to *Allow_prediction_resistance*.
- If the implementation cannot handle a personalization string, then *Test_personalization_string* **shall** be changed to *Null* in steps 1, 3 and 6.

The following process or its equivalent **shall** be used to test error handling by an instantiation process.

Test_Instantiation_Error_Handling ( ) :

Comment [ebb4]: Page: 79
Don't know how to check prediction resistance capability flag failure.

Input: integer *strength*.
Output: string *status*.
Process:

Comment : Test *requested_strength* check failure. The *strength* ≥ the last *strength* tested by **Test_DRBG (...).**

1. (*status*, *state_pointer*) = **Instantiate_DRBG** (*strength* + 1, *No_prediction_resistance*, *Test_personalization_string*, *DRBG_specific_parameters*, *Fixed_entropy_input*_1).

2. If (*status* = "Success"), then **Return** ("Accepted incorrect *strength*").
   Comment : Test **Get_entropy (...)** status check failure.

3. (*status*, *state_pointer*) = **Instantiate_DRBG** (*strength*, *No_prediction_resistance*, *Test_personalization_string*, *DRBG_specific_parameters*, *Failure*).

4. If (*status* = "Success"), then **Return** ("**Get_entropy** failure not detected").
   Comment : Test **the Find_state_space (...)** error handling process. Fill any unused state space.

5. For *i* = 0 to *last_state* do

   If (*state* (*i*) = *Empty*), then *state* (*i*) = *Test_not_empty*.

6. (*status*, *state_pointer*) = **Instantiate_DRBG** (*strength*, *No_prediction_resistance*, *Test_personalization_string*, *DRBG_specific_parameters*, *Fixed_entropy_input*_8).

7. If (*status* = "Success"), then **Return** ("Did not detect the full state space").
8. For $i = 0$ to *last_state* do

   If (*state* (*i*) = *Test_not_empty*), then *state* (*i*) = *Empty*.
9. **Return** ("Success").

9.9.2.7 Test_Generation_Error_Handling (...)

The following **Test_Generation_Error_Handling (...)** process **shall** be available when an implementation has the pseudorandom bit generation process. Calls to **DRBG (...) shall** be considered as calls to the generation process for the appropriate DRBG (e.g., **Hash_DRBG (...)**).

- Note that the *requested_strength* is the highest strength available for the implementation.
- If the implementation cannot handle *additional_input_text* or the *No_prediction_resistance* flag, then step 1 **shall** be modified to a call that can be handled (e.g., by changing to the *Allow_prediction_resistance* flag).
- Steps 1 and 2 **shall** be present when the generation process includes a check for an appropriate state pointer.
- Steps 3-7 **shall** be present when the generation process has no ability to automatically reseed.
- Steps 8 and 9 **shall** be present when the generation process checks for an appropriate security strength request.
- Steps 10 and 17 **shall** be present to test prediction resistance.
- Steps 11-13 **shall** be present when prediction resistance is supported, and the generation process checks whether a prediction resistance capability was instantiated.
- Steps 14-16 **shall** be present when both reseeding and prediction resistance are supported.
- Steps 18-20 **shall** be present when automatic reseeding is available and a check is made to determine if *reseed_interval* has been reached.
- Step 21 **shall** always be included.

The following process or its equivalent **shall** be used to test error handling by a pseudorandom bit generation process.

Test_Generation_Error_Handling ( ) :

Input: integer *requested_strength, state_pointer*.
Output: string *status*.
Process:

Comment : Test *state_pointer* checking.

1. (*status*, *entropy_input*) = **DRBG** (*Invalid_state_pointer*, *small_number_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
2. If (*status* = "Success"), then **Return** ("Accepted incorrect *state_pointer*").

Comment : Test abort when *reseed_interval* is reached and reseeding is unavailable.

3. *temp* = *state* (*state_pointer*).*reseed_counter*.
4. *state* (*state_pointer*).*reseed_counter* = *reseed_interval*.
5. (*status*, *entropy_input*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *No_prediction_resistance*, *Fixed_entropy_input*_1).
6. If (*status* = "Success"), then **Return** ("Incorrect operation when *reseed_counter* = *reseed_interval*").
7. *state* (*state_pointer*).*reseed_counter* = *temp*.

Comment : Test *requested_strength* checking.

8. (*status*, *entropy_input*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength* + 1, *additional_input_text*, *No_prediction_resistance*, *Fixed_no_of_bits*_1).
9. If (*status* = "Success"), then **Return** ("Accepted incorrect *requested_strength*").

Comment : Test inappropriate *prediction_resistance_request* checking.

10. *temp* = *state* (*state_pointer*).*prediction_resistance_flag*.
11. *state* (*state_pointer*).*prediction_resistance_flag* = *No_prediction_resistance*.
12. (*status*, *entropy_input*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *Provide_prediction_resistance*, *Fixed_no_of_bits*_2).
13. If (*status* = "Success"), then **Return** ("Incorrect handling of prediction resistance request").

Comment : Test reseeding error when prediction resistance requested.

14. *state* (*state_pointer*).*prediction_resistance_flag* = *Provide_prediction_resistance*.
15. (*status*, *entropy_input*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength*, *additional_input_text*, *Provide_prediction_resistance*, *Failure*).
16. If (*status* = "Success"), then **Return** ("Failure indication from reseed request when prediction resistance requested").
17. *state* (*state_pointer*).*prediction_resistance_flag* = *temp*.

Comment : Test reseeding when *reseed_counter* reaches *reseed_interval*.

18. *state* (*state_pointer*).*reseed_counter* = *reseed_interval* - 1.
19. (*status*, *entropy_input*) = **DRBG** (*state_pointer*, *small_no_of_bits*, *requested_strength* + 1, *additional_input_text*, *Provide_prediction_resistance*, *Failure*).

20. If (*status* = "Success"), then **Return** ("Incorrect reseed handling when *reseed_counter* ≥ *reseed_interval*").
21. **Return** ("Success").

9.9.2.8 Test_Reseeding_Error_Handling (...)

The following **Test_DRBG_Reseeding_Error_Handling (...)** process **shall** be available when an implementation has the reseeding process. Calls to **Reseed_Instantiation (...)** **shall** be considered as calls to the reseeding process for the appropriate DRBG (e.g., **Reseed_Hash_DRBG_Instantation (...)**).

- Steps 3 and 4 **shall** be present if entropy can be readily obtained.

The following process or its equivalent **shall** be used to test error handling by a reseeding process.

Test_Reseeding_Error_Handling ( ) :

Input: integer *state_pointer*.
Output: string *status*.
Process:

Comment : Test *state_pointers* check failure.

1. *status* = **Reseed_Instantiation** (*Invalid_state_pointer*, *Fixed_entropy_input*_2).
2. If (*status* = "Success"), then **Return** ("Accepted incorrect *state_pointer*").
   Comment : Test **Get_entropy (...)** status check failure.
3. *status* = **Reseed_ Instantiation** (*state_pointer*, *Failure*).
4. If (*status* = "Success"), then **Return** ("**Get_entropy** failure not detected").
   Comment : Test check of old and new entropy_input.
5. *state* (*state_pointer*).*transformed_seed* = *Fixed_entropy_input*_2.
6. *status* = **Reseed_ Instantiation** (*state_pointer*, *Fixed_entropy_input*_2).
7. If (*status* = "Success"), then **Return** ("Entropy input failure not detected").
8. **Return** ("Success").

9.9.2.9 ES_Selftest (...)

The concept of an entropy input source selftest is introduced in Part 1 of this Standard. This test **shall** consist of the following steps. Let *max_strength* be the maximum strength to be supported by the DRBG implementation; let *min_length* be the appropriate minimum length of the entropy input for the DRBG when it supports the maximum strength; and let *max_length* be the maximum length of the entropy input for the DRBG when it supports the maximum strength.

The following process or its equivalent **shall** be used to test the entropy input source.

ES_Selftest (...):

Input: None..
Output: string *status*.
Process:

Comment: Obtain two strings.

1. *min_entropy* = **max** (128, *max_strength*).
2. (*status*, *entropy_input*_1) = **Get_entropy** (*min_entropy*, *min_length*, *max_length*, *Normal_operation*).
3. If (*status* ≠ "Success"), then **Return** ("Failure indication returned by the **Get_entropy** source").
4. (*status*, *entropy_input*_2) = **Get_entropy** (*min_entropy*, *min_length*, *max_length*, *Normal_operation*).
5. If (*status* ≠ "Success"), then **Return** ("Failure indication returned by the **Get_entropy** source").

   Comment : Compare the two strings.

6. If (**len** (*entropy_input*_1) ≠ **len** (*entropy_input*_2)), then **Return** ("Success").
7. If (*entropy_input*_1 = *entropy_input*_2), then **Return** ("Entropy input source failure").
8. **Return** ("Success").

9.9.2.10 Abort_to_error_state (...)

Critical errors, such as the failure of the entopy input source, **shall** call the **Abort_to_error_state (...)** process specified below. Let *no_of_states* be the number of states available to the DRBG implementation.
The following or an equivalent process **shall** be used as the **Abort_to_error_state (...)** function:
Abort_to_error_state (...):

Input: string status.
Output: None.
Process:

1. **Display** ("*status*").

   Comment : Display the error indication message.

2. For *i* = 1 to *no_of_states*

   Uninstantiate_DRBG (*i*).

   Comment: Uninstantiate all states.

3. **Abort** ( ).

   Comment: Abort the DRBG.