10.1.2.3.2 Instantiation of Hash_DRBG (...)

The following process or its equivalent **shall** be used to instantiate the **Hash_DRBG (...)** process. Let **Hash (...)** be the Approved hash function to be used; let *outlen* be the output length of that hash function, and let *inlen* be the input length.

Instantiate_Hash_DRBG (...):

Input : integer (*usage_class, requested_strength, prediction_resistance_flag, personalization_string*).

Output : string *status*.

Process :

1. If *requested_strength* > the maximum security *strength* that can be provided for the hash function (see Table 1), then **Return** ("Invalid *requested_strength*").
2. If (*prediction_resistance_flag* = 1) and prediction resistance cannot be supported, then **Return** ("Prediction resistance cannot be supported").
3. Set the strength to one of the five security strengths.
   If (*requested_strength* ≤ 80), then *strength* = 80
   Else if (*requested_strength* ≤ 112), then *strength* = 112
   Else (*requested_strength* ≤ 128), then *strength* = 128
   Else (*requested_strength* ≤ 192), then *strength* = 192
   Else *strength* = 256.
4. Set up *t* in accordance with the indicated *usage_class*. If no value of *t* is available for the *usage_class*, then **Return** ("No value of *t* is available for the *usage_class*").
5. *min_entropy* = **max** (128, *strength*).
6. *min_length* = **max** (*outlen*, *strength*).

   Comment Get the *seed*.

7. (*status, entropy_bits*) = **Get_entropy** (*min_entropy, min_length, inlen*).
8. If (*status* = "Failure"), then **Return** ("Failure indication returned by the entropy source").
9. *seed_material* = *entropy_bits* || *personalization_string*.
10. *seedlen* = **max** (*strength* + 64, *outlen*).
11. If (*seedlen* > *inlen*), then *seedlen* = *inlen*.

    Comment: Ensure that the entropy is distributed throughout the seed.

12. *seed* = **Hash_df** (*seed_material*, *seedlen*).

    Comment : Perform a one-way function on the seed formlater comparison during reseeding.

13. *transformed_seed* = **Hash** (*entropy_bits*).
14. *ctr* = 1.
15. *V* = *seed*.
16. *C* = **Hash** (*t* || *V*).

17. *state* = {*usage_class*, *V*, *C*, *ctr*, *t*, *strength*, *seedlen*, *prediction_resistance_flag*, *transformed_seed*}.
18. **Return** ("Success").

Note that multiple *state* storage is required if the DRBG is used for multiple *usage_classes*.
If an implementation does not need the *usage_class* as a calling parameter (i.e., the implementation does not handle multiple usage classes), then the *usage_class* parameter can be omitted, step 4 must set *t* to the value to be used, and the *usage_class* indication in the *state* (see step 17) must be omitted.
If an implementation does not handle all five security strengths, then step 3 must be modified accordingly.
If no *personalization_string* will ever be provided, then the *personalization_string* parameter in the input may be omitted, and step 9 becomes *seed_material* = *entropy*.
If an implementation will never be reseeded using the process specified in Section 10.1.2.3.3, then step 13 may be omitted, as well as the *transformed_seed* in the *state* (see step 17).
If an implementation does not need the *prediction_resistance_flag* as a calling parameter (i.e., the **Hash_DRBG (....)** routine in Section 10.1.2.3.4 either always or never acquires new entropy in step 5), then the *prediction_resistance_flag* in the calling parameters and in the *state* (see step 17) may be omitted.

10.1.2.3.3 Reseeding a Hash_DRBG (...) Instantiation

The following process or its equivalent **shall** be used to reseed the **Hash_DRBG (...)** process. Let **Hash (...)** be the Approved hash function to be used; let *outlen* be the output length of that hash function, and let *inlen* be the input length.

Reseed_Hash_DRBG_Instantiation (...):

Input: integer (*usage_class*).
Output: string *status*.
Process:

1. If a *state* is not available for the indicated *usage_class*, then **Return** ("State not available for the indicated *usage_class*").
2. Get the appropriate *state* values for the indicated *usage_class*, e.g., *V* = *state.V*, *t* = *state.t*, *strength* = *state.strength*, *old_seedlen* = *state.seedlen*, *old_transformed_seed* = *state.transformed_seed*.
3. *min_entropy* = **max** (128, *strength*).
4. *min_length* = **max** (*outlen*, *strength*).
5. (*status*, *entropy_bits*) = **Get_entropy** (*min_entropy*, *min_length*, *inlen*).
6. If (*status* = "Failure"), then **Return** ("Failure indication returned by entropy source").

   Comment: Determine the larger of the key sizes so that entropy is not lost.
7. *seedlen* = **max** (*strength* + 64, *outlen*).

   Comment: Combine the new *entropy_bits* with the entropy present in *V*, and distribute throughout the *seed*.
8. *seed_material* = *entropy_bits* || *V*.
9. *seed* = **Hash_df** (*seed_material*, *seedlen*).

Comment: Perform a one-way function on the seed and compare with the old transformed seed.

10. *transformed_seed* = **Hash** (*entropy_bits*).
11. If (*transformed_seed* = *old_transformed_seed*), then **Return** ("Entropy source failure").
12. V = seed.
13. *ctr* = 1.
14. *C* = **Hash** (*t* || *V*).
15. Update the appropriate *state* values for the *usage_class*.
    15.1 *state.V* = *V*.
    15.2 *state.C* = *C*.
    153 *state.ctr* = *ctr*.
    15.4 *state.seedlen* = *seedlen*.
    15.5 *state.transformed_seed* = *transformed.seed*.
16. **Return** ("Success").

If an implementation does not need the *usage_class* as a calling parameter (i.e., the implementation does not handle multiple usage classes), then the *usage_class* parameter and step 1 can be omitted, and steps 2 and 15 will use the only *state* available.

10.1.2.3.4 Generating Pseudorandom Bits Using Hash_DRBG (...)

The following process or its equivalent **shall** be used to generate pseudorandom bits. Let **Hash (...)** be the Approved hash function to be used; let *outlen* be the output length of that hash function, and let *inlen* be the input length.

Hash_DRBG (...):

Input: integer (*usage_class*, *requested_no_of bits*, *requested_strength*, *additional_input*, *prediction_resistance_requested*).

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

Process:

1. If a *state* for the indicated *usage_class* is not available, then **Return** ("State not available for the indicated *usage_class*", Null).
2. Set up the *state* in accordance with the indicated *usage_class*, e.g., *V* = *state.V*, *C* = *state.C*, *ctr* = *state.ctr*, *strength* = *state.strength*, *seedlen* = *state.seedlen*, *prediction_resistance_flag* = *state.prediction_resistance_flag*.
3. If (*requested_strength* > *strength*), then **Return** ("Invalid *requested_strength*").
4. If ((*prediction_resistance_requested* = 1) and (*prediction_resistance_flag* = 0)), then Return ("Prediction resistance capability not instantiated").
5. If (*prediction_resistance_requested* = 1), then
   5.1 *status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class*).
   5.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
6. If (*additional_input* ≠ Null), then do
   6.1 *w* = **Hash** (*additional_input* || *V*).
   6.2 *V* = (*V* + *w*) mod $2^{seedlen}$.
7. *pseudorandom_bits* = **Hashgen** (*requested_no_of_bits*, *V*).
8. *V* = (*V* + *pseudorandom_bits* + *C* + *ctr*) mod $2^{seedlen}$.
9. *ctr* = *ctr* + 1.

10. If (*ctr* ≥ *max_updates*), then
    10.1 *status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class*).
    10.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
  Else Update the changed values in the *state*.
    10.3 *state.V* = *V*.
    10.4 *state.ctr* = *ctr*.
11. **Return** ("Success", *pseudorandom_bits*).

Hashgen (...):

Input: integer *requested_no_of_bits*, bitstring *V*.

Output: bitstring *pseudorandom_bits*.

Process:

1. $m = \left\lceil \frac{requested\_no\_of\_bits}{outlen} \right\rceil$.
2. *data* = *V*.
3. *W* = the Null string.
4. For *i* = 1 to *m*
   4.1 w_i = **Hash** (*data*).
   4.2 *W* = *W* || w_i.
   4.3 *data* = *data* + 1.

   [Note that in Figures 5 and 7, this step is shown a bit differently; a suggestion for reconciliation is welcome.]
5. *pseudorandom_bits* = Leftmost (*requested_no_of_bits*) bits of *W*.
6. **Return** (*pseudorandom_bits*).

If an implementation does not need the *usage_class* as a calling parameter (i.e., the implementation does not handle multiple usage classes), then the *usage_class* input parameter and step 1 can be omitted, and step 2 uses the only *state* available.
If an implementation does not need the *prediction_resistance_flag*, then the *prediction_resistance_flag* and steps 4 may be omitted. If prediction resistance is never used, then step 5 may be omitted.