10.1.2.3.3 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.

```
Instantiate_Hash_DRBG (...):
   Input : integer (requested_strength, prediction_resistance_flag,
                    personalization_string, mode ).
   Output : string status, integer state_pointer.
   Process :
      1. If (requested_strength > the maximum security strength that can be provided by
         the hash function (see Table 2)), then Return ("Invalid requested_strength",
         Invalid_state_pointer).
      2. If (prediction_resistance_flag = Allow_prediction_resistance) and prediction
         resistance cannot be supported, then Return ("Prediction resistance cannot be
         supported", Invalid_state_pointer).
      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. min_entropy = max (128, strength).
      5. min_length = max (outlen, strength).

                                          Comment Get the entropy_input.
      6. (status, entropy_input) = Get_entropy (min_entropy, min_length, max_length,
         mode).
      7. If (status ≠ "Success"), then Return ("Failure indication returned by the
         entropy_input source:" || status, Invalid_state_pointer).
      8. seed_material = entropy_input || personalization_string.
      9. seedlen = max (strength + 64, outlen).
                                          Comment: Ensure that the entropy is
                                          distributed throughout the seed.
      10. seed = Hash_df (seed_material, seedlen).

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

      11. transformed_entropy_input = Hash (entropy_input).

      12. reseed_counter = 1.

      13. V = seed.
      14. C = Hash (0x00 || V).           Comment: Precede V with a byte of
                                          zeroes.
      15. (status, state_pointer) = Find_state_space (mode).
```

```
16. If (status ≠ "Success"), then Return (status, Invalid_state_pointer).
17. state (state_pointer) = { V, C, reseed_counter, strength, seedlen,
        prediction_resistance_flag, transformed_entropy_input}.
18. Return ("Success", state_pointer).
```

10.1.2.3.5 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.

```
Hash_DRBG (...):
  Input: integer (state_pointer, requested_no_of bits, requested_strength,
        additional_input, prediction_resistance_request_flag, mode).
  Output: string status, bitstring pseudorandom_bits.
  Process:
    1. If ((state_pointer > max_no_of_states) or (state (state_pointer) = {Null, Null, 0,
       0, 0, 0, Null})), then Return ("State not available for the state_pointer", Null).
    2. Set up the state values, e.g., V = state(state_pointer).V, C =
       state(state_pointer).C, reseed_counter = state(state_pointer).reseed_counter,
       strength = state(state_pointer).strength, seedlen state(state_pointer).seedlen,
       prediction_resistance_flag = state(state_pointer).prediction_resistance_flag,
       old_transformed_entiopy_input =
       state(state_pointer).transformed_entropy_input.
                                  Comment: If reseed_counter ≥
                                  reseed_interval, then reseeding could not be
                                  done in step 12 (below) during the previous
                                  call.
    3. If (reseed_counter ≥ reseed_interval), then Return ("DRBG can no longer be
       used. Please re-instantiate or reseed", Null).
    4. If (requested_strength > strength), then Return ("Invalid requested_strength",
       Null).
    5. If ((prediction_resistance_request_flag = Provide_prediction_resistance) and
       (prediction_resistance_flag = No_prediction_resistance)), then Return
       ("Prediction resistance capability not instantiated", Null).
    6. If (prediction_resistance_request_flag = Provide_prediction_resistance), then
       6.1  min_entropy = max (128, strength).
       6.2  min_length = max (outlen, strength).
       6.3  (status, entropy_input) = Get_entropy (min_entropy, min_length,
            max_length, mode).
       6.4  If (status ≠ "Success"), then
              If (mode = Normal_operation) then Abort_to_error_state ("Failure
              indication returned by the entropy_input source during generation:" ||
              status, Null).
            Else Return ("Failure indication returned by the entropy_input source
            during generation:" || status, Null).
       6.5  transformed_entropy_input = Hash (entropy_input).
```

```
        6.6 If (transformed_entropy_input = old_transformed_entropy_input), then
            Return ("Entropy_input source failure during generation", Null).
        6.7 additional_input = entropy_input || additional_input.
        6.8 state(state_pointer).transformed_entropy_input =
            transformed_entropy_input.
    7. If (additional_input ≠ Null), then do
        7.1 $w$ = Hash (0x02 || $V$ || additional_input).
        7.2 $V = (V + w) \bmod 2^{seedlen}$.
    8. pseudorandom_bits = Hashgen (requested_no_of_bits, V).
    9. $H$ = Hash (0x03 || $V$).
    10. $V = (V + H + C + reseed\_counter) \bmod 2^{seedlen}$.
    11. reseed_counter = reseed_counter + 1.
    12. If (reseed_counter ≥ reseed_interval), then
            12.1 status = Reseed_ Hash_DRBG_Instantiation (state_pointer, Null,
                 mode).
            12.2 If (status ≠ "Success"), then Return (status, Null).
            12.3 Return ("Success", pseudorandom_bits).
    13. Update the changed values in the state.
            13.1 state(state_pointer).V = V.
            13.2 state(state_pointer).reseed_counter = reseed_counter.
    14. 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.
    5. pseudorandom_bits = Leftmost (requested_no_of_bits) bits of W.
6.  Return (pseudorandom_bits).
```