### 10.1.3 Hash Function DRBG Using Any Approved Hash Function (Hash_DRBG)

[Further work awaits the definition of a revised Hash_DRBG (...).]

#### 10.1.3.1 Discussion

Figures X and Y present a DRBG that uses any Approved hash function. **Hash_DRBG (...)** employs an Approved hash function that produces a block of pseudorandom bits using a seed (*seed*) and an application specific constant *(t)*. Optional additional input (*additional_input*) may be provided during each access of **Hash_DRBG (...)** to obtain bits; the size of the *additional_input* is arbitrary.

The **Hash_DRBG (...)** requires the use of a hash function at three points in the process, including the instantiation and reseeding processes (see Figures 8 and 9). The same hash function **shall** be used at all three points. The hash function to be used **shall** meet or exceed the desired security strength of the consuming application.

**Hash_DRBG (...)** has been designed to meet different security levels, depending on the hash function used. The security strengths that can be accommodated by each hash function, the associated entropy requirement and the seed lengths are specified in Table 2. For each security *strength*, the required minimum entropy (*min_entropy*) **shall** be the maximum of 128 and the security *strength* (i.e., *min_entropy* = **max** (128, *strength*)). The minimum length of the *seed* (*seedlen*) **shall** be the maximum of the hash output block size (*outlen*) and the security *strength*; the maximum length of the *seed* **shall** be the size of the hash input block (*inlen*); i.e., **max** (*outlen*, *strength*) ≤ *seedlen* ≤ *inlen*. Further requirements for the *seed* are provided in Section 9.4.

**Table 1: Security Strength, Entropy Requirement and Seed Length for Each Hash Function**

| Hash Function | Security Strength | Required Minimum Entropy | Seed Length |
|---|---|---|---|
| SHA-1 | 80 | 128 | 160-512 |
| | 112 | 128 | 160-512 |
| | 128 | 128 | 160-512 |
| SHA-224 | 80 | 128 | 224-512 |
| | 112 | 128 | 224-512 |
| | 128 | 128 | 224-512 |
| | 192 | 192 | 256-512 |
| SHA-256 | 80 | 128 | 256-512 |
| | 112 | 128 | 256-512 |
| | 128 | 128 | 256-512 |
| | 192 | 192 | 256-512 |
| | 256 | 256 | 384-512 |
| SHA-384 | 80 | 128 | 384-1024 |
| | 112 | 128 | 384-1024 |
| | 128 | 128 | 384-1024 |
| | 192 | 192 | 384-1024 |
| | 256 | 256 | 384-1024 |
| SHA-512 | 80 | 128 | 512-1024 |
| | 112 | 128 | 512-1024 |

| | | | |
|---|---|---|---|
| | 128 | 128 | 512-1024 |
| | 192 | 192 | 512-1024 |
| | 256 | 256 | 512-1024 |

The application-specific constant (*t*) **shall** be *outlen* bits in length. See Annex E.??? for some values for *t*.

Figures X1 and X1 depict the insertion of test input for the *seed*, the application-specific constant (*t*) and the additional input values (*additional_input*). The tests **shall** be run on the output of the generator.
Validation and operational testing are discussed in Section 11. Detected errors **shall** result in a transition to the error state.

#### 10.1.3.2 Interaction with Hash_DRBG (...)

##### 10.1.3.2.1 Instantiating Hash_DRBG (...)

Prior to the first request for pseudorandom bits, **Hash_DRBG (...) shall** be instantiated using the following call:

*status* = **Instantiate_Hash_DRBG** (*usage_class, requested_strength, prediction_resistance_flag, personalization_string*),

as described in Section 9.6.1.

##### 10.1.3.2.2 Reseeding a Hash_DRBG (...) Instantiation

When a DRBG instantiation requires reseeding (see Section 9.7), the DRBG **shall** be reseeded using the following call:

*status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class* )

as described in Section 9.7.2.

##### 10.1.3.2.3 Generating Pseudorandom Bits Using Hash_DRBG (...)

An application **shall** request the generation of pseudorandom bits by **Hash_DRBG (...)** using the following call:

(*status*, *pseudorandom_bits*) = **Hash_DRBG** (*usage_class*, *requested_no_of_bits*, *requested_strength*, *additional_input_flag*, *prediction_resistance_flag*)

as described in Section 9.8.2.

##### 10.1.3.2.5 Inserting Additional Entropy into the State Using the Hash_DRBG (...) Process

Additional entropy **may** be inserted into the state of the **Hash_DRBG (...)** between requests for pseudorandom bits as follows:

(*status*) = **Add_Entropy_to_Hash_DRBG** (*usage_class, request_sufficient_entropy_flag, always_update_flag*)

as described in Section 9.9.

#### 10.1.3.3 Specifications

##### 10.1.3.3.1 General

The instantiation and reseeding of **Hash_DRBG (...)** consists of obtaining a *seed* with at least the requested amount of entropy. The *seed* is used to derive elements of the initial *state*, which consists of:

1. (Optional) The *usage_class* for the DRBG instantiation; if the DRBG is used for multiple *usage_classes*, requiring multiple instantiations, then the *usage_class* parameter **shall** be present, and the implementation **shall** accommodate multiple *states* simultaneously; if the DRBG will be used for only one *usage_class*, then the *usage_class* parameter **may** be omitted).
2. A value (*V*) that is updated during each call to the DRBG.
3. A constant *C* that depends on the application-specific constant (*t*) and the *seed*.
4. A counter (*ctr*) that indicates the number of ipdates of *V* since the *seed* was acquired.
5. The application specific constant (*t*) (see Annex E).
6. The security *strength* of the DRBG instance.
7. The length of the *seed* (*seedlen*).
8. A *prediction_resistance_flag* that indicates whether or not prediction resistance is required by the DRBG, and
9. (Optional) A transformation of the *seed* using a one-way function for later comparison with a new *seed* when the DRBG is reseeded; this value **shall** be present if the DRBG will potentially be reseeded; it **may** be omitted if the DRBG will not be reseeded.

The variables used in the description of **Hash_DRBG (...)** are:

| | |
|---|---|
| *additional_entropy* | A string of bits containing entropy. |
| *additional_input* | Optional additional input. |
| *additional_input_flag* | A flag that indicates whether or not additional input is to be requested (see Section 9.6.3); its values are as follows:<br>0 = Do not request *additional_input*.<br>1 = Request *additional_input*, but return the Null string if no input is available. |
| *always_update_flag* | Indicates whether or not the state should be updated when entropy is not available. 1=Yes, 0=No. |
| *C* | An *outlen*-bit constant that is calculated during the instantiation and reseeding processes. |
| *ctr* | A counter that is used to update the *state* of **Hash_DRBG (...)** and records the number of times that *V* has been updated since the instantiation was seeded or reseeded. |
| *data* | The *data* to be hashed. |
| **Get_entropy** (*min_entropy, min_length, max_length*) | A function that acquires a string of bits from an entropy source. *min_entropy* indicates the minimum amount of entropy to be provided in the returned bits; *min_length* indicates the minimum number of bits to be returned. |

| | |
|---|---|
| | *max_length* indicates the maximum number of bits to be returned. See Section 9.6.2. |
| **Get_additional_input ( )** | Returns a value for *additional_input*. The specification of this function is left to the implementer. See Section 9.6.3. |
| **Hash (*a*)** | A hashing operation on data *a* using an appropriate Approved hash function. |
| *i* | A temporary value used as a loop counter. |
| *inlen* | The length of the input block of a hash function. |
| *m* | The number of iterations of the hash function needed to obtain the requested number of pseudorandom bits. |
| *max_updates* | The maximum number of updates of *V* for the DRBG. |
| *min_entropy* | The minimum amount of *entropy* to be provided in the *seed*. |
| *min_length* | The minimum length of the *seed*. |
| *old_seedlen* | The *seedlen* from the previous seeding of the instantiation. |
| *old_transformed_seed* | The *transformed_seed* from the previous seeding of the instantiation. |
| *outlen* | The length of the hash function output block. |
| *prediction_resistance_flag* | For instantiation, this flag indicates whether or not prediction_resistance may need to be provided upon request. 1 = requests may indicate a need for prediction resistance; 0 = prediction_resistance should never be provided. |
| *prediction_resistance_requested* | For pseudorandom bit generation, this flag indicates whether or not prediction resistance is required; 1 = yes, 0 = no. |
| *pseudorandom_bits* | The pseudorandom bits produced by the DRBG. |
| *requested_no_of_bits* | The number of bits to be generated. |
| *requested_strength* | The security strength to be associated with the pseudorandom bits obtained from the DRBG. |
| *seed* | The string of bits containing entropy that is used to determine the initial state of the DRBG during instantiation or reseeding. |
| *seedlen* | The length of the *seed* containing the required entropy. |
| *state* | The *state* of **Hash_DRBG (...)** that is carried between calls to the DRBG. In the following specifications, the entire *state* is defined as (*usage_class, V, C, cnt, t, strength, seedlen, prediction-resistance_flag, transformed_seed*). A particular element of the *state* is specified as *state element*, e.g., *state V*. |
| *status* | The status returned from a function call, where *status* = "Success", "No update performed" (informative message only) or an indication of failure. Failure messages are: 1. Invalid *requested_strength*. 2. No value of *t* is available for the *usage_class*. 3. Failure indication returned by the entropy source. 4. State not available for the indicated *usage_class*. |

**Comment [ebb1]:** Page: 1
This may not make sense.

| | |
|---|---|
| | 5. Invalid *additional_input_flag* value.<br>6. Failure from request for *additional_input*. |
| *strength* | The security strength provided by the instance of the DRBG. |
| *t* | The application-specific constant associated with the *usage_class* (see Annex E.3). |
| *transformed_seed* | A one-way transformation of the *seed* for the Hash_DRBG (...) instance. |
| *usage_class* | The usage class of a DRBG instance. |
| *V* | A value that is initially derived from the *seed*, but assumes new values based on optional additional input (*additional_input*), the pseudorandom bits produced by the generator (*pseudorandom_bits*), the constant (*C*) and the iteration count (*ctr*). |
| *w, W* | Intermediate values. |

#### 10.1.3.3.2 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 2), then **Return** ("Invalid *requested_strength*").
2. 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.
3. 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*").
4. *min_entropy* = **max** (128, *strength*).
5. *min_length* = **max** (*outlen*, *strength*).

   Comment Get the *seed*.

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

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

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

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

12. *transformed_seed* = **Hash** (*seed*).

13. *ctr* = 1.

14. *V* = *seed*.
15. *C* = **Hash** (*t* || *V*).

16. *state* = {*usage_class*, *V*, *C*, *ctr*, *t*, *strength*, *seedlen*, *prediction_resistance_flag*, *transformed_seed*}.

17. **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 3 must set *t* to the value to be used, and the *usage_class* indication in the *state* (see step 16) must be omitted.

If an implementation does not handle all five security strengths, then step 2 must be modified accordingly.

If no *personalization_string* will ever be provided, then the *personalization_string* parameter in the input may be omitted, steps 8 and 9 may be combined into *seedlen* = || *entropy_bits* ||, and step 10 may be omitted.

If an implementation will never be reseeded using the process specified in Section 10.1.3.3.3, then step 12 may be omitted, as well as the *transformed_seed* in the *state* (see step 16).

If an implementation does not need the *prediction_resistance_flag* as a calling parameter (i.e., the **Hash_DRBG (....)** routine in Section 10.1.3.3.4 either always or never acquires new entropy in step 10), then the *prediction_resistance_flag* in the calling parameters and in the *state* (see step 16) may be be omitted.

#### 10.1.3.3.3 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** (*old_seedlen*, || *entropy_bits* ||).

   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** (*seed*).
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.

#### 10.1.3.3.4 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_flag*, *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 ((*additional_input_flag* < 0) or (*additional_input_flag* > 1), then **Return** ("Invalid *additional_input_flag* value", Null).
5. If ((*prediction_resistance_requested* = 1) and (*prediction_resistance_flag* = 0)), then Return ("Prediction resistance capability not instantiated").
6. If (*prediction_resistance_requested* = 1), then
   6.1 *status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class*).
   6.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
7. If (*additional_input_flag* = 0), then *additional_input* = the Null string
   Else {
   7.1 (*status*, *additional_input*) = **Get_additional_input** ( ).
   7.2 If (*status* = "Failure"), then **Return** ("Failure from request for *additional_input*", Null).
8. If *additional_input* ≠ Null, then do
   8.1 *w* = **Hash** (*additional_input* || *V*)
   8.2 *V* = (*V* + *w*) mod $2^{seedlen}$
9. *pseudorandom_bits* = **Hashgen** (*requested_no_of_bits*, *V*).
10. *V* = (*V* + *pseudorandom_bits* + *C* + *ctr*) mod $2^{seedlen}$
11. *ctr* = *ctr* + 1.
12. If (*ctr* ≥ *max_updates*), then
    12.1 *status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class*).
    12.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
    Else Update the changed values in the *state*.
    12.3 *state.V* = *V*.
    12.4 *state.ctr* = *ctr*.
13. **Return** ("Success", *pseudorandom_bits*).

Comment [ebb2]: Page: 82
Does this make any sense for this DRBG ?

**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*).

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 will never request *additional_input*, then the *additional_input_flag* input parameter and step 4, 7 and 8 may be omitted.
If an implementation does not need the *prediction_resistance_flag*, then the *prediction_resistance_flag* and steps 5 and 6 may be omitted.

#### 10.1.3.3.5 Adding Entropy to Hash_DRBG (...)

If additional entropy is to be inserted into the DRBG other than during the instantiation, reseeding or the generation of pseudorandom bits, then the following process or its equivalent **shall** be used to insert additional entropy into the **Hash_DRBG (...)** state. It is recommended that the *request_sufficient_entropy_flag* be set to 1 (see Section 9.9). 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.

**Add_Entropy_to_Hash_DRBG (...)**:

**Input:** integer (*usage_class, request_sufficient_entropy_flag, always_update_flag*).
**Output:** string *status*.
**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*.
3. If (*request_sufficient_entropy_flag* = 1), then
   3.1 *min_entropy* = **max** (128, *strength*).
   3.2 *min_length* = **max** (*outlen*, *strength*).
   Else
   3.3 *min_entropy* = *min_length* = 1.
4. (*status*, *entropy_bits*) = **Get_entropy** (*min_entropy*, *min_length*, *inlen*).
5. If (*status* = "Failure"), then **Return** ("Failure from request for additional entropy").
6. If ((*entropy_bits* = Null) and (*always_update_flag* = 0)), then **Return** ("No update performed").
7. Perform steps 8-11 of **Hash_DRBG (...)**.

7.1. If *additional_input* ≠ Null, then do
   7.1.1 *w* = **Hash** (*additional_input* || *V*).
   7.1.2 *V* = (*V* + *w*) mod $2^{seedlen}$.
7.2. *pseudorandom_bits* = **Hashgen** (*requested_no_of_bits*, *V*).
7.3 *H* = (*V* + *pseudorandom_bits* + *C* + *ctr*) mod $2^{seedlen}$.

**Comment [ebb3]:** Page: 1
Not sure that this is right. Depends how **Get_entropy** is implemented.

7.4 *ctr* = *ctr* + 1.

8. If (*ctr* ≥ *max_updates*), then
   8.1 *status* = **Reseed_ Hash_DRBG_Instantiation** (*usage_class*).
   8.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
   Else Update the changed values in the *state*.
   8.3 *state.V* = *V*.
   8.4 *state.ctr* = *ctr*.
9. **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* input parameter and step 1 can be omitted, and step 2 uses the only *state* available.
If an implementation always requires sufficient entropy, then the *request_sufficient_entropy_flag* may be omitted as an input parameter, and step 3 may consist of only substeps 3.1 and 3.2. If an implementation never requires sufficient entropy, then the *request_sufficient_entropy_flag* may be omitted as an input parameter, and step 3 may consist of only substep 3.3.
If an implementation will always update the *state* even when no additional entropy is available, then the *always_update_flag* input parameter and step 6 may be omitted. If an implementation will never update the *state* unless additional entropy is available, then the *always_update_flag* input parameter and the reference to the flag in step 6 may be omitted, and step 7.1 can be changed to just steps 7.1.1.and 7.1.2.
Note that step 8 does not include a check for the *prediction_resistance_flag*. Since pseudorandom bits are not being produced by this process, and since whatever entropy was available is acquired in step 4, a check of the *prediction_resistance_flag* is not required.

#### 10.1.3.4 Generator Strength and Attributes

[To be determined]

#### 10.1.3.5 Reseeding

A new *seed* **shall** be generated to reseed the generator [How often?]