10.1.2Hash Function DRBG Using HMAC with Any Approved Hash (HMAC_DRBG)

10.1.2.1 Discussion

This section discusses a new DRBG based on using any approved hash function in the HMAC construction for making a keyed hash function. Any application that has access to an approved hash function can implement HMAC, though dedicated implementations of HMAC will be considerably more efficient.

10.1.2.2 Interaction with HMAC_DRBG

10.1.2.2.1 Instantiating HMAC_DRBG (...)

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

status = **Instantiate_HMAC_DRBG** (*usage_class, requested_strength, prediction_resistance_flag)*

as described in Section 9.6.1.

10.1.2.2.2 Reseeding a HMAC_DRBG (...) Instantiation

When a **HMAC_DRBG (...)** instantiation requires reseeding, the DRBG **shall** be reseeded using the following call:

status = **Reseed_HMAC_DRBG_Instantiation** (*usage_class*)

as described in Section 9.7.2.

10.1.2.2.3 Generating Pseudorandom Bits Using HMAC_DRBG (...)

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

(*status, pseudorandom_bits*) = **HMAC_DRBG** (*usage_class, requested_no_of_bits, requested_strength, additional_input_flag*)

as discussed in Section 9.8.2.

10.1.2.3 Specifications

10.1.2.3.1 General

The instantiation of **HMAC_DRBG (...)** consists of obtaining a *seed* with the appropriate amount of entropy, which is used to define the initial *state* of the DRBG. The *state* consists of:

1. 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. The value X, which is updated each time another N bits of output are produced (where N is the number of output bits in the underlying hash).
3. The value K, which is updated at least once each time the DRBG generates pseudorandom bits.
4. The size of the hash function output, N.
5. A *prediction_resistance_flag* that indicates whether or not prediction resistance is

	2. Failure indication returned by the entropy source. 3. State not available for the indicated *usage_class*. 4. Entropy source failure. 5. Invalid *additional_input_flag* value. 6. Failure from request for *additional_input*. 7. *additional_input* too large.
t	The initial value of the hash function. See Annex E.
temp	A temporary value.
transformed_seed	A one-way transformation of the *seed* for the **HMAC_DRBG(...)** instance.
usage_class	The purpose(s) of a DRBG instance.

10.1.2.3.2 Instantiation of HMAC_DRBG(...)

The following process or its equivalent **shall** be used to initially instantiate the **HMAC_DRBG (...)** process in Section 10.1.2.3.4:

Instantiate_HMAC_DRBG (...):

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

Output: string *status*.

Process:

1. If requested_strength>N, then Return("Invalid requested_strength)
2. (status,seed) = Get_entropy(N,N,2^{32})
3. If (*status* = "Failure"), then **Return** ("Failure indication returned by the entropy source").
4. K = HMAC(0x0000...00,0x0101..01||0x00||seed)
5. X = HMAC(K,X)
6. K = HMAC(K,X||0x01||seed)
7. *transformed_seed* = **X**
8. X = HMAC(K,X)
9. Set up *t* for the indicated *usage_class*. Comment: See Annex E.
10. state = {usage_class, prediction_resistance_flag,N,K,X}
11. **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* calling parameter may be omitted, step 7 must set *t* to the value to be used, and the *usage_class* indication in the *state* (see step 10) must be omitted.

If an implementation will never require more than N bits of security, then the *requested_strength* parameter and step 1 can be omitted.

If an implementation does not need the *prediction_resistance_flag* as a calling parameter (i.e., the **HMAC_DRBG (....)** routine in Section 10.1.2.3.4 either always or never acquires new entropy in step 9), then the *prediction_resistance_flag* in the *state* (see step 10) must be omitted.

If an implementation will never be reseeded using the process specified in Section 10.1.2.3.3, then step 6 may be omitted, as well as the *transformed_seed* in the *state* (see step 10). This does not preclude using the **Instantiate_HMAC_DRBG (...)** process to create a new instantiation.

5. Get the appropriate *state* values in accordance with the indicated *usage_class*, e.g., *K* = *state.K*, *t* = *state.t*, etc..
6. seed = ""
7. If (state.prediction_resistance_flag==1) then seed = Get_entropy(N,N,2^{32})
8. If (additional_input_flag==1) then seed = seed || Get_input()
9. If (additional_input_flag or state.prediction_resistance) then:
   9.1 K = HMAC(K,X||0x00||seed)
   9.2 X = HMAC(K,X)
10. temp = ""
11. while (len(temp)<requested_no_of_bits) do:
   11.1 X = HMAC(K,X)
   11.2 temp = temp || X
12. *pseudorandom_bits* = Leftmost (*requested_no_of_bits*) of (*temp*).
13. K = HMAC(K,X||0x01||seed)
14. X = HMAC(K,X)
15. state.X = X
16. state.K = K
17. Return("Success",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 parameter and step 3 can be omitted, and step 4 acquires the only state available.
If an implementation will never require more than N bits of security, then the *requested_strength* parameter and step 1 can be omitted.
If an implementation will never request additional input, then the *additional_input_flag* in the calling parameter may be omitted, and the *additional_input* term may be removed.
If an implementation does not use the *prediction_resistance_flag* in the *state* (see Section 10.1.2.3.2), then the *prediction_resistance_flag* is not acquired, and the reference to the *prediction_resistance_flag* is omitted.

10.1.2.4 Generator Strength and Attributes

The HMAC_DRBG provides outputs indistinguishable from ideal random outputs and reseeds securely if HMAC provides a pseudorandom function. It is instantiated securely if HMAC with an arbitrary key distills entropy from the input seed material.

10.1.2.5 Reseeding

Instantiation, reseeding, and generating pseudorandom bits with prediction resistance are all equivalent in security terms. Indeed, the mechanism for instantiating the HMAC_DRBG is nothing more than setting K and X to constant bitstrings, and then doing operations equivalent to generating N bits of DRBG output with prediction resistance, and reseeding the DRBG is nothing more than generating N bits of DRBG output with prediction resistance.