10.3 Deterministic RBGs Based on Number Theoretic Problems

10.3.1 Discussion

A DRBG can be designed to take advantage of number theoretic problems (e.g., the discrete logarithm problem). If done correctly, such a generator's properties of randomness and/or unpredictability will be assured by the difficulty of finding a solution to that problem. Section 10.3.2 specifies a DRBG based on elliptic curves; Section 10.3.3 specifies a DRBG based on the RSA integer factorization problem.

10.3.2 Dual Elliptic Curve Deterministic RBG (Dual_EC_DRBG)

10.3.2.1 Discussion

Dual_EC_DRBG (...) is based on the following hard problem, sometimes known as the "elliptic curve logarithm problem": given points P and Q on an elliptic curve of order n, find a such that $Q = aP$.

Dual_EC_DRBG (...) uses a seed m bits in length to initiate the generation of m-bit pseudorandom strings by performing scalar multiplications on two random points in an elliptic curve group, where the curve is defined over a field approximately 2^m in size. For efficiency, m **should** be kept as small as possible, subject to the security strength required by the application. For all the NIST curves given in this Standard, $m \geq 163$. Figure 18 depicts the **Dual_EC_DRBG (...)**.

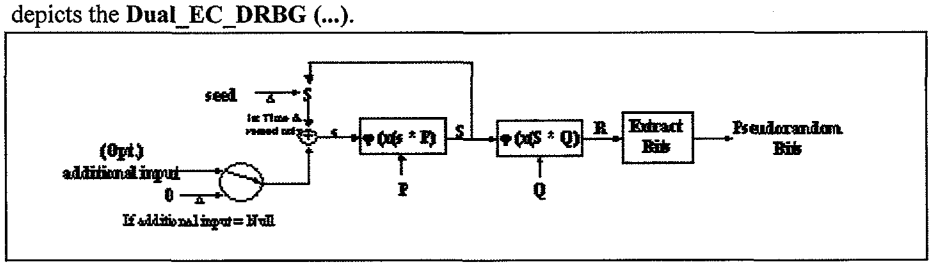

Figure 18: Dual_EC_DRBG (...)

The instantiation of this DRBG requires the selection of an appropriate elliptic curve and curve points specified in Annex E.4 for the desired security strength. The *seed* used to determine the initial value (S) of the DRBG **shall** have entropy that is at least the maximum of 128 and the desired security strength (i.e., *entropy* ≥ **max** (128, *strength*). Its length **shall** be m bits. Further requirements for the *seed* are provided in Section 8.4. When optional additional input (*additional_input*) is used, the value of *additional_input* is arbitrary, in conformance with Section 9.8.3, but it will be hashed to an m-bit string. Figure 19 depicts the insertion of test input for the *seed* and the *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.

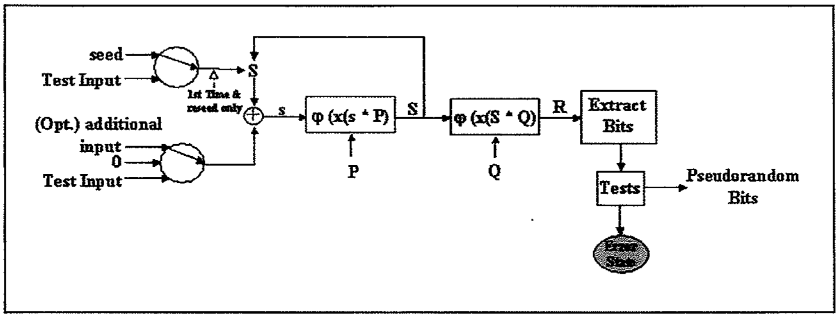

Figure 19: Dual_EC_DRBG (...) with Tests

10.3.2.2 Interaction with Dual_EC_DRBG (...)

10.3.2.2.1 Instantiating Dual_EC_DRBG (...)

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

status = **Instantiate_Hash_DRBG** ([*usage_class*,] *requested_strength*, *prediction_resistance_flag* [, integer ([*requested_curve_type*] [, *max_ctr*])])

as described in Section 9.6.1, with the addition of the optional *requested_curve_type* and *max_ctr* parameters. *requested_curve_type* is used to specify a class of elliptic curves from which the instantiated elliptic curve is to be selected. *max_ctr* indicates the maximum number of steps that may be taken along the curve before the DRBG must be seeded.

10.3.2.2.2 Reseeding a Dual_EC_DRBG (...) Instantiation

When a DRBG instantiation requires explicit reseeding (see Section 9.7), the DRBG **shall** be re- instantiated (i.e., reseeded) using the following call:

status = **Reseed_ Dual_EC_DRBG_Instantiation** ([*usage_class*,]

as described in Section 9.7.2.

10.3.2.2.3 Generating Pseudorandom Bits Using Dual_EC_DRBG (...)

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

(*status*, *pseudorandom_bits*) = **Dual_EC_DRBG** ([*usage_class*,] *requested_no_of_bits*, *requested_strength*, *additional_input_flag*)

as described in Section 9.8.2.

10.3.2.2.4 Inserting Additional Entropy into the State Using Dual_EC_DRBG (...)

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

(*status*) = **Add_Entropy_to_Dual_EC_DRBG** ([*usage_class*])

as described in Section 9.9.

10.3.2.3 Specifications

10.3.2.3.1 General

The instantiation of **Dual_EC_DRBG (...)** consists of selecting an appropriate elliptic curve and point pairing from Annex E.4 and obtaining a *seed* that is used to determine an initial value (S) for the DRBG that is one element of the initial *state*. The state consists of:

1. (Optional) The *usage_class* of the DRBG instantiation; if the DRBG is used for multiple *usage_classes*, requiring multiple instantiations, then the *usage_class* **shall** be indicated, and the implementation **shall** accommodate multiple *states* simultaneously; if the DRBG will be used for only one *usage_class*, then the *usage_class* **may** be omitted,
2. A counter (*ctr*) that indicates the number of requests to **Dual_EC_DRBG (...)** during the current instance,
3. An optional *max_counter* may be provided, which will be checked for automatic reseeding of the **Dual_EC_DRBG (...)**,
4. A value (S) that is updated during each request for pseudorandom bits,
5. The elliptic curve domain parameters (*curve_type*, m, [p], a, b, n), where curve_type indicates a prime field F_p, or a pseudorandom or Koblitz curve over the field is $F_2{}^m$; a and b are two field elements that define the equation of the curve, and n is the order of the point P; one of the binary curve types may be requested at initialization; otherwise, the default *curve_type* 0, indicating mod p, will be used,
6. Two points P and Q on the curve; the generating point of the curve will be used as P,
7. The security *strength* provided by the instance of the DRBG; the curve will be selected to provide a maximum of *requested_strength* bits of security,
8. A *prediction_resistance_flag* that indicates whether or not prediction resistance is required by the DRBG, and
9. (Optional) A record of the seeding material in the form of a one-way function that is performed on the *seed* 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 **Dual_EC_DRBG (...)** are:

a, b	Two field elements that define the equation of the curve.
additional_input	Optional additional input. A bitstring returned by **Get_additional_input**( ), a function that prompts the user to supply an input. It will be hashed and truncated to m bits.
additional_input_flag	A flag that indicates whether or not additional input may be used, with values as follows: 0 = None requested, return 0. 1 = Request *additional_input*, but return 0 if no input is available.
B	The output block length of the hash function.
ctr	A count of the number of iterations of the of **Dual_EC_DRBG (...)** since the last reseeding.
curve_type	Either 0,1,2 indicating a curve over a prime field, a random binary curve, or a Koblitz curve, respectively.

E	An elliptic curve defined over F_p or $F_2{}^m$
f	The cofactor of the curve: 1 for all prime field curves, 2 or 4 for the binary curves. Comment: This value will be implicit from the *curve_type* and *a*.
G	A generating point of prime order *n* on the curve.
Get_entropy (*min_entropy,min_length, max_length*)	A function that acquires a string of bits from an Approved entropy source. The parameters indicate the minimum entropy to be provided in the returned bit string, and the limits between which the length of that string must lie (i.e., *min_length* and *max_length*). **Dual_EC_DRBG (....)** will always specify (*min_length* - *max_length*) = *m*.
i	A temporary value that is used as a loop counter.
m	Length in bits of the internal state *S*; the curve is defined over a field with approximately 2^m elements.
max (*A*, *B*)	The maximum of the values *A* and *B*.
max_ctr	The maximum number of steps taken along the curve before the DRBG must be reseeded.When counter reaches 100,000, a new seeding is recommended (see Annex D.3.2).
min_entropy	A value used in the request to **Get_entropy (...)** to indicate the minimum entropy to be provided. Comment: In fact, the value of *strength* is used in this determination, and *strength* is always at least *requested_strength.*
n	The order of the point *P* on the curve.
old_transformed_ seed	A record of the *seed_material* used in this previous instance of the DRBG.
order_P	The order of the point *P*. Note: For the NIST approved curves, the order of *Q* equals the order of *P*.
p	The modulus when *curve_type* = 0 (prime field); an *m*-bit prime.
P, Q	Random points on the elliptic curve *E*, such that each generates a large cyclic subgroup on *E.* The generating point *G* will be used as *P*.
prediction_resistance_flag	An indication of whether or not prediction resistance is to be provided by the DRBG.
pseudorandom_bits	The pseudorandom bits produced by the DRBG.
R	A value from which pseudorandom bits are extracted.
requested_curve_type	The *curve_type* can be specified as input to **Initialize_Dual_EC_DRBG (...)**; if none is requested the default value of 0 is assigned.
requested_no_of_bits	The number of pseudorandom bits to be returned on a call to **Dual_EC_DRBG (...)**.
requested_stength	The security *strength* of the bits requested from the DRBG.
s	A temporary value.

S	A value that is initially determined by a *seed*, but assumes new values during each request of pseudorandom bits from the DRBG.
seed_material	The seed used to derive the initial value of *S*.
seedlen	The length of the *seed_material*.
state	The state of the DRBG that is carried between calls to the generator. In the following specifications, the entire state is ([*usage_class*,] *counter*, *max_counter*, *S*, *curve_type*, [*p*], *a*, *b*, *n*, *P*, *Q*, *strength*, *prediction_resistance_flag* [, *transformed_seed*]). A particular element of the *state* is specified as *state.element*, e.g., *state.S*.
status	The *status* returned from a function call, where *status* = "Success" or an indication of a failure. Failure messages are: 1. Invalid *requested_strength.* 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.*
strength	The maximum strength of an instance of the DRBG (i.e., 80, 112, 128, 192 or 256).
temp	A temporary value.
temp_input	A temporary value.
transformed_seed	A record of the *seed_material* used in the current instance of the DRBG.
Truncate (*bits, in_len, out_len*)	A function that inputs a bit string of *in_len* bits, returning a string consisting of the leftmost *out_len* bits of input. If *in_len* < *out_len*, the input string is padded on the right with (*out_len* - *in_len*) zeroes, and the result is returned.
usage_class	The *usage_class* of a DRBG instance. This optional integer parameter may be used to differentiate instantiations of the **Dual_EC_DRBG (...)**, e.g., when there are multiple purposes being serviced that require differing strengths.
$x(A)$	The *x*-coordinate of the point *A* on the curve *E*.
φ	A mapping from field elements to non-negative integers, which takes the bit vector representation of a field element and interprets it as the binary expansion of an integer. Section 10.3.2.2.5 includes details of this mapping.
*	Scalar multiplication of a point on the curve.

10.3.2.2.2 Instantiation of Dual_EC_DRBG (...)

The following process or its equivalent **shall** be used to instantiate the **Dual_EC_DRBG (...)** process. Let **Hash (...)** be an Approved hash function for the security strengths to be supported. If the DRBG will be used for multiple security strengths, and only a single hash function will be available, that hash function **shall** be suitable for all supported security strengths (see SP 800-57).

Instantiate_Dual_EC_DRBG (...):

Input: integer ([*usage_class*,] *requested_strength, prediction_resistance_flag* [, integer ([*requested_curve_type*] [, *max_ctr*])]

Output: string *status*.

Process:

1. If (*requested_strength* > 256) , then **Return** ("Invalid *requested_strength*").

   Comment : Determine m appropriate for the requested strength : this will depend on *curve_type*.

2. If (*requested_curve_type* = 0), then
   Comment : choose one of the prime field curves :

   If (*requested_strength* $\leq$ 96), then {*strength* = 96, m = 192}
   Else if (*requested_strength* $\leq$ 112), then {*strength* = 112, m = 224}
   Else if (*requested_strength* $\leq$ 128), then {*strength* = 128, m = 256}
   Else if (*requested_strength* $\leq$ 192), then {*strength* = 192 , m = 384}
   Else if (*requested_strength* $\leq$ 256), then {*strength* = 256, m = 521}

   Comment : There is no NIST curve with m = 512 .

3. If (*requested_curve_type* $\neq$ 0), then
   Comment : choose one of the binary curves.

   If (*requested_strength* $\leq$ 80), then {*strength* = 80, m = 163}
   Else if (*requested_strength* $\leq$ 112), then {*strength* = 112, m = 233}
   Else if (*requested_strength* $\leq$ 128), then {*strength* = 128, m = 283}
   Else if (*requested_strength* $\leq$ 192), then {*strength* = 192 , m = 409}
   Else if (*requested_strength* $\leq$ 256), then {*strength* = 256, m = 571.

4. Choose a suitable elliptic curve E defined over Fp based on *requested_curve_type*, where p is an m-bit prime, or $F_2{}^m$ from Annex E.4.
5. Set the point P to the generator G and determine the *order_of_P*.
6. Select the appropriate Q from Annex E.4.

   Comment: Request *seed_material* of length m and the appropriate amount of *entropy*.

7. *min_entropy* = **max** (128, *strength*).
8. (*status*, *seed_material*) = **Get_entropy** (*min_entropy*, m, m).
9. If (*status* = "Failure"), then **Return** ("Failure indication returned by the entropy source").
10. *seedlen* = ||*seed_material*||.
11. (Optional) Get additional input and combine with the *seed_material*.
    11.1 (*status, additional_input*) = **Get_additional_input** ( ).
    11.2 If (*status* = "Failure"), then **Return** ("Failure from request for additional input").
    11.3 *seed_material* = *seed_material* || *additional_input*.

Comment [ebb1]: Page: 128
Note that we don't have an official strength of 96 bits. Should we check for a requested strength of 80 bits ?

Comment [ebb2]: Page: 128
Don't see that this is explicitly used.

Comment [ebb3]: Page: 128
The same curve type as P ? Any other restrictions ?

Comment [ebb4]: Page: 129
Note that min_length is not requested_strength + 64 here.

Comment : Perform a one-way function on the *seed* values for later comparison

12. (Optional) *transformed_seed* = **Hash** (*seed_material*).
13. *ctr* = 0.
14. *S* = **Hash_df** (*seed_material*, *seedlen*). Comment : See Section 9.6.3.2.
15. If *max_ctr* not present as an input parameter, then *max_ctr* = 0.

    Comment : Setting *max_counter* = 0 means that there is no maximum.
16. *state* = {[*usage_class*,] *ctr*, *max_ctr*, *S*, *curve_type*, *m*, [*p*], *a*, *b*, *n*, *P*, *Q*, *strength*, *prediction_resistance_flag* [, *transformed_seed*]}.
17. **Return** ("Success").

Comment [ebb5]: Page: 129
Should a hash derivation function be used, or can we devise an EC derivation function ?

10.3.2.2.3 Reseeding of a Dual_EC_DRBG (...) Instantiation

The following process or its equivalent **shall** be used to reseed the **Dual_EC_DRBG (...)** process, after it has been instantiated. Let **Hash (...)** be an Approved hash function for the security strengths to be supported.

Reseed_Dual_EC_DRBG_Instantiation (...):

Input: integer [*usage_class*].

Output: string *status*.

Process:

1. If a *state* is not available for the indicated *usage_class*, **Return** ("State not available for the indicated *usage_class*").
2. Get the appropriate *state* values for the indicated *usage_class*, e.g., *S* = *state.S*, *m* = *state.m*, *strength* = *state.strength*, *old_transformed_seed* = *state.transformed_seed*.
3. Perform steps 7-13 of **Instantiate_Dual_EC_DRBG (...)**.
    - 3.1 *min_entropy* = **max** (128, *strength*).
    - 3.2 (*status*, *seed_material*) = **Get_entropy** (*min_entropy*, *m*, *m*).
    - 3.3 If (*status* = "Failure"), then **Return** ("Failure indication returned by the entropy source").
    - 3.4 *seedlen* = || *seed_material* ||.
    - 3.5 (Optional) Get additional input and combine with the *seed_material*.
        - 3.5.1 (*status*, *additional_input*) = **Get_additional_input** ( ).
        - 3.5.2 If (*status* = "Failure"), then **Return** ("Failure from request for additional input").
        - 3.5.3 *seed_material* = *seed_material* || *additional_input*.
    - 3.6 *transformed_seed* = **Hash** (*seed_material*).
    - 3.7 *ctr* = 0.
4. If (*transformed_seed* = *old_transformed_seed*), then **Return** ("Entropy source failure").
5. *temp* = **Hash_df** ((*S* || *seed_material*), *B*).
6. *S* = **Truncate** (*temp*, *B*, *m*).
7. Update the changed values in the *state*.
    - 7.1 *state.S* = *S*.
    - 7.2 *state.transformed_seed* = *transformed_seed*
    - 7.3 *state.ctr* = *ctr*.

8. **Return** ("Success").

10.3.2.2.4 Generating Pseudorandom Bits Using Dual_EC_DRBG (...)

The following process or its equivalent **shall** be used to generate pseudorandom bits.

Dual_EC_DRBG (...):

Input: integer ([*usage_class*,] *requested_no_of_bits*, *requested_strength*, *additional_input_flag*).

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

Process:

1. If a *state* is not available for the indicated *usage_class*, **Return** ("State not available for the indicated *usage_class*", Null).
2. Get the appropriate *state* values for the indicated *usage_class*, e.g., S = *state.S*, m = *state.m*, *strength* = *state.strength*, P = *state.P*, Q = *state.Q*, *ctr* = *state.ctr*, *max_ctr* = *state.max_ctr*, *prediction_resistance_flag* = *state.prediction_resistance_flag*.
3. If (*requested_strength* > *strength*), then **Return** ("Incorrect *requested_strength*", Null).
4. If ((*additional_input_flag* < 0) or (*additional_input_flag* > 1)), then **Return** ("Invalid *additional_input_flag* value", Null).
5. If (*additional_input_flag* = 0), then *additional_input* = 0   Comment: m zeroes
   Else do
   5.1 (*status*, *temp_input*) = **Get_additional_input** ( ).
   5.2 If (*status* = "Failure"), then **Return** ("Failure from request for *additional_input*", Null).
   5.3 *temp_input* = **Hash** (*temp_input*).
   5.4 *additional_input* = **Truncate** (*temp_input*, B, m).
   Comment: Determine whether reseeding is required.
6. *temp* = the Null string.
7. $i = 0$.
8. If ((*max_ctr* > 0) and (*ctr* = *max_ctr*)), then
   8.1 *status* = **Reseed_Dual_EC_DRBG** ([*usage_class*]).
   8.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
9. $s = S \oplus additional\_input$.   Comment: s is to be interpreted as an m-bit unsigned integer. To be precise, s should be reduced mod n; the scalar * will affect this.
10. $S = \varphi(\, x(s * P))$.   Comment: S is an m-bit number.
11. $R = \varphi(\, x(S * Q))$.   Comment: R is an m-bit number. See footnote [1].

[1] The precise definition of $\varphi(x)$ used in steps 12 and 13 depends on the field representation of the curve points. In keeping with the convention of FIPS 186-2, the following elements will be associated with each other:

B: $| c_{m-1} | c_{m-2} | \ldots | c_1 | c_0 |$, a bitstring, with c_{m-1} being leftmost

12. *temp* = *temp* || *R*.
13. $i = i + 1$.
14. *ctr* = *ctr*+1.
15. If (||*temp*|| < *requested_no_of_bits*), then go to step 7.
16. *pseudorandom_bits* = **Truncate** (*temp*, $i \times B$, *requested_no_of_bits*).
17. If (*prediction_resistance_flag* = 1), then
    17.1 *status* = **Reseed_Dual_EC_DRBG** ([*usage_class*]).
    17.2 If (*status* ≠ "Success"), then **Return** (*status*, Null).
    Else Update the changed values in the *state*.
    17.3 *state.S* = *S*.
    17.4 *state.ctr* = *ctr*.
18. **Return** ("Success", *pseudorandom_bits*).

Comment [ebb6]: Page: 132
This demands input each time unless the additional_input_flag = 0. Is this what is wanted ?

10.3.2.2.5 Adding Additional Entropy to Dual_EC_DRBG (...)

The **Dual_EC_DRBG (...)** may be reseeded at any time. There is also the *additional_input* parameter that allows a bitstring to be added to the current state (*seed*) whenever **Dual_EC_DRBG (...)** is invoked.

Add_Entropy_to_Dual_EC_DRBG (...):

Input: integer ([*usage_class*,] *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. Get the appropriate *state* values for the indicated *usage_class*, e.g., *S* = *state.S*, *m* = *state.m*, *strength* = *state.strength*, *P* = *state.P*, *Q* = *state.Q*, *ctr* = *state.ctr*, *max_ctr* = *state.max_ctr*, *prediction_resistance_flag* = *state.prediction_resistance_flag*.
3. (*status*, *additional_entropy*) = **Get_entropy** (1, 1, *inlen*).
4. If (*status* = "Failure"), then **Return** ("Failure from request for additional entropy").
5. If ((*additional_entropy* = Null) and (*always_update_flag* = 0)), then **Return** ("No update performed").
6. Perform steps 5.3-17 of **Dual_EC_DRBG (...)**.
    6.1 *temp_input* = **Hash** (*temp_input*).
    6.2 *additional_input* = **Truncate** (*temp_input*, *B*, *m*).

Comment: Determine whether reseeding is required.

Z: $c_{m-1}2^{m-1} + \ldots + c_2 2^2 + c_1 2^1 + c_0 \in Z$;
Fa: $c_{m-1}2^{m-1} + \ldots + c_2 2^2 + c_1 2^1 + c_0 \bmod p \in GF(p)$;
Fb: $c_{m-1}t^{m-1} \oplus \ldots \oplus c_2 t^2 \oplus c_1 t \oplus c_0 \in GF(2^m)$, when a polynomial basis is used;
Fc: $c_{m-1}\beta \oplus c_{m-2}\beta^2 \oplus c_{m-3}\beta^{2^2} \oplus \ldots \oplus c_0\beta^{2^{m-1}} \in GF(2^m)$, when a normal basis is used.

Thus, any field element *x* of the form *Fa*, *Fb* or *Fc* will be converted to the integer *Z* or bitstring *B*, and vice versa, as appropriate.

6.3 $temp$ = the Null string.
6.4 $i = 0$.
6.5 If (($max\_ctr > 0$) and ($ctr = max\_ctr$)), then
  6.5.1 $status$ = **Reseed_Dual_EC_DRBG** ([*usage_class*]).
  6.5.2 If ($status \neq$ "Success"), then **Return** (*status*, Null).
6.6. $s = S \oplus additional\_input$. Comment: s is to be interpreted as an m-bit unsigned integer. To be precise, s should be reduced mod n; the scalar * will affect this.
6.7 $S = \varphi(\, x(s * P))$. Comment: S is an m-bit number.
6.8 $R = \varphi(\, x(S * Q))$. Comment: R is an m-bit number. See footnote.
6.9 $temp = temp \,||\, R$.
6.10 $i = i + 1$.
6.11 $ctr = ctr+1$.
6.12 If ($||temp|| < requested\_no\_of\_bits$), then go to step 3.
6.13 $pseudorandom\_bits$ = **Truncate** ($temp, i \times B, requested\_no\_of\_bits$).
6.14 If ($prediction\_resistance\_flag = 1$), then
  6.14.1 $status$ = **Reseed_Dual_EC_DRBG** ([*usage_class*]).
  6.14.2 If ($status \neq$ "Success"), then **Return** (*status*, Null).
  Else Update the changed values in the *state*.
  6.14.3 $state.S = S$.
  6.14.4 $state.ctr = ctr$.
7. **Return** ("Success").

Comment [ebb7]: Page: 133
This demands input each time unless the additional_input_flag = 0. Is this what is wanted ?

10.3.2.3.7 Implementation Considerations

[To be inserted]

10.3.2.3 Generator Strength and Attributes

The particular curve used is based on *strength*, which is selected from one of five security levels and is always at least *requested_strength*. The curves and associated security levels are those given in Section 1.2 of FIPS 186-2; they are meant to correspond to the strengths of various standard symmetric encryption algorithms.

There are three curves associated with each security level, one defined over a prime field GF(p) and two over a binary field GF(2^m), where $2^m \approx p$. The mod p curves, assigned *curve_type* 0, are used by default. Any of these three curves may be used for the security level.

Initial seeding is accomplished with a call to **Get_entropy(…)**, which returns a bitstring of a specified length and entropy. The **Dual_EC_DRBG (...)** specifies **max** (128, *strength*) bits of entropy.

10.3.2.4 Reseeding and Rekeying

The reseeding process is covered in 10.3.2.2.3 . Reseeding may be performed "automatically", by using a nonzero value for *max_ctr* . Alternatively, or in addition, a call to **Reseed_Dual_EC_DRBG_Instantiation (...)** can be made at any time.

The **Dual_EC_DRBG (...)** is not keyed per se; however, the *additional_input* feature may be used to effect keying, if desired.