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 (...)**.

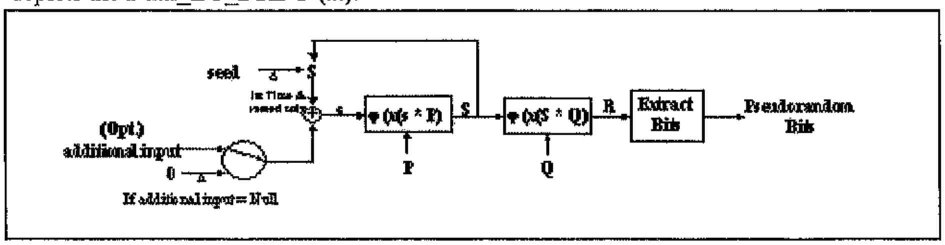

Figure 18: Dual_EC_DRBG (...)

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.

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.

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

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*.

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

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