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 modulo 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 using two random points in an elliptic curve group, where the curve is defined over a field approximately 2^m in size, where $m \geq 192$. Figure 16 depicts the **Dual_EC_DRBG (...)**.

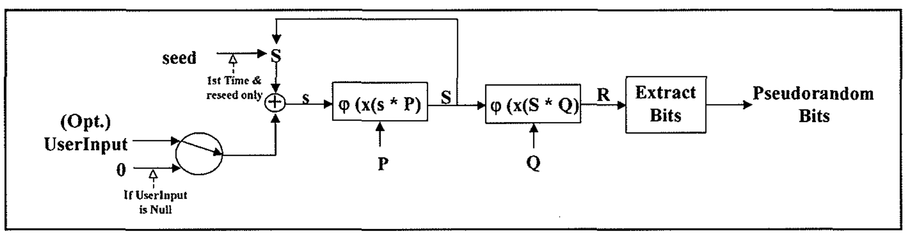

Figure 16: Dual_EC_DRBG (...)

The initialization of this DRBG requires the selection of an appropriate elliptic curve for the desired security strength. The curve **shall** be generated in accordance with ANSI X9.62 or X9.63, or **shall** be selected from the NIST Recommended curves. If the DRBG is reseeded, the same or a different curve may be used for the new DRBG instance.

Comment [ebb1]: Page: 91
The revised version.

Comment [ebb2]: Page: 91
What reference should be provided ? The revised X9.62 ? FIPS 186-2 ? Should there be other references ?

During initialization and reseeding, two random points (P and Q) **shall** be generated for the curve, such that each point generates a large cyclic group on the curve. These points may be considered as the equivalent of keys and **shall** be handled as keys (see SP 800-57). The points may be generated externally and entered into the DRBG during initialization and reseeding, or may be generated internally.

The *seed* used to define the initial value (S) of the DRBG **shall** have entropy that is at least twice the desired security *strength*. The length of the *seed* **shall** be m bits in length. Further requirements for the *seed* are provided in Section 9.4.

Comment [ebb3]: Page: 91
How is this guaranteed, given our current philosophy ? Presumably, when m = 192, 80 bits of security are obtained, requiring 160 bits of entropy. How does one get exactly 192 bits ? Do we use a KDF and run until we get enough bits ? Which KDF ?

When optional user input (*UserInput)* is used, the length and value of *UserInput* are arbitrary.

Figure 17 depicts the insertion of test input for the *seed* and the *UserInput*. 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 tothe error state.

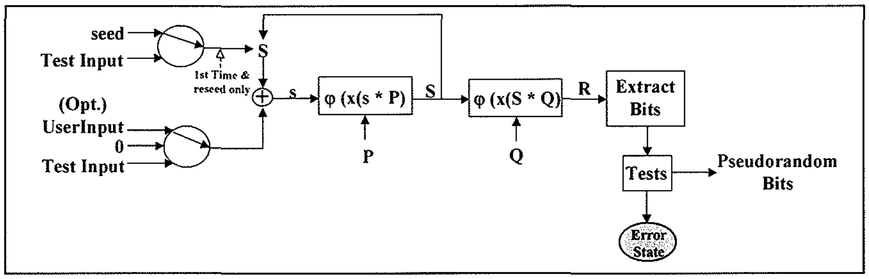

Figure 17: Dual_EC_DRBG (...) (with Tests)

10.3.2.2 Description

10.3.2.2.1 General

The initialization of **Dual_EC_DRBG (...)** consists of selecting an appropriate elliptic curve, selecting two random points on that curve 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 *purpose* of the DRBG instantiation; if the DRBG is used for multiple *purposes*, requiring multiple instantiations, then the *purpose* **shall** be indicated, and the implementation **shall** accommodate multiple *states* simultaneously; if the DRBG will be used for only one *purpose*, then the *purpose* **may** be omitted),
2. A value (S) that is updated during each request for pseudorandom bits,
3. (Optional) The m-bit prime modulus p for curves over F_p; if the DRBG will be using a curve over F_p, the value of p **shall** be present; otherwise, a value for p **shall not** be present,
5. The elliptic curve domain parameters (q, a, b, G, n [, h]), where q is the field size, a and b are two field elements that define the equation of the curve, G is a generating point of prime order on the curve, n is the order of the point G, and h is the optional cofactor,
6. The two random points on the curve (P and Q),
7. The maximum security *strength* provided by the instance of the DRBG, and
8. (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.

Comment [ebb4]: Page: 92
Note that the basis indication and the SEED are not present. Do we need them ?

The *state* **shall** be retained within the DRBG.

For each request of pseudorandom bits, a *requested_strength* is provided and checked against the *strength* indicated in the *state*. If the *requested_strength* exceeds the *strength* afforded by the DRBG, an indication of failure is returned.

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

a, *b*	Two field elements that define the equation of the curve.
E	An elliptic curve defined over F_{2^m} or over F_p.
entropy	The entropy of the *seed.*
G	A generating point of prime order on the curve.
h	The cofactor.
i	A temporary value that is used as a loop counter.
m	Length of the seed; $m \geq 192$.
n	The order of the point *G* on the curve.
no_of_bits_per_block	The number of pseudorandom bits to be used from each iteration of the elliptic curve process.
NRBG (*entropy*)	A function that acquires a string of bits from an Approved NRBG or an Approved DRBG (or chain of Approved DRBGs) that is seeded by an Approved NRBG. The parameter indicates the *entropy* to be provided in the returned bits.
order_P	The order of the point *P*.
order_Q	The order of the point *Q*.
p	The modulus; an *m*-bit prime, where $m \geq 192$.
P, *Q*	Random points on the elliptic curve *E*, such that each generates a large cyclic subgroup on *E*..
pseudorandom_bits	The pseudorandom_bits produced by the DRBG.
purpose	The *purpose* of a DRBG instance.
q	The field size of the curve.
R	A value from which pseudorandom bits are extracted.
random_bits	A temporary value.
random_for_P	A random value used to determine the point *P*.
random_for_Q	A random value used to determine the point *Q*.
requested_no_of_bits	The number of pseudorandom bits to be generated.
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.
save_ seed	A representation of the *seed* of this instance of the DRBG.
seed	The *seed* for this instance of the DRBG. The *seed* is used to derive the initial value of *S*.
selected_random_bits	A temporary value.
state	The state of the DRBG that is carried between calls to the generator. In the following specifications, the entire state is ([*purpose*,] *S*, *m*, [*p*,] *q*, *a*, *b*, *G*, *n*, [*h*], *P*, *order_P*, *Q*, *order_Q*, *strength* [, *save_initial_state*]). A particular element of the *state* is specified as *state.element*, e.g., *state.S*.
status	The *status* returned from the initialization, reseeding or **Dual_EC_DRBG (..)** processes, where *status* = Success or Failure.
strength	The maximum strength of an instance of the DRBG (see Table XX).
temp	A temporary value.
UserInput	Optional user input.
UserInput_flag	A flag that indicates whether or not user input may be used, with values as follows: 1 = Request *UserInput*, but return 0 if no input is available. 2 = Request *UserInput*; wait until *UserInput* is available before continuing. 3 = Obtain *UserInput* only during the first loop of the call. If no input is available, return 0. Subsequent loops should use the same value as the first loop. 4 = Obtain *UserInput* only during the first loop of the call; wait until *UserInput* is available before continuing. Subsequent loops should use the same value as the first loop. If an implementation does not require this flexibility, the implementation may include only the appropriate capability (e.g., if *UserInput* is not desired, then the step requesting *UserInput* may be eliminated).
x(*A*)	The *x*-coordinate of the point *A* on the curve *E*.
φ	A mapping from field elements to integers, which takes the bit vector representation of a field element and interprets it as the binary expansion of an integer.

Comment [ebb5]: Page: 95
Does this need to be defined here ?

* Scalar multiplication.

10.3.2.2.2 Initialization of Dual_EC_DRBG (...)

The following process or its equivalent **shall** be used to initialize the **Dual_EC_DRBG (...)** process. Let **One_Way (...)** be a one-way function.

Comment [ebb6]: Page: 95
We probably want to specify something here, e.g., an Approved hash function of the appropriate strength.

Initialize_Dual_EC_DRBG (...):

Input: integer ([*purpose*,] *m*, *requested_strength*).

Comment [ebb7]: Page: 95
Is this the entropy ?

Output: integer *status*, where *status* = Success or Failure.

Process:

1. If ($m < 192$), then **Return** (Failure).

   Comment : Determine the *strength* by the size of *m*.

2. If ($m < 224$), then *strength* = 80

   Else if ($m < 256$), then *strength* = 112

   Else if ($m < 384$), then *strength* = 128

   Else if ($m < 512$), then *strength* = 192

   Else *strength* = 256.

3. If (*requested_strength* > *strength*), then **Return** (Failure).

4. Choose a suitable elliptic curve E defined over F_{2^m} or over F_p, where p is an m-bit prime and $m \geq 192$. The curve may be entered from an external source, or may be generated interally. NIST Recommended elliptic curves are provided in FIPS 186-2/3. This process results in values for q, a, b, G, n [, h] where q is the field size, a and b are two field elements that define the equation of the curve, G is a generating point of prime order on the curve, n is the order of the point G, and h is the cofactor. For curves ov F_p, the value of p **shall** also be obtained.

   Comment: Select random values for two points, P and Q.

5. If P and Q are to be obtained from an external source, then

   5.1 Get P.

   5.2 Determine the *order_of_P*.

   5.3 If (*order_of_P* < ???), then go to step 5.1.

   Comment [ebb8]: Page: 96
   Is this right ?

   5.4 Get Q.

   5.5 Determine the *order_of_Q*.

   5.6 If (*order_of_Q* < ???), then go to step 5.4.

   5.7 Go to step 8.

Comment: Get points P and Q using the random values.

6. If the curve is over F_p:
   6.1 *entropy* = $\| p \|$ + 64.
   6.2 *random_for_P* = (**NRBG** (*entropy*)) mod *p*.
   6.3 Using *random_for_P*, determine a point *P*.
   6.4 Determine the *order_of_P*.
   6.5 If (*order_of_P* < ???), then go to step 6.2.
   6.6 *random_for_Q* = (**NRBG** (*entropy*)) mod *p*.
   6.7 Using *random_for_Q*, determine a point *Q*.
   6.8 Determine the *order_of_Q*.
   6.9 If (*order_of_Q* < ???), then go to step 6.6.
   6.10 Go to step 8.
7. If the curve is over $F_2{}^m$:
   7.1 *entropy* = *m* + 64.
   7.2 *random_for_P* = **NRBG** (*entropy*) mod *m*.
   7.3 Using *random_for_P*, determine a point *P*.
   7.4 Determine the *order_of_P*.
   7.5 If (*order_of_P* < ???), then go to step 7.2.
   7.6 *random_for_Q* = **NRBG** (*entropy*) mod *m*.
   7.7 Using *random_for_Q*, determine a point *Q*.
   7.8 Determine the *order_of_Q*.
   7.9 If (*order_of_Q* < ???), then go to step 7.6.
8. *entropy* = *m* + 64.

Comment: Request the *seed* material.

9. *S* = *seed* = **NRBG** (*entropy*) mod *m*.

Comment : Perform a one-way function on the state values for later comparison.

11. (Optional) *save_seed* = **One_Way** (*seed*).
12. *state* = {[*purpose*,] *S*, *m*, [*p*,] *q*, *a*, *b*, *G*, *n*, [*h*], *P*, *Q*, *strength* [, *save_seed*]}.
13. **Return** (Success).

Comment [ebb9]: Page: 96
In accordance with Section 11.3 of Part 1.

Comment [ebb10]: Page: 96
This will provide enough extra bits in steps 7.2 and 7.3 that a mod can be used to reduce to the desired number of bits (*m*).

10.3.2.2.3 Reseeding of Dual_EC_DRBG (...)

The following process or its equivalent **shall** be used to reseed the **Dual_EC_DRBG (...)** process. Let **One_Way (...)** be a one-way function.

Reseed_Dual_EC_DRBG (...):

Input: integer ([*purpose*,] *m*, *requested_strength*).

Output: integer *status*, where *status* = Success or Failure.

Process:

1. If ($m < 192$), then **Return** (Failure).
2. Get the appropriate *state* values for the indicated *purpose*, e.g., q = *state.q*, a = *state.a*, b = *state.b*, *save_seed* = *state.save_seed*. If a *state* is not available for the indicated *purpose*, **Return** (Failure).
3. If ($m < 224$), then *strength* = 80

   Else if ($m < 256$), then *strength* = 112

   Else if ($m < 384$), then *strength* = 128

   Else if ($m < 512$), then *strength* = 192

   Else *strength* = 256.

   Comment [ebb11]: Page: 97
   There is an assumption that the strength of the new DRBG instance and the old DRBG instance need not be the same.
4. If (*requested_strength* > *strength*), then **Return** (Failure).

   Comment : Select a new curve, if desired ; otherwise, use the old curve.
5. (Optional) Choose a suitable elliptic curve E defined over F_{2^m} or over F_p, where p is an m-bit prime and $m \geq 192$. The curve may be entered from an external source, or may be generated interally. NIST Recommended elliptic curves are provided in FIPS 186-2/3. This process results in values for q, a, b, G, n [, h] where q is the field size, a and b are two field elements that define the equation of the curve, G is a generating point of prime order on the curve, n is the order of the point G, and h is the cofactor. For curves ov F_p, the value of p **shall** also be obtained.
6. If P and Q are to be obtained from an external source, then

   6.1 Get P.

   6.2 Determine the *order_of_P*.

   6.3 If (*order_of_P* < ???), then go to step 6.1.

   6.4 Get Q.

   6.5 Determine the *order_of_Q*.

   6.6 If (*order_of_Q* < ???), then go to step 6.4.

   6.7 Go to step 9.
7. If the curve is over F_p:

7.1 *entropy* = $\| p \|$ + 64.

7.2 *random_for_P* = (**NRBG** (*entropy*)) mod *p*.

7.3 Using *random_for_P*, determine a point *P*.

7.4 Determine the *order_of_P*.

7.5 If (*order_of_P* < ???), then go to step 7.2.

7.6 *random_for_Q* = (**NRBG** (*entropy*)) mod *p*.

7.7 Using *random_for_Q*, determine a point *Q*.

7.8 Determine the *order_of_Q*.

7.9 If (*order_of_Q* < ???), then go to step 7.6.

7.10 Go to step 9.

8. If the curve is over F_2^m:

8.1 *entropy* = *m* + 64.

8.2 *random_for_P* = **NRBG** (*entropy*) mod *m*.

8.3 Using *random_for_P*, determine a point *P*.

8.4 Determine the *order_of_P*.

8.5 If (*order_of_P* < ???), then go to step 8.2.

8.6 *random_for_Q* = **NRBG** (*entropy*) mod *m*.

8.7 Using *random_for_Q*, determine a point *Q*.

8.8 Determine the *order_of_Q*.

8.9 If (*order_of_Q* < ???), then go to step 8.6.

Comment: Get points *P* and *Q* using the random values.

9. *entropy* = *m* + 64.

Comment: Request the *seed* material.

10. *S* = *seed* = **NRBG** (*entropy*) mod *m*.

Comment : Perform a one-way function on the state values for later comparison.

11. *temp* = **One_Way** (*seed*).

12. If (*temp* = *save_ seed*), then go to step 5.

13. *state* = {[*purpose*,] *S*, *m*, [*p*,] *q*, *a*, *b*, *G*, *n*, [*h*], *P*, *Q*, *strength* [, *save_seed*]}.

14. **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 ([*purpose*,] *requested_no_of_bits*, *no_of_bits_per_block*, *requested_strength*, *UserInput_flag*).

Output: integer (*status*, *pseudorandom_bits*), where *status* = Success or Failure.

Process:

1. Set up the *state* in accordance with the indicated purpose, e.g., S = *state.S*, m = *state.m*, p = *state.p*, P = *state.P*, Q = *state.Q*, *strength* = *state.strength, etc.*.
2. If ((*no_of_bits_per_block* < 0) or (*no_of_bits_per_block* > m)), then **Return** (Failure, 0).
3. If (*requested_strength* > *strength*), then **Return** (Failure, 0).
4. If ((*UserInput_flag* < 0) or (*UserInput_flag* > 4)), then **Return** (Failure, 0).
5. *temp* = then Null string.
6 If (*UserInput_flag* = 0), then *UserInput* = 0

   Else If ((*UserInput_flag* = 3) and ($i \neq 0$)), then go to step 7

   Else *UserInput* = **Get_userInput** (*UserInput_flag*).

7 $s = (S \oplus UserInput)$.

   Comment: If *UserInput* is shorter than S, then XOR *UserInput* to the rightmost bits of S.

8. $S = \varphi(\ x(s * P)\ )$
9. $R = \varphi(\ x(S * Q)\ )$.

   Comment: R is an m-bit number.

   Comment [ebb12]: Page: 99
   Do these require substeps ?

10. If the curve is over Fp:

    10.1 Do $i = m$ to 1 by -1

    If ((Bit i of p = 1) and (Bit i of R = 0)), then go to step 10.2.

    10.2 *random_bits* = Rightmost (i - 1) bits of R.

    10.3 Go to step 12.

11. If the curve is over F_2^m: *random_bits* = Rightmost (m-8) bits of R.
12. If (*no_of_bits_per_block* > || *random_bits*), then *selected_random_bits* = *random_bits*

    Else *selected_random_bits* = Leftmost *no_of_bits_per_block* of *random_bits*.

13. *temp* = *temp* || *selected_random_bits*.
14. If (|| *temp* || < *requested_no_of_bits*), then go to step 6.
15. *pseudorandom_bits* = Leftmost *requested_no_of_bits* of *temp*.

16. Update the changed values in the *state*, e.g., *state.S* = *S*.

17. **Return** (Success, *pseudorandom_bits*).

10.3.2.3 Generator Strength and Attributes

[To be determined]

10.3.2.4 Reseeding and Rekeying

[To be determined]