Below are Table 4 from Section 10.3.1, and the example in Appendix F.5. I have some questions,

1. In the table, which entries are required for the curve to work properly, and which entries are really dependent on the requested security strength during instantiation? For example, if a consuming application requests instantiation at the 112-bit security level, theoretically, any curve could be used. If it is determined that the P-256 curve will be used in this case, do all the values in the P-256 column need to be used, or can lesser values be used in some cases.
    a. Can the *min_length*, for example, be reduced to 224? Why is the *min_length* about twice the minimum entropy anyway? To give a big security cushion?
    b. Can the *seedlen* be 224, as it was for the P-224 curve, or must it be 256?
    c. Can the *max_outlen* be 208, or must it be 240?

    These same questions apply if any curve is used to support a security strength < the max. that it can support?
2. Please check the blue text that is highlighted in gray. Does it make sense?

Table 4: Definitions for the Dual_EC_DRBG

	~~P-224~~ **P-256**	**P-384**	**P-521**
Supported security strengths	See SP 800-57		
highest_supported_ security_strength	See SP 800-57		
Output block length (*max_outlen* = largest multiple of 8 less than *seedlen* - (13 + $\log_2$ (the cofactor))	~~208~~ 240	368	504
Required minimum entropy for instantiate and reseed	*security_strength*		
Minimum entropy input length (*min_length* = 8 × ⌈*seedlen*/8⌉)	~~224~~ 256	384	528
Maximum entropy input length (*max_length*)	$\leq 2^{13}$ bits		
Maximum personalization string length (*max_personalization_string_length*)	$\leq 2^{13}$ bits		
Supported security strengths	See SP 800-57		
Seed length (*seedlen* = *m*)	~~224~~ 256	384	521

	~~P-224~~ P-256	P-384	P-521
Appropriate hash functions	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	SHA-224, SHA-256, SHA-384, SHA-512	SHA-256, SHA-384, SHA-512
max_number_of_bits_per_request	*max_outlen* × *reseed_interval*		
Number of blocks between reseeding (*reseed_interval*)	$\leq 2^{32}$ blocks		

~~**The generate function is the same as that provided in Annex E.3.5.**~~

F.5 Dual_EC_DRBG Example

This example of **Dual_EC_DRBG** allows a consuming application to instantiate using any of the threefour prime curves. The elliptic curve to be used is selected during instantiation in accordance with the following:

requested_instantiation_security_strength	**Elliptic Curve**
≤ 112	P-256
113 – 128	P-256
129 – 192	P-384
193 – 256	P-512

A reseed capability is available, but prediction resistance is not available. Both a *personalization_string* and an *additional_input* are allowed. A total of 10 internal states are provided. For this implementation, the algorithms are provided as inline code within the functions.

The nonce for instantiation (*instantiation_nonce*) consists of a random value with *security_strength*/2 bits of entropy; the nonce is obtained by a separate call to the **Get_entropy_input** routine than that used to obtain the entropy input itself.

The internal state contains values for *s*, *seedlen*, *p*, *a*, *b*, *n*, *P*, *Q*, ~~*r_old*~~, *block_counter* and *security_strength*. In accordance with Table 4 in Section 10.3.1, security strengths of 112, 128, 192 and 256 may be supported. SHA-256 has been selected as the hash function. The following definitions are applicable for the instantiate, reseed and generate functions:

1. *highest_supported_security_strength* = 256.
2. Output block length (*outlen*): See Table 4.
3. Required minimum entropy for the entropy input at instantiation and reseed = *security_strength*.

4. Minimum entropy input length (*min _length*): See Table 4.
5. Maximum entropy input length (*max _length*) = 1000 bits.
6. Maximum personalization string length (*max_personalization_string_length*) = 800 bits.
7. Maximum additional input length (*max_additional_input_length*) = 800 bits.
8. Seed length (*seedlen*): See Table 4.
9. Maximum number of bits per request (*max_number_of_bits_per_request*) = 1000 bits.
10. Reseed interval (*reseed_interval*) = ~~10,000~~ 2^{32} blocks.

F.5.1 Instantiation of Dual_EC_DRBG

This implementation will return a text message and an invalid state handle (-1) when an **ERROR** is encountered. **Hash_df** is specified in Section 10.4.1.

Instantiate_Dual_EC_DRBG (...):

Input: integer (*requested_instantiation_security_strength*), bitstring *personalization_string*.

Output: string *status,* integer *state_handle*.

Process:

Comment : Check the validity of the input parameters.

1. If (*requested_instantiation_security_strength* > 256) then **Return** ("Invalid *requested_instantiation_security_strength*", -1).
2. If (**len** (*personalization_string*) > 800), then **Return** ("*personalization_string* too long", -1).

Comment : Select the prime field curve in accordance with the *requested_instantiation_security_strength*

3. If *requested_instantiation_security_strength* ≤ 112), then

   {*security_strength* = 112; *seedlen* = 256; *outlen* = 240; *min_entropy_input_len* = 256}

   Else if (*requested_instantiation_security_strength* ≤ 128), then

   {*security_strength* = 128; *seedlen* = 256; *outlen* = 240; *min_entropy_input_len* = 256}

   Else if (*requested_instantiation_security_strength* ≤ 192), then

   {*security_strength* = 192;, *seedlen* = 384; *outlen* = 368; *min_entropy_input_len* = 384}

Else {*security_strength* = 256;, *seedlen* = 521; *outlen* = 504;
min_entropy_input_len = 528}.

4. Select the appropriate elliptic curve from Appendix A using the Table in Appendix F.5 to obtain the domain parameters *p*, *a*, *b*, *n*, *P*, and *Q*.

Comment: Request *entropy_input*.

5. (*status*, *entropy_input*) = **Get_entropy_input** (*security_strength*, *min_entropy_input_length*, 1000).
6. If (*status* ≠ "Success"), then **Return** ("~~Failure indication returned by~~Catastrophic failure of the *entropy_input* source:" || *status*, -1).
7. (*status*, *instantiation_nonce*) = **Get_entropy_input** (*security_strength*/2, *security_strength*/2, 1000).
8. If (*status* ≠ "Success"), then **Return** ("Catastrophic failure of ~~Failure indication returned by~~ the random nonce source:" || *status*, -1).

Comment: Perform the instantiate algorithm.

9. *seed_material* = *entropy_input* || *instantiation_nonce* || *personalization_string*.
10. *s* = **Hash_df** (*seed_material*, *seedlen*).
11. ~~*r_old* = φ(*x*(*s* * *Q*)).~~
12. ~~*block_counter*~~ = 0.

Comment: Find an unused internal state and save the initial values.

~~13~~12. (*status*, *state_handle*) = **Find_state_space** ( ).

~~14~~13. If (*status* ≠ "Success"), then **Return** (*status*, -1).

~~15~~14. *internal_state* (*state_handle*) = {*s*, *seedlen*, *p*, *a*, *b*, *n*, *P*, *Q*, ~~*r_old*, *block_counter*~~, *security_strength*}.

~~16~~15. **Return** ("Success", *state_handle*).

F.5.2 Reseeding a Dual_EC_DRBG Instantiation

The implementation is designed to return a text message as the status when an error is encountered.

Reseed_Dual_EC_DRBG_Instantiation (...):

Input: integer *state_handle*, string *additional_input_string*.

Output: string *status*.

Process:

Comment: Check the input parameters.

1. If ((*state_handle* < 0) or (*state_handle* > 9) or (*internal_state* (*state_handle*).*security_strength* = 0)), then **Return** ("State not available for the *state_handle*").
2. If (**len** (*additional_input*) > 800), then **Return** ("Additional_input too long").

   Comment: Get the appropriate *state* values for the indicated *state_handle.*
3. *s* = *internal_state* (*state_handle*).*s*, *seedlen* = *internal_state* (*state_handle*).*seedlen*, *security_strength* = *internal_state* (*state_handle*).*security_strength.*

   Comment: Request new *entropy_input* with the appropriate entropy and bit length.
3. (*status*, *entropy_input*) = **Get_entropy_input** (*security_strength*, *min_ entropy_input_length*, 1000).
4. If (*status* ≠ "Success"), then **Return** ("Catastrophic failure of ~~Failure indication returned by~~ the entropy source:"|| *status*).

   Comment: Perform the reseed algorithm.
5. *seed_material* = **pad8** (*s*) || *entropy_input* || *additional_input.*
6. *s* = **Hash_df** (*seed_material*, *seedlen*).

   Comment: Update the changed values in the *state.*
7. *internal_state* (*state_handle*).*s* = *s.*
8. *internal_state.block_counter* = 0.
9. **Return** ("Success").

F.5.3 Generating Pseudorandom Bits Using Dual_EC_DRBG

The implemenation returns a *Null* string as the pseudorandom bits if an error is encountered.

Dual_EC_DRBG (...):

Input: integer (*state_handle*, *requested_security_strength*, *requested_no_of_bits*), bitstring *additional_input.*

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

Process:

Comment: Check for an invalid *state_handle.*

1. If ((*state_handle* < 0) or (*state_handle* > 9) or (*internal_state* (*state_handle*) = 0)), then **Return** ("State not available for the *state_handle*", *Null*).

Comment: Get the appropriate *state* values for the indicated *state_handle*.

2. *s* = *internal_state* (*state_handle*).*s*, *seedlen* = *internal_state* (*state_handle*).*seedlen*, *P* = *internal_state* (*state_handle*).*P*, *Q* = *internal_state* (*state_handle*).*Q*, ~~*r_old = internal_state (state_handle).r_old,*~~ *block_counter* = *internal_state* (*state_handle*).*block_counter*.

Comment: Check the rest of the input parameters.

3. If (*requested_number_of_bits* > 1000), then **Return** ("Too many bits requested", *Null*).
4. If (*requested_security_strength* > *security_strength*), then **Return** ("Invalid *requested_strength*", *Null*).
5. If (**len** (*additional_input*) > 800), then **Return** ("Additional_input too long", *Null*).

Comment: Check whether a reseed is required.

6. If $\left(block\_counter + \left\lceil \frac{requested\_number\_of\_bits}{outlen} \right\rceil > \text{~~}10{,}000_2{}^{32}\text{~~}\right)$, then

   6.1 **Reseed_Dual_EC_DRBG_Instantiation** (*state_handle*, *additional_input*).

   6.2 If (*status* ≠ "Success"), then **Return** (*status*).

   6.3 *s* = *internal_state* (*state_handle*).*s*, *block_counter* = *internal_state* (*state_handle*).*block_counter*.

   6.4 *additional_input* = *Null*.

Comment: Execute the generate algorithm.

7. If (*additional_input* = *Null*) then *additional_input* = 0

Comment: *additional_input* set to *m* zeroes.

Else *additional_input* = **Hash_df** (**pad8** (*additional_input*), *seedlen*).

Comment: Produce *requested_no_of_bits*, *outlen* bits at a time:

8. *temp* = the *Null* string.
9. *i* = 0.
10. $t = s \oplus additional\_input$.
11. $s = \varphi(x(t * P))$.

12. $r = \varphi(x(s * Q))$.

13. ~~If ($r = r\_old$), then **Return** ("ERROR: outputs match", *Null*).~~

~~14. $r\_old = r$.~~

~~15.~~ $temp = temp \,||$ (**rightmost** *outlen* bits of r).

~~16~~14. $additional\_input = 0^{seedlen}$. Comment: *seedlen* zeroes; *additional_input* is added only on the first iteration.

~~17~~15. $block\_counter = block\_counter + 1$.

~~18~~16. $i = i + 1$.

~~19~~17. If (**len** (*temp*) < *requested_no_of_bits*), then go to step 10.

~~20~~18. *pseudorandom_bits* = **Truncate** (*temp*, $i \times outlen$, *requested_no_of_bits*).

Comment: Update the changed values in the *state*.

~~21~~19. *internal_state.s* = *s*.

~~22~~20. ~~*internal_state.r_old = r_old.*~~

~~23.~~ *internal_state.block_counter* = *block_counter*.

~~24~~21. **Return** ("Success", *pseudorandom_bits*).

Email note to interested parties:
A draft NIST Special Publication (Draft SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*) is available for public comment at http://csrc.nist.gov/publications/drafts.html. Comments should be submitted to ebarker@nist.gov by Wednesday, February 1, 2006. Please place "Comments on SP 800-90" in the subject line.

Instructions to Patrick:
Please place the following on the csrc page:

December 16, 2005: A draft NIST Special Publication (Draft SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*) is available[EBB1] for public comment. Comments should be submitted to ebarker@nist.gov by Wednesday, February 1, 2006. Please place "Comments on SP 800-90" in the subject line.

The draft document is attached.

Instructions to Larry Bassham:
Please place the following on the http://csrc.nist.gov/CryptoToolkit/tkrng.html page:

A draft NIST Special Publication (Draft SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*) is available[EBB2] for public comment. Comments should be submitted to ebarker@nist.gov by Wednesday, February 1, 2006. Please place "Comments on SP 800-90" in the subject line.

Patrick will be placing the document on the drafts page.