Comments Received on SP 800-90A

On 9/11/13 12:06 AM, "Robert Bushman" <bob@traxel.com> wrote:

"Draft Special Publication 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators" cannot be trusted to secure our citizens and corporations from cyber-attack, for reasons that should be quite apparent. Please terminate it or replace the PRNG algorithm without the participation of the NSA.

On 9/21/13 1:25 PM, "peter bachman" <peterb@cequs.com> wrote:

See enclosed random.pdf for comment

(A picture was attached)

From: Mike Stephens <Mike.Stephens@microsoft.com>

Date: Friday, October 18, 2013 2:47 PM

[MS] 1	10.3	Page 60	te	Published cryptanalysis shows that Dual EC DRBG outputs are biased. This should disqualify the algorithm from	Remove DUAL_EC_DRBG from SP 800-90.

				inclusion in the standard.[i] [ii]	
[MS] 2	10.3	Page 60	ge	Recent publications regarding Dual EC DRBG have significantly compromised public trust in the algorithm to the extent that correcting any mathematical properties of the algorithm would not regenerate confidence in it. Continued inclusion of Dual EC DRBG will harm the credibility of the SP-800 series of standards.	Remove DUAL_EC_DRBG from SP 800-90.
[MS] 3	11	Page 72	ge	The SP-800 series standardizes algorithms, not implementation aspects. Self-test requirements should come from FIPS 140-2. The current self-test requirements in SP 800-90 do not align with or cooperate with FIPS 140-2 requirements. Should the self-test requirements remain in SP 800-90; then, harmonize these requirements with those in FIPS 140.	Remove self-test requirements from SP800-90.

On 11/3/13 7:58 AM, "Thomas Hales" <hales@pitt.edu> wrote:

Dear NIST committee 800-90A,

I am submitting two documents in response to your request for public comments on Special Publication 800-90A.

The first document "NSA back door to NIST" will be published in the Notices of the American Mathematical Society. The first part of that article discusses the familiar Ferguson-Shumow back door to the elliptic curve pseudo-random generator. The second part of that article shows that the

design of the back door is based on Diffie-Hellman key exchange. Because of specific mathematical

structures in the back door, I conclude that the design is not accidental.

The second document "Formalizing NIST cryptographic standards" argues that the standard 800-90A exhibits pervasive sloppiness in its use of mathematics and advocates formal verification methods as a remedy. NIST has suffered damage to its reputation as a result of 800-90A. That reputation cannot be recaptured by repeating the same old process, but it can be regained by elevating the standard through formal methods.

I offer my perspective not as a cryptographer, but as a mathematician who has devoted much of his career to making it possible to do mathematics more reliably by computer. I direct one of the the largest formal verification projects ever undertaken (the Flyspeck project for the formal verification of the Kepler conjecture), which is now almost complete. I speak from experience, when I conclude that it is technologically feasible for NIST to adopt formal methods.

My honors include the Moore Prize on reliable computing (2004), the Fulkerson Prize of the Mathematical Programming Society and the American Mathematical Society (2009), and the Robbins Prize of the AMS (2007). My speaking honors include an invited plenary address at the International Congress on Mathematical Software (Kobe Japan, 2010) and an invited talk at the International Congress of Mathematicians (Beijing China, 2002).

(Documents not yet included in this file)

On 11/5/13 12:52 PM, "Stephan Mueller" <stephan.mueller@atsec.com> wrote:

1 Suggestions For Changes To SP800-90A

With the reopening of the SP800-90A standard for comments, I would like to take this opportunity to inform the authors about issues that were identified during the implementation of all three DRBGs defined in this standard (skipping the Dual EC DRBG for obvious reasons).

The common theme of the issues identified has to do with the complexity of the state transition function supported by the derivation function defined by SP800-90A.

The discussion starts with some mathematical considerations on which the comments are based on to ensure that the mathematical model for maintaining entropy is preserved. Next, the general theme of the comments is outlined. Lastly, the general theme is then broken down to concrete comments regarding the three different types of DRBG.

1.1 Mathematical Considerations

Before starting the details of the issues, please consider the following mathematical backgrounds applied for the discussion.

The specification of the deterministic random numbers have an associated cryptographic strength which correlates with the strength of the underlying cipher. This strength defines the upper boundary of the entropy that can be maintained by the DRBG. If the internal state of the DRBG would contain more entropy than this strength, entropy will be lost in either the state transition function or the output function. Thus, to maintain entropy that equals to the associated strength of the DRBG, it is sufficient to have an internal state that equals that strength.

2 General Concerns

The general concerns brought forward is the sheer complexity of the DRBG. To illustrate the complexity, please consider the following numbers (the abbreviation LOC means Lines of Code). The following groups of numbers are to be read as: first number are the LOCs for the common DRBG core, the second number are the LOCs for the Hash-DRBG specific code, the third number are the LOCs for the HMAC-DRBG specific code, and the last number are the LOCs for the CTR-DRBG specific code. To obtain the complete number for one instance of DRBG, please add the specific code number with the common core code.

Implementation	LOCs Core	LOCs Hash DRBG	LOCs HMAC DRBG	LOCs CTR DRBG
OpenSSL	550	320	220	380

Mozilla	N/A	750	N/A	N/A
My own implementation	470	240	90	320

Please compare that to an implementation of a typical micro kernel which contains about 5000 lines of code. One can see that for the task of stirring numbers from entropy sources, about one fifth to one sixth of the size of a micro kernel is needed, which is way too much, especially when using the DRBG in sensitive environments, such as operating system kernels or high assurance environments. Every line of code that is not needed is one line too much.

The goal of the suggested changes is to reduce the complexity.

2.1 General State Maintenance Concerns

The different DRBGs defined in SP800-90A have a cryptographic strength associated with them. The cryptographic strengths are always equal to the cryptographic strength of the ciphers. That means that the state, which holds the entropy, does not need to be larger than the strength of the ciphers and thus the overall security rating of the DRBG. If the state is larger, and potentially contains more entropy than the cipher used for the output function, the additional entropy will be lost, because SP800-90A defines the DRBGs with an associated cryptographic strength. If that cryptographic strength is smaller than the entropy the state can hold, that additional entropy will be lost.

For example, the Hash DRBG specification mandates 440 bits of state for the SHA-256 Hash DRBG. That DRBG is defined to have a strength of 256 bits, equal to the strength of SHA-256 which is the output function for the DRBG. Therefore, the state of 440 can be considered too big for the DRBG, because any entropy above 256 bits that the state may have will be compressed to at most 256 bits. Therefore, the maintenance of a state above the cryptographic strength rating of the output function and thus the DRBG is considered to be not needed to retain the strength of the DRBG.

In addition to the entropy consideration, another aspect regarding the state maintenance should be considered. SP800-90A defines different state transition functions for the different DRBGs. These state transition functions include XORs or the addition of some data with the state of the DRBG. The implementation of such state transition functions adds complexity. However, it must be asked why not use the respective underlying cipher for the state transition function? The mathematical properties of the cipher can be considered to be at least as good as the state transition

functions defined in SP800-90A in terms of maintaining entropy. When using the cipher for the state transition, only one code invocation to the cipher is needed instead of multiple lines of code implementing some hand-crafted state transition logic.

When applying the consideration about the size of the state and its entropy, we can conclude that when having a state that is equal to the cryptographic strength of the cipher and applying the cipher for the state transition function, complexity can be reduced significantly.

The following chapters explain in detail the suggested changes for the different DRBG types.

3 HMAC DRBG

The HMAC DRBG can be used as a good example for the other two classes of DRBG (Hash, CTR), as the general state maintenance concerns raised above are covered:

- The state of the HMAC DRBG is equal to the cryptographic strength of the used hash.
- Furthermore, the state is equal in size to the blocksize of the cipher.
- Finally, the state transition of the HMAC DRBG is implemented with the HMAC function itself. The state transition can be defined with the following two formulas: $Key_{t+1}=HMAC(Key_t, V_t \| 0 \| Seed_t)$ $V_{t+1}=HMAC(Key_{t+1}, V_t)$ The limited complexity can be seen with the lines of code needed to implement the HMAC core: it is the smallest implementation of the three types. However, one small complexity is added that is considered to be not warranted: SP800-90A section 10.1.2.2 requires the calculation of the Key_{t+1} and V_{t+1} using two rounds of the above mentioned formula as follows, if the caller provides seed material (i.e., additional input): $Key_{interim}=HMAC(Key_t, V_t \| 0 \| Seed_t)$ $V_{interim}=HMAC(Key_{interim}, V_t)$ $Key_{t+1}=HMAC(Key_{interim}, V_{interim} \| 1 \| Seed_t)$ $V_{t+1}=HMAC(Key_{t+1}, V_{interim})$ It is unclear why the last two operations to obtain a new value for K and V again are needed. The entropy that is present in the seed is transferred into K and V with the first two operations already. Moreover, using the cipher for the state transition, any concerns about secrecy of the state are not warranted. Therefore, I would recommend to drop steps 4 and 5 in the HMAC DRBG update process defined in section 10.1.2.2 – the last two calculation steps mentioned above shall simply be dropped in case of additional input present. Backtracking resistance is still maintained as the values for K and V are still recalculated. However, if that second iteration is required for ensuring backtracking or prediction resistance, please disregard this request.

4 Hash DRBG

The Hash DRBG is significantly more complex than the HMAC DRBG as both concerns regarding the state maintenance are applicable: the state is bigger than the cipher and the state transition is performed with an operation different than the cipher.

Figure 1: Hash DRBG State Transition (Not included in this file)

Figure 1 outlines the state transition for the example of a SHA-256 Hash DRBG. That figure skips details about the input to the hash (V_t plus additional input) as well as the subsequent concatenation of V_{t+1} with C. These skipped steps may be preserved and are thus not further discussed.

The state of 440 bits is hashed using the cipher, generating a resulting value of 256 bits. This resulting value is now added to the state at always the same location.

The dark gray shaded boxes are the bits that will always be touched by the addition operation. However, the state bits above 256 bits are being touched by the addition operation infrequently. The more the bits are above from the 256 threshold, the less often the bit is touched – this is symbolized by the lighter shadings of gray which fade out to white. The reason for this is that these bits above the 256 bit threshold are only touched if the addition operation produces an overflow. An analogy to help put this in perspective: the state is a 7 digit integer (i.e., up to 9,999,999). Now the value to be added into the state only has three digits (i.e., at most 999). How often will the digit for the thousand, the ten thousand, the hundred thousand or the million be changed? Much less than the lower three digits which are changed every time. The same applies to the bits above the 256 threshold.

Moreover, considering the cryptographic strength of the SHA-256 hash to have 256 bits, the state size of 440 bits is not warranted.

Therefore, I propose the following:

- The state of the Hash DRBG is equivalent to the block size of the cipher. For up to SHA- 256 that implies that the state is also equal to the cryptographic strength of the cipher defined by SP800-90A. For SHA-384 and SHA-512, this is not true anymore, but the reason for the larger state size will be clear with the next bullet point. In any case, the state size is sufficient to hold as much entropy to fulfill the mathematical background outlined above.

- The state transition function of the Hash DRBG shall be the cipher itself, when considering that the different addition operations (section 10.1.1.4, step 2.2 and 5, step 4.3 in the hashgen process) should all be replaced and consolidated into an invocation of the cipher where the output of the cipher is the new state value.

- Due to the reduction in size of the state, the constant C must be equally reduced in size. This also implies that the DF function will be much smaller as it only needs to Last update: 2013-11-05 Classification: atsec confidential Status: Released Version: 1.0 ©2013 atsec information security corporation perform one cipher operation instead of invoking the cipher in a loop and truncate the last block.

The basic idea is to use the concept defined for the HMAC DRBG for the Hash DRBG as well.

5 CTR DRBG

The CTR DRBG is the most complex of the three DRBGs. One of the reasons is the problem of translating the size of buffers using the bijective function of AES. The other two DRBGs use the surjective function of hashes which automatically implement the compression of input data into a fixed output length if the input data length (e.g., the seed data) is larger than the block size of the cipher. When using that block size of the cipher for the state maintenance (as suggested), no further translation of buffer sizes is needed. However, for a bijective function, any translation of buffer sizes (shrink a larger input buffer to a smaller output buffer or vice versa) requires additional operations beyond the invocation of the cipher.

Therefore, the basic requirement should always be to either prevent translation of buffer sizes, or only use one type of translation (i.e., either shrinking or enlarging), but not both at the same time.

Figure 2: CTR DRBG: DF and BCC Buffer Size Translation I (Not included in this file)

However, the derivation function together with the BCC function implements just that: first the input is shrunken and then subsequently enlarged. Figure 2 outlines the operation with some sample buffer sizes. This figure illustrates the operation of the DF and BCC function on the seed and additional input/personalization string data. In this example, the seed has a length of 72 bytes. The first step is to collapse the 72 bytes into a string of 16 bytes, i.e., the block size of AES. Subsequently, the data is now enlarged using multiple AES rounds to the required seed size.

The first question that can be raised is the appropriateness of this operation for the mathematical model to maintain entropy. The collapse of data to one AES block and the subsequent enlargement of that block implies that entropy of the seed data is reduced to 16 bytes. The subsequent enlargement will not increase the entropy any more.

In addition, the two translation operations of the buffer size by first shrinking and then enlarging the buffer requires the implementation of two

loops: the first loop applies the AES cipher blockwise on the seed data and XORs the output of the different AES rounds (the BCC operation) and the subsequent enlargement to the right seed size by invoking the BCC operation several times. This complicated processing of input data could be made less complex by considering the following suggestions.

5.1 Use of Compression Translation Function

The first consideration is the only use of a compression translation operation by processing the seed data as follows: The internal state is defined as a bit string with the key size of K of the cipher. When iterating blockwise over the input data and apply the AES encryption operation, XOR the output of the AES of the first input data block with the first 16 bytes of K. If K is 32 bytes (in case of AES 256), XOR the AES output of the second input data block with the second 16 bytes of K, otherwise again with the first 16 bytes – marked as round robin schema below:

```
for (i=1, i <= num_of_input_blocks, i++)

    }

    *K = *K XOR AES(input_block_i)

    /* implement a round-robin schema to modify each AES-blocksize portion of K equally as much as possible – if there are numbers
    of input blocks that are not multiples of (length(K)/block_length(AES)), the code treats this as unproblematic */

    K = K + (16*i mod (length of K in bytes))
```

The state value V to be encrypted can be derived the same way as for the HMAC DRBG by simply encrypting 16 bytes of the resulting value K.

If there is the concern that the first 16 bytes of K are modified too often compared to the second 16 bytes, a reminder can be stored which of the two 16 byte blocks of K was modified last. For a complete new round of this logic, that reminder is considered and the respective other 16 byte block is used as start value.

On the other hand, if the derivation of V from K is not considered to be appropriate. The above mentioned loop can be implemented to create the interim value I that holds the concatenation of K and V. This loop would look like:

```
I = K || V

length(I) = length(K) + block_length(AES)

for (i=1, i <= num_of_input_blocks, i++)

{

      *I = *I XOR AES(input_block_i)

                  /* implement a round-robin schema to modify each AES-blocksize portion of I equally as much as possible – if
                  there are numbers of input blocks that are not multiples of (length(I)/block_length(AES)), the code treats this as
                  unproblematic */

        I = I + (16*i mod (length of I in bytes))

}

 K = first 16 or 32 bytes of I

 State V = remaining 16 bytes
```

In the case of using AES192, the above mentioned loop is applied with K assumed to be 32 bytes in length (equal to 256). When using the value K, the implementation simply truncates the superfluous 8 bytes from the 32 byte value K to obtain the 24 byte key needed for AES192.

The above loops can be used for a shrinking of the buffer size or for a state function where the buffer size of the input equals that to the output. But it does not enlarge a buffer.

The entire step 9 in section 10.4.2 including the invocation of the BCC function can be replaced with this simple logic.

The discussed loops are applicable for a state transition with or without additional input:

- Without additional input: the input to the loop is the value K (or I in case the second approach is used) which is equal in size than the required output size K (or I). Thus, the bijective operation of AES does not need to perform a translation of buffer sizes.

- With additional input: the loop is appropriate to shrink the larger input buffer size to the needed output size.

5.2 State Transition Function

When using the result of the loop above for the value K and the state value, the seed data is already mixed into the state of the DRBG. It then has the same basic logic as used for the HMAC DRBG.

Therefore, the subsequent XORing of the data from the DF operation as specified in steps 3 through 7 in section 10.2.1.2 would not be needed anymore. The update of the state consisting of K and V is implemented with the above specified state transition.

5.3 Additional Considerations

In addition to re-define the DF and BCC function, the following smaller items add complexity that may not be needed as they do not add entropy and the avalanche effect of AES ensures an appropriate update of the state without adding some more constants:

- The definition of L and N to be used in the state transition (step 2 through 4 in section 10.4.2 – only the padding of the input size is needed).
- The definition and maintenance of the value "i" (step 7 section 10.4.2) is not needed any more.

From: <Nicholls>, Tom <Tom.Nicholls@thalesesec.com>

Date: Wednesday, November 6, 2013 10:46 AM

Legend (type of comment); E = Editorial; G = General; T = Technical

ID	SECTION, SUBJECT & PARA.	TYPE	COMMENT	RESOLUTION
1	10.3.1	G	Since the potential for a 'backdoor' in the recommended parameters for EC_Dual_DRBG was pointed out the year after its publication (Shumow and Ferguson, 2007) it's been regarded with extreme suspicion, which has recently been more or less vindicated. At the very least the 'recommended' (unsourced) choices of P and Q should be removed. Even apart from that issue, it has been shown to have significant biases in its output (e.g. Schoenmakers and Sidorenko, 2006), as well as being extremely slow, so should probably be removed entirely from the standard. If it is desired to include a DRBG based on the hardness of a number- theoretical problem then this should be the focus of a future consultation.	Please remove the EC_Dual_DRBG from the standard.
2	10.2.1 & E.3	T	It is implied that the security strength of a CTR_DRBG scheme matches that of the key -- so that a 256-bit AES key would give 256-bit security for the DRBG, for example. However, the security strength is really determined by the block size of the cipher (128 bits in this case). Campagna (2006) constructs a distinguisher between the DRBG output and a random source, provided one knows at what points reseeding occurs. It works by requesting some number M of L-bit blocks (where L is the cipher block size) that are known not to include a reseed, and checking if they are all distinct. This is repeated q times. If all q sequences consisted of M distinct blocks, we guess that the sequence is the DRBG output; otherwise we guess is it random. This distinguisher has an advantage of about q * M^2 / 2^L. The key size doesn't come into it (provided it's at least L). (For maximum advantage, choose M to be the reseeding interval in bits divided by L.)	A conseqence is that the 3-key TDEA, AES- 192 and AES-256 variants do not provide the claimed strength. The security claims made for these should therefore be downgraded.

3	8.6.5	T	Bullet point 3 implies that it is possible to 'stretch' an entropy source indefinitely. Suppose we have a DRBG D_1, seeded from real entropy, that requires reseeding with K bits of 'entropy' once every I bits of output. We use the output of this solely to reseed another identical DRBG D_2. Then D_2 will produce (I/K)^2 bits of output per bit of entropy input. And so on: by chaining r of these together we can get a DRBG D_r that produces (I/K)^r bits of output per bit of entropy -- i.e., as much as we want. Furthermore, the work required to generate a bit from D_r is only slightly more (by a factor of at most 1/(1-K/I)) than the work required for a single bit from D_1. SP 800-90C section 7 discusses this configuration in more detail. The only proviso is that: "If the target DRBG is intended to support requests for prediction resistance, then an SEI that has access to a Live Entropy Source shall be used." However, even a DRBG that does not support requests for prediction resistance may have a maximum reseed interval. Indeed, all those specified in SP 800-90A do. This requirement becomes pointless if the new seed comes from another DRBG within the same security boundary. Indeed, in such a configuration each individual DRBG would remain prediction-resistant in the sense that if you know its internal state you can only predict up to I of its future values. But this is not true of the chain as a whole: knowing the internal state of all of the DRBGs would allow you to predict an unbounded number of future values.	Please explain what value the compulsory reseed interval in the approved DRBGs actually adds, since any guarantees of prediction resistance that it may provide can be circumvented by the chaining construction. Alternatively remove the compulsory reseed interval in these scenarios.
4	Pages 43, 66, 68, 81 (2), 83, 84, 89, 90 (3), 97, 113.	E	Some elements of the document (I guess they're equations) have been rendered as empty boxes on the draft! We can only presume that they are unchanged from the previously approved version of the document.	Please re-populate these unchanged sections into the standard.

From: Brian Smithson <bsmithson@ricohsv.com>

Date: Wednesday, November 6, 2013 2:41 PM

Please consider the following comment, sent on behalf of Ricoh.

Comment: For software DRBGs, the following tests should not be mandatory.

- Known-answer testing of the generate function "at reasonable intervals" (11.3.3 para 1),
- Known-answer testing of the instantiate function "prior to creating each operational instantiation" (11.3.2 para 1), and,
- Self-testing of the reseed function whenever the reseed function is invoked (11.3.4 para 3 list item 2)

Rationale: Software does not tend to wear out, and it is as likely that the test program will be broken as that the program being tested will be broken. Therefore, the value of such testing is not worth the burden of performing the tests, except after power-up.

On 11/6/13 4:41 PM, "Sandy Maitland" <SMAITLAND@spyrus.com> wrote:

General Comments on SP 800-90A:

1) The removal of Dual_EC_DRBG from SP 800-90A and related standards is supported. Given the development in the press, there is no way the reputation of this algorithm can be rehabilitated. We would go further to say that the category of "number-theoretic" DBRGs, in general, has become tainted in the industry and public opinion. It may well be wise not to attempt to replace it with anything in that general category for the time being. The SP 800-90A hash-based and block-cipher based algorithms seem well suited to fill the void.

2) Given that this "problem" was first noted in the 2006 timeframe, should there not be a more proactive review of DBRG algorithms that responds to public domain flaw reporting? It takes very little to cast doubt on security standards and technology, and the public goodwill cannot afford to be taken for granted. For those in the industry who wholeheartedly and in good faith support NIST standards in their products and through FIPS 140-2 validations, the risk of future repeat occurrences of this can lead to serious loss of market share and revenue.

3) Would it be possible at this time to extend the base of the Hash_DRBG and possibly HMAC_DRBG to include SHA-3?

From: Rene Struik <rstruik.ext@gmail.com>

Date: Thursday, November 7, 2013 8:11 AM

I have the following comments on the NIST SP 800-90A draft:

1. Some of the formulae seem missing, e.g., p. 66, Step 1 of EC-DRBG Generate process.

2. I have two recommendations re the EC-DRBG generator:

a) Include the output of the verifiably random pick for G and Q in the specification (e.g., in Appendix A.2.1).

b) Change the EC-DRBG random number generator more fundamentally, so as to

(1) remove reliance on the public key Q;

(2) lower distinguishability of the output bit string;

(3) tighten security reductions;

(4) provide potential resilience against quantum cryptographic attacks (should these become a long-term threat).

c) For RNGs based on number-theoretic problems (such as EC-DRBG), it would be beneficial to produce outputs in the underlying field, rather than bit strings, while specifying post-processing that would map elements of the underlying field to binary strings.

For more details re #2 above, please see the attached summary document of how this could potentially be realized in detail.

I would be happy to discuss the technical proposal re "tweaks" to EC-DRBG in more detail with you.

(A paper titled “Visiting Discrete-Logarithm Based Random Number Generators (A Summary)” was attached to the email, and is provided on the following pages below.)

Revisiting Discrete-Logarithm Based Random Number Generators

—

(Summary Findings)

René Struik

rstruik.ext@gmail.com

Abstract. We revisit constructions for deterministic random bit generators based on discrete logarithm problems, triggered by the reopening of the review of the NIST SP 800-90A specification [15, 18]. Constructions in the literature mostly rely on the intractability of the Diffie-Hellman problem and crucially depend on the difficulty of computing the discrete logarithm of some public key Q relative to a base point G in the Diffie-Hellman group. We suggest various constructions, so as to (1) remove reliance on the public key Q; (2) lower distinguishability of the output bit string; (3) tighten security reductions; (4) provide potential resilience against quantum cryptographic attacks (should these become a long-term threat).

CAUTIONARY NOTE: This summary report is based on a careful analysis of the literature on discrete-log based random number generators, intractability assumptions, and, e.g., results on finite fields. However, so far, results have only been partially vetted and formal proofs are still lacking (since requiring more time than afforded by the review time window).

1 The Elliptic Curve Random Number Generator

Let $E(\mathbb{F}_q)$ be an elliptic curve with cyclic subgroup $\mathbb{G}$ of prime order n, generated by some base point G. One has $|E(\mathbb{F}_q)| = n \cdot h$, where the co-factor h assumes a small value (with NIST curves, one has $h = 1$ if it concerns a prime curve, and $h = 2$ or $h = 4$ if it concerns a binary curve). Each point P on this curve (except the so-called point of infinity) can be represented as a pair (x, y) in $\mathbb{F}_q \times \mathbb{F}_q$, in which case one denotes $x := x(P)$.

The elliptic curve random number generator EC-DRBG specified in [6, 5, 15] is defined in terms of an elliptic curve $E(\mathbb{F}_q)$, base point G and *random* point Q in $\mathbb{G}$ and produces, upon a random input $k \in \mathbb{Z}_q$, a deterministic sequence of outputs $\text{out}_1, \ldots, \text{out}_\ell$, each in $\mathbb{Z}_b$, as depicted in Algorithm 1 below[1].

[1] We omit details that are irrelevant for our exposition (such as re-seeding).

Algorithm 1 EC-DRBG Generator

Input: $k \in \mathbb{Z}_q, b < q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_b$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S) \leftarrow (kG, kQ)$;
    Set $(k, \text{out}_i) \leftarrow (x(R) (\textbf{mod } q), x(S) (\textbf{mod } b))$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

With EC-DRBG, the value of parameter b is a power of two, so that each output is obtained from the x-coordinate of an elliptic curve point via truncation. With EC-DRBG, the bit-size of b is at least $13 + \lceil \log_2 h \rceil$ less than the bit-size of the order of the finite field $\mathbb{F}_q$ and byte-length oriented. While smaller values of b are allowed, it is recommended that b is picked as large as possible, subject to the above constraints [15, §10.3.1.4].

With EC-DRBG, default values of G and Q are specified for the NIST prime curves P-256, P-384, and P-521 [15, Appendix A.1], although picking alternative points G and Q is allowed, provided these are generated *verifiably at random* [15, Appendix A.2].

In what follows, we mainly consider curves with co-factor $h = 1$.

2 Security of the Elliptic Curve Random Number Generator

The security of the elliptic curve random number generator was analyzed in [1, 13, 4], with main results as follows:

1. The procedure by which the default points G and Q specified in [15, Appendix A.1] have been picked is unknown (and cannot be checked to have been generated verifiably at random). As a result, one does not know whether $\log_G(Q)$ is known to those who specified these points. Here, one should observe that if $d := \log_G(Q)$, the internal state R can be efficiently determined from S, since $R := d^{-1}S$. Since the only two points with x-coordinate $x(S)$ are S and $-S$ and truncation only removes roughly 16 bits from $x(S)$, it follows that in that case one can determine R from an observed output in roughly 2^{16} guesses for S [13].
2. The output of the EC-DRBG is distinguishable from a random bit string. Here, one should observe that the set of x-coordinates of valid elliptic curve points form a subset of $\mathbb{F}_q$ of cardinality roughly $q/2$. Since one can easily check whether a value $x \in \mathbb{F}_q$ is in this set, one can efficiently distinguish an output from the EC-DRBG (without truncation) from a random element of $\mathbb{F}_q$: the latter elements pass this check only half of the time, whereas the

former always do. Differentiability also remains with truncation, if one does not remove sufficiently many bits from $x(S)$ [1, 4]. For binary curves, the x-coordinate of an elliptic curve point in $\mathbb{G}$ has a fixed so-called trace value, thereby potentially resulting in an affine relationship between a known set of bit positions (dependent on the basis for $\mathbb{F}_{2^m}$). Here, truncation might need to be quite extensive to mitigate this effect. For the NIST curves B-409 and K-409, the lowest-order output bit always has a fixed value, so no truncation is capable of removing this bias [1]).

3. The hardness of the so-called x-logarithm problem, on which the security of EC-DRBG relies, is difficult to quantify and the security reduction of a related security problem (AXLP) to the well-known decisional Diffie-Hellman problem (DDH) is rather loose [1].

The extent to which these drawbacks present a problem depends on context: with Result 1 above, any exposed output (e.g., if used as nonce in a challenge-response protocol), would result in complete exposure of the internal state of the EC-DRBG and, thereby, in a total break; with Result 2, if the output is used as input to a key derivation function (and the curve has co-factor $h = 1$), the impact may be less severe in practice. Whether or not Result 3 presents a security problem in practice, is hard to evaluate (since not all schemes believed to be secure have tight security reductions).

3 Improvements to the Elliptic Curve Random Number Generator

3.1 Addressing Part of the Problem

The attack in the event of a potential backdoor (Result 1 above) can be prevented by truncating the output to roughly $(\log_2 q)/2$ bits, since then inverting the mapping $x(S) \rightarrow x(S)(\textbf{mod } b)$ requires $O(\sqrt{n})$ operations, which is the same computational complexity as generic DLP solvers (Pollard's rho method, etc.). This would also most likely make each output of the EC-DRBG (for curves with co-factor $h = 1$) indistinguishable from a random element of $\mathbb{Z}_b$. Since this would reduce the efficiency of the EC-DRBG by a factor two, this would make the EC-DRBG – already a relatively slow random number generator in the first place – very unattractive to use in practice. We, therefore, do not consider this as a viable option.

Of course, any potential backdoor can be prevented, by simply picking Q verifiably at random, as also suggested as alternative in [15, Appendix A.2]. Here, one does not need to generate both G and Q verifiably at random, as [15] seems to suggest: fixing G and picking Q subsequently at random suffices. Here, it is preferable to just generate Q in this way and publish the result in the specification, so that this is easy to verify once and for all. This would avoid each implementer having to implement verification tests (and having to implement SHA-512).

3.2 Addressing the Entire Problem

Both changes above would remove suspicion that the public key Q was concocted in a particular way or would simply thwart exploiting a concocted key. However, this does not address the other drawbacks of the EC-DRBG mentioned above. Here, we will show that one could potentially mitigate all drawbacks, via a simple change in the EC-DRBG itself. We consider several alternatives, each with slightly different properties.

Construction A: DDH Generator with Tight Reduction
Our first construction depends on three random public keys $Q_1, \ldots, Q_3$ (rather than one, as in the original EC-DRBG). The random number generator is as depicted in Algorithm A below. Notice that $b := q$ (i.e., there is no need to truncate outputs).

Algorithm A DDH Generator with tight reduction
Input: $k \in \mathbb{Z}_q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_q$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S_1, S_2, S_3) \leftarrow (kG, kQ_1, kQ_2, kQ_3)$;
    Set $(k, \text{out}_i) \leftarrow (x(R) + x(S_1), x(S_2) + x(S_3))(\textbf{mod } q)$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

Assuming the intractability of the DDH problem, one may assume that all elliptic curve points in each round are random points and can show that both the updated value of k and the output element are indistinguishable from random elements of $\mathbb{F}_q$. This results in a tight reduction.

Construction B: Optimized DDH Generator with Tight Reduction
Our second construction depends on two random public keys $Q_1,\ Q_2$ (rather than one, as in the original EC-DRBG). The random number generator is as depicted in Algorithm B below. Notice that $b := q$ (i.e., there is no need to truncate outputs).

Algorithm B Optimized DDH Generator with tight reduction
Input: $k \in \mathbb{Z}_q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_q$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S_1, S_2) \leftarrow (kG, kQ_1, kQ_2)$;
    Set $(k, \text{out}_i) \leftarrow (x(R) + x(S_1), x(S_1) + x(S_2))(\textbf{mod } q)$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

Assuming the intractability of the DDH problem, one may assume that all elliptic curve points in each round are random points and can show that both the updated value of k and the output element (and any nontrivial linear combina-

tion of both) are indistinguishable from random elements of $\mathbb{F}_q$. This results in a tight reduction.

Construction C: DDH Generator with Looser Reduction

Our third construction depends on one random public key Q (as did the original EC-DRBG). The random number generator is as depicted in Algorithm C below. Notice that $b := q$ (i.e., there is usually no need to truncate outputs [caveat discussed below]).

Algorithm C Efficient DDH generator (with looser reduction)
Input: $k \in \mathbb{Z}_q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_q$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S) \leftarrow (kG, kQ)$;
    Set $(k, \text{out}_i) \leftarrow (2x(R) + x(S)), x(R) + x(S))(\textbf{mod } q)$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

Assuming the intractability of the DDH problem, one may assume that all elliptic curve points in each round are random points and can show that both the updated value of k and the output element are indistinguishable from random elements of $\mathbb{F}_q$. However, in this case, if one has access to an output, the security now relies on the hardness of the x-logarithm problem (as did the original EC-DRBG). This results in a looser reduction (assuming the same reduction as the AXLP problem). If one has access to internal state, but not to the output, one can show that this last output can be distinguished from random elements from $\mathbb{Z}_q$ with probability roughly half. Whether this is problematic remains to be seen, since this happens only once internal state becomes available (presumably a rare event). If this would be problematic, one could resort to truncation of the output.

Notice the strong similarity with the original EC-DRBG, where small changes result in significantly improved security assurances.

Construction D: Preferred DDH Generator with Tight Reduction

Our fourth construction is similar to Construction B, but does *not* depend on any random public keys (rather than one, as in the original EC-DRBG). The random number generator is as depicted in Algorithm D below. Notice that $b := q$ (i.e., there is no need to truncate outputs).

Algorithm D Preferred DDH Generator with tight reduction

Input: $k \in \mathbb{Z}_q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_q$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S_1, S_2) \leftarrow (kG, k^2G, k^3G)$;
    Set $(k, \text{out}_i) \leftarrow (x(R) + x(S_1), x(S_1) + x(S_2))(\textbf{mod } q)$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

Assuming the intractability of the DDH problem and the decisional square DH problem (DSqDH [8]) and decisional cube DH problem[2], one may assume that all elliptic curve points in each round are random points and can show that both the updated value of k and the output element (and any nontrivial linear combination of both) are indistinguishable from random elements of $\mathbb{F}_q$. This results in a tight reduction.

Notice that this construction does not require authentic storage of any public keys and, thereby, also does not require protection against key swaps. Moreover, since all scalar multiples are relative to the same base point, one can take advantage of more efficient mechanisms for batch computing of multiple points (or pre-compute tables).

Construction E: Preferred DDH Generator with Looser Reduction
Our fifth construction is similar to Construction C, but does *not* depend on any random public keys (rather than one, as in the original EC-DRBG). The random number generator is as depicted in Algorithm E below. Notice that $b := q$ (i.e., there is no need to truncate outputs).

Algorithm E Preferred DDH Generator (with looser reduction)

Input: $k \in \mathbb{Z}_q, \ell \geq 0$
Output: ℓ pseudorandom numbers in $\mathbb{Z}_q$
  for $i := 1$ **to** ℓ **do**
    Set $(R, S) \leftarrow (kG, k^2G)$;
    Set $(k, \text{out}_i) \leftarrow (2x(R) + x(S)), x(R) + x(S))(\textbf{mod } q)$;
  end for
  Return $(\text{out}_1, \ldots, \text{out}_\ell)$

Assuming the intractability of the DDH problem and the decisional square DH problem (DSqDH [8]), one may assume that all elliptic curve points in each round are random points and can show that both the updated value of k and the output element. However, as with Construction C, if one has access to an output, the security now relies on the hardness of the x-logarithm problem (as did the original EC-DRBG). This results in a looser reduction (assuming the same reduction as the AXLP problem). If one has access to internal state, but

[2] One can easily show that the computational versions of these problems are equivalent to the ordinary CDH.

not to the output, one can show that this last output can be distinguished from random elements from $\mathbb{Z}_q$ with probability roughly half. Whether this is problematic remains to be seen, since this happens only once internal state becomes available (presumably a rare event). If this would be problematic, one could resort to truncation of the output.

Notice that this construction does not require authentic storage of any public keys and, thereby, also does not require protection against key swaps. Moreover, since all scalar multiples are relative to the same base point, one can take advantage of more efficient mechanisms for batch computing of multiple points (or pre-compute tables).

4 Concluding Remarks

All the constructions of the previous section produced as output pseudo-random elements of $\mathbb{Z}_q$, rather than pseudo-random bit strings. However, it is trivial to map elements of $\mathbb{Z}_q$ to $\mathbb{Z}_b$ via the mapping $x \rightarrow x(\textbf{mod } b)$. One can easily pick b, so that the statistical distance of truncating an element of $\mathbb{Z}_q$ can be bounded above by ε. For NIST prime curves, the prime is always close to a power of two, in which case the statistical distance introduced by truncation is quite small. For Brainpool curves, however, one may have to truncate the output considerably, should one wish to obtain a good indistinguishability bound. Please do note that for some applications it seems wasteful to *not* work with elements of $\mathbb{Z}_q$, e.g., with generation of ephemeral private keys for the same curve (or its twist) – in that case, working with $\mathbb{Z}_q$-entries results in a factor two efficiency improvement compared to working with binary strings.

All the constructions of the previous section produced as output $s := x(F) + x(G)(\textbf{mod } q)$, where $F,\ G$ are elliptic curve points. For each nonzero $s \in \mathbb{F}_q$, there are roughly $q/4$ solutions (F, G). Moreover, for each *candidate solution* (F, G), it is not immediately clear how one could check whether this solution is valid. It would be of interest to check whether this offers resilience in the event quantum cryptographic attacks would become a threat: a quantum set search on these pairs takes roughly $O(\sqrt{q})$ operations [23]. In this case, Algorithms D and E could be of special interest, since these do not depend on storage of long-term public keys Q_i and would require online quantum computing calculations to be effective. This would also provide resilience in case the underlying curve would have some heretoforth unknown hidden weakness that, at some moment in the future, might be exposed.

All constructions presented here can be extended to work for binary curves, prime curves with co-factor $h > 1$, Montgomery curves, Edwards curves, and twisted Edwards curves, with essentially the same results.

FINAL NOTE: Remember the caveat that results are preliminary at this stage. The author is happy to discuss the material further.

References

1. D.R.L. Brown, K. Gjøsteen, 'A Security Analysis of the NIST SP 800-90 Elliptic Curve Random Number Generator,' in *Proceedings of Advances of Cryptology – CRYPTO 2007*, A. Menezes, Ed., Lecture Notes in Computer Science, Vol. 4622, pp. 466-481, Berlin: Springer, 2007.
2. R.R. Farashahi, B. Schoenmakers, A. Sidorenko, 'Efficient Pseudorandom Generators Based on the DDH Assumption,' in *Proceedings of 10th International Conference on Practice and Theory in Public-Key Cryptography – PKC 2007*, T. Okamoto, X. Wang, Eds., Lecture Notes in Computer Science, Vol. 4450, pp. 426-441, Berlin: Springer, 2007.
3. P.L. Montgomery, 'Speeding the Pollard and Elliptic Curve Methods of Factorization,' *Mathematics of Computation*, Vol. 48, No. 177, pp. 243-264, 1987.
4. B. Schoenmakes, A. Sidorenko, 'Cryptanalysis of the Dual Elliptic Curve Pseudorandom Generator,' IACR ePrint 2006-190.
5. NIST SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, National Institute of Standards and Technology, Department of Commerce, 2007.
6. ANS X9.82: Part 3-2007, *Random Number Generation, Part 3: Deterministic Random Bit Generators*, American National Standard for Financial Services, Accredited Standards Committee X9, Annapolis, MD, 2007.
7. D. Brown, R. Gallant, 'The static Diffie-Hellman Problem,' International Association for Cryptologic Research, IACR ePrint 2004-306.
8. N. Koblitz, A. Menezes, 'Intractable Problems in Cryptography,' in *Finite Fields: Theory and Applications*, Contemporary Mathematics, 518, pp. 279-300, 2010.
9. P-A. Fouque, A. Joye, M. Tibouchi, 'Injective Encodings to Elliptic Curves,' in *18th Australasian Conference on Information Security and Privacy – ACISP 2013*, C. Boyd, L. Simpson, Eds., Lecture Notes in Computer Science, Vol. 7959, pp. 203-218, New York: Springer, 2013.
10. D.J. Bernstein, M. Hamburg, A. Krasnova, T. Lange, 'Elligator: Eliptic-Curve Points Indistinguishable from Uniform Random Strings,' in *Proceedings of 20th ACM Conference on Computer and Communications Security – ACM-CCS 2013*, 2013.
11. D.F. Aranha, P.S.L.M. Barreto, G.C.C.F. Pereira, J.E. Ricardini, 'A Note on High-Security General-Purpose Elliptic Curves,' International Association for Cryptologic Research, IACR ePrint 2013-647.
12. J.H. Cheon, 'Discrete Logarithm Problems with Auxiliary Inputs,' J. of Cryptology, 2010.
13. D. Shumow, N. Ferguson, 'On the Possibility of a Backdoor in the NIST SP 800-90 Dual-ECC PRNG,' Rump Session, Crypto 2007.
14. FIPS Pub 186-2, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2, US Department of Commerce/National Institute of Standards and Technology, Springfield, Virginia, January 27, 2000. (Includes Change Notice, October 5, 2001.)
15. NIST SP 800-90A, *Recommendation for Random Number Generation Using the Deterministic Random Bit Generators*, Revision 1, Draft, National Institute of

Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 9, 2013.
16. NIST SP 800-90B, *Recommendation for the Entropy Sources Used for Random Bit Generation*, Draft, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 6, 2012.
17. NIST SP 800-90C, *Recommendation for Random Bit Generation Constructions*, Draft, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 6, 2012.
18. NIST, *NIST Supplemental ITL Bulletin for September 2013*, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 9, 2013.
19. D.R. Hankerson, A.J. Menezes, S.A. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2003.
20. R. Lidl, H. Niederreiter, *Finite Fields*, 2nd Edition, Cambridge: Cambridge University Press, 1997.
21. Bundesamt für Sicherheit in der Informationstechnik, *Technical Gideline TR-03111 – Elliptic Curve Cryptography*, Version 1.11, April 17, 2009.
22. RFC 5639, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, March 2010.
23. L.K. Grover, 'A Fast Quantum Mechanical Algorithm for Database Search,' STOC 1996.

Revisiting Discrete-Log Based Random Number Generators (or: How to Fix EC DRBG?)

–

René Struik
(Struik Security Consultancy)

e-mail: rstruik.ext@gmail.com

Outline

1. Notation
2. NIST EC-DRBG
   - Description of Generator
   - Security Caveats
3. EC-DRBG “Fixes”
   - Main Objectives
   - Five Constructions
4. Conclusions

Notation

$E(\mathbf{F}_q)$: elliptic curve over field $\mathbf{F}_q$
G: cyclic subgroup of $E(\mathbf{F}_q)$, of prime-order n
G: base point of **G**
h: co-factor (usually, small)

One has $|E(\mathbf{F}_q)| = n \cdot h$

$x(P)$: x-coordinate of point P on the curve (not being point at infinity), when represented in affine coordinates

NIST EC-DRBG Generator

Algorithm 1: EC-DRBG Generator

Input: $k \in \mathbf{Z}_q$, $b \leq q$, $l \geq 0$

Output: l pseudorandom numbers in $\mathbf{Z}_b$

for i:=1 **to** l **do**

Set $(R, S) \leftarrow (kG, kQ)$;

Set $(k, out_i) \leftarrow (x(R) \ (\textbf{mod}\ q), x(S) \ (\textbf{mod}\ b))$;

end for

Return $(out_1, \ldots, out_l)$

NIST EC-DRBG:

- b: a power of two (i.e., output obtained via truncation of x-coordinate)
- b: at least $13 + \log_2 h$ bits less than bit-size of order of finite field $\mathbf{F}_q$ (byte-oriented)
  (recommendation was to pick b as large as possible, for efficiency reasons)
- $E(\mathbf{F}_q)$: NIST prime curves P-256, P-384 (and others)
- G, Q: default values specified for NIST prime curves P-256, P-384
  (alternative values allowed, provided generated *verifiably at random*)

Security of NIST EC-DRBG

1. Potential back-door EC-DRBG

Unknown whether default base point G and public key Q generated verifiably at random
Unknown if $\log_G(Q)$ known to those who specified G and Q

- If $d := \log_G(Q)$, one can determine internal state R from S, since $R := d^{-1}S$
- One can determine S from $x(S)$, since only two points with same x-coordinate
- One can determine $x(S)$ from truncated version, since only roughly 16 bits removed

So, if $\log_G(Q)$ known, then internal state leaked from observed output out_i

2. Output EC-DRBG distinguishable from random bit string

- Set of x-coordinates of valid point forms subset of $\mathbf{F}_q$ of cardinality roughly $q/2$ and easy to check whether $x \in \mathbf{F}_q$ is in this set. So, output of EC-DRBG (without truncation) is easily distinguished from random element of $\mathbf{F}_q$
- Distinguishability remains with truncation, if one does not remove sufficiently many bits from $x(S)$

3. Loose security reduction

Hardness of so-called x-Logarithm Problem, on which security of core EC-DRBG relies, is hard to quantify and security reduction of related security problem (AXLP) to Diffie-Hellmann problem (DDH) is rather loose

NIST EC-DRBG “Fixes”

Minor “tweaks” of EC-DRBG suffice to obtain the following properties:

1. Reduce/remove reliance on public key *Q*
2. Lower distinguishability of output bit string
3. Tighten security reductions
4. Provide potential resilience against quantum cryptographic attacks (should these become a long-term threat)

Claims:

− Techniques apply to short Weierstrass curves (e.g., NIST, Brainpool), Montgomery curves, Edwards and twisted Edwards curves, binary curves.
− Techniques do not add additional computational cost (mostly, far more efficient)
− Techniques can do without public key *Q*, thus eliminating key substitution attacks

NOTE: builds upon existing cryptanalysis EC-DRBG ([1])
− uses tight bounds on character sums and Kloosterman sums ([18])
− uses presumed difficulty of Diffie-Hellman problems ([7])

Example of 'Fix' (roughly "Construction C")

Original EC-DRBG Generator

Input: $k \in \mathbf{Z}_q$, $b \le q$, $l \ge 0$
Output: l pseudorandom numbers in $\mathbf{Z}_b$

for i:=1 **to** l **do**
  Set $(R, S) \leftarrow (kG, kQ)$;
  Set $(k, out_i) \leftarrow (x(R) \ (\mathbf{mod}\ q), x(S) \ (\mathbf{mod}\ b))$;
end for
Return $(out_1, \ldots, out_l)$

"Algorithm C": DDH Generator

Input: $k \in \mathbf{Z}_q$, $l \ge 0$
Output: l pseudorandom numbers in $\mathbf{Z}_b$

for i:=1 **to** l **do**
  Set $(R, S) \leftarrow (kG, kQ)$;
  Set $(k, out_i) \leftarrow (x(R) \ (\mathbf{mod}\ q), (x(R) + x(S)) \ (\mathbf{mod}\ b)$;
end for
Return $(out_1, \ldots, out_l)$

NIST EC-DRBG vs. New DDH Constructions

Construction	NIST	A	B	C	D	E	D(k)
*#Public keys **Q***	1	3	2	1	–	–	–
≈ # rnd. bits/curve size	1	1	1	1	1	1	k
Rate[1]	1/2	1/4	1/3	1/2	1/3	1/2	$k/(k+2)$
Backdoor possible?	*Yes*	*unlikely*	*unlikely*	*unlikely*	*No*	*No*	*No*
Indistinguishable output	*poor*						
*- if state **R** not known*		*tight*	*tight*	*tight*	*tight*	*tight*	*tight*
*- if state **R** known*		*tight*	*tight*	*poor*	*tight*	*poor*	*tight*
Reduction next state	*AXLP*						
- if output not known		*tight*	*tight*	*tight*	*tight*	*tight*	*tight*
- if output known		*tight*	*tight*	*AXLP*	*tight*	*AXLP*	*tight*
Quantum-crypto secure?	*No*	*perhaps*	*perhaps*	*perhaps*	*likely*	*likely*	*likely*

Notes:

- Five constructions submitted to NIST (as comment re-opened SP 800-90A spec)
- Full details in draft technical paper

[1]Rate: #random bits (as multiple of bit-size curve)/#scalar multiplications

Conclusions

Security weaknesses EC-DRBG relatively easy to fix

- Five constructions, with slightly differing properties
- Simplest fix: only change w.r.t. original EC-DRBG is *single modular addition*
- Some suggested fixes possibly resistant to quantum-cryptographic attacks

Constructions work for “short” Weierstrass curves (e.g., NIST, Brainpool), Edwards curves, twisted Edwards curves, Montgomery curves

Contrary to popular belief, NIST EC-DRBG can be made highly secure

Notes:

- Main constructions submitted to NIST
- Full details to appear in technical paper

Further Reading

1. D.R.L. Brown, K. Gjøsteen, "A Security Analysis of the NIST SP 800-90 Elliptic Curve Random Number Generator," in *Proceedings of Advances of Cryptology – CRYPTO 2007*, A. Menezes, Ed., Lecture Notes in Computer Science, Vol. 4622, pp. 466-481, Berlin: Springer, 2007.
2. R.R. Farashahi, B. Schoenmakers, A. Sidorenko, "Efficient Pseudorandom Generators Based on the DDH Assumption," in *Proceedings of 10th International Conference on Practice and Theory in Public-Key Cryptography – PKC 2007*, T. Okamoto, X. Wang, Eds., Lecture Notes in Computer Science, Vol. 4450, pp. 426-441, Berlin: Springer, 2007.
3. B. Schoenmakes, A. Sidorenko, "Cryptanalysis of the Dual Elliptic Curve Pseudorandom Generator," IACR ePrint 2006-190.
4. NIST SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, National Institute of Standards and Technology, Department of Commerce, 2007.
5. ANS X9.82: Part 3-2007, *Random Number Generation, Part 3: Deterministic Random Bit Generators*, American National Standard for Financial Services, Accredited Standards Committee X9, Annapolis, MD, 2007.
6. D. Brown, R. Gallant, "The static Diffie-Hellman Problem," International Association for Cryptologic Research, IACR ePrint 2004-306.
7. N. Koblitz, A. Menezes, "Intractable Problems in Cryptography," in *Finite Fields: Theory and Applications*, Contemporary Mathematics, 518, pp. 279-300, 2010.
8. P-A. Fouque, A. Joye, M. Tibouchi, "Injective Encodings to Elliptic Curves," in *18th Australasian Conference on Information Security and Privacy – ACISP-2013*, C. Boyd, L. Simpson, Eds., Lecture Notes in Computer Science, Vol. 7959, pp. 203-218, New York: Springer, 2013.

Further Reading (cont'd)

9. D.J. Bernstein, M. Hamburg, A. Krasnova, T. Lange, "Elligator: Eliptic-Curve Points Indistinguishable from Uniform Random Strings," in *Proceedings of 20th ACM Conference on Computer and Communications Security – ACM-CCS 2013*, 2013.
10. D.F. Aranha, P.S.L.M. Barreto, G.C.C.F. Pereira, J.E. Ricardini, "A Note on High-Security General-Purpose Elliptic Curves," International Association for Cryptologic Research, IACR ePrint 2013-647.
11. J.H. Cheon, "Discrete Logarithm Problems with Auxiliary Inputs," J. of Cryptology, 2010.
12. D. Shumow, N. Ferguson, "On the Possibility of a Backdoor in the NIST SP 800-90 Dual-ECC PRNG," Rump Session, Crypto 2007.
13. FIPS Pub 186-2, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2, US Department of Commerce/National Institute of Standards and Technology, Springfield, Virginia, January 27, 2000. (Includes Change Notice, October 5, 2001.) \
14. NIST SP 800-90A, *Recommendation for Random Number Generation Using the Deterministic Random Bit Generators*, Revision 1, Draft, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 9, 2013.
15. NIST SP 800-90B, *Recommendation for the Entropy Sources Used for Random Bit Generation*, Draft, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 6, 2012.
16. NIST SP 800-90C, *Recommendation for Random Bit Generation Constructions*, Draft, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, September 6, 2012.

Further Reading (cont'd)

17. D.R. Hankerson, A.J. Menezes, S.A. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2003.
18. R. Lidl, H. Niederreiter, *Finite Fields*, 2nd Edition, Cambridge: Cambridge University Press, 1997.
19. BSI , *Technical Guidance TR-03111 – Elliptic Curve Cryptography, Version 2.0*, June 28, Bundesamt fur Sicherheit in der Informationstechnik, Bonn, Germany, 2012.
20. RFC 5639, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, March 2010.
21. L.K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," STOC 1996.

On 11/8/13 4:54 PM, "David Johnston" <dj@deadhat.com> wrote:

1 Comment - Typo

Location: **SP800-90A, Page 1, heading 1**

Type: **Editorial**

Issue:

Extraneous ‘b’ on the clause heading

Proposed resolution:

Replace

1 Authorityb

With

1 Authority

2 Comment – SEI Definition

Location: **SP800-90A, Page 9, SEI Definition**

Type: **Normative**

Issue:

The specification restricts an SEI to include only an approved entropy source, within the context of the SP800-90 definition of an approved entropy source.

Common practice in common cryptographic RNGs, especially in operating systems, E.G the linux kernel random number service, take the approach that multiple entropy sources be mixed. The security of the resulting system relying on at least one of those sources being sufficiently entropic. If any sources are malicious, the system will still be secure provided at least one source is entropic. This is a powerful security construct, since it enables

- Systems to be built with redundant sources from different entropy source types.
- Users to guarantee the security of their random numbers by adding in a source they control that they know to be entropic, while the RNG continues to use the provided source at the same time.

By limiting to only approved SEIs, it is guaranteed that such robust approaches to a secure RNG are non compliant. This makes SP800-90 compliant RNG unacceptable to a large audience of educated users and system builders. It also leads to fragile systems with single points of failure.

The types of SEI are listed as three optional things. You need at least one, so the clause should be written to make it mandatory that the source be one of three types.

Proposed resolution:

Replace

> A component of a DRBG that outputs bitstrings that can be used as entropy input by a DRBG mechanism. An SEI may be an approved entropy source, an approved RBG employing an approved entropy source to obtain entropy input for its DRBG mechanism, or a nested chain of approved RBGs whose initial member employs an approved entropy source to obtain entropy input for its DRBG mechanism.

With

> A component of a DRBG that outputs bitstrings that can be used as entropy input by a DRBG mechanism. An SEI shall employ at least one approved entropy source. An SEI **shall** be one of (1) An approved entropy source with 0 or more additional entropy sources, (2) an approved RBG employing an approved entropy source with 0 or more additional entropy sources to obtain entropy input for its DRBG

mechanism, or (3) a nested chain of approved RBGs whose initial member employs an approved entropy source and 0 or more additional entropy sources to obtain entropy input for its DRBG mechanism.

3 Comment – Ambiguous optionality

Location: **SP800-90A, Page 13, clause 7**

Type: **Normative**

Issue:

The diagram shows inputs to functions including entropy, personalization string, nonce and Additional input. However the text calls out only the nonce input as being optional. By omission it leaves the optionality of the other inputs ambiguous. In a specification, where there is a list of items, some optional, some mandatory, it is necessary to identify the optional or mandatory nature of every item.

Also, “depending on the implementation” is redundant and adds no meaning.

Proposed resolution:

Replace

> Figure 1 provides a functional model of a DRBG (i.e., one type of RBG). A DRBG uses a DRBG mechanism and a source of entropy input, and may, depending on the implementation of the DRBG mechanism, include a nonce source. The components of this model are discussed in the following subsections.

With

> Figure 1 provides a functional model of a DRBG (i.e., one type of RBG). A DRBG **shall** implement an approved DRBG algorithm and at least one approved source of entropy input, and **may** include additional optional sources including a nonce source, personalization string, and additional input. The components of this model are discussed in the following subsections.

4 Comment – Inappropriate use of the term ‘bias’

Location: **SP800-90A, Page 13, Clause 7.1, second paragraph**

Type: **Normative**

Issue:

The text says “however, the DRBG mechanisms have been specified to allow for some bias in the entropy input by allowing the length of the entropy input to be longer than the required amount of entropy”

This is neither sufficient nor true. Bias is the least likely form of deviation from pure randomness in a deliberately designed physical entropy source. Circuits designed specifically to be an entropy source typically use feedback to ensure even bias, while introducing correlation and non stationarity.

The text should explain that non full entropy inputs are suitable inputs where there is sufficient total entropy.

Also, entropy is a bulk property of groups of things. Entropy is not bits or data, it is measured in bits, but groups of bits and data have entropy, they are not themselves entropy. The use of language should reflect this.

Proposed resolution:

Replace

> Ideally, the entropy input will have full entropy; however, the DRBG mechanisms have been specified to allow for some bias in the entropy input by allowing the length of the entropy input to be longer than the required amount of entropy (expressed in bits). The entropy input can be defined to be a variable length (within limits), as well as fixed length.

With

> Ideally, the entropy input will have full entropy; however, the DRBG mechanisms have been specified to allow for non-full entropy input by allowing the length of the entropic input data to be longer than the required entropy (expressed in bits) such that the total entropy of the data meets the requirements of the algorithm. The entropy input can be defined to be a variable length (within limits), as well as fixed length.

5 Comment – Ambiguous optionality

Location: **SP800-90A, Page 14, clause 7.2, second paragraph**

Type: **Normative**

Issue:

“may be required” is no sort of language for a specification. It is mandatory or is it optional? Optionally mandatory is not a useful definition.

Proposed resolution:

Replace

> During DRBG instantiation, a nonce may be required, and if used, it is combined with the entropy input to create the initial DRBG seed. The nonce and its use are discussed in Sections 8.6.1 and 8.6.7.

With

> During DRBG instantiation, a nonce **may** be supplied as input, and if used, it is combined with the entropy input to create the initial DRBG seed. The nonce and its use are discussed in Sections 8.6.1 and 8.6.7.

6 Comment – Ambiguous optionality

Location: **SP800-90A, Page 14, Clause 7.2, third paragraph**

Type: **Unclear**

Issue:

What does “strongly advises” mean? Is it mandatory or is it optional?

I cannot know if this is an informative section or a normative section with this sort of language. Does the strength of the advice constitute a requirement? I cannot know from reading the text.

The idea that an instantiation can have a personalization string implies that the context of the instantiation is ephemeral. That the state comes into being with the instantiation and there may be more than one. This is a reasonable property of a software algorithm, but any true RBG necessarily includes a hardware component and a hardware implementation does not have ephemeral state. The flip-flops of a hardware implementation are instanced at the semiconductor factory, not at the call of a software function.

This type of language and the ‘strong recommendation’ therein is symptomatic of the software centric approach of this specification. The standard should describe requirements independent of the means of implementation.

The specification makes a requirement that the personalization string be unique. However this is an input to the system. The compliant implementation has no control over the uniqueness of the personalization string, so this should not be a mandatory requirement. ‘should’ is the appropriate term, indicating that there are consequences if the personalization string is not unique.

Proposed resolution:

Replace

> This Recommendation strongly advises the insertion of a personalization string during DRBG instantiation; when used, the personalization string is combined with the entropy input bits and possibly a nonce to create the initial DRBG seed. The personalization string shall be unique for all instantiations of the same DRBG mechanism type (e.g., all instantiations of HMAC_DRBG). See Section 8.7.1 for additional discussion on personalization strings.

With

> A personalization string **may** be an input during DRBG instantiation; when used, the personalization string is combined with the entropy input bits and possibly a nonce to create the initial DRBG seed. The personalization string **should** be unique for all instantiations of the same DRBG mechanism type (e.g., all instantiations of HMAC_DRBG). See Section 8.7.1 for additional discussion on personalization strings.

7 Comment – Unnecessary restriction on sources

Location: **SP800-90A, Page 21. Clause 8.6.5, first sentence.**

Type: **Normative**

Issue:

The argument here is the same as in comment 2.

Proposed resolution:

Replace

> The source of the entropy input (SEI) **shall** be either:

With

> The source of the entropy input (SEI) **shall** include at least one of:

8 Comment – Incompatible control flow descriptions

Location: **SP800-90A, Page 21. Clause 8.6.5, final paragraph**

Type: **Unclear**

Issue:

The text indicates that with the first two options, the SEI is the slave of the DRBG, providing entropic bits at the request of the DRBG.

The text also indicates that the third option, the DRBG is a slave to the supplying DRBG, providing fresh entropy only when the upstream DRBG supplies fresh entropy.

However the chain in the third type much always terminate in a DRBG of the first or second type. So the chain of dependency is in conflict.

The reality is that SEIs may be caused to produce entropy by the consumer, but it will only produce entropy at the rate imposed by the physical limits of the device. Ultimately, the DRBG is left waiting for the SEI, not the other way around.

It is unnecessary for the specification to concern itself with the locus of control in a DRBG, it should concern itself only in the algorithms and the entropic properties of the data processed by those algorithms.

The paragraph adds nothing, and appears to constrain solutions that cause a reseed when SEI data is available, rather than causing data to be available when a reseed is desired.

Proposed resolution:

Strike the indicated parts of the paragraph.

> ~~In cases 1 and 2, the SEI provides fresh entropy bits upon request. In case 3, the SEI can provide fresh entropy bits only if it has access to an entropy source or NRBG at the time of the request; otherwise, the entropy input provided by the SEI has a maximum security strength that is (at most) the security strength of the DRBG serving as the SEI.~~ Further discussion about entropy and entropy sources is provided in [SP 800-90B]; further discussion on RBG construction and SEIs is provided in [SP 800-90C].

9 Comment – Incompatibility with FIPS140-2

Location: **SP800-90A, Page 23. Clause 8.7, both paragraphs**

Type: **Technical**

Issue:

The text addresses other input to the DRBG besides the SEI, when they may be provided, the role of the provider and the secrecy of the values to others and to the provider.

However this text and the text it refers to (8.6.1 & 8.7.2) are undermined by FIPS 140-2. A FIPS 140-2 compliant DRBG is required by that standard to comply with SP800-90. However 140-2 places explicit rules on data and control passing over the FIPS security boundary into the DRBG. Namely, any input, whether data, control or configuration (except for a human-typed password) must be from an authenticated entity and the data must be authenticated.

This presents a chicken and egg problem. The DRBG in a FIPS 140-2 crypto system would provide random values necessary for key agreement, secure encryption, nonces, keyed hashes and other primitives that would be used in the authentication and session establishment necessary to authenticate and securely communicate with authenticated entities. But those are the same entities that FIPS 140-2 requires be authenticated before they can input additional entropy, nonces and personalization strings, necessary to get the random numbers necessary to do the authentication.

The only identified effective way of complying with both FIPS140-2 and SP800-90 is to have no inputs - no nonces, no personalization strings, no extra entropy and no configuration input. Then there is no authentication requirement.

SP800-90 needs to carve out an exception for the authentication of entities supplying input data and configuration at an SP800-90 boundary that is coincident with a FIPS boundary.

Proposed resolution:

Add a new paragraph to section 8.7

> Entities supplying data and configuration inputs to an SP800-90 DRBG, including the request parameters, nonces, personalization strings and additional entropy are exempt from the entity authentication requirements as described in section 4 of FIPS 140-2.

10 Comment – Ambiguous optionality

Location: **SP800-90A, Page 24. Clause 8.7.2**

Type: **Normative**

Issue:

“is allowed” is not clear specification language. Is it mandatory, optional or informative?

In this case, the addition entropy input may optionally be supported. This language has led directly to misinterpretation by certification test labs that implementations must support additional entropy in case the consuming application wants to supply it. This is wrong and the language must be clarified in terms of the requirements on the implementation not the data.

Proposed resolution:

Replace

> During each request for bits from a DRBG and during reseeding, the insertion of additional input is allowed. This input is optional, and the ability to enter additional input may or may not be included in an implementation. Additional input may be either secret or publicly known; its value is arbitrary, although its length may be restricted, depending on the implementation and the DRBG mechanism. The use of additional input may be a means of providing more entropy for the DRBG internal state that will increase assurance that the entropy

requirements are met. If the additional input is kept secret and has sufficient entropy, the input can provide more assurance when recovering from the compromise of the entropy input, the seed or one or more DRBG internal states.

With

A DRBG **may** support the insertion of additional input during reseeding and generate requests. This input is optional for both the DRBG and the consuming application, and the ability to enter additional input may or may not be included in an implementation. Additional input may be either secret or publicly known; its value is arbitrary, although its length may be restricted, depending on the implementation and the DRBG mechanism. The use of additional input may be a means of providing more entropy for the DRBG internal state that will increase assurance that the entropy requirements are met. If the additional input is kept secret and has sufficient entropy, the input can provide more assurance when recovering from the compromise of the entropy input, the seed or one or more DRBG internal states.

11 Comment – Resource constrained solutions – lack thereof.

Location: **SP800-90A, Section 10, all of it**

Type: **Technical**

Issue:

The 4 approved DRBGs (hash, hmac, ctr and dual_ec) are all based on cryptographic primitives that take substantial resources in terms of state, silicon area or time. Since this specification began development we have:

- New mathematics describing smaller more efficient PRNGs
- New hash algorithms with more efficient native PRNG modes
- New NSA published algorithms, such as Simon and Speck aimed directly as resource constrained environments
- PRNG algorithms where the reseeding (SP800-90A) and conditioning (SP800-90B) are both performed as part of the same reseeding algorithm.
- A growing market for very small, lightweight, resource constrained devices

The algorithms on offer render this specification obsolete, with respect to modern cryptography and device requirements. In order to remain relevant, it must adopt new algorithms that support resource constrained devices, such as wearable electronics, smart cards, RFIDs and isolated on chip security functions.

A basic fast 10 clock AES silicon implementation within inline key schedule takes between 30k and 100k NAND gate equivalents, depending on the synthesis constraints. The majority of the logic going to implement the SBOXes. This is incompatible with the power and size profile of RFIDs, smartcards and heavily re-used on chip circuits.

With current knowledge it is possible to create very compact, resource efficient RNGs that condition raw entropy and output full entropy, with no intermediate DRBG. The DRBG's primary role in an RBG is to match a slow entropy source to a high demand for random numbers. Modern metastability based entropy sources are fast compared to digital DRBG implementations. A simple serial XOR accumulator, sufficiently iterated can reach full entropy from a serial entropy source. Simple serial pattern counting online health test algorithms can be more discriminatory that the tests described in this specification.

Proposed resolution:

Adopt one or more new approved DRBG functions in section 10 with application aimed directly at resource constrained environments. Solicit input for such functions for review at the next RNG workshop.

12 Comment – Dual EC DRBG

Location: **SP800-90A, Page 60. Clause 10.3, all of it**

Type: **Normative**

Issue:

The Dual EC DRBG is sad and unloved.

Proposed resolution:

Remove section 10.3 and all reference in this document and SP800-90B and C to the Dual EC DRBG.

Remove appendix A.

Remove appendix C.

Remove appendix D.5

13 Comment – FIPS 140-2 incompatible output tests

Location: **FIPS 140-2, 4.9.2**

Type: **Unclear**

Issue:

Section 4.9.2 of FIPS 140-2 says:

> "If each call to a RNG produces blocks of n bits (where n > 15), the first n-bit block generated after power-up, initialization, or reset shall not be used, but shall be saved for comparison with the next n-bit block to be generated. Each subsequent generation of an n-bit block shall be compared with the previously generated block. The test shall fail if any two compared n-bit blocks are equal."

This completely undermines SP800-90. A true random sequence will generate adjacent identical values in sequences of 16 bits frequently. By eliminating those adjacent values, the randomness of the sequence is reduced below the (1-epsilon) threshold for full entropy defined in SP800-90.

It is an ad-hoc RNG output test that ignores the testing required by SP800-90.

Proposed resolution:

Add text to exempt SP800-90 DRBGs and ENRNGs from 4.9.2 of FIPS 140-2. Fix 140-2.

Comments Received on SP 800-90B

On 10/11/13 3:22 PM, "Stephanie Eckgren" <seckgren@infogard.com> wrote:

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
1	General	There is not a clear mapping between the validation requirements in Section 4.1 (Entropy Estimation and Validation), Section 6.1 (General Requirements for Design and Validation), and Section 7.0 (Validation Data and Documentation Requirements).	A validation or documentation requirement should not be listed more than once within the document. Consider revising the structure slightly so that the developer and tester are working from the same "validation" requirements.	It appears that the developer might work from Sections 4.1 and 6.1 but the tester will work from 7.0. This may cause issues in the validation.
2	General	The term "developer" is used throughout SP 800-90B. Does this mean that an outside party cannot test/analyze a purchased entropy source?	Clarify the term "developer". Consider using a more general term (e.g., "vendor" as used in FIPS 140).	The "developer" of the entropy source will not always be the one writing documentation and obtaining a validation.
3	General	Will FIPS 140-2 Annex C be updated to include SP 800-90B and C once these are out of draft and published?	Clarify if there will be a new category in Annex C for Approved Entropy Source. If not, clarify where SP 800-90B Approved mechanisms will be listed.	To mesh well with FIPS 140-2 and inform vendors and laboratories on how these guidelines fit into the validation scheme, this point should be addressed prior to final publication of the SP 800-90 suite.

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
4	Section 6.1, Page 23, #2	This states that the security boundary “shall be the same as or be contained within a [FIPS 140] cryptographic module boundary”. This implies that an SP 800-90B implementation must have both a CAVP certificate AND a CMVP certificate to be valid. How will this be handled during validation?	Consider allowing a standalone CAVP certificate. Then if the vendor chooses, they can embed it within a FIPS 140 module.	This is different than other algorithms because they can have a standalone CAVP certificate and still be valid. It needs to be clarified if this is not the case for SP 800-90B.
5	Section 6.5.1.2, Page 30	Many implementations of RNG use a HW RNG to generate a seed and/or seed key, and the HW RNG is not otherwise used by the module. Is continuous testing meaningful in this scenario?	Clarify intent for this scenario.	Assuming a HW RNG must meet SP 800-90B SEI requirements, the value of continuous testing of the SEI in this one-shot use scenario is not clear. The current Implementation Guidance recognizes this scenario in the Additional Comments section of IG 9.8: "If the design of the cryptographic module is such that the Approved RNG or RBG is only seeded (seed and seed key) once from an NDRNG after cryptographic module power-on and never re-seeded (seed and seed key) until the module is powered-off, and the NDRNG is not used for any other function or purposes, than the module does not need to implement a Continuous Random Number Generator

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
				Test on the output of the NDRNG." Is NIST policy for SP 800-90B continuous testing expected to have a similar provision?
6	Section 6.5.1.2, Page 30	FIPS 140-2 guidance will be required to clarify that Repetition Count Test supersedes the current AS09.42 requirement.	Update FIPS 140-2 guidance coincident with publication of SP 800-90B.	A vendor should not fail AS09.42 if Section 6.5.1.2 is implemented. Section 6.5.1.2 appears more permissive.
7	Section 6.5.1.2.2, Page 32	Definition of A is not entirely clear. The intent seems to be that A is the previous value, not the current value.	Define A as the previous sample value (if S != N).	There are 2 samples in this algorithm: the previous value and the current value. - A is defined as the “current sample value” (item #1 in definitions). - Process step 3b: “If A = the current sample value…” So it would always be true.

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
8	Section 7.1, Page 38, Paragraph 1	The following statement is very important and one of the main purposes of SP 800-90B: "The entropy source will have no more entropy than that provided by the noise source, and as such, the noise source requires special attention during validation testing." Unfortunately, there does not appear to be any concrete information in this standard about good/bad noise sources.	Add an Appendix to SP 800-90B titled "Known Noise Sources". This appendix can have tables listing known good noise sources and known bad noise sources. Or tables could potentially list pros/cons of certain noise sources.	The hardest part of SP 800-90B for the developer and testing lab is determining whether the noise source (in theory) is sufficient. If developers and testing labs are given no common concrete information on good/bad noise sources, almost any "noise source" could be accepted with a reasonable argument (even if it is inaccurate).
9	Section 7.1, Page 38, Paragraph 1	Again, the statement that "the noise source requires special attention during validation testing" is very important and one of the main purposes of SP 800-90B. Has NIST determined if the noise source(s) will be categorized and requested by the CAVP?	Consider requiring noise source categories to be, at a minimum, submitted to the CAVP/CMVP during testing (e.g., selections in the CAVS tool). The categories could align with the "Known Noise Sources" appendix suggested in the previous comment.	This will help introduce accountability into the SP 800-90B validations. If this information is not requested by NIST, there may be some hand waving by developers/testers that do not yet fully understand "noise sources".
10	Section 7.1, Page 38, #1	There is no specific requirement for the tester to verify and record the version(s) of the entropy source equipment.	Insert a bullet #1 that says: "Data shall be collected from the entropy source under validation. The version(s) shall be recorded prior to gathering entropy."	It is a crucial step that must be taken. Though it seems obvious, it is sometimes not setup properly by the developer or the tester may miss the step.

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
11	Section 7.1, Page 39, Bullet 1	"Data collection will be performed in one of two ways 1) by the developer with a witness from the testing lab, or 2) by the testing lab itself." Option 1 leaves many interpretations - "witness" can be interpreted as being physically present at all times or as being remotely present for just the setup of data collection software. This can be complicated because gathering 1,000,000 samples could potentially take weeks and/or many overnight running scripts. The meaning and intention of "witness" should be clarified.	Recommendation #1: Remove the language for how the lab must perform the data collection. Simplify the sentence as follows: "Data collection will be performed in accordance with NVLAP standards." Recommendation #2: If Recommendation #1 is unacceptable, add the following footnote to option 1 ("by the developer with a witness from the testing lab"): "At a minimum, the witness must be remotely present (e.g., via a video conference or WebEx) for the startup of the entropy gathering equipment. Physical presence of a witness is highly recommended but not required."	Physical presence of a witness can be expensive, time consuming, and difficult for those travelling. This is especially true if physical presence is needed for potentially weeks of entropy gathering. At a minimum, a remote presence is sufficient in order to gain assurance that entropy is being gathered properly and that the gathering methodology is consistent with developer documentation.

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
12	Section 7.1, Page 39, Bullet 2	In some scenarios, noise sources run under normal operating conditions only during start up (e.g., entropy data is collected within the first 20 seconds at start up). In order to collect 1,000,000 samples, will a source need to be restarted many times?	Permit a single collection session only if the vendor provides an acceptable rationale. Otherwise, require the vendor to collect data under their normal operating conditions (i.e., with intervening power cycles between entropy data collection). Note that the practicality of this approach must be examined (how long will this data collection require?). Consider adding a footnote to clarify this scenario and what is permitted.	It is a common occurrence that entropy data is collected at start up of a module. A known mistake in this case is to simply start up the module once and collect 1,000,000 samples (without restarting). Doing this can cause misleading entropy data (with likely a higher min-entropy). If the vendor cannot provide some acceptable rationale for gathering without multiple start ups, then samples need to be collected over multiple start ups.
13	Section 7.1, Page 39, Bullet 4	Please clarify the meaning of “consecutive” samples and make an allowance if gathering “consecutive” samples is not possible.	Allow for the case where gathering 1,000,000 “consecutive” samples is not possible (e.g., where multiple start ups must occur to gain 1,000,000 samples). Add the following statement: “If generating 1,000,000 consecutive samples is not possible, concatenation of smaller consecutive samples is allowed.”	Collecting 1,000,000 samples can be challenging, and collecting 1,000,000 “consecutive” samples even more so. There needs to be an allowance for consecutive samples smaller than 1,000,000 (i.e., by concatenating).
14	Section 7.1, Page 39, Bullet 4	If claiming “full entropy”, should more than 1,000,000 samples be required?	If this is true, please add a statement about the number of sample required in claiming full entropy.	In the past, we have required a very large amount of data in order to pass full entropy tests in tools such as STS. Min-entropy tools can handle much less data than full entropy tools.

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
15	Section 7.1, Page 40, #3, Bullet 3	This specifically allows the technical argument for the noise source to be “in broad terms” and with “a rough description of the behavior of the noise source”. This is going to allow a developer to simply write a ½ page argument for their noise source(s).	Consider creating an appendix that contains an example argument for a noise source. This example should represent what is expected at a minimum.	As it stands, bullet #3 will allow for the noise sources to be described with minimal information from the developer, and the test laboratory will have no choice but to accept most arguments (even if they are poor). Since the noise sources are so important and are essential to the security of a module, more than a “rough description” “in broad terms” should be required.
16	Section 7.1, Page 40, #3, Bullet 3	Bullet 3 does not clearly require the developer to define their entropy rate prior to writing a technical argument and testing their data.	Bullet 3 should be clarified as follows: “Documentation shall provide an explicit statement of the expected entropy rate and provide a technical argument for why the noise source can support that entropy rate.” Additionally, the CAVS tool (or equivalent) should request the expected entropy rate as input. If the calculated min-entropy is less than the expected min-entropy, the test fails.	If documentation is not specifically required for the “expected entropy rate” before validation testing, developers will be likely to write the documentation after the fact and just insert the min-entropy given by the CAVP. Instead, it is important for the developer to have an estimate of their min-entropy in order to back up their technical argument (before sending entropy data to the CAVP).

#	Section, Paragraph, or Page	Comment	Suggested Revisions	Rationale for Revisions
17	Section 7.1, Page 40, #3, Bullet 4	It should be specifically asked whether all noise sources will be available when the module is deployed in the field. Also, if all noise sources will not be available, how should the entropy estimate be affected?	Add a bullet or a sentence to bullet #4 stating the following: "Documentation shall specify if all noise sources will be available in the field". If all noise sources will not be available in the field, entropy data collected should not include data from a noise source that will not be available in the field.	It seems obvious that any noise source being discussed will be available in the field, but this is not always the case.
18	Section 7.1, Page 40, #3	It will be beneficial to add a bullet requiring documentation about known vulnerabilities and the potential attacker. Example: If part of the noise source comes from human key strokes, could the potential attacker methodically repeat key strokes to get the same entropy data?	Add a bullet to #3 (Documentation for Validation Testing) as follows: "Documentation shall describe the known vulnerabilities of the entropy source and the potential attacker."	If a noise source is very vulnerable, then the noise source itself is very weak.
19	Section 8.4, Page 47	This section seems mislabeled; the content is about an alternative approach to health testing if the specified methods are not implemented.	This content seems to be covered by statements in Sections 6.5.1.2 and 10. Remove this section, consolidating any non-redundant information into Section 6.5.1.2 or Section 10.	Section 6.5.1 specifies health test requirements; Section 6.5.1.2 specifies the requirements for continuous testing.

On 10/22/13 6:43 PM, "Tom Tkacik" <tom.tkacik@freescale.com> wrote:

I would like to thank NIST and the authors for the publication of the two new documents SP800-90B and SP800-90C. These fill a hole in reliable implementation of Random Bit Generation. Without being able to measure the entropy from an entropy source, it has been difficult to know one is indeed any good. That measurement process will lead to overall improvements in entropy generation.

I do have a question about the concept of full entropy. SP800-90B gives a definition of full entropy, that an n-bit string have at least (1-e)n bits of entropy (e <= 2^-64). (The definition does not state if this is shannon entropy or min-entropy. I assume it is meant to be min-entropy.) Can the raw, unconditioned, output of an entropy source ever be claimed to have full-entropy?

Here are some more specific comments on SP800-90B:

Section 6.4.2 specifies how to assess the min-entropy of conditioned output. However, the assessed output entropy is not a monotonic function of the input entropy. An input string S with m=n-1 bits of entropy, will assess the output Y with n-1 bits of entropy. But an input string S with m=n bits of entropy, will assess the output Y with n/2 bits of entropy. Increasing the entropy of the input will result in an output entropy decrease. I do not know the intention of this assessment, but it cannot be correct.

Figure 2 has the X and Y labels reversed.

Section 6.5.1.2.2.1.2 gives values for the cutoff value C. For N=64 and N=256, the values are not correct. (They are correct for N=4096 and N=65536.) Here are the correct values (using the critbinom() function given in footnote 1):

N	64	256
H		
1	55	176
2	39	108
3	27	68
4	20	44
5	15	29
6	11	21
7	9	15
8	7	11
9	6	9
10	5	7
11	4	6
12	4	5
13	3	4
14	3	4
15	3	3

16	2	3
17	2	3
18	2	2
19	2	2
20	2	2

Section 8.2.1.a.i states "the entropy estimate for each output of the conditioning component is min(outlen, entropy_in), where outlen is the length of the output from the conditioning component, and entropy_in is the amount of entropy per bit in the input to the conditioning component". I believe that "entropy per bit in the input" should be "entropy per sample in the input". However, this is still inconsistent with section 6.4.2.1 where the output has 1/2 the entropy of the input.

The relationship between entropy input to an approved conditioning function, and the entropy of the output needs to be better understood, or articulated.

Section 8.2.1.b.ii.a.ii states "Let S be the entropy estimate for the noise source (i.e., the number of bits of entropy per bit of noise source output.)" I believe that S is actually the number of bits of entropy per sample of noise source output, not entropy per bit.

Section 8.3.1.a.i states "For full-entropy sources with no conditioning component: Credit for full-entropy will be given only if the data is verified to be IID, and a full-entropy estimate is produced by the tests in this Recommendation." The test is given in Section 9.2. However, greater than 2^128 samples would be required before raw binary data could be estimated to have full-entropy, according to this test. This is not practical. If the raw output of an entropy can never achieve full-entropy, the document should state that a condition of achieving full entropy is to be the output of an approved conditioning function.

If raw output can achieve full entropy, the document should give realistic requirements that can be actually be met.

Section 9.1.2.4.1.b "W_i = hamming_weight(s_i, ..., s_i+7)" should be "W_i = hamming_weight(s_8i, ..., s_8i+7)".

From: <Scott>, Michael2 <michael2.scott@rsa.com>

Date: Sunday, November 3, 2013 10:28 PM

The feedback from the RSA BSAFE development team concerning the draft of SP800-90B from August 2012 is:

1. Section 6.5.1.3 states: *At a minimum, the start-up tests **shall** consist of one full cycle of the continuous tests to ensure that the continuous tests have had an opportunity to verify that the device is working before it is used*. Could a precise definition of *one full cycle* be given for each of the continuous tests?
2. In section 6.5.1.2.2.1.2 the footnote for the critical value calculation states: *C would be computed as =CRITBINOM(N,2^(-H),1-T)*. However the term *T* is not defined in the document. Should the term 1-T be replaced with α?
3. In the same section, Table 2 illustrates the cutoff values for given entropy per sample and window size. If the table is extended to entropy per sample values greater than 20 then zeroes appear in table. Should the occurrence of a zero values be indicated by the document and their impact on tests be defined?

From: <Thomas>, Ryan <ry.thomas@cgi.com>

Date: Tuesday, November 5, 2013 3:49 PM

Please find the CGI ITSETF's (an NVLAP-accredited laboratory and a candidate NIAP Common Criteria Laboratory) comments on NIST DRAFT SP 800-90B in response to the public comment period on the NIST Computer Security Publications website.

NIST SP 800-90B

1. Section 7.1 General Validation Requirements on page 39:

"*Data collection will be performed in one of two ways 1) by the developer with a witness from the testing lab, or 2) by the testing lab itself. The entropy source shall contain an interface that enables access to raw bits from the noise source and conditioned outputs from the conditioning component (if utilized). This interface shall consume the noise source outputs (i.e., these outputs shall not be used for anything else once received by the interface). The interface shall be accessible during validation testing but may be disabled, otherwise.*"

CGI Comment: We have concerns with the witness testing requirements for data collection in the 90B draft SP as a CST Laboratory that will be responsible for facilitating and/or performing this testing as a part of FIPS 140-2 module validations. In our experience this testing can take upwards of 2-3 days to complete and it is not feasible/practical/economical from a cost perspective for a lab member to be on-site the entire time for witness testing.

There is not much to observe during this process and it can take a very, very long time. The requirement for a lab member to witness the entropy testing will likely mean additional expenses and cost to the lab/vendor and in our opinion it adds very little value in terms of assurance of the testing process and results.

In our opinion, the objective of assurance in the data collection during the validation process can be met in several other ways - a couple of suggestions:

- An affirmation form signed by the vendor stating that the samples collected are from the entropy source and were tested on a certain date;
- Sample testing that can be performed on the entropy source during the FIPS functional testing to verify the results are consistent with previously collected results

We believe that a review of the entropy source (and entropy testing once 90B is published) should be one of the first things performed during the module validation process. Currently, we are seeing a need to make substantial changes to the entropy and entropy sources in the module or the module's operating environment (random drivers in the underlying operating system) in order to meet the requirements in 90B and the FIPS 140-2 Implementation Guidance. From a lab perspective we request that our clients implement make the required changes prior to performing CAVP algorithm testing and FIPS 140-2 functional testing.

If this remains the case it is very likely the lab will have to make multiple trips - this is not cost effective or practical for the lab or many vendors (example: a North American lab that has a vendor based in Asia or Europe). We request that NIST please consider adding additional (more practical) methods for ensuring the integrity of the data collection process.

2. Section 7.1 General Validation Requirements on page 39:

"Data shall be collected from the noise source and conditioning component (if available) under normal operating conditions (i.e., when it is reasonable to expect entropy in the outputs)."

CGI Comment: What can vendors/labs do if there are constraints and no direct access to the module's raw entropy source (before conditioning) is exposed? - ie. output of a COTS chip that is conditioned internally.

Can NIST please provide a brief definition of "Normal Operating Conditions" so that vendors and labs have a clear and unambiguous understanding of the expectations for when it is acceptable for samples to be gathered (vs. it when it not acceptable)?

1. Section 7.1 General Validation Requirements on page 39:

"Data collected from the noise source for validation testing shall be raw, digitized, but otherwise unprocessed, sample values. NIST will provide guidance as to the appropriate format of the data for input to the validation tests."

CGI Comment: Can NIST please advise where this guidance when (and in what document) guidance on the appropriate format of the data for input to the validation tests will be provided?

1. Section 7.1 General Validation Requirements on page 39:

"*One long dataset of at least 1,000,000 consecutive sample values obtained directly from the noise source (i.e., raw and unprocessed samples) shall be collected for validation*"

CGI Comment: Can NIST please elaborate or provide an explanation/rationale as to why NIST selected 1,000,000 samples in an Appendix? Can NIST provide some additional guidance on general principles should the lab or vendor be aware of when making sure the sample size selected is a sufficient sample size for testing/validation purposes?

2. Page 28 of NIST 800-90B states the following:

"Assessed entropy (m)

Output length (n)

Credit entropy output (Y)

Relationship	if(m >= 2n) then Y = n
	if(2n > m >= n) then Y = m/2
	if(m < n) then Y = m

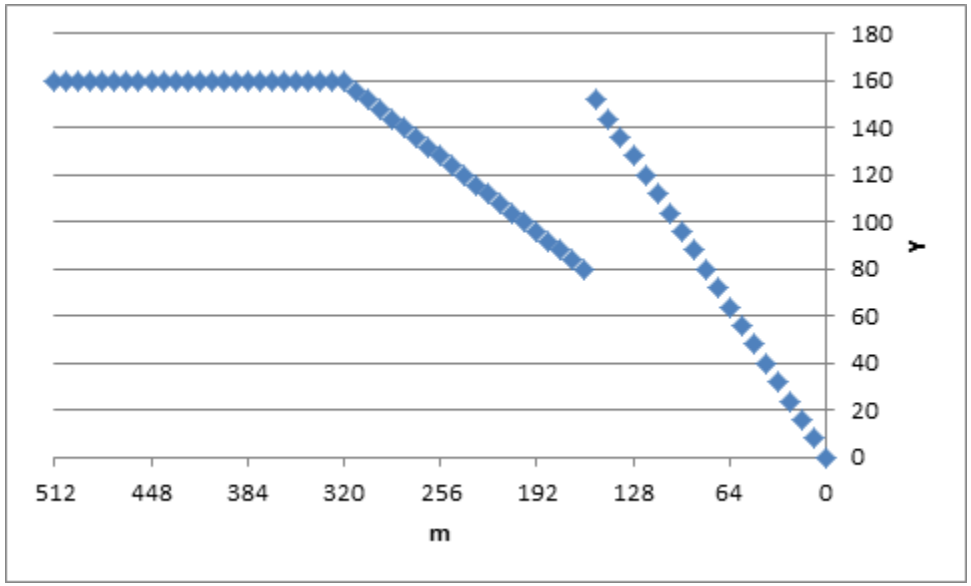
(For the purposes of this exercise, I am going to let n = 160 bits because I am interested in the SHA1 hashing output.)"

CGI Comment: It is the lab's understanding that there are some distributive properties of the SHA hash function (well, any acceptable has function, really) that NIST is making use of when trying to credit entropy after it has gone through the hash.

The problem, in our opinion is when you actually graph m vs. Y you get an odd discontinuity at 160 bits (since *n* is a constant). You end up with the state where having a certain amount of MORE entropy than your output length is a detriment to your credited entropy. That makes no sense to me, but I'm not a crypto expert.

From a Common Criteria perspective, NIAP has asked the labs to follow 800-90B as a guideline in their NDPP v1.1 Addendum (http://www.niap-

ccevs.org/pp/pp_nd_v1.1-add1.pdf). We want to make sure that we are not unduly penalizing vendors if they happen to get an entropy rate that makes them fall into a weird part of the graph. It would be greatly appreciate if we could have something that assumes an average rate of decline for the time-being as a reasonable intermediate analysis step.



During some research on this subject CGI located and interesting set of slides on the NIST website at:
http://csrc.nist.gov/groups/ST/rbg_workshop_2012/kelsey_800-90b.pdf

Slide 19 states: “Our math in SP 800-90B is messed up”

Since this could mean ***anything*** it is hard to determine the intention and implication of this statement. However, since the scope of that slide is

on the “assessed” entropy and the NEXT slide discusses the whole “2n” relationship, I wonder if they are, in fact, talking about this discontinuity.

From a validation perspective can the testing lab be more lenient if a developer claims a hashing function on an input string assessed at some level of entropy and they fall into this “weird” discontinuity?

Please do not hesitate to let me know if you require any clarification on the comments. Thank-you in advance for your consideration**.**

On 11/6/13 2:48 PM, "Brian Smithson" <bsmithson@ricohsv.com> wrote:

Please consider the following comment, sent on behalf of Ricoh.

Comment: Entropy source documentation requirements (6.1, 7.1 list item 3) is difficult, and in some circumstances, not possible when third-party sources are used. Nonetheless, those requirements are mandatory even when the intended purpose of conforming to the Recommendation is to fulfill requirements of another validation process at a far lower level of information assurance.

The SP 800-90 Recommendations would be more broadly useful if they provided two or more levels of conformance requirements, similar to FIPS 140, so that references to SP 800-90 series could be made at a more appropriate level. At the lower conformance level, testing methods specified in Chapter 9 should be sufficient without also requiring detailed documentation.

Rationale: Other validation processes, such as Common Criteria certification in NIAP, references the SP 800-90 series. And yet, current NIAP

policies for Common Criteria certification limit the assurance level to EAL 2 or lower. Detailed documentation is not required at CC EAL 2, and so it is remarkably inconsistent to require such documentation in the referenced NIST recommendation.

From: Rene Struik <rstruik.ext@gmail.com>

Date: Thursday, November 7, 2013 8:11 AM

I have the following comments on the NIST SP 800-90B draft:

1) At the review last year (2012) I expressed concern that the language of the draft would effectively rule out using physically unclonable functions as a source of true randomness. This would be highly unfortunate, since this could prove to become quite an attractive low-cost true randomness source with high min-entropy. Since the draft did not change since last year, I thought I should re-emphasize this point.

2) I would like to make you aware of the recently published paper

Key Derivation Without Entropy Waste (Yevgeniy Dodis, Krzysztof Pietrzak, Daniel Wichs, IACR ePrint 2013-708). It seems worth investigating whether the bounds on entropy-loss in that paper would justify loosening the requirements on how much min-entropy a true randomness source would need to have to justify derived keys to be considered fully random (it seems one would gain a factor almost 2x in efficiency this way).

This relates to my comment from the review last year re

Section 6.2, p. 24, 2nd item: It is not entirely clear where the “double block size requirement” comes from. Assuming the input to the conditioning

function (with output size *n* bits) to be “full entropy”, this suggests that this requires at least 2*n*-bit entropy. Again, this suggests that – with the use of an only the SHA-x family of hash functions as approved conditioning functions – one is stuck with noise sources with 2*n*> 320 of entropy and cannot use a noise source offering *n*=128 bits of min-entropy in case on throws in a conditioning function. This needs some more explanation.

On 11/8/13 4:54 PM, "David Johnston" <dj@deadhat.com> wrote:

Location: **SP800-90B, , Page ii, abstract KEY WORDS**

Type: **Informative**

Issue:

The primary role of this spec is to define NRBGs up to and including the conditioning part. So it would be appropriate to include them in the keywords

Proposed resolution:

Replace:

> KEY WORDS: deterministic random bit generator (DRBG); entropy; hash function; random number generator; noise source; entropy source; conditioning component

With

> KEY WORDS: deterministic random bit generator (DRBG); entropy; hash function; random number generator; non-deterministic random bit generator (NRBG); entropy source; entropy conditioner

14 Comment – Taxonomy of RBGs

Location: **SP800-90B Page 8, 1.0 Scope, paragraph 1, SP800-90B Page 10, Definition, SP800-90B, page 12, Definition, SP800-90B Page 13, Definitions**

Type: **Informative**

Issue:

The taxonomy of RBGs appears to be unclear. Paragraph one states that SP800-90 A & C covers DRBGs and RBGs. However SP800-90C differentiates NRBGs and Enhanced NRBGs and the definitions in SP800-90C defines enhanced-NRBG but not NRBG, while the text in SP800-90B uses entropy-source and NRBG interchangeably.

There should be a clear taxonomy, with the definitions in sync. E.G. following the diagram below:

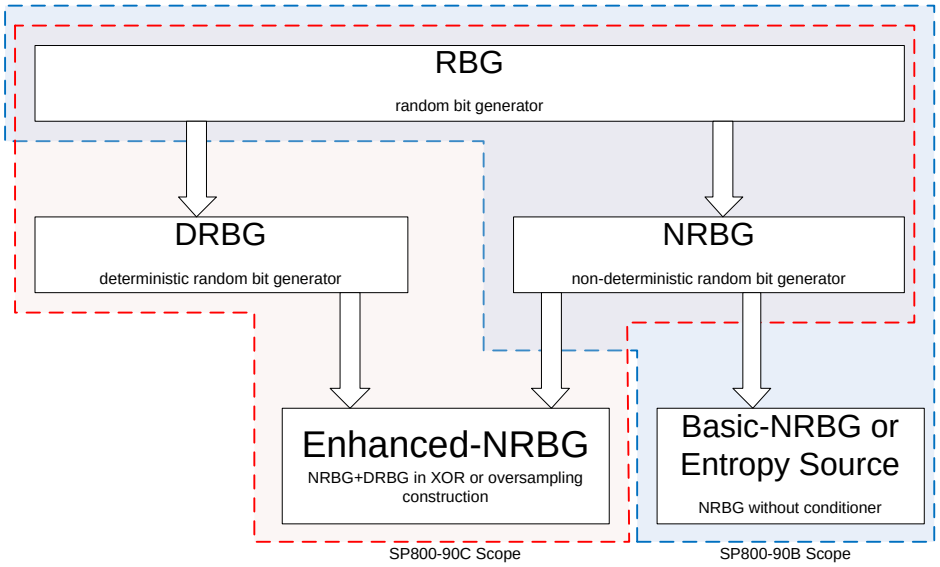


Figure 1 Hierarchy of RNG Types

Enhanced-NRBGs are out of scope in SP800-90B. ‘Noise Source’ is a subset of ‘Entropy Source’ and ‘Entropy Source’ is equivalent to ‘Basic-NRBG’.

So the SP800-90B definitions should cover RBG, NRBG, Noise Source, Entropy Source and Basic-NRBG

The SP800-90C definitions should cover RBG, NRBG, Noise Source, Entropy Source, Basic-NRBG and Enhanced-NRBG.

Proposed resolution:

Replace at SP800-90B Page 8, 1.0 Scope, paragraph 1:

> Cryptography and security applications make extensive use of random numbers and random bits. However, the generation of random bits is problematic in many practical applications of cryptography. The purpose of NIST Special Publication (SP) 800-90B is to specify the design and testing requirements for entropy sources that can be validated as approved entropy sources by NIST‘s CAVP and CMVP. SPs 800-90A and 800-90C address the construction of approved Deterministic Random Bit Generator (DRBG) mechanisms and approved Random Bit Generators (RBGs) that utilize the entropy sources and DRBG mechanisms, respectively.

With

> Cryptography and security applications make extensive use of random numbers and random bits. However, the generation of random bits is problematic in many practical applications of cryptography. The purpose of this Recommendation is to specify approved Non-deterministic Random Bit Generators (NRBG) for use in random bit generators (RBGs). NIST Special Publication 800-90A addresses the construction of approved Deterministic Random Bit Generator (DRBG) mechanisms. NIST Special Publication 800-90C addresses approved Random Bit Generators (RBGs) that utilize Enhanced-Non-Deterministic Random Bit Generators (Enhanced-NRBG) using both NRBG and DRBG mechanisms together.

Replace at SP800-90B Page 10, Definition:

> Deterministic Random Bit Generator (DRBG)

An RBG that employs a DRBG mechanism and a source of entropy input. A DRBG produces a pseudorandom sequence of bits from an initial secret value called a seed (and, perhaps additional input). A DRBG is often called a Pseudorandom Bit (or Number) Generator.

With the authoritative text from SP800-90A

Deterministic Random Bit Generator (DRBG)

An RBG that includes a DRBG mechanism and a source of entropy input. The DRBG produces a pseudorandom sequence of bits from a secret initial value called a seed, along with other possible inputs. A DRBG is often called a Pseudorandom Number (or Bit) Generator.

Add at SP800-90B, page 12, Definitions:

Enhanced Non-Deterministic Random Bit Generator (Enhanced-NRBG)

An RBG that uses both an NRBG and a DRBG together to produce a full entropy output.

Replace at SP800-90B Page 14, Definitions

Random Bit Generator (RBG)

A device or algorithm that is capable of producing a random sequence of (what are effectively indistinguishable from) statistically independent and unbiased bits. An RBG is classified as either a DRBG or an NRBG.

With

Random Bit Generator (RBG)

A device or algorithm that outputs a random sequence. An RBG is one of a DRBG, an NRBG or an Enhanced-NRBG.

Location: **SP800-90B 6.5.1.2**

Type: **Technical (very)**

Issue:

The continuous testing section 6.5.1.2 appears to conflate the false positive rate of the noise source health tests with the failure tests of the entropy source. These are not the same thing. Two tests are defined, but the first is a test suitable for per-sample health tests and the second is appropriate for producing an entropy source failure indication.

Continuous tests fall into to two types: per-sample health tests and longer term failure tests. The text should reflect this so that the appropriate requirements can apply to the appropriate test type.

The requirement for a low false positive error rate on the noise source test is not a conservative requirement. There is a linear trade-off between false positive error rate and false negative error rate. To be conservative in noise source testing, we want to require low false negative error rate, which implies a high false positive error rate at the noise source. At the entropy source level we want the identified unhealthy noise source data to either be used but not counted or not used at all. Any conditioning present must take this into account so that the entropy source output has a very low failure rate. This implies short range per-sample tests for the noise source to tag samples as healthy or not, while having longer term metrics that determine if the rate of unhealthy tagged samples are indicative of a failed source.

The tests specified do not fit with these concepts. They appear as independent tests on the noise source that raise an entropy source failure, with a low probability. The tests require a large amount of buffering and are not realizable in hardware at reasonable cost in many applications.

The focus of the tests should be separated into distinguishing healthy samples from unhealthy samples, and distinguishing a failed from a functional noise source. The latter tests may depend on the output of the former tests. Sliding windows tests that apply bitwise tests to a range of bits as they ‘slide’ by in a shift register are appropriate for fast hardware implementation since the incremental work per sample can be low, while the statistical strength of the test over the sliding window can be high.

Any test identifying healthy from unhealthy samples is simply separating all the possible bit strings into two sets, the healthy ones and the unhealthy ones. For a binary symmetric source, all strings are equally likely, so in practice we need to choose tests that favor strings that are less indicative of hardware failure but should be conservative so that we reject more data than is necessary to ensure that hardware failures are detected quickly.

In a hypothetical example system with noise source tests that test 512 bit samples, and tag them as healthy or unhealthy, a subsequent long range window test need only buffer and analyze 1 bit of health data per 512 bits of noise source data, while effectively monitoring the rate of health output. In combination with a practice of “use every sample, but only count the healthy samples”, the high false positive rate of the noise source tests will not diminish the entropy rate out of the entropy source, but will diminish the entropy throughput by the ratio of false positive error

samples to total number of samples. e.g. A 10% false positive error rate on a healthy noise source will reduce the throughput by 10% but will have very conservative failure detection properties.

The two tests provided are specific algorithms. It would be better to specify the conditions that must be detected and let the developer show that their implemented test detects those conditions. For a well modeled noised source, it should be simple to show the false positive error detection rate for a functioning noise source and simple to show the false negative rate for any specific hardware failure.

It seems inappropriate to suggest that data may be output before the data has been tested. However in a practical hardware implementation this is not necessary. It is easy to design hardware to produce the result of a test in parallel with the production of the samples, so that test result and sample are available simultaneously. The specification should be stricter about this, since the implementation impacts of the stricter model are negligible. The parallel form of implementation is the more natural form in traditional RTL design.

The tests defined appear to operate over a stream of values, updating their state as values are generated and passed on. The determination of an error may occur only at the end of the defined failure sequence and the contributing bits have already been passed to the conditioner or entropy source output.

While noise sources will typically be single bit serial, or have relatively narrow fundamental bit widths, conditioning algorithms, such as CBC-MAC or HMAC will typically have wide block width inputs. Therefore it is appropriate to permit the test hardware to gather test data units of multiple bits (that are likely to match an integer multiple of the input width of the conditioner) and perform the tests over the test data unit such that the health status of the test data unit is known before the data unit is passed to the conditioner or entropy source output. It is wrong to allow noise source data to be used until its health has been assessed.

It is not reasonable to specify test algorithms without reference to the implementation context (e.g. hardware vs. software, parallel vs. sequential, undoable vs. non undoable results), particularly when the necessary tests will be focused on the expected failure modes specific to the noise source in ways that cannot be anticipated by this specification. Minimum test conditions should be specified but it should be expected that the test implementation will go beyond the minimums, to test expected failure modes specific to the noise source.

The text of 6.5.1.2 uses bullet points for the specific requirements, but previous text uses numerical prefixes, so the requirements are numbered. This section should follow the numbered form.

I do not see how the adaptive proportion test finds the most frequent value. It appears to only count the frequency of the first sample seen in every N bits. Is there an else clause missing from test part 3?

I propose that the recommendations around false positive error rate be substantially changed to require a high enough false positive rate to have a low false negative rate.

I propose that these constraints be applied to the noise source per sample tests but not the entropy source failure mode detection tests, which should have a low probability of false positive error at the cost of having a per time throughput diminished by the false positive error rate of the per sample test, while maintaining a reliable high entropy rate.

I propose that instead of specifying specific test algorithms, the conditions the algorithms detect be specified, so that the developers are free to implement more general or more stringent tests that detect the specified conditions while potentially detecting a broader set of error conditions in the same test suite.

I propose that the adaptive proportion test be dropped in favor of a test that uses the long term ratio of healthy to unhealthy samples be used to determine the failure of the noise source. The adaptive proportion test appears to be aimed at the failure mode of certain specific implementations, whereas it would be powerless against other implementations that wouldn't fail in a way detectable by this test. The requirement that the developer documents known likely failure modes and tests for them is stronger than this test. A simple inspection of the noise source will reveal the nature of likely failure modes and it can be seen if the provided tests detect them. The adaptive proportion test would for instance be powerless against a serial noise source that oscillated 0101010101, as would the stuck bit tests. However in the text proposed below, a design prone to such an output would be obliged to test for the condition.

The writers of this specification should be aware that modern fast electronic entropy gathering circuits are just as likely to fail to an oscillating state as to a constant state, given the feedback necessary in such designs. This commenter has witnessed such behavior in on silicon implementations taken outside their design envelope.

I propose that the terms continuous noise source health tests (CNSH tests) and continuous noise source failure tests (CNSF tests) be used to differentiate the two classes of continuous test. I also propose that the requirements on these two classes of test be defined separately.

I propose that bullet points be replaced by a numeric index so that requirements are enumerated.

Proposed Resolution:

Replace 6.5.1.2 with

Continuous tests include

- Continuous Noise Source Health Tests (CNSH tests)
- Continuous Noise Source Failure Tests (CNSF tests)

The purpose of noise source health testing is to allow the entropy source to detect many kinds of disastrous failures in its underlying noise source. These tests are run continuously on all noise source outputs.

The purpose of noise source failure tests is to detect a catastrophic failure of the entropy source or underlying noise source by detecting a failure or failures that persist over time, to differentiate them from transient statistical false positive errors.

In the case of per-sample health tests of noise source data, there is a linear tradeoff between sensitivity to the false positive and false negative error detection properties of the applied test or tests. To ensure that it is unlikely that low entropy samples from a defective noise source be unrecognized as such by the applied continuous test, the test sensitivity should be set to reject healthy entropic data from a functioning noise source at a relatively high rate, e.g. In a specific implementation, between 1% and 10% might be an appropriate percentage of the per sample false positive error rate, to achieve a very low false negative error rate.

The requirements for the continuous per-sample health tests are:

1. The per-sample health test or tests shall detect the excessive repetition count condition described below in 6.5.1.2.1.
2. The per-sample health test or tests may detect other conditions in excess of those defined in 6.5.1.2.1.
3. The developer shall describe the test algorithms
4. The developer shall show how the test algorithms detect the condition in 6.5.1.2.1.

Entropy source failure tests that produce the final failure mode determination should be tolerant of a relatively high per sample error rate. Such tests should raise a failure signal when indications of per sample failures happen too frequently, for example in a specific implementation, a per sample error rate above 50% over a certain window of time might be a threshold that triggers a failure indication. It is easy to compute the false positive failure rate for a sliding window over per sample health test results and it is practical to achieve a very low probability of a false positive failure rate. The developer is required to show that the probability of a failure indication from an entropy source with a functioning noise source is low enough that it is unlikely to be seen over the lifetime of a device.

The developer is required to define the expected lifetime of the device and the acceptable false positive failure rate for the entropy source. These values may be very dependent on the application.

The requirements for the continuous entropy source failure tests are:

1. The developer shall define the expected lifetime of the device

2. The developer shall define the acceptable false positive error rate of the device

3. The entropy source failure test or tests shall detect the conditions described below in 6.5.1.2.2.

4. The entropy source failure test or tests may detect other conditions in excess of those defined in 6.5.1.2.2.

5. The developer shall show that the continuous entropy source failure test or tests yield a false positive error rate equal to or below the acceptable false positive error rate for the device.

6. The developer shall document any known likely failure modes of the device

7. The developer shall describe the false negative error rate of the implemented tests for the known likely failure modes of the device.

8. The developer shall describe the test algorithms

9. The developer shall show how the test algorithms detect the condition in 6.5.1.2.2.

Change title of 6.5.2.1 to Excessive Repetition Condition

At SP800-90B 6.5.2.1 Repetition Count Test, change text to

Where :

A test data unit is a number of bits collected from the noise source that is tested by the per sample health tests independent from other tests data units.

N = The number of bits of data gathered from a noise source into each test data unit.

S = The number of values of data gathered into each test data unit.

H = The per value entropy of the output data of the noise source.

C = The sequential cutoff limit.

w = The fundamental width of the noise source output in bits.

Note that N = wS.

W = The acceptable false positive error probability per test data unit

and where C=⌈ 1+((-log(W)))/H⌉

The excessive repetition count condition is defined as true when the following condition hold true, over a test data unit of N bits:

There exists a contiguous sequence of values of the same value, equal to or longer than C.

At SP800-90B 6.5.2.2 Adaptive Proportion Test for the Most Common Value: Change title to Noise Source Failure Condition

Change text of 6.5.1.2.2 to:

Where :

A tested data unit is a number of bits collected from the noise source that is tested by the per sample health tests, along with the result of the per sample health test. It includes the binary health result and the N bit data value.

N = The number of bits of data gathered from a noise source into each tested data unit.

R = The result of the per-sample test where 1=healthy and 0=not healthy.

P = The failure cutoff ratio.

m = The length of a memory M of the health of the last m tested data units

M = The memory of the health status of the last m tested data units.

H = The number of bits = 1 in the memory M.

The noise source failure condition is defined as true when the following condition hold true, over the last m health status bits:

(H/(m-H))<P

Therefore it is required that of the last m tested data units from the noise source at least P/M of them are healthy.

Delete SP800-90B 6.5.1.2.2.1

Delete SP800-90B 6.5.1.2.2.2.2

Delete SP800-90B 6.5.1.2.3

From: <Nicholls>, Tom <Tom.Nicholls@thalesesec.com>
Date: Wednesday, November 6, 2013 10:46 AM

Legend (type of comment)
E = Editorial G = General T = Technical

ID	SECTION, SUBSECT & PARA.	TYPE	COMMENT	RESOLUTION
1	6.1.1 & 6.3.2	G	Estimates of entropy from physical sources can only really be justified with reference to details of the hardware device that may not be in the public domain.	Consider modifying the requirements to facilitate the use of 3rd party hardware noise sources by reducing the level of detail required for documenting the internal structure and/or Intellectual Property required to satisfy this

				specification.
2	6.1.5	G	An interface to the raw noise must be available, and capable of being sampled at a sufficient rate; this may be difficult under normal operating conditions.	Consider modifying the requirements to facilitate the use of 3rd party hardware noise sources by reducing the level of detail required for documenting the internal structure and/or Intellectual Property required to satisfy this specification.
3	6.2	G	The required information on the conditioning algorithm in use within third-party components may not be available due to 3rd party Intellectual Property concerns.	Consider modifying the requirements to facilitate the use of 3rd party hardware noise sources by reducing the level of detail required for documenting the internal structure and/or Intellectual Property required to satisfy this specification. Alternatively, if such documentation cannot be provided could the specification say that the output must be treated as if it were unconditioned entropy?
4	6.2	T	We would expect that the amount of input data required by the conditioning function to be correlated to the equivalent bit strength of the underlying cryptographic primitive (as defined in SP800-57, Part 1) and the entropy per bit ratio in order to deliver its full security strength given the	Update the second sentence in the second bullet-point to reflect this or clarify to the contrary. Is this related to the statement in section 6.4.2.2 that an approved conditioning

			intended operation.	function should produce n bits of full-entropy output from an input having 2n bits of entropy, if so should this statement be referenced here?
5	6.3.3, 6.3.4, 6.5.1 & 7.1	G	When using 3rd party components which provide the entropy source it may not always be possible to get direct access to the noise source (i.e. bypassing the conditioning function).	Provide a method for validating entropy sources in these scenarios rather than the noise source.
6	6.4	T	If using a 3rd party entropy source that internally has an unapproved conditioning component (e.g. a Von Neumann algorithm for bias removal) is it possible to use an approved conditioning component on the entropy source (after the Von Neumann) to allow the vendor to claim 'full entropy'?	Please clarify this scenario.
7	6.4.2.1.1 & 6.4.2.2	T	These sections describe approved (keyed and unkeyed) conditioning functions. They both say that entropy should be assessed as follows: "If the input string S was assessed at 2n bits of min-entropy or more (i.e., m >= 2n), then Y may be considered to have n bits of full entropy output. If S was assessed at m bits of min-entropy and 2n > m >= n, then Y shall be assessed at m/2 bits of min-entropy. If S was assessed at m bits of min-entropy and m < n then Y shall be assessed at m bits of min-entropy." We be wrong. In particular, it defines a function from the	We think it should read: "If the input string S was assessed at 2n bits of min-entropy or more (i.e., m >= 2n), then Y may be considered to have n bits of full entropy output. If S was assessed at m bits of min-entropy and m < 2n, then Y shall be assessed at m/2 bits of min-entropy."

			entropy m of the input string to the assessed entropy of the output that is non-monotone: it has a downward step from n to n/2 at m = n.	
8	General comment	G	As a general comment we feel that the specification should be more closely linked to the statistical test suite for random and pseudorandom number generators for cryptographic applications (SP800-22).	If this is a deliberate separation please clarify why this standard does not reference SP800-22.
9	General comment	G	We would like to see credit given to entropy sources that are already validated to other certifications/standards (e.g. AIS 31).	Please clarify this scenario.
10	General Comment	G	What happens in the scenario where a FIPS 140-2 Level validated modules uses an external source for it entropy. A situation could arise when this entropy source has not been evaluated to the same levels (e.g. environmental factor testing).	We propose that all level 3 and level 4 modules should not be allowed to have an external entropy source. This will also promote good security practice by keeping the entropy within the validated enclosure.
11	General Comment	G	Entropy validation testing currently requires a lab to collect the raw data or for the lab to be present at data collection. This is contrary to the current algorithm testing format. There has to be a level of trust between NIST, the lab and the vendor. With current algorithm testing a vendor could run the vectors against a completely different module and the lab	We propose that this requirement be dropped to bring this testing in line with the current algorithm testing procedures. The current requirement has the potential to dramatically increase both time and costs of entropy source validation for vendors and labs.

			would be none-the-wiser.	
12	General Comment	G	Is the importing of entropy into a module (i.e. from outside the security boundary of the system) permitted? For example, is the import of entropy from a live source via a file allowed? If so, is there any guidance for this type of operation?	Please clarify this scenario.

Comments Received on SP 800-90C

On 10/22/13 6:43 PM, "Tom Tkacik" <tom.tkacik@freescale.com> wrote:

Here are some more specific comments on SP800-90C (only typos):

Section 5.6. "An SEI may be an entropy source that conforms" should be "An SEI may be an entropy source that conforms".

Section 8, line 2. "and examples of DRBGs are provided in Appendix B" should be
"and examples of DRBGS are provided in Appendix D".

Section C.1, bottom of page. "Note that using a single permutation algortihm" should be "Note that using a single permutation algorithm".

*****

On 11/8/13 4:54 PM, "David Johnston" <dj@deadhat.com> wrote:

1 Comment – Health Test Requirements

Location: SP800-90C definitions, SP800-90C

Type: **Technical**

Issue:

The text on the health tests on the conditioning component appears to have unnecessary discussion of matters not pertinent to the testing. The cryptographic nature of a conditioner is immaterial to how to test its correct construction. Verifying the BIST testing of the correctness of construction of the logic gates is a practice called ‘fault grading’ and is a practice that works without reference to the underlying algorithm being tested. This proposed change addresses both the ‘bias vs. entropy rate’ and the health testing issues that would otherwise be addressed in a later section.

Proposed Resolution:

Replace at SP800-90B, 6.5.2

> 6.5.2 Health Tests on the Conditioning Component
>
> The role of the conditioning component is to reduce the bias that would otherwise be present in the entropy source output and/or to ensure that the output bitstrings provide entropy at an acceptable rate. The conditioning component will implement a deterministic algorithm.
>
> The functional requirements for the health tests of the conditioning component are:
>
> 1. The conditioning component shall be tested during start-up with known answer tests necessary to establish that the conditioning component is working as designed.
>
> 2. The developer shall describe the health tests implemented for the conditioning component to include the failure conditions covered by the tests chosen.

With

6.5.2 Health Tests on the Conditioning Component

The role of the conditioning component is increase where possible the entropy rate of the data at the conditioning component output relative to the data present in the entropy source output and/or to ensure that the output bitstrings provide entropy at an acceptable rate.

The conditioning component shall implement a deterministic algorithm.

The requirements for the health testing of the conditioning component are

1. The conditioning component shall be tested during the startup period with known answer tests necessary to establish that the conditioning component is working as designed.

2. The developer shall describe the health tests implemented for the conditioning component

3. The developer shall show the fault coverage of the health tests implemented for the conditioning component.

4. The developer shall show that the fault coverage of the health tests is sufficient to ensure a negligible probability of a construction error in the conditioning component going undetected.

From: <Nicholls>, Tom <Tom.Nicholls@thalesesec.com>
Date: Wednesday, November 6, 2013 10:46 AM

Legend (type of comment)
E = Editorial G = General T = Technical

ID	SECTION, SUBSECT & PARA.	TYPE	COMMENT	RESOLUTION
1	4.2.1	G	Why does the specification require $2n$ bits of entropy to be input into an approved conditioning function, where n is the length of the output block of the approved derivation function (SP800-90B) to enable the entropy source to achieve full entropy?	Please explain the rationale for choosing $2n$.
2	4.2.1	G	How do security strengths and block sizes relate to conditioning algorithms? As an example; using the approved CMAC conditioning function with AES-256 as the underlying cipher would provide the same level of entropy as a CMAC function with AES-128 as the underlying cipher because the block size of both is 128 bits. Is there any benefit to using a higher security strength conditioning function or is it purely based on the block size?	Please explain why the input/output sizes of the conditioning functions are based upon block sizes and not security strengths.

3	5.4 & 7.2	T	The specification asserts that a DRBG without access to a 'Live' entropy source cannot achieve full entropy output or output with prediction resistance. Is it permitted to buffer and securely store (in a manner equivalent to the internal state of the DRBG) the output of a 'Live' entropy source (to avoid delays in the production of entropy or when operating in a power-limited environment) for later use? Then when this entropy is used for it to be still considered a 'Live' entropy source? Whether the entropy is being directly provided at the time of reseeding or from a stored buffer becomes irrelevant assuming the level of protection on the buffered entropy is sufficient.	Please clarify this scenario.
4	7.4	E	We believe 'than' should be 'then'.	Resolve grammatical mistake.
5	8.2	E	There appears to be a contradiction between 'A (target) DRBG without access to an entropy source after instantiation....' and the second bullet 'An entropy source or NRBG is made available to the target DRBG expressly for the purposes of reseeding'. Are you referring to a scenario where the DRBG has periodic access to an NRBG for reseeding purposes?	Resolve or clarify the contradiction.