Comments Received on SP 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Michael Graham, Advantura....................2

William Gill, EPA....................3

Robert Zuccherato, Entrust....................4

Werner Schindler, Bundesamt für Sicherheit in der Informationstechnik (BSI) 9

William L. Millan, ISI, QUT 11

Kristen Lauter, Microsoft....................16

From: Michael Graham <mgraham@advantura.com>
Date: Saturday, December 31, 2005 1:47 AM

Within SP 800-90, page 12 (in the definition of entropy source in the table) and page 91 (first paragraph on page and Figure C-1) refer to "conditioning" or "optional conditioning" of the bits provided by the digitalization of the entropy source. This process is more generally referred to as "entropy distillation" and is briefly explained in SP 800-22 in Section 1.1.3.

A number of methods for obtaining random numbers from computer hardware in a PC are already in use and some of the new VIA processors have hardware random number generators built in. Not all of these methods condition the output of the entropy source (although the VIA processors do, using a von Neumann whitener). I believe it would be very useful to include a brief section on how to condition the output of an entropy source within SP 800-90. This would allow readers whose hardware has a potential entropy source to be able to use that source with some confidence that the output of the source is not overly biased. One of the most common conditioning methods, the von Neumann whitener, is relatively easy to explain and is easy to code in a variety of computer languages.

I believe the inclusion of this information would require less than a page and would be of benefit to many readers of SP 800-90. Please consider including it.

Sincerely,

Michael F. Graham
Vice President
Advantura LLC

From: Gill.William@epamail.epa.gov <Gill.William@epamail.epa.gov>
Date: 1/12/06 11:53 AM

EPA has no comments on the subject draft.

Thank you for the opportunity to comment.

William Gill, CISSP
Technology and Information Security Staff
Office of Technology Operations and Planning
Office of Environmental Information

From: Robert Zuccherato <robert.zuccherato@entrust.com>
Date: Fri, 27 Jan 2006 08:57:26 -0500

1. Section 2: The last paragraph of this section states, "If the seed is kept secret, and the algorithm is well designed, the bits output by the DRBG will be unpredictable, up to the security strength of the DRBG algorithm." Strictly speaking, this is not correct. The bits output by the DRBG will be unpredictable up to the minimum of the seed entropy and the security strength of the DRBG algorithm.

2. Section 7.2: The third paragraph of this section states, "The personalization string shall be unique for all instantiations of the same DRBG type." This requirement is also stated without rationale. It would seem then that this would preclude the use of a personalization string that does not change or only changes rarely. However, there may be valid and good reasons for including this type of personalization string. For example, an implementation may not be able to create a personalization string that is unique every time that a DRBG is instantiated, but it may be able to use a constant personalization string (e.g., device serial number that remains constant between instantiations, but is different for each device). There is value in using this type of personalization string since it will still provide some variability, although not as much as when it changes for each instantiation. We strongly suggest that unique personalization strings should be recommended, not required. Note also that the text and the suggested personalization string contents in Section 8.7.1 seem to support a more open interpretation of the personalization string.

3. Section 7.3: Replace "contents" with "content" in the last sentence.

4. Section 8.2: The second paragraph says, "when reseeded, the seed SHALL be different than the seed used for instantiation." This makes it a system requirement to ensure the seed is always different, which can easily be burdensome, and also does not say that a reseed seed should be different from another reseed seed or that initial seeds should be different across instantiations. Suggest rewording along the lines of "All seeds used SHOULD be different and SHALL NOT knowingly be identical to a previously-used seed."

5. Section 8.2: It is probably also worth mentioning at this point that implementations should not re-use the entropy input from the instantiation as the entropy input for the reseeding (and then think that they have obtained prediction resistance). Suggest adding, "When reseeded the entropy input used SHALL NOT be knowingly identical to a previously-used entropy input."

6. Section 8.6.3: Replace "it" with "or" in the second sentence.

7. Section 8.6.8: This section claims that reseeding is preferred over a new instantiation. This had better not be true. If entropy assessment is failing, we have no idea how to assess whether a system is secure or not, so it should be assessed as not secure and not used.

8. Section 8.6.10: As plural nouns are used, the exact requirement can be unclear. Suggest using a singular form, so any implementation choice can be seen to be either conforming or non-conforming. The current text could be interpreted as requiring that a given entropy input shall not be used to instantiate a DRBG more than once. However, the real requirement is that the seeds be different (so that the DRBGs do not output identical bits), not that entropy input be different. There may be good and valid reasons for instantiating a DRBG with the same entropy input more than once (e.g., new entropy input is not available and the alternative is to just keep a single DRBG instantiated for a very long period of time). We suggest rewording this to: "A seed used by DRBGs shall not knowingly be used again as a seed for another DRBG or for any other purposes (e.g., domain parameter or prime number generation). Similarly, the specific entropy input used to create a seed shall not be used for other purposes, although it may be used again to instantiate another DRBG, as long as the created seed is different."

9. Section 8.7.1: It is not clear what is meant by "The value of any secret information contained in the personalization string." It appears that this sentence is trying to make the point that the DRBGs cryptographic mechanisms will protect the secret information and thus information should not be used that requires greater protection. We suggest rewording this sentence to "Secret information should not be used in the personalization string if it requires a level of protection that is greater than the claimed strength of the DRBG."

10. Section 8.8: In regards to backtracking resistance and prediction resistance, the system states cannot be distinguished from random "with less work than associated with the security level of the instantiation".

11. Section 9.1: Some of the comments occur between lines and so it is not clear whether the comment applies above or below the line, the reader should not have to need to figure this out. An example of this is in Section 9.1 after "Required info not provided…" This occurs elsewhere, also.

12. Section 9.1: This section states (within a comment) that when instantiating a DRBG the entropy input and the nonce "shall not be provided by the consuming application as an input parameter during the instantiate request." (A similar comment is made in Section 9.2 regarding reseeding.) This requirement is stated without rationale. We note that this requirement will make toolkit implementations of DRBGs without entropy sources more difficult. It will make it impossible for a toolkit to implement a validated DRBG without also implementing a validated entropy source. Given the difficulty in implementing validated software entropy sources, this will make implementing a validated software DRBG very difficult. We see no reason to not allow a consuming application to obtain a given amount of entropy (possibly from a hardware source) and provide it to the DRBG. In addition, this will force the DRBG implementation to place any entropy provided by the consuming application into the personalization string, which does not necessarily seem to be the right place for it to go. We also note that the requirement is ambiguous as written. Does it allow the consuming application to provide a callback that would supply the entropy input and nonce? Does it allow a non-validated DLL associated with the consuming application to be registered that would supply the entropy input and nonce? Thus, we suggest removing the requirement in this comment.

13. Section 9.1: This section (and subsequent sections) assumes that there will be a *state_handle* to identify the internal state associated with various instantiations. However, if an implementation only supports the instantiation of a single DRBG at a time, then an explicit *state_handle* is not required. Thus, the *state_handle* should be made optional in this case.

14. Section 9.3.1: This section states that if a reseed capability is not available, then steps 6 and 7 (where the DRBG is reseeded because the *reseed_interval* has been reached) can be removed and replaced with an Uninstantiate call. However, there may be valid reasons why an application may wish to uninstantiate at this point instead of reseeding, even if reseed capabilities exist. Thus, replacing steps 6 and 7 by an Uninstantiate call should be made strictly optional.

15. Section 9.5: The second paragraph of this section states that if errors occur during testing, the condition causing the failure must be fixed and the DRBG must be re-instantiated. However, it is not clear that this is always the best course of action. For example, if the error occurs while testing the Uninstantiate function, naively following this advice may open the application to further risks (if the DRBG is uninstantiated first). The intent of this section is clear and valid, it just seems like more detailed advice should be given to avoid potential problems.

16. Section 9.5.1: This section requires known answer tests on the instantiate function prior to creating each instantiation unless several instantiations will be performed in quick succession using the same input parameters. At this point, the term "input parameters" is not clearly defined. If they include the entropy input, nonce and/or personalization string, then the same input parameters should never be used and this requirement would them seem a bit excessive. The term "input parameters" should be clarified to not include these values, which are required to change with each call.

17. Section 9.5.2: It is not clear in the first paragraph of this section if the requirement to perform known answer tests on the Generate function before its first use applies to the first use per instantiation or the first use ever.

18. Figure 8: This figure indicates that the Counter starts at 1, however from the textual description it looks like the counter starts at 0.

19. Section 10.1.1.2: There is no English word "preceed", suggest "prefix", as precede or proceed may be confusing. This occurs elsewhere (example: 10.1.1.3) so suggest a search to find all of them.

20. Section 10.1.1.4: We note in the design of this DRBG that in the Generate process, if in step 5 we get $H+C+reseed\_counter \bmod 2^{seedlen}$ to be a value less than about 2^{12}, then it is possible on two successive calls to Generate to get substantial substrings matching in the output. Now, the probability of this happening is negligible, so a change is not required. However, it would be nice if the DRBG did not have this property. One way to deal with it now would be to very slightly change the Hashgen process. Step 4.1 of that process could be changed to $w_i=Hash(0x04||V)$.

21. Section 10.1.2: We question the design of some aspects of this DRBG. First note at the top of page 49 it is stated that the values *V* and *Key* are the "secret values" of the internal state. This is a true statement related to the state between Generate calls. However, note that in step 4 on page 52, during the Generate call the current value of *V* is made public as part of the *returned_bits*. Thus, immediately following the completion of step 4, the only secret value is *Key*. Hence the state space of this DRBG is at most *outlen* instead of *2*outlen*. One way to deal with this reduction in state space would be to modify step 4 as follows:

```
4a. Let output_block = V.
4b. While (len (temp) < requested_number_of_bits) do:
    4.1 output_block=HMAC(Key,output_block).
    4.2 temp=temp||output_block.
```

In addition, it is not clear why *Key* and *V* are each updated twice as part of the Update process. This results in four calls to the HMAC function, when perhaps only two would be sufficient, resulting in a lack of efficiency.

22. Section 10.2.1.3.1: We note that in Step 6 of the Instantiate process, the call to Update() does not appear to provide any real advantage over simply parsing the created *seed_material* string into *Key* and *V*. Thus, we question the inclusion of this Update call here.

23. Section 10.2.1.3.2: As with the previous comment, the Update() call in Step 5 does not appear to provide any real advantage over simply parsing the *seed_material* string. Thus, we question the inclusion of this Update call here.

24. Section 10.2.1.3.2: At the top of page 59 modified instructions are given for what to do if a *personalization_string* will never be given. However, these instructions do not say what to do with the *nonce*, if provided.

25. Section 10.3.1: The Dual_EC_DRBG allows an output block length just slightly less than the size of the base field. However, the paper available at http://eprint.iacr.org/2005/324 (particularly Section 5) seems to indicate that perhaps less than half the bits of the x-coordinate of an elliptic curve point are indistinguishable from random. The results of this paper should be taken into account, or at least explained in relation to the allowed block length.

26. Section 10.3.1.1: Item 1 subitem c: From a security perspective, it cannot be worse and may be better if both P and Q are generated randomly, so this should at least be allowed.

27. Section 10.3.1.3: Again, at the top of page 69, the *nonce* has not been taken into account in this implementation note.

28. Section 10.4.2: We note that in step 8 of the Block_Cipher_df process all of the data in the *input_string* is condensed into *keylen+outlen* bits, and then these bits are then expanded into *number_of_bits_to_return* bits in step 12. This condensing and then expanding seems unfortunate. It would seem to preserve the entropy in *input_string* better if *number_of_bits_to_return* bits were produced in step 8 and then simply returned at that point without continuing on to step 12.

29. Section A.1: This section mentions FIPS 186-3, but this is not released yet.

30. Section A.2: Use MAY instead of SHOULD in the second sentence, as this is one acceptable way to avoid potentially weak points, but is probably not the best way from a security perspective, which is to generate P and Q randomly.

31. Section A.2.1: Regarding P, if a vendor wants to use randomly generated P, this should be allowed.

32. Section C.3: The last paragraph on page 93 discusses concatenating samples from an entropy source and the property of min-entropy that allows one to simply sum the min-entropy of the parts to get the min-entropy of the concatenation. However, this property only holds if the samples are not correlated. This is not made clear in the present text and should be pointed out.

33. Section C.4: It is not clear why there will be at least 256 coin flips, when the smallest security level is 112. The text does not reflect the procedure in the recent Entrust submission to ANSI X9F1 for X9.82 Part 2 on Simplified Coin Toss Entropy Assessment. Is there new information or what?

34. Section E.2: The paper mentioned in our comment on Section 10.3.1 should be taken into account here.

35. Section E.2: Remove "already sluggish" as being too negative a phrase.

36. Appendix G: This section needs a table to summarize the discussion. Also, as these methods need to be coded and tested, a column in the table could give the performance times on some common machine, as a rough comparison as to what to expect in terms of throughput. For example, something like the following:

DRBG	**Dominating Cost/Block**	**Constraints**	**Output/Second**
Hash	2 hash	2^{48} calls of 2^{19} bits	
HMAC	4 hash	2^{48} calls of 2^{19} bits	
CTR (TDEA)	1 TDEA encrypt	2^{32} calls of 2^{13} bits	
CTR (AES)	1 AES encrypt	2^{48} calls of 2^{19} bits	
Dual EC	2 EC points	2^{32} blocks	

37. G.4: The example of a request for 2 million bits requires the computational expense of at least 2 elliptic curve points seems jarring, at this example will need a lot more than 2. Suggest rewording this sentence to make the point that any call needs at least 2 computations.

Robert Zuccherato, Chief Cryptographer

Entrust

From: Werner.Schindler@bsi.bund.de

Date: Wed, 1 Feb 2006 19:03:18 +0100

Here are some notes about the draft NIST SP 800-90 "Recommendation for Random Number Generation Using Deterministic Random Bit Generators"

(December 2005).

Section 4) Terms and Definitions:

a) To "Non-Deterministic Random Bit Generator (Non-Deterministic RBG) (NRBG)":

   The term "physical source" should be specified in Section 4. Does it include human interaction and system data, for instance, or is it restricted to entropy sources that use dedicated hardware to measure the physical characteristics of a sequence of events in the real world? (cf. ISO FDIS 18031, "Random Bit Generation", Sect. 7.1)

b) To "Random Number":

   The formulation "... a value in a set that has an equal probability of being selected from the total population of possibilities ..." seems to be too strict. It does not exactly match with the definition of the term "unpredictable".

   Proposal: "equal probability" should be replaced by " ... has almost an equal probability..."

c) To "Random Bit Generator" and "Random Number Generator": These definitions seem to be too restrictive. Strictly interpreted they exclude even very small-biased physical RNGs, for instance (cf. also comment b) to Appendix D).

d) To "Security Strength": To "...that is required to break a cryptographic algorithm or system..." should e.g. be added "with known techniques".

e) To "Seed": The formulation "... and its entropy must be ..." should be replaced by "... and the entropy of the determined portion of the internal state must be ..."

Appendix D: (Normative) Constructing a Random Bit Generator (RBG) from Entropy Sources and DRBG Mechanisms:

a) D1.b uses the terms "Approved NRBG" and "Approved entropy source". What does this exactly mean? Do approved NRBGs or approved entropy sources in the meaning of this draft already exist?

b) Remark: Fig. D-1: NRBG contains the term "FULLENTROPY OUTPUT". Interpreted in a strict sense this requires a very precise mathematical model of the entropy source.

c) Assume that DRBG A (re-)seeds DRBG B. It should be pointed out in this subsection that this operation reduces the amount of entropy of DRBG A that is currently available for (re-)seeding DRBG B. (Of course, this amount of entropy will be increased by the next reseeding of DRBG A.)

From: Bill Millan millan@isrc.qut.edu.au

Date: Thu, 02 Feb 2006 10:57:40 +1000

1 Introduction

NIST is seeking comments on their draft recommendation (SP 800-90 [1]) regarding deterministic random bit generators. We note that such algorithms are different to stream ciphers in that they allow for additional sources of entropy and variation than just a secret key and a known IV. From the purpose and environment of the DRBG, it seems that what we would call a traditional stream cipher is in fact distinctly more powerful than a DRBG. We notice that NIST has created standard block ciphers, hash functions and elliptic curves, but no dedicated stream cipher is yet approved. They construct the DRBG from all three of these previously approved algorithm types. Looking closer we see than in many cases the instance creating and re-seeding operations use two or more instances of a secure block cipher or hash function, when we might think that a single instance would be sufficient. Thus opportunities should exist for designing a dedicated stream cipher that provides better performance than the currently proposed over designed systems.

This document offers detailed comments on SP 800-90. We consider overall intention, structure, design choices and their means of expression. Especially we seek to check self-consistency and correctness in comparison with existing theoretical knowledge. Some choices may have implications in future scenarios involving advances in cryptanalysis.

2 Main Comments

The bullet points contain the page number.

- (p.10) The scope of the document does not include the device health tests. Such BIST (built in self test) techniques tend to use an LFSR with a primitive feedback polynomial. Perhaps if such a component were available anyway, it could be re-used as a component in a dedicated stream cipher? For a more useful structure, the feedback polynomial could be made a variable. Having access to a dedicated hardware LFSR inside an otherwise software device could provide a better speed security trade-off than seems to be currently provided.

- (p.11-p.17) The terms and definitions are generally well done. However, the entry for block cipher does not even mention that the mapping, for every key, must be a permutation.
- (p.19, p.28) The personalization string is an attempt to make each instance of the same DRBG unique. It does not need to contain identification information and it could be misused to demand such data. This is a problem for multi-purpose devices now and for any device in the future after a security compromise then threatens more than just the users keys, but all of the users private and identity information.

(p.22) Security strength is made variable and separated from the key length. This is a good idea, but it could be misused. An underlying fault here is the assumption that an increased key size provides increased security, like a sliding scale. But security should be absolute. It should be on or off. If k is the security level and it is unbroken, then $2k$ security level is not more secure, since it is equally well not attackable. Making the security level a sliding scale puts pressure on the user to specify a suitable security level for their application. But users should not have insecure options available at all, since some will inevitably make a poor choice, or a corrupted device will make deliberately poor choices.

(p.23) There exists an option to create the entropy source within the DRBG boundary. However, this allows a modified or corrupted device to create low entropy data and lie about its quality. Such a subverted device happily interacts with non-corrupted devices and yet it is easier to break and it may reduce the security of devices which depend on the entropy of its output.

(p.26) Why is the minimum entropy of the nonce given by $security\ strength/2$, when theory suggests it could be useful to have $security\ strength * 2$ bits of uncertainty in the nonce to prevent time-memory trade-off (TMTO) attacks that operate faster than a security strength level exhaustive search?

(p.27) There are good rules overall for initialising, re-seeding and protecting the state data from other processes.

(p.28) There is danger in revealing personal info in the personalization string when the system is broken in the future. Using private info is specifically recommended here, where other ways of creating this string would have the same effect on security, without bringing the security of vital private info down to the security of this device. Asking the user to make a critical decision regarding the security of personal info and other devices is asking for trouble and allowing some to make poor choices. Users should not be asked to have specialist knowledge and they should not be offered possibly insecure options.

(p.29) Backtracking resistance requires update be one way, therefore it must use pseudorandom function (PRF) not pseudorandom permutation (PRP) in dedicated designs. Thus the state must be 4* security level to also account for TMTO attacks.

(p.29) Prediction resistance is provided by frequent re-keying. However the Prediction Resistance flag does not guarantee this property, it just asks for it. Perhaps some re-keying verification is warranted? Also, some implementations do not use this flag at all, so check that the interpretations for it being missing are not catastrophic.

(p.32) Why has the importance of consuming applications checking the status of the DRBG instantiation been downgraded to "Should" rather than set at "Shall"? Faulty instantiations could be victims of fault attacks and should not be used.

(p.39) Exactly how many instantiations can be consecutively created with the same parameters without checking the known-answer tests? Too many invites the induced fault attack.

(p.40) The capability to test on demand could be exploited by a fault attack, yet they say implementations "Should" have this feature.

(p.41) Testing that the uninstantiate function really has zeroised the state could be expressed in the following way: that the residual traces of previous memory (and these do exist for a long time afterward, see recent advances in computer forensics) must be below some threshold level (perhaps linked to the technology of readers). Beware: a more sensitive reading device may be able to see the previously deleted memory traces, even if it has been declared all-zero.

(p.41) Why is it a *catastrophic* error when the entropy source has any error? The stream cipher primitive has no external entropy so it is a much stronger algorithm than what they are asking for in a DRBG with non-zero external entropy source. Also, what are the consequences of the entropy source being revealed to the attacker? Some DRBG will be broken or at least collapse to the same as a stream cipher. However, the recommendation does not state that the entropy source should have no other client applications. Including the entropy source within the DRBG boundary solves this but opens the way for a fault attack that makes the device use lower entropy and lie about its quality to queries.

(p.44) Notice they use the hash function in 3 places, including twice in series. Can this be improved (using fewer of them) or could reduced round hash functions suffice?

(p.45) They prepend a constant to the SEED before hashing. There are at least 4 places where the constant byte is prepended, with different constants for each use. However, this is perhaps theoretically contradictory with the stated requirement that all these hash functions should be the same. If this *is* what they want, perhaps this structure should promote the adoption of a new statistical test for hash function randomness, when comparing the inputs spaces with different first byte values?

(p.47) (*) The leftmost bits are not the most recently generated from the iterated process, according to my interpretation of line 4.2 in HASHGEN. This observation applies many other places in the document. If they took all the most recently generated data then that would give some resistance against any future break of the single hash function.

(p.49) Why should the last stage be omitted when the provided data = NULL when it is well defined anyway? Does this mean it is OK for no provided data applications to have less security? If it is not less secure without the extra stage, then why do it at all?

(p.52) See * again.

(p.53-p.54) Key length of 3-TDEA is given as 168, but isn't there an attack with complexity 2^{112}. Should blocksize (outlen) be judged during security level checks? [Answer implicit on page 54: it affects the reseed interval maximum]. Should this connection be made more explicit? There is an incompatible criteria at work here: the reseed interval is given as 2^{48} for the AES-128. Why is it not 2^{64}, which would be consistent with using 2^{32} for Triple-DES. Either one

2^{64}, which would be consistent with using 2^{32} for Triple-DES. Either one security margin is too high or the other is too low, but which is it? Note the total data output is the maximum output size * the reseed interval. For security we want the total data less than $2^{\frac{blocksize}{2}}$ in all cases. This needs to be considered more concretely and the security margins made uniform. How secure is TDES when it produces $2^{13+32} = 2^{45}$ bits?

(p.55) They recommend a fixed-key block cipher simply iterated plaintext sequence = counter output.

(p.58) See * again.

(p.62) See * again.

(p.63) See * again.

(p.72-73) See * again, twice.

(p.76) If health tests stay within the boundary, then a modified device could lie about the results. In contrast, if health testing is done from the outside, as a separate entity, then a single device modification will not be able to hide the true health result.

(p.77) The health test data is never output. The data is fixed before hand, so the output is not random: it has no entropy. This protects against stupid implementors. *However*, the health test results cannot now be verified! So we need a health test for the health test, and a test for that test...and so on...eventually some test must be able to be verified. How to avoid this infinite regress?

(p.81) To avoid weak points the fixed ones "Should" be used, but this is not strong enough and still allows some evil (deliberately weakened) implementations to use weak points. What is wrong with the fixed points "Shall" be used? Most implementations will use them...so why not ensure that all do?

(p.87-88) There is a trade-off between skew and irregular timing.

(p.91) Is the health test for the entropy source the same as or different compared with the DRBG health tests?

93, p.95) When the entropy source is a higher-level DRBG, the max entropy for longer strings is upper bounded by DRBG entropy input.

(p.94) If they are going to explain coin tossing why not explain dice rolling or selecting from a deck of cards? There exist four-sided and eight-sided dice which can produce 2 or 3 uniformly distributed bits each time.

(p.97) Linking entropy input to protocol data might give attackers a way to reduce the effective entropy. (Remember problems with the Netscape browser? That system used local timestamps which were a source of weakness in the initialisation.)

(p.101) There might be a problem with the reasoning, since uniform strings imply uniform substrings, which is in contradiction with what is written. The last sentence is especially vague!

(p.103-p.111) Why are the maximum limits encoded as fixed constants? This seems inflexible.

(p.119) Fixing EC parameters to particular security strengths does not allow for future improvements in cryptanalysis. Thus this standard could become compromised or obsolete.

(p.125) Statements in support of re-using available hash or block cipher ignores the possibility of a single break destroying the security of the entire device. Why not use multiple mechanisms to provide some robustness?

(p.128) They assume best attack on ECC size m has $2^{\frac{m}{2}}$ complexity. This ignores future cryptanalysis. Compare with the Lenstra et.al. paper, which allowed for the possibility for future ECC cryptanalysis. They ignore this possibility by hiding behind claims the structure is based on a hard problem in number theory. It is only *believed* to be hard, and omitting the reference to belief could be construed as misleading the users about their risk. Note that the strength of the hash functions and block ciphers is similarly only a belief.

References

1. NIST, SP 800-90, "Recommendation for Random Number Generation Using deterministic Random Bit Generators", Draft seeking comment, available on-line, December 2005.

From: "Kristin Lauter" klauter@microsoft.com

Date: Wed, 1 Feb 2006 18:56:13 -0800
We are pleased to submit the following comments on the December 2005 Draft of SP 800-90 on behalf of Microsoft Corporation, Redmond, WA.

1. General Comments

Our first comment concerns the relationship between SP 800-90 and the related ANSI X9.82 standard currently under development. While we appreciate NIST's desire for a FIPS standard for DRBGs, we would strongly advise NIST to wait on finalizing SP 800-90 until the ANSI X9.82 committee has completed its work. We believe that any significant arbitrary differences between ANSI X9.82 and SP 800-90 could be detrimental to the widespread adoption of both future standards by implementers. We recognize that, for a variety of reasons one specification may need to be a superset of the other; if that case arises we believe that ANSI X9.82 should be the subset.

Additionally, in general we favor parameter generation that can be performed and/or verified independently through an open and transparent process.

2. Choice of Algorithms Included in SP 800-90

It is not clear to us from reading SP 800-90 why FIPS 186-2 style random number generator engines are excluded from this recommendation. Assuming that there is no known weakness in the general purpose FIPS 186-2 PRNG, it would appear to be prudent to enable support in SP 800-90 for FIPS 180-2 hash functions in FIPS 186-2 random number generation engines. (It's possible that this support more correctly belongs in FIPS 186-3 over SP 800-90, but we believe it should be addressed in one of those two locations.)

If NIST desires a number theoretic security reduction for the PRNGs in SP 800-90, then we would recommend that NIST adopt one or more of the following possible options: RSA PRBG, Blum-Blum-Shub, or the following cryptographically secure PRBG based on walks on expander graphs:

Cryptographically secure PRBG based on walks on expander graphs.

Input: a random integer x_0 to be the *seed*

Output: a pseudorandom bit sequence $z_1, z_2, \ldots, z_n$

Step 1. The set-up is to generate secretly and randomly an expander graph and a starting vertex of the graph (for example from the leading bits of x_0). For example the graph could be an expander graph with optimal expansion properties such as 1) the Lubotzky-Philips-Sarnak graphs or 2) the Pizer graph of supersingular elliptic curves over the field of p^2 elements with k-isogenies, where p is a large prime of cryptographic size and k is a small prime such as 3.

Step 2. Use the first bit(s) of the seed to choose a path from the first vertex to a next vertex. Set z_1 to be the least significant bit of the next vertex. In general, use the i^{th} bit or chunk of bits of the seed to choose a path from the $(i-1)^{th}$ vertex to the i^{th} vertex. Set z_i to be the least significant bit of the i^{th} vertex.

Note that to improve efficiency, more bits can be extracted from each vertex.

More details on the graphs suggested in Step 1 of this approach are available in the related paper "Cryptographic hash functions from expander graphs" by Charles, Goren and Lauter, available on-line at http://eprint.iacr.org/2006/021. An analysis of the properties of the resulting PRBGs will be forthcoming in a document in preparation by those same co-authors.

3. Detailed comments and requests

1. We would like to request that a set of "Known Answer Tests" be included in SP 800-90 (perhaps in an Appendix). We think such tests would be especially helpful in this case for determining implementation correctness.
2. Page 29, Prediction resistance sub-item (b): If one were to interpret this in terms of the definition of internal_state given in section 7.1, then the statement given here is incorrect. Given the complete internal state one can easily predict the next state of the DBRG. A corrected version could read: "…given the knowledge of the output of the internal state."
3. Page 40, Section 9.5.3: It is very difficult to test if a source has a certain min-entropy. Since min-entropy is a property of the distribution, not of particular values. Could the draft propose one or more specific tests for the re-seed function?
4. Page 42, Table 2: The draft currenly lists SHA-1 along with the SHA-2 family of hash functions. Given the recent theoretical results on finding SHA-1 collisions we believe that the use of SHA-1 in SP 800-90 should at least be discouraged if not prohibited completely. Ideally, SP 800-90 would include a statement deprecating use of SHA-1 for PRBGs.

5. Page 48, Section 10.1.2: This section should include a list of approved keyed hash functions for HMAC_PRBG (or a pointer to another section of the document where such a list occurs).
6. Page 53, Section 10.2 (DBRGs Based on Block Ciphers): Note that using the described mechanism for producing a block of pseudorandom bits by encryption of V, V+1,.., V+k means that one should not use any encryption method that has a related message attack. This section should include a note to this effect.
7. Page 64, Section 10.3.1: The function \phi is shown in Figure 13 and used in equations on Page 65, but \phi is not defined until Page 69. A short sentence describing the purpose of \phi near the beginning of Section 10.3 should suffice.
8. Page 64, Section 10.3: We note that the Dual-EC method described in this section can also be implemented on finite fields by taking two numbers g and h in a finite field, and letting $s_i = g^{s_{i-1}}$ and $r_i = h^{s_i}$, where r_i being the output of the random number generator.
9. Page 64, Section 10.3: Since P and Q are n-torsion points, there is a possibility that Q = rP for some r. In this case, the system reduces to having a secret r and outputting the x-coordinate of $s_i r*P$. This situation may be more or less secure than when we have P and Q defined over different pieces of the n-torsion and hence being linearly independent. We believe that the case when Q = rP is actually more secure; since, in the other case we must have that the elliptic curve has low embedding degree, and hence, the discrete log will be reduced to that over finite fields. In particular, this property should be examined for the curves given in the Appendix.
10. Page 74: Auxiliary Functions: 10.4.3. Block Cipher Hash Function (Block_Cipher_Hash) The length of data_to_hash (input data) is required to be an integral multiple of the block cipher's block size (outlen). This requirement depends on the checks in Block_Cipher_df function. The Block_Cipher_Hash function would stand on its own more independently if this requirement is removed, and a conventional padding scheme is instituted instead. A potential and a commonly-implemented padding scheme is defined in PKCS#5 and in RFC 1423.