1. Overview: Constructing a DRBG from Algorithms and Entropy Sources

The rest of this document is primarily concerned with the algorithms for generating pseudorandom outputs and how they are to be implemented. The source of seeding material for the DRBGs is mostly left to the designer to get right. In this appendix, we briefly describe how this can be done.

2. Internally Seeded DRBG

The ideal situation for a full DRBG is to have ready access to some entropy source. The entropy source provides bit strings along with a promise about how much entropy the bit strings have. An example of an entropy source would be a ring oscillator sampled one hundred times per second, where extensive analysis had been done to ensure that each sequence of 100 bits sampled had at least 80 bits of entropy. Any DRBG with an internal source of entropy can be used to access its underlying entropy source: if the source DRBG promises *k* bits of security, then each new request for *k* or more bits of output with prediction resistance from the source DRBG can be assumed to contain *k* bits of min-entropy.

When the DRBG has an internal source of entropy, reseeding and instantiation can be done on demand, requests for prediction resistance can be honored, and when a DRBG hits a required reseed interval after having generated too many outputs, it can simply reseed.

An internally seeded DRBG may use a seedfile, as described below, but does not require one.

3. Externally Seeded DRBG

Many implementations of DRBGs will not have access to an entropy source. We call these externally seeded DRBGs. An externally seeded DRBG has the following requirements:

- The DRBG must be instantiated at a time when the DRBG has access to some entropy source, and the entropy provided for instantiation must be provided over a secure (private and authentic) channel. In some applications, the entropy source is only available during manufacture or device setup; in others, it is occasionally available (e.g., when a user is moving the mouse around on a laptop).
- The DRBG must maintain its working state for as long as the DRBG may be called upon to generate outputs. This typically requires some kind of persistent memory to avoid losing state during power down. This may be maintained directly as the state of the DRBG, or maintained in a seedfile, as described below.

Over time, an externally seeded DRBG may be able to accumulate entropy from additional inputs provided by the user or consuming application. For this reason, the DRBG implementation should accept additional input whenever possible.

Implementations that have values which may have entropy, such as timestamps, nonces from protocol runs, etc., should provide them to the DRBG as additional inputs.

4. Using a Persistent DRBG as a Seedfile

A seedfile is persistent storage kept for a DRBG. It is used both to protect against silent failure of its entropy source, and also to allow externally seeded DRBGs to instantiate itself on power up and save all the entropy in its state back to the seedfile whenever necessary. Seedfiles are used and described in many cryptographic PRNGs, including /dev/random and Yarrow-160.

In X9.82, a seedfile is simply a DRBG instance whose working state is stored in some kind of persistent storage. Let SEEDFILE be the DRBG which is being used as a seedfile, and whose working state is stored in persistent storage. The following explains how the seedfile is used to support a DRBG (CURRENT) whose state is stored in volatile storage, but which may accumulate entropy over time from additional inputs:

- SEEDFILE is instantiated from some entropy source (possibly another RBG) when it is available, such as during manufacturing or device setup.
- At power up, CURRENT is instantiated. This is done by requesting a seed from SEEDFILE's generate routine and using that seed to instantiate CURRENT. Any additional input which is available from the application should be provided in the personalization string during CURRENT's instantiation.
- During operation, application data which might contain some entropy should be stored up, and periodically used as additional input in one-byte generate requests to SEEDFILE. The resulting one-byte outputs are discarded.
- At power down, CURRENT generates a *k* bit output. This output is used as additional input, along with any other available application data which might have some entropy, in a one-byte generate request to SEEDFILE. The one byte output is discarded.
- If the application rarely or never has a power down, then a *k*-bit value from CURRENT should periodically (e.g., once a day) be generated and used as additional input in a one-byte generate request to SEEDFILE, and the resulting output byte should be discarded.

5. Seeding Many DRBGs from One DRBG

Some applications may benefit from using different DRBGs for different applications. However, this must be done with some care, to avoid introducing new weaknesses. In order to use multiple different DRBGs for different consuming applications, the following steps shall be done:

- The parent DRBG is first instantiated with as much entropy as is available.
- Each child DRBG is instantiated with seed material acquired from a generate request to the parent DRBG.

Dear Dr. Schneier,

In light of your November 14, 2007 Wired commentary (http://www.wired.com/politics/security/commentary/securitymatters/2007/11/securitymatters_1115), we would like to take the opportunity to provide a few clarifications on NIST Special Publication 800-90.

NIST would never knowingly support the inclusion of an algorithm with secret features such as a "back door" in its standards. We do not think there is an intentionally placed back door or any other secret feature in the Dual_EC_DRBG pseudorandom-number generator.

If we discovered a back door in any algorithm in a NIST standard, we would withdraw the algorithm as soon as practical. We have no evidence that someone knows the existence of the "secret numbers" that Dan Shumow and Niels Ferguson have shown would provide advance information about the pseudorandom numbers that Dual_EC_DRBG would generate. Therefore, we have no plans to withdraw the algorithm at this time.

As you note, the Dual_EC_DRBG algorithm has also been approved as an ANSI international standard. The algorithm was vetted through the ANSI X9 subcommittee, of which Neils Ferguson (one of authors of the paper that claims a back door) is[EBB1] a participant. As Drs. Shumow and Ferguson state in their presentation, they do not believe that NIST would have intentionally created a back door in Dual_EC_DRBG, and they state that even the algorithm's designer may not have been aware of having potentially created such a feature.

It is also worth noting that no one is required to use Dual_EC_DRBG or any other algorithm based on its appearance in NIST Special Publication 800-90. Moreover, as you point out in your column, Appendix A of SP 800-90 gives users the information that is needed to generate alternative values which should preclude any chance of the secret trap door in the scenario that Shumow and Ferguson have presented.

NIST special publications, including this one, undergo a rigorous review process, including a public comment period. We take all comments on our publications very seriously and regularly update topics in our special publications. We appreciate the opportunity to comment on this standard[EBB2].

Sincerely,