Deterministic Random Bit Generators NIST techniques comparison

Tomas Bedoya
Daniel Andres Jaimes Cardenas
Eder Leandro Carbonero Baquero
Universidad de los Andes, Colombia

***Abstract*—Aquí va el resumen del artículo, una breve descripción del problema, la metodología y los resultados.**

I. Introduction

II. General defintions

With the aim of providing a broader understanding of the various concepts addressed in this document, the following are definitions of some of the acronyms that will be mentioned throughout the text.

Acronym	**Meaning**
CTR_DRBG	Counter mode Deterministic Random Bit Generator
NIST	National Institute of Standards and Technology
AES	Advanced Encryption Standard
RBG	Random Bit Generator
DRBG	Deterministic Random Bit Generator
PRNG	Pseudorandom Number Generator

Table I
Acronyms definitions.

III. methodologies

IV. Methology

A. Random Number Generation

For this exercise, three different forms of random number generation were performed. The first was through the *CTR_DRBG* algorithm, which used random keys generated by the `random` function of the Python programming language version 3.9 as seeds and was executed on a MacBook Pro M3 Pro computer. This detail may lead to different results if the experiment is replicated. However, we assume that Python's `random` function generates a uniform randomness distribution, simulating an entropic system for generating random bits.

For the second option, a sequential range of seeds was chosen, which were formatted in 32 bits to ensure compatibility with AES encryption. The sequence started at 1 and increased by one until reaching 50 million seeds. These seeds were then used to execute the *CTR_DRBG* algorithm, which generates the pseudo-random bit values.

The third approach made use of the Beacon system. In this case, the official NIST API was consumed to retrieve the last 15,600 random bits, generated from 512-bit values, obtained from the historical records maintained by the system.

V. techniques

VI. CTR_DRBG Counter mode Deterministic Random Bit Generator

CTR_DRBG (Counter mode Deterministic Random Bit Generator) is a standardized method for constructing a deterministic random bit generator (PRNG) using a block cipher operating in counter (CTR) mode. This technique is defined in NIST Special Publication 800-90A, titled *"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"*. Essentially, CTR_DRBG transforms a secure symmetric cipher—such as AES—into a cryptographically strong source of pseudorandom bits. The counter mode ensures that each generated block is unique by systematically incrementing a counter value for each new data request, thus preventing the repetition of output sequences under the same key and seed. This approach is widely used in cryptographic applications requiring high security and reliability in random data generation, such as key generation, initialization vectors, and session tokens.

How It Works

The internal state of the CTR_DRBG consists of two components:

- **V**: A state vector that acts as a counter.
- **Key**: A symmetric encryption key (commonly 128 or 256 bits).

To generate random output, the algorithm encrypts sequential values derived from `V` using the current `Key`. For each block of output, the counter `V` is incremented.

A secure seed—typically composed of entropy input, a nonce, and optional personalization string—is required to properly initialize the internal state.

CTR_DRBG utilizes an approved block cipher algorithm in counter (CTR) mode, as described in Barker and Kelsey [2015]. Unlike the standard CTR mode, it allows the counter field to occupy only a portion of the cipher input block, as specified in Dworkin [2007].

For context, the block size depends on the cipher: 64 bits for TDEA and 128 bits for AES. The same block cipher algorithm and key length are used consistently throughout all encryption operations in the DRBG. These parameters must meet or exceed the required security level of the intended application.

The construction of *CTR_DRBG* revolves around an internal update function, `CTR_DRBG_Update`, illustrated in Figure 1. This function is invoked during instantiation, reseeding, and random bit generation to update the internal state using newly provided entropy or additional input. It also ensures that the internal state changes appropriately after every generation step.

Figure 2 outlines the full operation of *CTR_DRBG* in three stages: instantiation, generation, and reseeding.

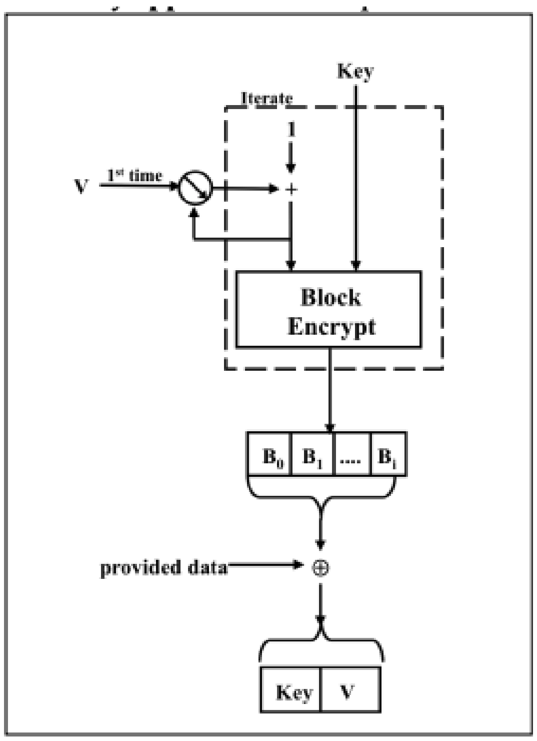

Figure 11: CTR_DRBG Update Function

Figure 1. CTR_DRBG Update function as shown in Barker and Kelsey [2015].

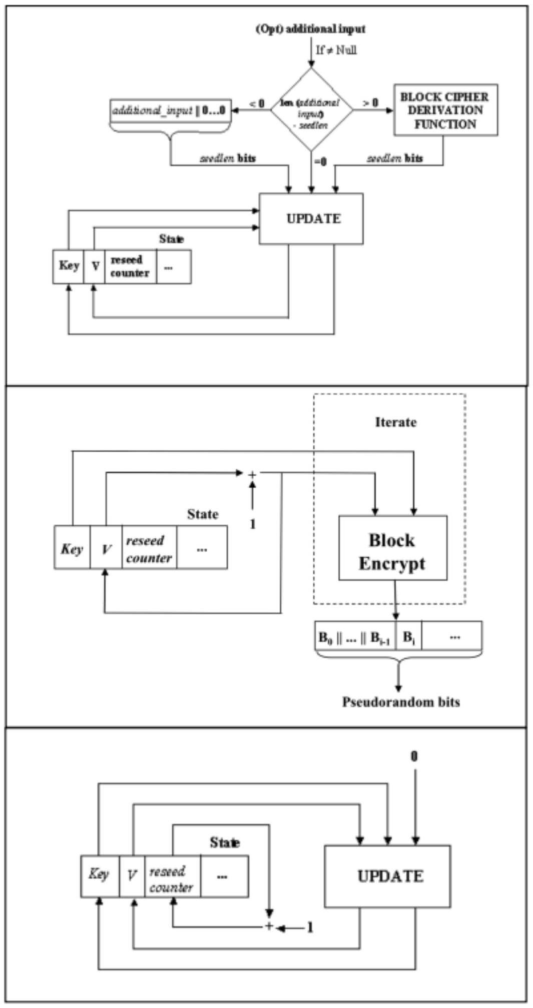

Figure 12: CTR-DRBG

Figure 2. The three stages of CTR_DRBG operation as illustrated in Barker and Kelsey [2015].

To provide an overview of what can be found in the source article, we can say that *CTR_DRBG* is a deterministic random bit generator that uses an approved symmetric encryption algorithm, such as AES, in counter (CTR) mode to produce secure random bits. Its internal state consists of a counter vector called `V` and a secret key, `Key`, which are updated through an internal function named `CTR_DRBG_Update` during instantiation, generation, and reseeding, thus ensuring the cryptographic security of the generator. At each step, `V` is encrypted with `Key` and incremented to generate non-repeating pseudorandom blocks. It requires a secure initial seed with sufficient entropy to avoid predictability and can be periodically reseeded to maintain long-term security. This flexible design allows for the use of variable block sizes and key lengths depending on the algorithm (e.g., 128 bits for AES) and meets robust cryptographic standards, making it reliable for applications that require secure random number generation.

A. *AES - Advanced Encryption Standard*

AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used across various applications for securing data. It was established by the National Institute of Standards and Technology (NIST) in 2001 as a replacement for the older Data Encryption Standard (DES). AES operates on fixed-size blocks of data (128 bits) and supports key sizes of 128, 192, or 256 bits, providing a high level of security against brute-force attacks. The algorithm employs a series of transformations, including substitution, permutation, and mixing, to encrypt plaintext into ciphertext and vice versa. Its efficiency and robustness have made it the de facto standard for encryption in many cryptographic protocols and systems.

VII. Results

A. *Histogram analysis*

B. *Distribution of Random Bits Generated with Python Random Seeds*

The distribution results of each technique are separated by the method used to obtain the seed versus the technique used to distill and obtain the random big number, so we can observe different results for each case.

To view the distribution of values, distribution graphs were used, either standard or uniform, depending on the input data. In our experiments, uniform distributions were shown. 8-bit segments were taken, dividing the entire input data set into byte blocks, obtaining bytes from 0 to 255. This range of possibilities gives us a visual of how the results are distributed.

We begin with the first technique, which was the generation of 6.25 million 32-byte seeds that were used to generate 6.25 million random bytes, or 50 million random bits, using the *CTR_DRBG* technique. The graph we see below leads us to the conclusion that there is a uniform distribution of the results without pronounced peaks, which are a symptom of an effective system in generating random bits.

C. *Distribution of Random Bits Generated with Sequential Seeds*

For the following test, sequential seeds were used starting at 1 and reaching 6.5 million in the set of natural numbers. This number was converted to binary and turned into a 32-bit number, then it was used as a seed to generate the distribution with the *CTR_DRBG* method, showing a uniform distribution without much variation. Unexpectedly, it is seen that the distribution behaves very similar to obtaining reliable seeds with true entropy. Below is the graph.

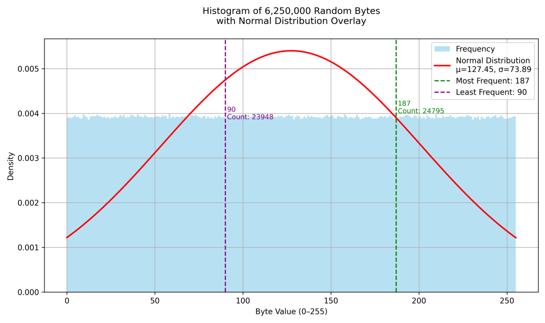

Figure 3. Random bits *CTR_DRBG* seed generation with Python random function.

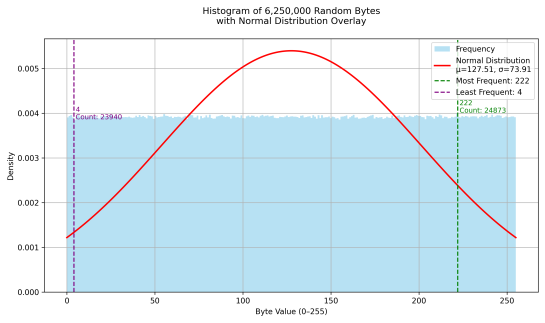

Figure 4. Sequential seed from 1 to 6.5M using *CTR_DRBG*.

VIII. Conclusions

References

Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators (revised). Technical Report Special Publication 800-90A Revision 1, National Institute of Standards and Technology (NIST), June 2015. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical Report Special Publication 800-38D, National Institute of Standards and Technology (NIST), November 2007. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.