Benchmarking Deterministic Random Bit Generators in Rust

Edoardo Simonetti Spallotta
Student ID: 2077561
Sapienza University of Rome / ECSAI
simonettispallotta.2077561@studenti.uniroma1.it

Abstract

This report documents three deterministic random bit generators (DRBGs) implemented in Rust: a ChaCha20 construction built on `rand_chacha`, an AES-256 counter-mode DRBG that expands keying material with BLAKE3, and a keyed BLAKE3 extendable-output DRBG. Each generator emits packed binary strings for lengths between 10^4 and 10^7 bits, while the benchmarking harness measures latency, space consumption, and the proportion of generated ones. Experiments conducted on Apple Silicon (Darwin 24.6, arm64) show that the BLAKE3 XOF path is the fastest at 1.98 ms for 10^7 bits, ChaCha20 follows closely at 2.20 ms, and the AES-256-CTR design trades speed for a conservative block-cipher core at 7.47 ms. All generators stay within $\pm 0.8\%$ bias at 10^4 bits and converge to $\pm 0.03\%$ by 10^7 bits, demonstrating balanced output alongside efficient memory usage from bit packing.

1 Introduction

Cryptographically secure pseudorandom number generators underpin key generation, nonces, and masking schemes. When only binary strings are needed, a deterministic random bit generator (DRBG) with a compact output representation simplifies downstream consumers. The goal of this work is to build and compare three modern DRBGs in Rust, focusing on execution time, memory footprint, and bit balance over output sizes ranging from 10^4 to 10^7 bits.

2 Generator Designs

2.1 Shared Packing and Seeding

All generators accept a byte slice seed. A BLAKE3-derived stream labelled per generator derives either a 256-bit key plus counter or a standalone keyed hash key, ensuring domain separation. Output bits are packed into byte vectors so that a n-bit string occupies $\lceil n/8 \rceil$ bytes.

2.2 ChaCha20 DRBG

The ChaCha20 variant relies on `rand_chacha`'s `ChaCha20Rng`. The derived 256-bit seed instantiates the RNG, and bytes from the keystream are truncated to the requested bit length. This path benefits from well-reviewed library code and constant-time primitives.

2.3 AES-256-CTR DRBG

The AES construction uses the `aes` and `ctr` crates. BLAKE3 expands the seed into a 256-bit AES key and a 128-bit initial counter. Each call seeds a fresh CTR instance, and the internal counter is advanced by the number of AES blocks consumed, mirroring SP 800-90 AES-CTR DRBG behaviour without pulling in heavy dependencies.

2.4 BLAKE3 XOF DRBG

A keyed BLAKE3 hasher with an incrementing 64-bit counter feeds the extendable-output reader. This yields a compact code path with minimal mutable state and no reliance on block-cipher operations, making it a strong baseline for high-throughput bit generation.

2.5 Code Excerpt

Listing 1: Core DRBG contract and packed bit counting

```
#[allow(dead_code)]
pub trait Drbg {
    fn name(&self) -> &'static str;
    fn reseed(&mut self, seed: &[u8]);
    fn generate_bits(&mut self, bits: usize) ->
        BitString;
}

#[derive(Clone)]
pub struct BitString {
    pub bits: usize,
    pub bytes: Vec<u8>,
}

impl BitString {
    pub fn count_bits(&self) -> (u64, u64) {
        let full = self.bits / 8;
        let mut ones = 0u64;
        for byte in self.bytes.iter().take(full)
            {
            ones += byte.count_ones() as u64;
        }
        let rem = self.bits % 8;
        if rem > 0 {
            let byte = self.bytes[full];
            for i in 0..rem {
                if ((byte >> (7 - i)) & 1) == 1 {
                    ones += 1;
                }
            }
        }
```

```
        (self.bits as u64 - ones, ones)
    }
}
```

3 Methodology

Benchmarks run via `cargo run --release` on Apple Silicon (Darwin 24.6, arm64). Four output sizes are covered: 10^4, 10^5, 10^6, and 10^7 bits. For each generator-length pair, the harness records wall-clock time using `std::time::Instant`, the packed byte length, and counts of ones and zeros. Outputs are written to `results/metrics.csv`, and Plotters renders visual summaries in `results/plots/`.

4 Results

4.1 Timing

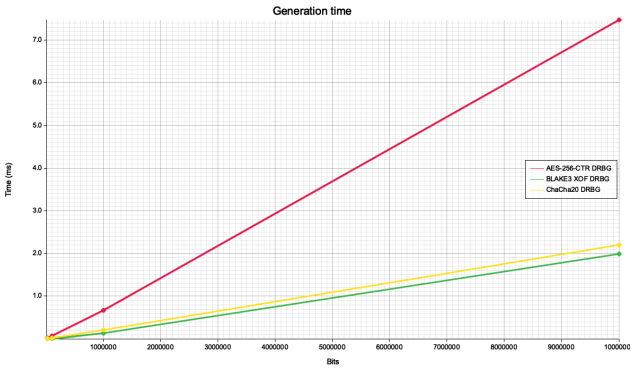

Figure 1: Generation time versus output length.

Timing scales linearly with output size. At 10^7 bits, the BLAKE3 XOF DRBG completes in 1.98 ms, the ChaCha20 path in 2.20 ms, and the AES-256-CTR variant in 7.47 ms. Even at 10^4 bits, AES is roughly three times slower than the hash-based generator because of block-cipher setup overhead.

4.2 Space

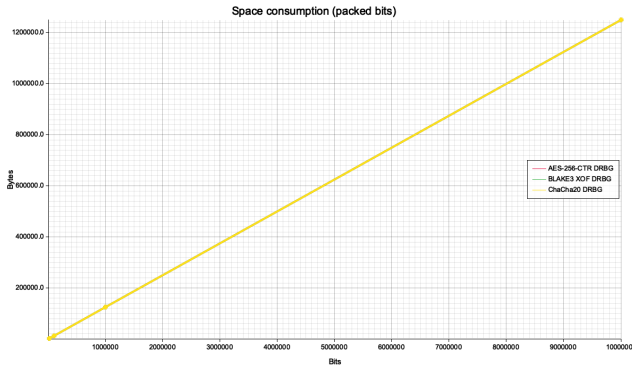

Figure 2: Packed bytes required per output length.

Space grows predictably with the packed representation: 1,250 bytes at 10^4 bits, 125,000 bytes at 10^6 bits, and 1.25 MB at 10^7 bits. All generators share this footprint because bits are stored, not streamed to disk.

4.3 Bit Balance

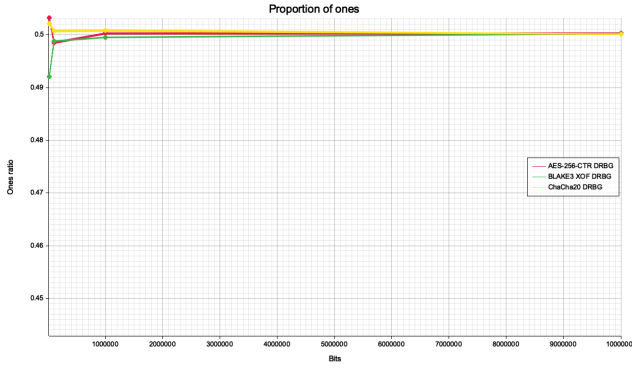

Figure 3: Proportion of ones across output lengths.

Bias remains low. The worst case occurs at 10^4 bits, where the BLAKE3 generator yields 0.492 ones ratio (a 0.8% deviation). By 10^7 bits all generators remain within $\pm 0.033\%$ of the ideal 0.5 ratio, indicating healthy diffusion.

5 Discussion

The results highlight complementary strengths. The BLAKE3 XOF DRBG delivers the best latency thanks to its hash-centric design and negligible per-call setup. ChaCha20 is only slightly slower while retaining stream-cipher familiarity and relying on well-audited crates. AES-256-CTR trails because of heavier block rounds and counter management, yet it offers a conservative choice when AES hardware acceleration or FIPS-aligned designs are required.

All three generators keep storage predictable via bit packing and maintain low bias across the evaluated lengths.

6 Source Layout and Reproducibility

The repository root contains `Cargo.toml` and `src/`. `src/drbg.rs` houses the three generators and shared helpers, while `src/main.rs` orchestrates the benchmarks and plotting. Running `cargo run --release` regenerates `results/metrics.csv` alongside refreshed figures in `results/plots/`. Adjusting the `TARGET_LENGTHS` constant in `src/main.rs` explores additional sizes.

7 Conclusion

Three Rust DRBGs—ChaCha20, AES-256-CTR, and BLAKE3 XOF—were implemented and benchmarked for binary string generation. The BLAKE3 design provides the lowest latency, ChaCha20 offers a balanced middle ground, and AES-256-CTR supplies a conservative option with predictable overhead. All maintain tight bit balance and compact storage. The provided harness and figures simplify future evaluations over alternative seeds or output lengths.