

# fn sample\_bernoulli\_rational

Michael Shoemate

May 27, 2025

This proof resides in “**contrib**” because it has not completed the vetting process.

**Warning 1** (Code is not constant-time). `sample_bernoulli_rational` takes in an optional `trials` parameter to denote the number of trials to run. The current implementation does not guard against other types of timing side-channels that can break differential privacy, e.g., non-constant time code execution due to branching.

## PR History

- [Pull Request #473](#)

This document proves that the implementations of `sample_bernoulli_rational` in `mod.rs` at commit [f5bb719](#) (outdated<sup>1</sup>) satisfies its proof definition.

At a high level, `sample_bernoulli` considers the binary expansion of `prob` into an infinite sequence `a_i`, like so:  $\text{prob} = \sum_{i=0}^{\infty} \frac{a_i}{2^{i+1}}$ . The algorithm samples  $I \sim \text{Geom}(0.5)$  using an internal function `sample_geometric_buffer`, then returns `a_I`.

## 0.1 Hoare Triple

### Preconditions

- User-specified types:
  - Variable `prob` must be of type `T`
  - Variable `constant_time` must be of type `bool`
  - Type `T` has trait `Float`. `Float` implies there exists an associated type `T::Bits` (defined in `FloatBits`) that captures the underlying bit representation of `T`.
  - Type `T::Bits` has traits `PartialOrd` and `ExactIntCast<usize>`
  - Type `usize` has trait `ExactIntCast<T::Bits>`

### Pseudocode

```
1 # returns a single bit with some probability of success
2 def sample_bernoulli_rational(prob: RBig, trials: Optional[int]) -> bool:
3     number, denom = prob.into_parts()
4     return number > UBig.sample_uniform_int_below(denom, trials)
```

<sup>1</sup>See new changes with `git diff f5bb719..f0b74b0 rust/src/traits/samplers/bernoulli/mod.rs`

## Postcondition

**Definition 0.1.** For any setting of the input parameters **prob** of type  $\mathbb{T}$  restricted to  $[0, 1]$ , and optionally **trials** of type `usize`, `sample_bernoulli_rational` either

- raises an exception if there is a lack of system entropy or if **trials** is set and it runs more than **trials** times, or
- returns **out** where **out** is  $\top$  with probability **prob**, otherwise  $\perp$ .

If **trials** is set, the implementation's runtime is constant.

*Proof.* An integer sample is taken uniformly at random from  $[0, \text{denom})$ , where **denom** is the denominator of **prob**. The implementation then returns  $\top$  if the sample is less than the numerator of **prob**, and  $\perp$  otherwise. Since only at most **numer** outcomes of  $\top$  are possible, out of **denom** possible outcomes, the implementation returns  $\top$  with probability **prob**.

The implementation runs in constant-time if **trials** is set.  $\square$