

# fn sample\_bernoulli\_float

Vicki Xu, Hanwen Zhang, Zachary Ratliff

September 14, 2025

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

**Warning 1** (Code is not constant-time). `sample_bernoulli_float` takes in a boolean `constant_time` parameter to protect against timing attacks on the Bernoulli sampling procedure. However, 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This document proves that the implementation of `sample_bernoulli_float` in `mod.rs` at commit [f5bb719](#) (outdated<sup>1</sup>) satisfies its proof definition.

`sample_bernoulli_float` 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_float(prob: T, constant_time: bool) -> bool:
3     if prob == 1: #
4         return True
5
```

<sup>1</sup>See new changes with `git diff f5bb719..1478d47 rust/src/traits/samplers/bernoulli/mod.rs`

```

6  # prepare for sampling first heads index by coin flipping
7  max_coin_flips = usize::exact_int_cast(T::EXPONENT_BIAS) + usize::exact_int_cast(
8      T::MANTISSA_BITS
9  ) #
10
11  # find number of bits to sample, rounding up to nearest byte (smallest sample size)
12  buffer_len = max_coin_flips.inf_div(8) #
13
14  # repeatedly flip fair coin and identify 0-based index of first heads
15  first_heads_index = sample_geometric_buffer( #
16      buffer_len, constant_time
17  )
18
19  # if no events occurred, return early
20  if first_heads_index is None: #
21      return False
22
23  # find number of zeroes in binary rep. of prob
24  leading_zeroes = (
25      T::EXPONENT_BIAS - 1 - prob.raw_exponent()
26  ) #
27
28  # case 1: index into the leading zeroes
29  if first_heads_index < leading_zeroes: #
30      return False
31
32  # case 2: index into implicit bit directly to left of mantissa
33  if first_heads_index == leading_zeroes: #
34      return prob.raw_exponent() != 0
35
36  # case 3: index into out-of-bounds/implicitly-zero bits
37  if first_heads_index > leading_zeroes + T::MANTISSA_BITS: #
38      return False
39
40  # case 4: index into mantissa
41  mask = 1 << (T::MANTISSA_BITS + leading_zeroes - first_heads_index)
42  return (prob.to_bits() & mask) != 0

```

## Postcondition

**Definition 0.1.** For any setting of the input parameters `prob` of type `T` restricted to  $[0, 1]$ , and `constant_time` of type `bool`, `sample_bernoulli_float` either

- raises an exception if there is a lack of system entropy,
- returns `out` where `out` is  $\top$  with probability `prob`, otherwise  $\perp$ .

If `constant_time` is set, the implementation’s runtime is constant.

## 0.2 Proof

*Proof.* To show the correctness of `sample_bernoulli` we observe first that the base-2 representation of `prob` is of the form

$$\text{leading\_zeroes} \parallel \text{implicit\_bit} \parallel \text{mantissa} \parallel \text{trailing\_zeroes}$$

and is represented *exactly* as a normal floating-point number. The [IEEE-754 standard](#) represents a normal floating-point number using an exponent  $E$ , and a mantissa  $m$ , using a base-2 analog of scientific notation.

**Definition 0.2** (Floating-Point Number). A  $(k, \ell)$ -bit floating-point number  $z$  is represented as

$$z = (-1)^s \cdot (B.M) \cdot (2^E)$$

where

- $s$  is used to represent the *sign* of  $z$
- $B$  is the implicit bit; 1 for normal floating-point numbers and 0 for subnormal floating point numbers
- $M \in \{0, 1\}^k$  is a  $k$ -bit string representing the part of the mantissa to the right of the radix point, i.e.,

$$1.M = \sum_{i=1}^k M_i 2^{-i}$$

- $E \in \mathbb{Z}$  represents the *exponent* of  $z$ . When  $\ell$  bits are allocated to representing  $E$ , then  $E \in [-(2^{\ell-1} - 2), 2^{\ell-1}] \cap \mathbb{Z}$ . Note that the range of  $E$  is  $2^\ell - 2$  rather than  $2^\ell$  as the remaining two numbers are used to represent special floating point values. When  $E = -(2^{\ell-1} - 2)$ , then the floating point number is considered *subnormal*.

We now use the technique for **arbitrarily biasing a coin in 2 expected tosses** as a building block. Recall that we can represent the probability `prob` as  $\text{prob} = \sum_{i=0}^{\infty} \frac{a_i}{2^{i+1}}$  for  $a_i \in \{0, 1\}$ , where  $a_i$  is the zero-indexed  $i$ -th significant bit in the binary expansion of `prob`. Then let  $I \sim \text{Geom}(0.5)$  and observe that the random variable  $a_I$  is an exact Bernoulli sample with probability `prob` since  $P(a_I = 1) = \sum_{i=0}^{\infty} P(a_i = 1 | I = i) P(I = i) = \sum_{i=0}^{\infty} a_i \cdot \frac{1}{2^{i+1}} = \text{prob}$ . It is therefore sufficient to show that for any  $(k, \ell)$ -bit float `prob` =  $\sum_{i=0}^{\infty} \frac{a_i}{2^{i+1}}$ , `sample_bernoulli` returns the value  $a_I$  with  $I \sim \text{Geom}(0.5)$ .

First, we observe that by line 3, if `prob` = 1.0 then `sample_bernoulli` returns `true` which is correct by definition of a Bernoulli random variable. Otherwise, the variable `max_coin_flips` is computed to be the value `T::EXPONENT_BIAS + T::MANTISSA_BITS` which equals  $2^{\ell-1} - 1 + k$  for any  $(k, \ell)$ -bit float. Since `prob` has finite precision, there is some  $j$  for which  $a_i = 0$  for all  $i > j$ . For all  $(k, \ell)$ -bit floating-point numbers,  $j \leq 2^{\ell-1} - 1 + k$  by definition. Then `sample_bernoulli` calls `sample_geometric_buffer` with a buffer of length  $\lceil \frac{\text{max\_coin\_flips}}{8} \rceil$  bytes (as shown in lines 9 and 12) which returns `None` if and only if  $I > 8 \cdot \lceil \frac{2^{\ell-1}-1+k}{8} \rceil$ , where  $I \sim \text{Geom}(0.5)$  (by Theorem 2.1). In this case, since  $I > j$  this index appears in the `trailing_zeros` part of the binary expansion of `prob` and should always return `false`, i.e.,  $a_I = 0$  for all  $I > j$ . We can therefore restrict our attention to when `sample_geometric_buffer` returns an index  $I \leq \text{max\_coin\_flips}$  and show that `sample_bernoulli` always returns  $a_I$ .

Assuming that `sample_geometric_buffer` returns some  $I < j$ , `sample_bernoulli` computes the number of leading zeroes in the binary expansion of `prob` to be `leading_zeros` = `T::EXPONENT_BIAS` - 1 - `raw_exponent(prob)`, where `raw_exponent(prob)` is the value stored in the  $\ell$  bits of the exponent. This value is correct by the specification of a  $(k, \ell)$ -bit float. `sample_bernoulli` then matches on the value `first_heads_index` corresponding to  $I \sim \text{Geom}(0.5)$  returned by the function `sample_geometric_buffer`:

**Case 1** (`first_heads_index` < `leading_zeros`).

This corresponds to `sample_geometric_buffer` returning a value  $I$  such that  $a_I$  indexes into the `leading_zeros` part of the `prob` variable's binary expansion. Therefore, for any  $I < \text{leading\_zeros}$ , it follows that  $a_I = 0$  and we should return `false`. In this case, `sample_bernoulli` returns `false`.

**Case 2** (`first_heads_index` == `leading_zeros`).

This corresponds to `sample_geometric_buffer` returning a value  $I$  such that  $a_I$  indexes into the `implicit_bit` part of the `prob` variable's binary expansion. When `prob` is a normal floating point value, i.e.,  $E \neq -(2^{\ell-1} - 2)$  then the implicit bit  $a_I = 1$ . Otherwise, when `prob` is a subnormal floating point value, i.e.,  $E = -(2^{\ell-1} - 2)$ , the implicit bit  $a_I = 0$ . Since `raw_exponent(prob)` corresponds to the exponent  $E$  for any  $(k, \ell)$ -bit floating point number `prob`, `sample_bernoulli` returns `true` when `raw_exponent(prob)`  $\neq 0$  and `false` otherwise.

**Case 3** (`leading_zeros` + `T::MANTISSA_BITS` <  $I$ ). This corresponds to the case where `sample_geometric_buffer` returns a value  $I$  where  $I > j$ , but  $I < \text{max\_coin\_flips}$  and therefore  $a_I$  indexes into the `trailing_zeros` part of `prob`'s. In this case, `sample_bernoulli` returns `false` since  $a_I = 0$  for all bits in the `trailing_zeros` part of `prob`'s.

binary expansion.

**Case 4** ( $\text{leading\_zeroes} < \text{first\_heads\_index} < \text{leading\_zeroes} + T::\text{MANTISSA\_BITS}$ ).

This corresponds to `sample_geometric_buffer` returning a value  $I$  such that  $a_I$  indexes into the `mantissa` part of the `prob` variable's binary expansion. In this case, `sample_bernoulli` left-shifts the value 1 by  $(\text{MANTISSA\_BITS} + \text{leading\_zeroes} - \text{first\_heads\_index})$  digits, the index into the mantissa corresponding to the digit  $a_I$  in the binary representation of `prob`. Since the operation between the left-shifted 1 and the binary representation of `prob` at that position is a bitwise AND, if the bit in question is 1 (matching the left-shifted 1), `sample_bernoulli` will return `true`. Otherwise, `sample_bernoulli` will return `false`.

Therefore, for any value of `prob`, the function `sample_bernoulli` either raises an exception or returns the value `true` with probability exactly `prob`. □