fn make_randomized_response_bool

Vicki Xu, Hanwen Zhang, Zachary Ratliff

October 24, 2025

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

Proves soundness of `make_randomized_response_bool` in `mod.rs` at commit f5bb719 (outdated[1]).

`make_randomized_response_bool` accepts a parameter `prob` of type `f64` and a parameter `constant_time` of type `bool`. The function on the resulting measurement takes in a boolean data point `arg` and returns the truthful value `arg` with probability `prob`, or the complement `!arg` with probability $1 -$ `prob`. The measurement function makes mitigations against timing channels if `constant_time` is set.

Warning 1 (Code is not constant-time)**.** `make_randomized_response_bool` takes in a boolean `constant_time` parameter that protects 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.

1 Hoare Triple

Preconditions

- Variable `prob` must be of type `f64`
- Variable `constant_time` must be of type `bool`

Pseudocode

```python
def make_randomized_response_bool(prob: f64, constant_time: bool):
    input_domain = AtomDomain(bool)
    input_metric = DiscreteMetric()
    output_measure = MaxDivergence()

    if (prob < 0.5 or prob > 1):  #
        raise Exception("probability must be in [0.5, 1]")

    if prob == 1.0:
        c = float("inf")
    else:
        c = prob.inf_div((1).neg_inf_sub(prob)).inf_ln()

    def privacy_map(d_in: u32) -> f64:  #
        if d_in == 0:
            return 0
        else:
```

[1] See new changes with `git diff f5bb719..a7353b4 rust/src/measurements/randomized_response/mod.rs`

```python
            return c

    def function(arg: bool) -> bool:  #
        sample = not sample_bernoulli_float(prob, constant_time)

        return arg ^ sample

    return Measurement(input_domain, function, input_metric, output_measure, privacy_map)
```

Postcondition

For every setting of the input parameters `(prob, constant_time)` to `make_randomized_response_bool` such that the given preconditions hold, `make_randomized_response_bool` raises an error (at compile time or run time) or returns a valid measurement. A valid measurement has the following properties:

1. (Data-independent runtime errors). For every pair of members x and x' in `input_domain`, `invoke`(x) and `invoke`(x') either both return the same error or neither return an error.

2. (Privacy guarantee). For every pair of members x and x' in `input_domain` and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map(d_in)` does not raise an error, and `privacy_map(d_in)` = `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

2 Proof

Proof. **(Privacy guarantee.)**

> **Note 1.** The following proof makes use of the following lemma that asserts the correctness of a Bernoulli sampler function.
>
> **Lemma 2.1.** `sample_bernoulli_float` satisfies its postcondition.

`sample_bernoulli` can only fail due to lack of system entropy. This is usually related to the computer's physical environment and not the dataset. The rest of this proof is conditioned on the assumption that `sample_bernoulli` does not raise an exception.

Let x and x' be datasets in the input domain (either $\top$ or $\bot$) that are `d_in`-close with respect to `input_metric`. Here, the metric is `DiscreteMetric` which enforces that $\texttt{d\_in} \geq 1$ if $x \neq x'$ and $\texttt{d\_in} = 0$ if $x = x'$. If $x = x'$, then the output distributions on x and x' are identical, and therefore the max-divergence is 0. Consider $x \neq x'$ and assume without loss of generality that $x = \top$ and $x' = \bot$. For shorthand, we let p represent `prob`, the probability that `sample_bernoulli_float` returns $\top$. Observe that $p = [0.5, 1.0]$ otherwise `make_randomized_response_bool` raises an error.

We now consider the max-divergence $D_\infty(Y||Y')$ over the random variables $Y = \texttt{function}(x)$ and $Y' = \texttt{function}(x')$.

$$
\begin{aligned}
\max_{x\sim x'} D_\infty(Y||Y') &= \max_{x\sim x'} \max_{S\subseteq Supp(Y)} \ln\left(\frac{\Pr[Y\in S]}{\Pr[Y'\in S]}\right) \\
&\le \max_{x\sim x'} \max_{y\in Supp(Y)} \ln\left(\frac{\Pr[Y=y]}{\Pr[Y'=y]}\right) && \text{Lemma 3.3 [1]} \\
&= \max_{x\sim x'} \max\left(\ln\left(\frac{\Pr[Y=\top]}{\Pr[Y'=\top]}\right), \ln\left(\frac{\Pr[Y=\bot]}{\Pr[Y'=\bot]}\right)\right) \\
&= \max\left(\ln\left(\frac{p}{1-p}\right), \ln\left(\frac{1-p}{p}\right)\right) \\
&= \ln\left(\frac{p}{1-p}\right)
\end{aligned}
$$

We let $c =$ `privacy_map`(`d_in`) $=$ `f64.inf_ln(prob.inf_div(1.neg_inf_sub(prob)))`. The computation of `c` rounds upward in the presence of floating point rounding errors. This is because `1.neg_inf_sub(prob)` appears in the denominator, and to ensure that the bound holds even in the presence of rounding errors, the conservative choice is to round down (so the quantity as a whole is bounded above). Similarly, `inf_div` and `inf_ln` round up.

When `d_in` > 0 and no exception is raised in computing `c` $=$ `privacy_map`(`d_in`), then $\ln\left(\frac{p}{1-p}\right) \le$ `c`.

Therefore we've shown that for every pair of elements $x, x' \in \{\bot, \top\}$ and every $d_{DM}(x, x') \le$ `d_in` with `d_in` ≥ 0, if x, x' are `d_in`-close then `function`(x), `function`(x') $\in \{\bot, \top\}$ are `privacy_map`(`d_in`)-close under `output_measure` (the Max-Divergence). □

References

[1] Shiva P. Kasiviswanathan and Adam Smith. On the "semantics" of differential privacy: A bayesian formulation. *Journal of Privacy and Confidentiality*, 6(1), June 2014.