fn make_binary_randomized_response_bool

Vicki Xu, Hanwen Zhang, Zachary Ratliff

July 18, 2024

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

Proves soundness of `make_binary_randomized_response_bool` in `mod.rs` at commit f5bb719 (outdated[1]).

`make_randomized_response_bool` accepts a parameter `prob` of type `Q` 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.

PR History

- Pull Request #490

1 Hoare Triple

Preconditions

- Variable `prob` must be of type `QO`
- Variable `constant_time` must be of type `bool`
- Type `bool` must have trait `SampleBernoulli<QO>`
- Type `QO` must have trait `Float`

Pseudocode

```
def make_randomized_response_bool(prob: QO, constant_time: bool):
    input_domain = AllDomain(bool)
    output_domain = AllDomain(bool)
    input_metric = DiscreteMetric()
    output_measure = DiscreteDivergence(QO)

    if (prob < 0.5 or prob >= 1):
```

[1] See new changes with `git diff f5bb719..31fdd61 rust/src/measurements/randomized_response/mod.rs`

```
        raise Exception("probability must be in [0.5, 1)")

    c = QO.inf_ln(prob.inf_div(1.neg_inf_sub(prob)))
    def privacy_map(d_in: IntDistance) -> QO:
        if d_in == 0:
            return 0
        else:
            return c

    def function(arg: bool) -> bool:
        return arg ^ !bool.sample_bernoulli(prob, constant_time)

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

Postcondition

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

1. (Privacy guarantee). For every pair of elements x, x' in `input_domain` and for every pair (`d_in`, `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 exception, and `privacy_map(d_in)` $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

2 Proof

Proof.

1. **(Privacy guarantee.)**

   > **Note 1** (Proof relies on correctness of Bernoulli sampler)**.** The following proof makes use of the following lemma that asserts the correctness of the Bernoulli sampler function.
   >
   > **Lemma 2.1.** If system entropy is not sufficient, `sample_bernoulli` raises an error. Otherwise, `sample_bernoulli(prob, constant_time)`, the Bernoulli sampler function used in `make_randomized_response_bool`, returns `true` with probability (`prob`) and returns `false` with probability (1 - `prob`).

   `sample_bernoulli` can only fail when the OpenSSL pseudorandom byte generator used in its implementation fails 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 v and w be datasets that are `d_in`-close with respect to `input_metric`. Here, the metric is `DiscreteMetric` which enforces that `d_in` ≥ 1 if $v \neq w$ and `d_in` $= 0$ if $v = w$. If $v = w$, then the output distributions on v and w are identical, and therefore the max-divergence is 0. Consider $v \neq w$ and assume without loss of generality that $v =$ `true` and $w =$ `false`. For shorthand, we let p represent `prob`, the probability that `sample_bernoulli` returns `true`. Observe that $p = [0.5, 1.0)$ otherwise `make_randomized_response_bool` raises an error.

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

$$D_\infty(Y||Z) = \max_{S \subseteq Supp(Y)} \left[\ln\left(\frac{\Pr[Y \in S]}{\Pr[Z \in S]} \right) \right]$$

$$= \max\left(\ln\left(\frac{\Pr[Y = \texttt{true}]}{\Pr[Z = \texttt{true}]} \right), \ln\left(\frac{\Pr[Y = \texttt{false}]}{\Pr[Z = \texttt{false}]} \right) \right)$$

$$= \max\left(\ln\left(\frac{p}{1-p} \right), \ln\left(\frac{1-p}{p} \right) \right)$$

$$= \ln\left(\frac{p}{1-p} \right)$$

We let $c = \texttt{privacy\_map(d\_in)} = \texttt{QO.inf\_ln(prob.inf\_div(1.neg\_inf\_sub(prob)))}$. 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 $\texttt{d\_in} > 0$ and no exception is raised in computing $\texttt{c} = \texttt{privacy\_map(d\_in)}$, then $\ln\left(\frac{p}{1-p}\right) \leq \texttt{c}$.

Therefore we've shown that for every pair of elements $v, w \in \{\texttt{false}, \texttt{true}\}$ and every $d_{DM}(v, w) \leq \texttt{d\_in}$ with $\texttt{d\_in} \geq 0$, if v, w are `d_in`-close then $\texttt{function}(v), \texttt{function}(w) \in \{\texttt{false}, \texttt{true}\}$ are $\texttt{privacy\_map(d\_in)}$-close under `output_metric` (the Max-Divergence).

□