# fn make_rappor

Abigail Gentle

June 16, 2024

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

Proves soundness of `make_rappor` in `mod.rs` at commit cfd1bec5 (outdated[1]).

## 1 Introduction

RAPPOR is a protocol for local differentially private (DP) frequency estimation. In the local model, each user is guaranteed $\varepsilon$-DP for their response. This is achieved by computing the `xor` of each input vector with a noise vector. Because the noise is added mechanically we can efficiently account for the bias introduced with the function `debias_basic_rappor`, which sums and normalises a vector of private outputs.

In the simplest case, each category is represented by an index $i \in [k]$, where $[k] = \{0, 1, \ldots, k-1\}$, and each row with non-empty category $i$ is transformed into a one-hot vector which has 0's everywhere and a 1 at index $i$. Therefore, the number of set bits in the input is 1, and the input domain should be a `BitVectorDomain` with `max_weight` $= 1$. Vikrant: I would define $m$ here, and then explain through the two examples you have provided. Otherwise, the reader will have to search for its meaning while reading the actual proofs. Let's just also mention somewhere that $2m + 1 \leq k$, as we had discussed on GitHub.

In order to estimate the frequencies of strings drawn from a potentially unbounded set, inputs can also be hashed onto a Bloom filter using $h < k$ hash functions. As such the number of set bits is at most $h$ and the input domain should be a `BitVectorDomain` with `max_weight` $= h$. As we demonstrate in Theorem 3.2, the privacy of the protocol will degrade linearly with the number of hash functions used (although the hash functions will make the life of an adversary trickier, as each bit set in the Bloom filter could map to multiple possible strings).

## 2 Hoare Triple

### 2.1 Preconditions

- Variable `input_domain` must be of type `BitVectorDomain`, with `max_weight`.
- Variable `input_metric` must be of type `DiscreteDistance`.
- Variable `f` must be of type `f64`.
- Variable `constant_time` must be of type `bool`.

[1] See new changes with `git diff cfd1bec5..8e0d3009 rust/src/measurements/rappor/mod.rs`

### 2.2 Pseudocode

```
def make_rappor(f: float64, constant_time: bool):
    input_domain: BitVectorDomain
    input_metric: DiscreteDistance
    output_domain = BitVectorDomain
    output_measure = MaxDivergence(f64)

    if (f <= 0.0 or f > 1):
        raise Exception("Probability must be in (0.0, 1]")

    m = input_domain.max_weight
    if m == None:
        raise Exception("RAPPOR requires a maximum number of set bits")
    eps = (2*m)*log((2-f)/f)
    def privacy_map(d_in: IntDistance):
        return eps
    def function(arg: BitVector) -> BitVector:
      k = len(arg)
      noise_vector = [bool.sample_bernoulli(f/2, constant_time) for _ in range(k)]
      return xor(arg, noise_vector)

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

### 2.3 Postcondition

For every setting of the input parameters `(f, m, constant_time)` to `make_rappor` such that the given preconditions hold,
`make_rappor` 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 $u, v$ 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 $u, v$ 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`($u$), `function`($v$) are `d_out`-close under `output_measure`.

## 3 Proof

### 3.1 Privacy

**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 3.1.** If system entropy is not sufficient, `sample_bernoulli` raises an error. Otherwise, `sample_bernoulli(f/2, constant_time)`, the Bernoulli sampler function used in `make_randomized_response_bool`, returns `true` with probability (`prob`) and returns `false` with probability (1 - `f/2`).

**Theorem 3.2** ([1])**.** The program `make_rappor` satisfies $\varepsilon$-DP, where

$$\varepsilon = 2m \log\left(\frac{2-f}{f}\right).$$

**Lemma 3.3.** We have the following for the output of the algorithm.

$$\mathbb{P}[y_i = 1 \mid x_i = 1] = 1 - \frac{1}{2}f \tag{1}$$

$$\mathbb{P}[y_i = 1 \mid x_i = 0] = \frac{1}{2}f \tag{2}$$

*Proof.* Let $Y = (y_1, \ldots, y_k)$ be a randomised report generated by `make_rappor`. Then the probability of observing any given report $Y$ is $\mathbb{P}[Y = y|X = x]$. Suppose $x = (x_1, \ldots, x_k)$ is a single Boolean vector with at most $m$ ones. Without loss of generality assume that $x^* = (x_1 = 1, \ldots, x_m = 1, x_{m+1} = 0, \ldots, x_k = 0)$, then we have

$$\mathbb{P}[Y = y \mid X = x^*] = \prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i} \times \prod_{i=m+1}^{k} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i}.$$

Then let $D$ be the ratio of two such conditional probabilities with distinct values $x_1$ and $x_2$, and let $S$ be the range of `make_rappor`. Vikrant: The notations $x_1, x_2$ are getting overloaded in the following, I believe. If they're denoting neighbouring vectors, I would just use $x, x'$.

$$\begin{aligned}
D &= \frac{\mathbb{P}[Y \in S \mid X = x_1]}{\mathbb{P}[Y \in S \mid X = x_2]} && (3)\\
&= \frac{\sum_{y \in S} \mathbb{P}[Y = y \mid X = x_1]}{\sum_{y \in S} \mathbb{P}[Y = y \mid X = x_2]} \\
&\leq \max_{y \in S} \frac{\mathbb{P}[Y = y \mid X = x_1]}{\mathbb{P}[Y = y \mid X = x_2]} \\
&= \max_{y \in S} \frac{\prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i} \times \prod_{i=m+1}^{k} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i}}{\prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i} \times \prod_{i=m+1}^{2m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i} \times \prod_{i=2m+1}^{k} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i}} \\
&= \max_{y \in S} \frac{\prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i} \times \prod_{i=m+1}^{2m} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i}}{\prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i} \times \prod_{i=m+1}^{2m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i}} && (4)\\
&= \max_{y \in S} \left[\prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{2(1-y_i)} \left(1 - \frac{1}{2}f\right)^{2y_i} \times \prod_{i=m+1}^{2m} \left(\frac{1}{2}f\right)^{2y_i} \left(1 - \frac{1}{2}f\right)^{2(1-y_i)}\right] && (5)
\end{aligned}$$

Notice that, by Equation 4, the privacy is not dependent on $k$ and Equation 5 is maximised when $y_1 = 1, \ldots, y_m = 1$, and $y_{m+1}, \ldots, y_{2m} = 0$, giving

$$\begin{aligned}
D &\leq \left(1 - \frac{1}{2}f\right)^{2m} \times \left(\frac{1}{2}f\right)^{-2m} \\
&= \left(\frac{2 - f}{f}\right)^{2m}
\end{aligned}$$

Therefore,

$$\varepsilon \leq 2m \log\left(\frac{2 - f}{f}\right), \tag{6}$$

which completes the proof. □

## 3.2 Utility

**Theorem 3.4.** The program `debias_basic_rappor` is an unbiased frequency estimator.

*Proof.* We denote the input domain as $X$, which is the set of all vectors with hamming weight at most $m$. The set of all users is $X^n$, where each user $x_1, \ldots, x_n$ holds a single input. Vikrant: We are overloading the notation $x$ quite a bit. In the previous subsection, we said $x_i$ is essentially a $k$-dimensional vector that is each user's data, but now, we are saying that $x_i$ is every user's data. There is a mismatch. I would change the notation a bit in the previous subsection, or introduce superscripts to make it less confusing. Also, "single input" is confusing. what does that mean here? Does every user just contribute one bit or $m$ bits? Each input $x_j$ is transformed into $y_j$ using `make_rappor`. Let $Y$ be the sum of $n$ received outputs, where $Y_i = \sum_{j=1}^{n} y_i$ is the number of received bits at index $i \in [k]$. Let $N_i = \sum x_{j,i}$ be the true (non-private) count vector of users with bit $i$ set. Our goal is to estimate the frequencies $q_i = N_i/n$ with minimal error. $Y_i$ is a sum of two binomials

$$\mathbb{E}[Y_i] = \mathbb{E}\left[\text{Bin}\left(N_i, 1 - \frac{f}{2}\right) + \text{Bin}\left(n - N_i, \frac{f}{2}\right)\right] \tag{7}$$

$$= N_i\left(1 - \frac{1}{2}f\right) + (n - N_i)\frac{f}{2}. \tag{8}$$

Therefore, by rearranging (8) we obtain an unbiased estimator for $\hat{N}_i$,

$$\hat{N}_i = \frac{Y_i - n\frac{f}{2}}{1 - f}. \tag{9}$$

Our goal, however, is to estimate the frequency of each element $\hat{q}_i$ so we need to normalise by dividing by $n$, which gives us

$$\hat{q}_i = \frac{\mathbb{E}[\hat{N}_i]}{n} = \frac{\frac{Y_i}{n} - \frac{f}{2}}{1 - f}.$$

This completes our proof. □

**Theorem 3.5.** The program `debias_basic_rappor` is a frequency estimator with mean squared error

$$\mathbb{E}[\|q - \hat{q}\|_2^2] = \frac{k\left(f - \frac{f^2}{2}\right)}{2n(1-f)^2}.$$

*Proof.* Let $q = q(X^n) = \sum_{j=1}^{n} x_j$ be the non private frequency vector of inputs. Our goal is to find the mean squared error of our estimate from this vector. Using the fact from Equation 7 that $Y_i$ is a sum of two Binomials with equal variance (by the symmetry of their probabilities), we get the following.

$$\text{Var}(Y_i) = n\frac{f}{2}\left(1 - \frac{f}{2}\right) \tag{10}$$

$$\mathbb{E}[\|\hat{q} - q\|_2^2] = \mathbb{E}\left[\sum_{i=1}^{k}(\hat{q}_i - q_i)^2\right] = \sum_{i=1}^{k}\mathbb{E}[(\hat{q}_i - q_i)^2]$$

$$= \sum_{i=1}^{k}\mathbb{E}[(\hat{q}_i - \mathbb{E}[\hat{q}_i])^2] \qquad \text{(By Theorem 3.4)}$$

$$= \sum_{i=1}^{k}\text{Var}(\hat{q}_i) = \sum_{i=1}^{k}\text{Var}\left(\frac{\frac{\hat{Y}_i}{n} - \frac{f}{2}}{1 - f}\right) \qquad \text{(By Equation 9)}$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \mathrm{Var}\left(\frac{Y_i}{n(1-f)}\right) \\
&= \sum_{i=1}^{k} \frac{\mathrm{Var}(Y_i)}{n^2(1-f)^2} \\
&= k\left(\frac{n\frac{f}{2}\left(1-\frac{f}{2}\right)}{n^2(1-f)^2}\right) && \text{(By Equation 10)} \\
&= \frac{k\left(f-\frac{f^2}{2}\right)}{2n(1-f)^2}
\end{aligned}
$$

This completes the proof. □

# References


[1] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, Scottsdale, Arizona, 2014.