MakeNoise<DI, MI, MO> for DiscreteLaplace

Michael Shoemate

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

Proves soundness of the implementation of `MakeNoise` for `DiscreteLaplace` in `mod.rs` at commit f5bb719 (outdated[1]).

This is an intermediary compile-time layer whose purpose is to dispatch to either the integer or floating-point variations of the mechanism, depending on the type of data in the input domain.

It does this through the use of the `Nature` trait, which has concrete implementations for each possible input type. This layer makes interior layers simpler to work with, and does not have privacy implications.

1 Hoare Triple

Precondition

Compiler-Verified

`MakeNoise` is parameterized as follows:

- `DI` is of type `VectorDomain<AtomDomain<T>>`
- `MI` is of type `L1Distance<T>`
- `MO` implements trait `Measure`

The following trait bounds are also required:

- Generic `T` implements trait `Integer`
- Generic `MO` implements trait `Measure`
- Type `usize` implements trait `ExactIntCast<T>`
- Type `RBig` implements trait `TryFrom<T>`
- Type `ZExpFamily<1>` implements trait `NoisePrivacyMap<L1Distance<RBig>, MO>`. This bound requires that it must be possible to construct a privacy map for the combination of `ZExpFamily<1>` noise distribution, distance type and privacy measure. Since the `ConstantTimeGeometric` distribution is equivalent to `ZExpFamily<1>`, maps built for `ZExpFamily<1>` can be used for `ConstantTimeGeometric`.

User-Verified

None

[1] See new changes with `git diff f5bb719..33f2250 rust/src/measurements/noise/distribution/geometric/mod.rs`

Pseudocode

```
1 # analogous to impl MakeNoise<VectorDomain<AtomDomain<T>>, L1Distance<T>, MO> for ConstantTimeGeometric<T> in Rust
2 class ConstantTimeGeometric:
3     def make_noise(
4         self, input_space: tuple[DI, MI]
5     ) -> Measurement[DI, DI_Carrier, MI, MO]:
6         input_domain, input_metric = input_space
7         scale, (lower, upper) = self.scale, self.bounds
8         if lower > upper: #
9             raise "lower may not be greater than upper"
10
11        distribution = ZExpFamily(scale=RBig.from_f64(scale))
12        output_measure = MO.default()
13
14        privacy_map = distribution.noise_privacy_map(
15            L1Distance.default(), output_measure
16        )
17
18        p = (1.0).neg_inf_sub((-scale.recip()).inf_exp()) #
19        if not (0.0 < p <= 1.0):
20            raise f"Probability of termination p ({p}) must be in (0, 1]. This is likely because the noise scale is so large that conservative arithmetic causes p to go negative"
21
22        def function(arg: Vec[T]) -> Vec[T]:
23            return [
24                sample_discrete_laplace_linear(v, scale, (lower, upper)) for v in arg
25            ]
26
27        return Measurement.new(
28            input_domain,
29            Function.new_fallible(function),
30            input_metric,
31            output_measure,
32            PrivacyMap.new_fallible(lambda d_in: privacy_map.eval(RBig.try_from(d_in))),
33        )
```

Postcondition

Theorem 1.1. For every setting of the input parameters (`self, input_space, T, MO`) to `make_noise` such that the given preconditions hold, `make_noise` 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 (`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 error, and `privacy_map(d_in)` = `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

Data-independent errors. Due to the check on line 8, the preconditions for `sample_discrete_laplace_linear` are satisfied. Therefore the postcondition guarantees data-independent errors. Since this is the only source of errors in the function, errors from the function are data-independent. ☐

Privacy guarantee. The privacy guarantee breaks down into three parts:

1. A 1-stable clamping pre-processor.

2. The noise perturbation mechanism.

3. A post-processing clamp.

Clamping the inputs is necessary because, to make constant-time sampling tractable, samples are only drawn up to the magnitude of the distance between the bounds. In the extreme case, consider when a adjacent inputs are located at `L + U` and `L + U - 1` without input clamping: then outputting `U - 1` is a distinguishing event.

This motivates the need for a clamping transformation.

$$\max_{x \sim x'} |\texttt{clamp}(x, L, U) - \texttt{clamp}(x', L, U)|_1 \tag{1}$$

$$\leq \max_{x \sim x'} |x - x'|_1 \tag{2}$$

$$= \texttt{d\_in} \tag{3}$$

The clamping transformation can only make datasets more similar, so the clamp is a 1-stable transformation.

Similarly the perturbed value also needs to be clamped: consider adjacent inputs at `L` and `L + 1`, then outputting `L - (U - L)` is a distinguishing event, due to the limited range of the noise distribution.

Notice that the output distribution is equivalent to sampling from the discrete laplace distribution with infinite support, and then clamping as post-processing. Therefore the privacy guarantee from NoisePrivacyMap`<L1Distance<RB` `MO>` applies. □