

MakeNoise<AtomDomain<T>, AbsoluteDistance<T>, MO> for ConstantTimeGeometric<T>

Michael Shoemate

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

Proves soundness of the implementation of **MakeNoise** over scalars for **ConstantTimeGeometric** in **mod.rs** at **commit f5bb719** (outdated¹).

The intuition of this implementation is that a vector-valued mechanism can be used to privatize a scalar-valued input, by transforming the input into a singleton vector, applying the vector mechanism, and then unpacking the resulting singleton vector.

1 Hoare Triple

Precondition

Compiler-Verified

MakeNoise is parameterized as follows:

- DI is of type **AtomDomain<T>**
- MI is of type **AbsoluteDistance<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

¹See new changes with `git diff f5bb719...c3b1c15 rust/src/measurements/noise/distribution/geometric/mod.rs`

Pseudocode

```
1 # analogous to impl MakeNoise<AtomDomain<T>, AbsoluteDistance<T>, MO> for
   ConstantTimeGeometric<T> in Rust
2 class ConstantTimeGeometric:
3     def make_noise(
4         self, input_space: tuple[AtomDomain[T], AbsoluteDistance[QI]]
5     ) -> Measurement[AtomDomain[T], T, AbsoluteDistance[QI], MO]:
6         t_vec = make_vec(input_space) #
7         m_noise = self.make_noise(t_vec.output_space()) #
8
9         return t_vec >> m_noise >> then_index_or_default(0) #
```

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`.

Proof. Neither constructor `make_vec` nor `MakeNoise.make_noise` have manual preconditions, and the postconditions guarantee a valid transformation and valid measurement, respectively. `then_index_or_default` also does not have preconditions, and its postcondition guarantees that it returns a valid postprocessor.

The chain of a valid transformation, valid measurement and valid postprocessor is a valid measurement. □