Modular Verification of Differential Privacy in Probabilistic Higher-Order Separation Logic

PHILIPP G. HASELWARTER, Aarhus University, Denmark
ALEJANDRO AGUIRRE, Aarhus University, Denmark
SIMON ODDERSHEDE GREGERSEN, New York University, USA
KWING HEI LI, Aarhus University, Denmark
JOSEPH TASSAROTTI*, New York University, USA
LARS BIRKEDAL, Aarhus University, Denmark

Differential privacy is the standard method for privacy-preserving data analysis. The importance of having strong guarantees on the reliability of implementations of differentially private algorithms is widely recognized and has sparked fruitful research on formal methods. However, the design patterns and language features used in modern DP libraries as well as the classes of guarantees that the library designers wish to establish often fall outside of the scope of previous verification approaches.

We introduce a program logic suitable for verifying differentially private implementations written in complex, general-purpose programming languages. Our logic has first-class support for reasoning about privacy budgets as a separation logic resource. The expressiveness of the logic and the target language allow our approach to handle common programming patterns used in the implementation of libraries for differential privacy, such as privacy filters and caching. While previous work has focused on developing guarantees for programs written in domain-specific languages or for privacy mechanisms in isolation, our logic can reason modularly about primitives, higher-order combinators, and interactive algorithms.

We demonstrate the applicability of our approach by implementing a verified library of differential privacy mechanisms, including an online version of the Sparse Vector Technique, as well as a privacy filter inspired by the popular Python library OpenDP, which crucially relies on our ability to handle the combination of randomization, local state, and higher-order functions. We demonstrate that our specifications are general and reusable by instantiating them to verify clients of our library. All of our results have been foundationally verified in the Rocq Prover.

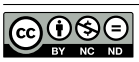
CCS Concepts: • **Software and its engineering** → **Software verification**; • **Theory of computation** → **Probabilistic computation**; **Separation logic**; • **Security and privacy** → **Data anonymization and sanitization**; **Logic and verification**.

Additional Key Words and Phrases: Differential Privacy, Separation Logic, Relational Program Verification

ACM Reference Format:
Philipp G. Haselwarter, Alejandro Aguirre, Simon Oddershede Gregersen, Kwing Hei Li, Joseph Tassarotti, and Lars Birkedal. 2026. Modular Verification of Differential Privacy in Probabilistic Higher-Order Separation Logic. *Proc. ACM Program. Lang.* 10, PLDI, Article 233 (June 2026), 25 pages. https://doi.org/10.1145/3808311

*Also affiliated with Amazon Web Services. This paper does not reflect the views of Amazon Web Services.

Authors' Contact Information: Philipp G. Haselwarter, Aarhus University, Denmark, pgh@cs.au.dk; Alejandro Aguirre, Aarhus University, Denmark, alejandro@cs.au.dk; Simon Oddershede Gregersen, New York University, USA, s.gregersen@nyu.edu; Kwing Hei Li, Aarhus University, Denmark, hei.li@cs.au.dk; Joseph Tassarotti, New York University, USA, jt4767@cs.nyu.edu; Lars Birkedal, Aarhus University, Denmark, birkedal@cs.au.dk.



ACM 2475-1421/2026/6-ART233
https://doi.org/10.1145/3808311

1 Introduction

Differential privacy [18, 19] (DP) is a collection of programming techniques to release aggregate information from a database while providing statistical guarantees about the privacy of individual user data. DP has been used widely in government and industrial applications to protect critical personal information (medical, financial, demographic, behavioral). The correctness of the implementations of DP is therefore crucial: logic- or implementation-level bugs can lead to catastrophic failure of the promised privacy guarantees.

The importance of having trustworthy implementations of DP is widely recognized. On the one hand, it has led to significant research in programming language and program verification. On the other, industrial developments of DP are routinely accompanied by pen-and-paper proofs of their claimed privacy properties. The OpenDP collaboration[1] even goes as far as to collect a proof for each element of the library, which are checked by a "privacy proof review board".

DP provides strong *statistical* guarantees that the data contributed by an individual does not influence the result of the analysis by too much, and hence cannot be recovered by observing the outcome. This is achieved by adding a small amount of random noise to each step of data analysis that could leak private information. This way, even if the input data changes by a small amount (say, the data of one user in a database), we can expect the output of the noisy program output to be similar, and an observer will not be able to tell whether the change comes from the noise or a change in inputs. The strength of the privacy guarantee of differentially private (dp) programs is governed by a parameter $\varepsilon \in \mathbb{R}_{\geq 0}$, that controls the overall probability that any user's privacy is compromised; a lower ε means a more private program.

Managing the Privacy Budget. A central idea of programming with DP is to think of the parameter ε that controls the noise distribution as the "privacy budget". To ensure that the whole program P, written as a sequence of computations $P = c_0; c_1; \ldots; c_n$, is ε-differentially private (ε-dp), we can "allocate" some part ε_i of our budget to each part c_i of the program and check that the probability that c_i compromises privacy is controlled according to ε_i. So long as the sum of the error terms ε_i is below the global budget ε, the program as a whole is ε-dp. This principle is known as the sequential composition theorem of DP, and it justifies our budget intuition of the privacy parameter: ε represents a *resource* that can be *split* according to the structure of the program, and is *consumed* by computing noisy results from a database.

The most widely used deployments of DP are based on *dynamic* techniques for tracking the privacy budget via a trusted library such as OpenDP (written in Python, Rust) or Google's Differential Privacy library (C++, Go, Java). Rather than specifying exactly how much ε_i each part of a program consumes, the budget consumption of the completed computations $c_0; \ldots; c_{i-1}$ is tracked, and a runtime check ensures that enough budget for c_i remains.

1.1 The Challenges of Verifying DP Frameworks

Most practical DP frameworks use "advanced" programming language features like higher-order functions (*e.g.*, in the form of classes or function pointers) and dynamically allocated, local state (*e.g.*, via class-private attributes). This is necessary to enable the *modular* construction of private programs through an API, where a library client does not have to concern themselves with the management of the privacy budget and correct application of noise to the results. To be able to reason compositionally, privacy proofs of a library API should likewise support *modular specifications*. We consider three representative challenges that arise from reasoning about libraries for DP:

(i) encapsulating dynamic, fine-grained budget accounting,

[1] OpenDP was initiated as a collaboration between Harvard University and Microsoft.

(ii) interactive or "online" data analysis,
(iii) budget minimization via caching.

As we shall see shortly, all three of these techniques rely on stateful randomized higher-order functions to implement features that are essential to the modular construction of dp programs. To verify modular API specifications we must thus support higher-order reasoning about local state. We illustrate the programming patterns via Python code snippets inspired by OpenDP, but the challenges also arise in, *e.g.*, the C++ implementation of Google's Differential Privacy library.[2] Although the verification approach developed in this paper is for an ML-like core language (§2.3) rather than Python, it can faithfully represent the salient aspects of these Python programs.

1.1.1 Dynamic Budget Accounting. An essential functionality of these libraries is that they offer *privacy filters* [45] that encapsulate the intricate reasoning about the privacy budget in a core, trusted API. A privacy filter tracks the privacy cost that a program has incurred up to the current execution point $c_0; \ldots; c_{i-1}$, and only executes c_i if enough budget remains. The example implementation of a simplified `PrivacyFilter` in Figure 1 is initialized with some `budget` and provides a single `try_run(cost, f)` method, which only runs `f` if there is indeed at least enough budget left to cover its `cost`. So long as each mechanism `f` correctly reports its budget consumption (`cost`), the filter will ensure that the total privacy cost will never exceed the global budget.

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, f):
    if self.budget_left < cost :
      return None
    self.budget_left -= cost
    return f()
```

Fig. 1. PrivacyFilter ensures that no client can exceed the privacy budget.

Centralized dynamic budget management simplifies the privacy analysis of programs and allows for scaling to industrial applications. Importantly, dynamically computed bounds enable a tighter analysis of the privacy cost compared to static type checks [21, 34]. Recall that the usual sequential composition theorem of DP requires that all ε_i be chosen upfront. The adaptive composition theorem [45] for DP lifts this restriction and allows the analyst to *adaptively* chose each ε_i depending on the results of previous (private) computations. Despite the successes of type- and program-logic-based analyses of DP, none of the existing systems can be used to specify and verify the correctness of implementations of privacy filters.

1.1.2 Interactive Data Analysis. *Interactivity* is central to real-world data analysis under DP. Given a new dataset, a data analyst does not just issue a static set of queries. Instead, they may compute some summary statistics to explore what ranges of values or categories of responses are of interest, and construct further queries based on those observations. This interactive or "online" style of analysis is crucial for practical utility [15, 19, 32]. A standard way to represent interactive computations is as streams: given a stream of queries, a private interactive mechanism should produce a stream of results.

The `AboveThreshold` mechanism in Figure 2 constructs a stream of booleans from a stream of queries as an iterator. The n-th boolean indicates whether q_n was above the threshold T, after suitable Laplace noise was added. Crucially, the queries themselves are provided as an iterator rather than as a precomputed list, and q_n is only evaluated in the n-th call to `__next__`. The

[2] See, *e.g.*, https://github.com/google/differential-privacy/blob/0a3b05a65f7ed8de/cc/accounting/accountant.cc#L46

`AboveThreshold` mechanism is ε-dp because the stream stops producing results after the first time `True` is returned; this is a standard result in DP but challenging to prove formally because it does *not* follow simply from the sequential composition theorem.

```
class AboveThreshold:
  def __init__(self, eps, T, queries, db):
    self.eps = eps
    self.T = Laplace(T, eps/2)
    self.queries = queries
    self.db = db
    self.halted = False

  def __iter__(self):
    return self

  def __next__(self):
    if self.halted:
      raise StopIteration
    q = next(self.queries)
    v = Laplace(q(self.db), self.eps/4)
    b = (self.T <= v)
    self.halted = b
    return b
```

Fig. 2. The classic Above Threshold interactive mechanism.

Any realistic verification framework for DP must therefore capture not only isolated, static mechanisms but also their behavior as interactive, stateful processes that maintain internal state and privacy budget across calls.

1.1.3 Budget Minimization via Caching. Certain results (*e.g.*, the number of non-zero entries) have to be calculated many times over as part of different analysis passes on a dataset. This can be wasteful in practical applications, since each private computation consumes some of the privacy budget, even if the same result has already been computed elsewhere in the program. Caching provides a solution to this problem: if the noisy results of queries are cached, then a repeated query can simply reuse the prior noisy result without incurring any privacy cost. Recent implementations of privacy frameworks have studied increasingly sophisticated caching strategies for DP workloads [32, 33, 39, 43], but even simple implementations of caching via memoization can lead to substantial savings in privacy budget in practical workloads [43][32, Fig. 3].

The implementation of memoization `mk_query_cache` in Figure 3 locally allocates a `cache` associated to a mechanism `add_noise` and a dataset `db` and returns a closure `f` which can be used subsequently to privately evaluate queries on `db`. Although it is a simple general-purpose caching mechanism, studying it formally requires reasoning about local state and higher-order functions. This places it outside of the scope of existing systems.

```
def mk_query_cache(add_noise, db):
  cache = {}
  def f(query):
    if query in cache:
      return cache[query]
    v = add_noise(query(db))
    cache[query] = v
    return v
  return f
```

Fig. 3. A generic caching mechanism.

1.2 Formal Guarantees for Differential Privacy

DP lends itself well to the analysis via standard PL methods due to its compositional nature. We distinguish two main classes of approaches. Type-based approaches [2, 10, 16, 21, 34, 35, 42, 44, 49–51] use a static typing discipline to ensure that the programs accepted by the system are dp. These approaches benefit from high degrees of automation. Most type-based approaches, however, do not handle mutable state. More generally, the complex nature of the type systems required (dependent-,

linear-, contextual-, or refinement types, or combinations thereof), hinders their integration with mainstream programming languages.

On the other hand, relational probabilistic Hoare logics such as apRHL [8, 9, 11], are more expressive but require more user effort in the form of manual or interactive proofs. These methods works particularly well when the program logic is defined with respect to a relatively simple denotational semantics of programs as subdistributions. This, however, generally restricts the applicability of the method to first-order programs. HO-RPL [3] extends the previous approach to support higher-order functions by constructing a more sophisticated semantic model, but general recursion or dynamic allocation and higher-order state are still unsupported. In a similar vein, some projects verified the privacy of sampling algorithms and simple mechanisms directly in the denotational semantics of programs [17, 48]. In summary, these approaches work well for the verification of algorithms in isolation, but do not focus on supporting *modular* program verification.

1.3 Modular Verification for DP Libraries in Clutch-DP

To address the challenges arising from the verification of DP libraries, we introduce the relational approximate probabilistic higher-order separation logic Clutch-DP. Clutch-DP supports higher-order functions such as `PrivacyFilter` and `mk_query_cache` via quantification over specifications, and references storing closures as they occur in the implementation of iterators (*e.g.*, `AboveThreshold`) via impredicative invariants à la Iris [31]. Building on previous relational separation logics [24, 27], we internalize the privacy budget of DP in Clutch-DP as a first-class separation logic resource. These *privacy credits* can be tracked in invariants and interact with the heap and with higher-order functions that consume a privacy budget in flexible ways. We prove that the usual rules for (sequential) relational separation logic [20] are sound in the probabilistic setting. To reason about privacy of primitives, we prove novel relational sampling rules for the Laplacian; in particular, we internalize a proof technique based on choice couplings [5] as a logical rule, which, in particular, is used in the privacy proof of `AboveThreshold`. We apply Clutch-DP to a number of case studies inspired by the challenge problems described above. Our specifications of, *e.g.*, `AboveThreshold` are compositional and reusable: we derive the privacy of clients and more complex mechanisms from the specifications of the building blocks, without referring to implementation details.

Contributions. To summarize, we make the following contributions.

(1) *A higher-order separation logic for DP* which internalizes privacy credits as first-class, composable logical resources and which supports heap allocation and higher-order closures.
(2) *New probabilistic sampling rules*, including a Laplacian rule that enables selective recovery of privacy credits, which can be used to verify examples whose privacy analysis goes beyond composition theorems.
(3) *A library of reusable, abstract specifications* for common DP primitives (Laplace, AboveThreshold, Sparse Vector Technique, Report-Noisy-Max, Privacy Filters, Caching via Memoization) that cleanly separate mechanism proof from client reasoning.
(4) *Client case studies* that highlight both expressiveness and reusability of specifications and that demonstrate that our approach successfully addresses the challenges outlined above (dynamic budget accounting, interactive data analysis, and budget minimization via caching).
(5) *Mechanized proofs*: a foundational formalization of the logic and all case studies in the Rocq Prover together with an adequacy theorem connecting our logic to standard DP.

Outline. In §2 we briefly recall the basic notions of DP and define the RandML programming language used in the remainder of the paper. In §3 we define the Clutch-DP logic and explain how it relates to DP. In §4 we illustrate reasoning in Clutch-DP via a number of case studies addressing

the challenges set out in §1.1. The soundness of Clutch-DP is addressed in §5. Finally, in §6 we survey related work, before concluding in §7.[3]

2 Differential Privacy and Programming Language Preliminaries

We briefly recall the elements of probability theory §2.1 and DP §2.2 we need to refer to, and formally define the programming language §2.3 used throughout the paper.

2.1 Probability Theory

Since we do not assume a priori that all programs we study terminate, we allow programs to "lose mass" on diverging runs and define the operational semantics using probability *sub*-distributions.

Definition 2.1. *A* discrete subdistribution *(henceforth simply* distribution*) on a countable set A is a function $\mu : A \to [0,1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. The distributions on A are denoted by $\mathcal{D}(A)$.*

In §2.3, we will define the operational semantics of RandML in terms of the distribution monad.

Lemma 2.2. *The* discrete distribution monad *induced by $\mathcal{D}$ has operations*

$$\mathrm{ret} : A \to \mathcal{D}(A) \qquad \mathrm{bind} : (A \to \mathcal{D}(B)) \to \mathcal{D}(A) \to \mathcal{D}(B)$$

$$\mathrm{ret}(a)(a') \triangleq \begin{cases} 1 & \text{if } a = a' \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{bind}(f,\mu)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$$

We write $(\mu \gg\!= f)$ for $\mathrm{bind}(f,\mu)$.

2.2 Differential Privacy

For background information on DP see, *e.g.*, Dwork and Roth [19] and Cowan et al. [15].

Defining Privacy. The definition of DP captures the intuition that it is hard to reconstruct information about any individual in a database from the output of a dp program.

Definition 2.3. *A function $f : \mathrm{DB} \to \mathcal{D}(X)$ is (ε,δ)-differentially private (short: "f is (ε,δ)-dp") if $\Pr_{f(x)}[\phi] \leq e^{\varepsilon} \cdot \Pr_{f(y)}[\phi] + \delta$ for all adjacent $x, y : \mathrm{DB}$ and all predicates $\phi \subseteq X$. If f is $(\varepsilon, 0)$-dp we simply say that f is ε-dp.*

A function is thus (ε,δ)-dp if it amplifies the probability of *any* observation ϕ by at most e^{ε}, or by more than that with probability at most δ. A small value for ε and δ thus means strong privacy guarantees. The definition of DP is parametrized by a type of databases DB and an adjacency relation. Clutch-DP can work with any database type and adjacency relation, but a common choice is to think of a database as a list of rows where each row is a tuple of a fixed size. If the type of databases comes with a notion of distance $d_{\mathrm{DB}} : \mathrm{DB} \to \mathbb{R}_{\geq 0}$ we say that two databases x, y are adjacent if $d_{\mathrm{DB}}(x,y) \leq 1$. For instance, if $d_{\mathrm{DB}}(x,y)$ is the number of rows where x and y differ, then adjacency means they only differ in one row.

Adding Noise: The Laplace Mechanism. To prove that any program is dp, we need primitives that add random noise. The prototypical example of a noise mechanism that achieves DP is the Laplacian distribution. However, since the Laplacian is a continuous distribution on $\mathbb{R}$, an implementation of a Laplacian sampler would have to work with exact real arithmetic, since implementations using floats lead to well-known privacy bugs [40].

[3] The extended version of this paper including the appendix is available as [26].

We therefore work with the *discrete Laplacian* (Fig. 4), the distribution on $\mathbb{Z}$ obtained by discretizing the continuous Laplacian. The discrete Laplacian with scale parameter ε and mean m has as probability mass function[4] [29, 30]:

$$\mathcal{L}^m_\varepsilon(v) \triangleq \frac{1}{W} \cdot e^{-\varepsilon \cdot |v-m|} \qquad \text{where } W \triangleq \sum_{z \in \mathbb{Z}} e^{-\varepsilon|z|} \tag{1}$$

Interpreting adjacency on $\mathbb{Z}$ as being at distance at most c, we have the following result.

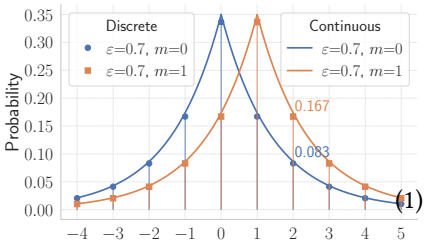


Fig. 4. Continuous and discrete Laplacian, with $\mathcal{L}^m_{0.7}(v)$ for v=2 and $m \in \{0, 1\}$, demonstrating 0.7-DP: $\mathcal{L}^1_{0.7}(2) \le e^{0.7} \cdot \mathcal{L}^0_{0.7}(2)$.

Theorem 2.4 ([22]). *$(\lambda\, m.\ \mathcal{L}^m_{\varepsilon/c}) : \mathbb{Z} \to \mathcal{D}(\mathbb{Z})$ is ε-dp.*

As a direct consequence, given any function $f : \mathsf{DB} \to \mathbb{Z}$ such that $|f(x) - f(y)| \le 1$ for adjacent datasets $x, y : \mathsf{DB}$, the function $(\lambda\, m.\ \mathcal{L}^m_\varepsilon) \circ f : \mathsf{DB} \to \mathcal{D}(\mathbb{Z})$ is ε-dp.

Composing DP. Differentially private functions satisfy a number of composition laws that can help structure and simplify the privacy verification of larger programs.

Lemma 2.5. *DP is* stable under post-processing*: if $f : A \to \mathcal{D}(B)$ is (ε, δ)-dp then for any $g : B \to C$, the function $\lambda\, x.\ (f(x) \gg\!= \lambda\, y.\ \mathrm{ret}\ g(y)) : A \to \mathcal{D}(C)$ is (ε, δ)-dp.*

When two dp functions are composed sequentially, their privacy parameters add up. Due to the post-processing property this holds even if the later computations can see the results of earlier computations. We only show the case for two functions, but the lemma directly generalizes to arbitrary k-fold composition for $k \in \mathbb{N}$.

Lemma 2.6 (Sequential Composition). *Let $f : \mathsf{DB} \to \mathcal{D}(B)$ be $(\varepsilon_1, \delta_1)$-dp and let g be a function $g : \mathsf{DB} \times B \to \mathcal{D}(C)$ such that $(\lambda\, x.\ g(x, b)) : \mathsf{DB} \to \mathcal{D}(C)$ is $(\varepsilon_2, \delta_2)$-dp for all $b \in B$. Then $\lambda\, x.\ (f(x) \gg\!= \lambda\, b.\ g(x, b))$ is $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-dp.*

Another useful composition law holds for functions which increase the distance between inputs by at most a fixed amount c in the following sense.

Definition 2.7. *We say that $f : A \to B$ is* c-sensitive *(also: "c-stable") if for all $x, y \in A$, the bound $d_B(f(x), f(y)) \le c \cdot d_A(x, y)$ holds, where the distances d_A, d_B are taken with respect to a metric space structure on A and B.*

Note that if $f : A \to B$ is c-sensitive and $g : B \to C$ is d-sensitive then $g \circ f$ is $(c{\cdot}d)$-sensitive. The following *metric composition law* then generalizes the remark following Theorem 2.4.

Lemma 2.8. *If $f : \mathsf{DB} \to \mathbb{Z}$ is c-sensitive then $(\lambda\, m.\ \mathcal{L}^m_{\varepsilon/c}) \circ f : \mathsf{DB} \to \mathcal{D}(\mathbb{Z})$ is ε-dp.*

2.3 The Language: Randomized ML

The RandML language that we consider is an ML-like language with higher-order recursive functions and higher-order state that we extend with an operator Laplace *a b m* that samples from the

[4]NB: The weight W is a geometric series over $\mathbb{Z}$ (hence why $\mathcal{L}_\varepsilon$ is also called the (two-sided) ε-geometric) with the closed form $\mathcal{L}^m_\varepsilon(v) = \frac{e^\varepsilon - 1}{e^\varepsilon + 1} \cdot e^{-\varepsilon \cdot |v-m|}$ [13]. We adopt the convention that $\mathcal{L}^m_\varepsilon = \mathrm{ret}(m)$ if $\varepsilon \le 0$.

Laplacian with scale (a/b) and mean m. The syntax is defined by the grammar below.

$$v, w \in \mathit{Val} ::= z \in \mathbb{Z} \mid b \in \mathbb{B} \mid () \mid \ell \in \mathit{Loc} \mid \mathsf{rec}\ \mathsf{f}\ \mathsf{x} = e \mid (v, w) \mid \mathsf{inl}\, v \mid \mathsf{inr}\, v$$
$$e \in \mathit{Expr} ::= v \mid \mathsf{x} \mid \mathsf{rec}\ \mathsf{f}\ \mathsf{x} = e \mid e_1\ e_2 \mid e_1 + e_2 \mid e_1 - e_2 \mid \ldots \mid \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid (e_1, e_2) \mid \mathsf{fst}\, e \mid \ldots$$
$$\mathsf{ref}\ e_1 \mid\ !\, e \mid e_1 \leftarrow e_2 \mid \mathsf{Laplace}\ e_1\ e_2\ e_3 \mid \ldots$$
$$K \in \mathit{Ectx} ::= - \mid e\, K \mid K\, v \mid \mathsf{ref}\ K \mid\ !\, K \mid e \leftarrow K \mid K \leftarrow v \mid \mathsf{Laplace}\, e_1\ e_2\ K \mid \mathsf{Laplace}\, e\ K\ v \mid \ldots$$
$$\sigma \in \mathit{State} \triangleq \mathit{Loc} \xrightarrow{\mathsf{fin}} \mathit{Val} \qquad \rho \in \mathit{Cfg} \triangleq \mathit{Expr} \times \mathit{State}$$

In RandML, $\mathsf{ref}\ e_1$ allocates a new reference containing the value returned by e_1, $!\, e$ dereferences the location e evaluates to, and $e_1 \leftarrow e_2$ evaluates e_2 and assigns the result to the location that e_1 evaluates to. We may refer to a recursive function value $\mathsf{rec}\ \mathsf{f}\ \mathsf{x} = e$ by its local name f. The heap is represented as a (partial) finite map from locations to values, and evaluation happens right to left as indicated by the evaluation context grammar *Ectx*.

The expression $\mathsf{Laplace}\ a\ b\ m$ samples from the discrete Laplacian with scale $\varepsilon = a/b$ and mean m. To avoid unnecessary complications with adding real numbers to the programming language, we require the scale ε to be a rational number. Formally, in RandML, Laplace takes three integers as input, but to keep our notation free from clutter we will simply write $\mathsf{Laplace}\ \varepsilon\ m$ instead of $\mathsf{Laplace}\ a\ b\ m$ with $\varepsilon = a/b$.

Operational Semantics. Program execution is defined by iterating $\mathsf{step} : \mathit{Cfg} \to \mathcal{D}(\mathit{Cfg})$, where $\mathsf{step}(\rho)$ is the distribution induced by the single step reduction of the configuration ρ. The semantics is mostly standard. We first define head reduction and then lift it to reduction in an evaluation context K. All non-probabilistic constructs reduce deterministically as usual, *e.g.*, $\mathsf{step}((\lambda\, \mathsf{x}.\ e)\, v, \sigma) = \mathsf{ret}(e[v/\mathsf{x}], \sigma)$. We write $e \rightsquigarrow e'$ if the evaluation is deterministic and holds independently of the state, *e.g.*, $(\lambda\, \mathsf{x}.\ e)\, v \rightsquigarrow e[v/\mathsf{x}]$ and $\mathsf{fst}(v_1, v_2) \rightsquigarrow v_1$. The sampling operator $\mathsf{Laplace}\ a\ b\ m$ reduces according to the Laplacian with scale a/b and mean m, *i.e.*,

$$\mathsf{step}(\mathsf{Laplace}\ a\ b\ m, \sigma)(v, \sigma) \triangleq \begin{cases} \mathcal{L}^m_{a/b}(v) & \text{for } v \in \mathbb{Z}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

With the single step reduction $\mathsf{step} : \mathit{Cfg} \to \mathcal{D}(\mathit{Cfg})$ defined, we next define a step-stratified execution probability $\mathsf{exec}_n : \mathit{Cfg} \to \mathcal{D}(\mathit{Val})$ by induction on n:

$$\mathsf{exec}_0(e, \sigma)(v) \triangleq \begin{cases} 1 & \text{if } e \in \mathit{Val} \wedge e = v, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathsf{exec}_{n+1}(e, \sigma)(v) \triangleq \begin{cases} 1 & \text{if } e \in \mathit{Val} \wedge e = v, \\ \sum_{(e', \sigma') \in \mathit{Expr} \times \mathit{State}} \mathsf{step}(e, \sigma)(e', \sigma') \cdot \mathsf{exec}_n(e', \sigma')(v) & \text{otherwise.} \end{cases}$$

That is, $\mathsf{exec}_n(e, \sigma)(v)$ is the probability of stepping from the configuration (e, σ) to a value v in at most n steps. The probability that an execution, starting from configuration ρ, reaches a value v is taken as the limit of its stratified approximations, which exists by monotonicity and boundedness:

$$\mathsf{exec}(\rho)(v) \triangleq \lim_{n \to \infty} \mathsf{exec}_n(\rho)(v)$$

The interpretation of programs as distributions induces a natural notion of (ε, δ)-DP for RandML programs. Concretely, if $f \in \mathit{Expr}$ is a RandML function then for adjacent databases x, y it should be the case that for all states σ,

$$\Pr_{\mathsf{exec}(f\ \mathsf{inj}(x),\ \sigma)}[\phi] \leq e^{\varepsilon} \cdot \Pr_{\mathsf{exec}(f\ \mathsf{inj}(y),\ \sigma)}[\phi] + \delta \tag{3}$$

where $\mathsf{inj} : \mathsf{DB} \to \mathit{Val}$ embeds the type of databases into RandML values (we usually omit inj).

Note that in particular $f = (\lambda\, \mathsf{x}.\ \mathsf{Laplace}\, \varepsilon\, \mathsf{x})$ is ε-dp in the sense of (3) by the definition of the operational semantics (2) and Theorem 2.4.

3 Program Logic

In this section, we introduce the Clutch-DP logic, the soundness theorem of Clutch-DP, which connects it to DP, and the new logical connectives and rules pertaining to the privacy budget reasoning. Our logic takes inspiration from Approxis [27], but we remark that the underlying model is different, and that it is designed to support our novel reasoning principles for DP via multiplicative error credits, rules for the Laplacian, and new composition laws.

The Logical Connectives. Clutch-DP is built on top of the Iris separation logic framework [31] and inherits many of Iris's logical connectives, a selection of which is shown below. Most of the propositions are standard, such as separating conjunction $P * Q$ and separating implication $P \mathrel{-\!\!*} Q$.

$$P, Q \in \mathit{iProp} ::= \mathsf{True} \mid \mathsf{False} \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid \forall x.\, P \mid \exists x.\, P \mid P * Q \mid P \mathrel{-\!\!*} Q \mid$$
$$\ell \mapsto v \mid \ell \mapsto_{\mathrm{s}} v \mid \lightning^{\times}(\varepsilon) \mid \lightning^{+}(\delta) \mid \{P\}\ e \precsim e'\ \{v\ v'.\, Q\} \mid \ldots$$

Since DP is a relational property, Clutch-DP is a relational program logic that proves properties about the execution of two programs e and e'. The central logical connective capturing the relationship between e and e' is the *Hoare quadruple*:

$$\{P\}\ e \precsim e'\ \{v\ v'.\, Q\} \tag{4}$$

Intuitively, this quadruple asserts that under the precondition P, if the two programs e and e' evaluate to results v and v' respectively, then the postcondition Q holds. Since the pre- and postcondition range over arbitrary Clutch-DP assertions, they can refer to the state or contain nested Hoare quadruples, which will be useful to write specifications about higher-order functions.

Logical assertions about the heap come in two versions, one for e and one for e'. The heap points-to assertion that denotes ownership of location ℓ for the left-hand side program e is written as $\ell \mapsto v$, and $\ell' \mapsto_{\mathrm{s}} v'$ denotes ownership of ℓ' for the right-hand side program e'.

Clutch-DP defines two kinds of resources that are novel with respect to standard Iris and which are inspired by Approxis [27]. The proposition $\lightning^{\times}(\varepsilon)$ asserts ownership of ε *multiplicative privacy credits* and $\lightning^{+}(\delta)$ similarly asserts ownership of δ *additive privacy credits*. These credits are a logical representation of the privacy budget of DP. Just as reasoning about the physical state of a program is logically captured by the points-to connective, so is reasoning about the privacy budget (ε, δ) logically expressed via privacy credits.[5]

Internalizing Privacy. We now have all of the ingredients to define DP in Clutch-DP.

Definition 3.1. *A RandML function* f *is* internally (ε, δ)-dp *if the following quadruple holds:*

$$\forall \mathsf{db}\, \mathsf{db}'.\ \left\{\mathsf{Adj}(\mathsf{db}, \mathsf{db}') * \lightning^{\times}(\varepsilon) * \lightning^{+}(\delta)\right\}\ \mathsf{f}\ \mathsf{db} \precsim \mathsf{f}\ \mathsf{db}'\ \{v, v'.\, v = v'\} \tag{5}$$

As with Definition 2.3, this notion is parametrized by an adjacency relation $\mathsf{Adj}(\mathsf{db}, \mathsf{db}')$ on inputs. For instance, we could represent the databases as lists, where each element represents one entry, and say two databases are adjacent if they differ only in one entry (when considered as multisets). We abbreviate the Hoare quadruple (5) as (ε, δ)-iDP(f) when the adjacency relation is clear from context, and simply ε-iDP(f) for the case where $\delta = 0$.

We call Definition 3.1 *internal* DP since it is a property of the RandML program f stated entirely in terms of the program logic. All reasoning about concrete probabilities is abstracted away via the use

[5] Contrary to heaps, the privacy budget pertains to a pair of program executions rather than to the left or right program, and hence there is no left- and right-hand version of credits; instead, they are shared.

of privacy credits. In contrast, the usual *external* definition of DP (Def. 2.3) for arbitrary functions and for RandML programs (Eq. (3)) is given in terms of plain mathematics, probabilities, and the operational semantics. Working with internal DP has two advantages: (1) It uses an expressive program logic, which allows us to reason about complex programs, whereas reasoning directly about the operational semantics would be infeasible. (2) We can "pay" some privacy credits to get an equality in the postcondition of Eq. (5), so that a client of f can simply assume that the results of (f db) and (f db′) are *identical* instead of having to work with the assumption that they are distributions at distance (ε, δ).

The precise meaning of Hoare quadruples and privacy credits can be understood through the following theorem which connects Clutch-DP to DP in the sense of Equation (3).

Theorem 3.2 (Soundness). *If* f *is internally* (ε, δ)*-dp then* f *is also externally* (ε, δ)*-dp.*

This theorem is a consequence of the adequacy of the semantic model of Clutch-DP (see §5).

3.1 Relational Separation Logic Rules

The standard rules of relational separation logic as used in, *e.g.*, [20] are also available in Clutch-DP. For instance, the Frame rule enables local reasoning by framing out R, and the Bind rule allows us to focus on sub-programs in evaluation contexts. The usual load and store rules for heap locations come in a left- and a right version, requiring ownership of the corresponding points-to connective.

$$\frac{\{P\}\ e \precsim e'\ \{Q\}}{\{P * R\}\ e \precsim e'\ \{Q * R\}}\ \textsc{Frame} \qquad \frac{\{P\}\ e \precsim e'\ \{v\,v'.\ R\} \qquad \forall v\,v'.\ \{R\}\ K[v] \precsim K'[v']\ \{Q\}}{\{P\}\ K[e] \precsim K'[e']\ \{Q\}}\ \textsc{Bind}$$

$$\frac{\{\ell \mapsto w\}\ w \precsim e'\ \{Q\}}{\{\ell \mapsto w\}\ !\ell \precsim e'\ \{Q\}}\ \textsc{Load-l} \qquad \frac{\{\ell' \mapsto_s w\}\ e \precsim w\ \{Q\}}{\{\ell' \mapsto_s w\}\ e \precsim\ !\ell'\ \{Q\}}\ \textsc{Load-r} \qquad \frac{\{\ell \mapsto w\}\ () \precsim e'\ \{Q\}}{\{\ell \mapsto v\}\ \ell \leftarrow w \precsim e'\ \{Q\}}\ \textsc{Store-l}$$

Fig. 5. Excerpt of the non-probabilistic rules of Clutch-DP.

It is worth noting that the rules in Figure 5 do *not* mention distributions, despite the fact that the operational semantics of RandML is probabilistic in general and references can, for instance, store randomly sampled values. Clutch-DP thus provides a convenient basis for program verification where reasoning about privacy is integrated as an *orthogonal* feature while preserving the familiar rules of relational higher-order separation logic.

3.2 Privacy Credit Laws

The privacy credit resources logically track two non-negative real numbers ε and δ corresponding to the privacy budget. In light of this intuition, one would expect that they support laws such as sequential composition (Lemma 2.6). Indeed, we can derive an internal version of this law from the primitive rules pertaining to privacy credits show in Figure 6 together with the structural rules for RandML in Figure 5.

$$↯^{\times}(\varepsilon_1 + \varepsilon_2) \dashv\vdash ↯^{\times}(\varepsilon_1) * ↯^{\times}(\varepsilon_2) \qquad ↯^{+}(\delta_1 + \delta_2) \dashv\vdash ↯^{+}(\delta_1) * ↯^{+}(\delta_2) \qquad ↯^{+}(1) \vdash \mathsf{False}$$

Fig. 6. Privacy credit laws.

Just as in Lemma 2.6, both multiplicative and additive privacy credits can be split into their summands[6] (and recombined). The last rule states that from 1 additive error credit, we can derive

[6]NB: Both kinds of privacy credit are split into separating conjunctions by using addition, not multiplication, as the resource algebra operation. The name "multiplicative privacy credit" for $↯^{\times}(\varepsilon)$ derives from the multiplicative factor e^{ε} in Def. 2.3.

False and hence anything. This makes sense if we recall that δ models a bound on an inequality between *probabilities* in the definition of DP (Def. 2.3), since any probability is bounded by 1, and hence any program is trivially $(\varepsilon, 1)$-dp.

The fact that the privacy budget is treated like any other separation logic resource and being able to split the budget enables *flexible* reasoning about privacy. Privacy credits are animated by the rules that govern their interaction with the operational semantics of RandML via sampling.

3.3 Rules for Sampling Noise

Privacy credits are consumed by the rules that reason about sampling operations. We introduce a rule Laplace-shift to reason about a pair of Laplacians, with the same parameter ε and means m, m'. By setting $k = 0$ (and thus, $|m - m'| \le c$), the rule lets us spend $\pounds^{\times}(c \cdot \varepsilon)$ to ensure that both Laplacians return the same result. If, instead, $k > 0$ and $c = 0$ (and thus, $|m - m'| = k$), the rule lets us conclude, without consuming privacy, that we will get two samples at distance k. The rule combines the two reasoning principles into one:

$$\frac{|k + m - m'| \le c}{\left\{\pounds^{\times}(c \cdot \varepsilon)\right\}\ \mathsf{Laplace}\ \varepsilon\ m \precsim \mathsf{Laplace}\ \varepsilon\ m'\ \{z\ z' .\ z' = z + k\}}\ \textsc{Laplace-shift}$$

A consequence of Laplace-shift (when $k = 0$) is that Laplace is internally private, *i.e.*, the Clutch-DP statement ε-iDP(λ x. Laplace ε x) is derivable. This internalizes DP of the Laplacian (Thm. 2.4).

Finally, we have the following novel rule for the Laplacian which can be used to *recover* some privacy credits. The idea is to partition the results into two groups of outcomes depending on some threshold T. If both Laplacians sample a result above T (in fact, above $T + 1$ for the right-hand side) the privacy credits are consumed. If, on the other hand, both results remain below their respective thresholds, then the privacy credits can be recovered. This yields a useful reasoning principle for the Laplacian that is applied, *e.g.*, in the verification of the `AboveThreshold` mechanism (§4.1.1).

$$\frac{|m - m'| \le 1 \qquad T \in \mathbb{Z}}{\left\{\pounds^{\times}(2\varepsilon)\right\}\ \mathsf{Laplace}\ \varepsilon\ m \precsim \mathsf{Laplace}\ \varepsilon\ m'\ \left\{z\ z' .\ \begin{array}{l}(T \le z\ \wedge\ T + 1 \le z') \quad \vee \\ (z < T\ \wedge\ z' < T + 1\ *\ \pounds^{\times}(2\varepsilon))\end{array}\right\}}\ \textsc{Laplace-choice}$$

Intuitively this rule is sound because it partitions the outcomes into disjoint events such that the total privacy budget for the two is bounded by the initial privacy budget. Despite its conceptual simplicity, constructing a sound model that validates Laplace-choice required substantial new insights in the form of a new composition theorem (see Theorem 5.2).

3.4 Internal Composition Laws

Working in a program logic makes it simple to internalize the various composition laws for DP, as presented in §2. First, we remark that the internal version of post-processing holds. This is stated in our logic through the following lemma:

Lemma 3.3 (Internal post-processing). *Let* f *be internally* (ε, δ)*-dp and assume* g *is "safe to execute" in the sense that* $\forall w.\ \{\mathsf{True}\}\ \mathrm{g}\ w \precsim \mathrm{g}\ w\ \{v\ v' .\ v = v'\}$ *holds. Then* $\mathrm{g} \circ \mathrm{f}$ *is internally* (ε, δ)*-dp.*

This lemma can be proven entirely within the logic as an immediate consequence of the fact that Clutch-DP validates the Bind rule. Similarly to the definition of internal DP, we can also recast the notion of sensitivity (Def. 2.7) in Clutch-DP.

Definition 3.4. *A RandML function* f *is* internally c-sensitive *(c-sens(f)) if the following holds:*

$$\forall x\ y : A.\ \{\mathsf{True}\}\ \mathrm{f}\ x \precsim \mathrm{f}\ y\ \{v\ v' .\ d_B(v, v') \le c \cdot d_A(x, y)\}$$

As before, the definition is parametrized by two distances at types A and B, which can then be internalized as distances between values. The internal equivalent of metric composition then follows.

Lemma 3.5. $\forall\, \mathsf{f}\, c\, \varepsilon.\ c\text{-sens}(\mathsf{f}) \multimap \varepsilon\text{-iDP}(\lambda\, \mathsf{x}.\ \mathsf{Laplace}\ (\varepsilon/c)\ (\mathsf{f}\ \mathsf{x}))$ *is derivable in Clutch-DP.*

4 Reusable Specs for Privacy Mechanisms

Privacy *mechanisms* are the primitive building blocks of DP. Mechanisms sample the appropriate noise for a given data processing task. We give specifications for some widely used mechanisms and demonstrate that clients can be verified based on these abstract specs.

Our first example will be the Above Threshold (AT) mechanism, and we will see how to (1) prove that it satisfies an abstract specification capturing its privacy, (2) build the Sparse Vector Technique (SVT) from it, and (3) use AT to calculate the clipping bounds required to privately compute averages over a dataset. Both (2) and (3) use the same abstract specification of AT.

Besides the case studies presented hereafter, we also verified the privacy of the Report Noisy Max mechanism, which is of note because—as with AT—its privacy does not follow from composition laws but requires careful manipulation of the privacy budget and the use of budget- and state-dependent invariants. Details can be found in the Appendix of [26]. We will present the most interesting ideas for each proof. Full proofs from the rules of Clutch-DP are available in our Rocq formalization.

4.1 Sparse Vector Technique

Suppose we have an incoming sequence of (1-sensitive) queries on a database and a fixed privacy budget. The Sparse Vector Technique (SVT) allows us to fix a threshold T and release, in a private manner, whether the result of each query exceeds T or not. The benefit of SVT is that one only has to spend privacy budget on the "successful" queries which do indeed exceed T and are released; the results of the queries that do not exceed the threshold can be computed (and discarded) without incurring a privacy cost. We can thus set in advance a maximum number N of successful queries to be released, and keep answering incoming queries interactively until N is reached. The SVT is usually implemented in terms of the Above Threshold mechanism (AT), which finds a *single* query above T. SVT then simply runs N iterations of AT. The privacy cost of SVT is N times the cost of finding one query above T.

The SVT is of interest for verification because several buggy privacy proofs have been published (see Lyu et al. [37] for a survey). As the survey explains, the SVT is particularly interesting in the interactive setting; in the non-interactive setting, one can use the Exponential Mechanism instead and get more accurate results. We explore two subtleties of SVT: how to perform the privacy analysis of AT, and how to build an interactive algorithm out of AT. The privacy analysis of AT (and SVT) is challenging because it requires fine-grained reasoning about the privacy budget that *cannot* be justified by the sequential composition law alone.

4.1.1 Above Threshold. The Above Threshold mechanism can be used to evaluate queries on a database until one query returns a result that exceeds a specified threshold T. The implementation of above_threshold in Figure 7 initializes the noisy threshold $\hat{\mathsf{T}}$ and returns a function to run queries interactively. This function receives a query, computes its result, adds additional noise to it and checks whether the noisy result exceeds $\hat{\mathsf{T}}$.

Example 4.1. Suppose we want to privately compute the number of even numbers in a list and check whether it exceeds a threshold of 3. If we run AT with a high privacy budget (*e.g.*, $\varepsilon = 10$) there is very little noise added and we are very likely to disclose the true result and disclose private

```
let above_threshold ε T = let T̂ = Laplace (ε / 2) T in
                          let f q db = let x = q db in let y = Laplace (ε / 4) x in T̂ ≤ y
                          in f
```

Fig. 7. The Above Threshold mechanism.

information. Concretely, with probability about 92%,

```
above_threshold 10 3 (List.count (λ x. x mod 2 = 0))   [1, 2, 3, 4, 5]     →  false
above_threshold 10 3 (List.count (λ x. x mod 2 = 0))   [1, 2, 3, 4, 5, 6]  →  true
```

As the value of ε *decreases*, we are more likely to observe true in the first query or false in the second, *i.e.*, privacy improves while accuracy deteriorates.

Our specification (6) captures the idea that above_threshold is *interactive*: after initialization, the function f can be used to compare a query to the (noisy) threshold T̂ until a result above T̂ is found, but queries can be supplied and *chosen* one by one after observing the result of previous queries.

$$\begin{array}{l}\{\lightning^{\times}(\varepsilon)\}\\ \quad \mathsf{above\_threshold}\ \varepsilon\ \mathsf{T} \precsim \mathsf{above\_threshold}\ \varepsilon\ \mathsf{T}\\ \left\{\begin{array}{ll} & \exists \mathit{AUTH}.\ \mathit{AUTH} * \\ \mathsf{f}\ \mathsf{f}'. & \quad \forall\ \mathsf{db}\ \mathsf{db}'\ \mathsf{q}. \quad \begin{array}{l}\{\mathit{AUTH} * \mathsf{Adj}(\mathsf{db}, \mathsf{db}') * \text{1-sens}(\mathsf{q})\}\\ \mathsf{f}\ \mathsf{q}\ \mathsf{db} \precsim \mathsf{f}'\ \mathsf{q}\ \mathsf{db}'\\ \{b\ b'.\ b = b' * \mathsf{if}\ \mathsf{not}\ b\ \ \mathsf{then}\ \mathit{AUTH}\}\end{array}\end{array}\right\}\end{array} \tag{6}$$

Let us unpack the specification piece by piece. After initializing the mechanism with a privacy budget of ε, we obtain a pair of functions f and f′, where both functions represent the same computation but with a priori different randomly sampled values of T̂, as well as an abstract "authorization token" *AUTH*. The Hoare quadruple for f, f′ in the postcondition indicates that so long as we have the *AUTH* token, running the functions on a 1-sensitive query q and adjacent datasets (1) produces the same result for both q db and q db′, *i.e.*, the computation is private, and (2) only consumes the *AUTH* token if q db is above T̂. If the result of the comparison is false, *AUTH* can be recovered in the postcondition and we can continue to privately look for a query that exceeds T̂. Note that the initial budget $\lightning^{\times}(\varepsilon)$ is only spent once at initialization, but we can still privately run as many queries as it takes to get a result above the threshold.

Proving Privacy. The proof of the specification (6) proceeds as follows:

(1) We split $\lightning^{\times}(\varepsilon)$ into $\lightning^{\times}(\varepsilon/2) * \lightning^{\times}(\varepsilon/2)$ and use the first half to pay for the Laplace-shift rule with parameters $m = m' = \mathsf{T}, c = 1, k = 1$. We thus obtain related results T̂, T̂′ for the left- and right-hand program such that T̂′ = T̂ + 1. Forcing the two noisy thresholds to be at distance 1 rather than equal is a standard "trick" in the analysis of Above Threshold.
(2) We pick $\mathit{AUTH} \triangleq \lightning^{\times}(\varepsilon/2)$ and use the remaining budget to provide the initial *AUTH* token.
(3) We now have to show the specification for

$$\mathsf{f}\ \ \mathsf{q}\ \mathsf{db} \ \triangleq\ \mathsf{let}\ \mathsf{x}\ = \mathsf{q}\ \mathsf{db}\ \ \mathsf{in}\ \mathsf{let}\ \mathsf{y}\ = \mathsf{Laplace}\ (\varepsilon\ /\ 4)\ \mathsf{x}\ \ \mathsf{in}\ \hat{\mathsf{T}}\ \leq \mathsf{y}$$

$$\mathsf{f}'\ \mathsf{q}\ \mathsf{db}' \ \triangleq\ \mathsf{let}\ \mathsf{x}' = \mathsf{q}\ \mathsf{db}'\ \mathsf{in}\ \mathsf{let}\ \mathsf{y}' = \mathsf{Laplace}\ (\varepsilon\ /\ 4)\ \mathsf{x}'\ \mathsf{in}\ \hat{\mathsf{T}}' \leq \mathsf{y}'$$

(4) By sensitivity of q and adjacency of the databases, x and x′ are at distance at most 1.
(5) By the definition of *AUTH*, the precondition of the refinement confers us a privacy budget of $\lightning^{\times}(\varepsilon/2)$. We use this budget to apply the Laplace-choice rule and partition the outcomes (y, y′) of the remaining Laplace sampling into two mutually exclusive cases:
   • T̂ ≤ y and T̂′ ≤ y′. In this case, both of the comparisons return true.

- $y < \hat{T}$ and $y' < \hat{T}'$. In this case, both comparisons return false, the rule does not consume the privacy budget and we can return $AUTH$.

Either way, f and f′ return the same result b, and if b = false then $AUTH$ is returned too. □

To see that the specification (6) is indeed useful, we will now use it to verify the privacy of the interactive sparse vector technique.

4.1.2 An interactive Sparse Vector Technique. We prove privacy of the interactive SVT directly from the abstract specification of the AT mechanism (6). SVT orchestrates repeated invocations of above_threshold in order to identify the first N of the queries that exceed the threshold T. Unlike a purely batch-style algorithm, the SVT exposes a streaming interface that allows queries to be submitted interactively, *i.e.*, depending on the results of earlier queries. This makes the verification of privacy significantly more challenging, as the set of queries cannot be fixed in advance but may depend on previously released (noisy) information. In Clutch-DP however, the proof that sparse_vector is dp is a relatively straightforward consequence of the privacy of above_threshold.

The implementation in §4.1.2 works as follows. It maintains two pieces of mutable state: a reference AT to the current Above Threshold instance and a counter that tracks how many additional true results (*i.e.*, queries that exceed the threshold) may still be released. Each invocation of the returned function f runs the current above_threshold instance on a new query q and database db, producing a boolean result b. If b is true and the counter has not yet reached zero, the mechanism consumes ε privacy credits and reinitializes AT with a fresh Above Threshold instance. Otherwise, the counter and function reference remain unchanged. The caller may then use the result b to decide which query to issue next.

```
let sparse_vector ε T N =
    let AT = ref (above_threshold ε T) in
    let counter = ref (N − 1) in
    let f q db = let b = (! AT) q db in
                 if ! counter > 0 && b then
                   ( counter ← (! counter − 1) ;
                     AT ← above_threshold ε T ) ;
                 b
    in f
```

```
let SVT_stream ε T N QS db =
    let f = sparse_vector ε T N in
    let rec iter i bs =
      if i = N then List.reverse bs
      else let q = QS bs in
           let b = f q db in
           iter (if b then (i + 1) else i) (b :: bs)
    in iter 0 []
```

Fig. 8. The Sparse Vector Technique and a streaming client.

The specification (7) formalizes the intuition that the interactive SVT behaves like a private state machine that can be queried multiple times while consuming a fixed privacy budget.

$$\left\{ \lightning^{\times}(\mathsf{N} \cdot \varepsilon) \right\}$$
$$\mathsf{sparse\_vector}\ \varepsilon\ \mathsf{T}\ \mathsf{N} \precsim \mathsf{sparse\_vector}\ \varepsilon\ \mathsf{T}\ \mathsf{N}$$
$$\left\{ \mathsf{f}\ \mathsf{f}'\ . \quad \begin{array}{l} \exists\, iSVT.\ iSVT(\mathsf{N}) * \\ \forall\ \mathsf{db}\ \mathsf{db}'\ \mathsf{q}\ n. \quad \begin{array}{l} \{iSVT(n+1) * \mathsf{Adj}(\mathsf{db}, \mathsf{db}') * \text{1-sens}(\mathsf{q})\} \\ \mathsf{f}\ \mathsf{q}\ \mathsf{db} \precsim \mathsf{f}'\ \mathsf{q}\ \mathsf{db}' \\ \{b\ b'\ .\ b = b' * iSVT(\text{if } b \text{ then } n \text{ else } n+1)\} \end{array} \end{array} \right\} \tag{7}$$

Initially, the mechanism owns a total privacy resource of $\lightning^{\times}(\mathsf{N} \cdot \varepsilon)$, corresponding to N possible above-T releases. The abstract token $iSVT(K)$ in the postcondition tracks the remaining number K of true results we can release. The returned function f satisfies the nested quadruple: if we own at least $iSVT(1)$, we can run a 1-sensitive query on adjacent databases and ensure we get the same result; if the result is true (i.e., the query exceeds the threshold), we decrease $iSVT$ by 1, otherwise we get back our initial token. This specification thus captures both the privacy accounting and the

interactive behavior of SVT: the mechanism remains private for any adaptively chosen sequence of 1-sensitive queries until the allotted number N of positive releases has been exhausted.

Notably, the proof of this specification treats AT abstractly and relies only on its specification (6), not its implementation, which gives us a more modular analysis. The proof relies on keeping a simple invariant over the counter, the AT reference and the remaining privacy budget.

To summarize, this implementation and our verification of it has several noteworthy features:

- the SVT is a client of AT via an abstract specification,
- its interactive interface is specified through nested Hoare quadruples, and
- it requires storing a private higher-order function in a reference.

Next, we demonstrate that our specifications are expressive enough for reuse by different clients.

4.1.3 SVT Client: Streams of Queries. The standard textbook account of SVT [19] presents it as an algorithm that takes in a *stream* of queries QS and produces a list of booleans bs where b_i indicates whether the i-th query was above the noisy threshold. The stream is represented as a (possibly stateful) function that produces a new query on each invocation, and interactivity is modeled by the fact that each time a new query is requested, QS gets access to the booleans bs resulting from the preceding queries.

We can directly prove that the implementation SVT_stream (§4.1.2) is private by applying the generic specification for SVT. A user of SVT_stream only has to satisfy the textbook assumption that all of the queries are indeed 1-sensitive. The privacy reasoning is encapsulated in (7).

4.1.4 Above Threshold Client: auto_avg. As a last application of the Above Threshold mechanism, we analyze the auto_avg client that privately computes the average of a dataset. This example is taken from the online textbook [41, Chapter 10].

The fact that both sparse_vector and auto_avg can both be verified against the same abstract interface for above_threshold is good evidence that our specifications are indeed reusable and can be used to verify libraries without having to worry about implementation details.

```
let auto_avg bnds ε db =
  let bound = get_clip_bound bnds ε db in
  let sum = clip_sum bound db in
  let sum_noisy = Laplace (ε / bound) sum in
  let count_noisy = Laplace ε (List.length db) in
  sum_noisy / count_noisy

let get_clip_bound bnds ε db =
  let qs = List.map (λ b. (b, mk_query b)) bnds in
  let (bound, _) = AT_list ε 0 db qs in
  bound

let mk_query b db =
  (clip_sum b db) − (clip_sum (b + 1) db)

let clip_sum bound db =
  List.sum (List.clip bound db)

let AT_list ε T db qs =
  let AT = above_threshold ε T in
  List.find (λ (bound, q). AT q db) qs
```

Fig. 9. Privately computing the average of a list of data.

To privately compute the average of a dataset it is not enough to first compute the average and then add ε Laplacian noise for a fixed ε, as this may leak information about the size of the dataset. The noise has to be calibrated to the largest element—but that value in itself is private information!

The solution adopted in the implementation of auto_avg in Figure 9 is to "clip" the elements of the database to lie in a bounded range $[0, B]$. If two adjacent databases are clipped to the same bound, their sum can differ by at most B. In other words, we can prove that clip_sum B is internally B-sensitive. We can apply internal metric composition (Lem. 3.5) to show that computing $\mathsf{sum}_{\mathsf{noisy}}$ by adding to the clipped sum Laplacian noise with scale (ε/B) is ε-private. Therefore, auto_avg

achieves $(3{\cdot}\varepsilon)$-DP, where the budget is divided equally between the call to get_clip_bound and the two calls to Laplace:

$$\left\{ \lightning^{\times}(3 \cdot \varepsilon) \ * \ \mathsf{Adj}(\mathsf{db}, \mathsf{db}') \right\} \ \mathsf{auto\_avg}\ \mathsf{bnds}\ \varepsilon\ \mathsf{db} \precsim \mathsf{auto\_avg}\ \mathsf{bnds}\ \varepsilon\ \mathsf{db}'\ \{x\ x'.\ x = x'\}$$

The utility of auto_avg stems from carefully choosing B. Given a list of candidate bounds bnds we can do this privately via the AT mechanism. The function get_clip_bound to finds the first value b in bnds such that the sum of elements in db stops increasing if the clipping bound is relaxed from b to b + 1. Testing this for all values in bnds via mk_query is 1-sensitive. Therefore we can directly apply the specification (6) for above_threshold to derive that get_clip_bound is ε-dp.

4.2 Privacy Filters

A common implementation technique for DP in general-purpose programming languages is to explicitly track the remaining privacy budget as a program variable. At the beginning of a data analysis, this variable is initialized to the global privacy budget ε, and it must remain non-negative throughout the program execution. So long as care is taken to decrement the budget every time (noisy) data is released, the whole data analysis is ε-private. To ensure that these rules are respected, the management of the privacy budget is commonly encapsulated in a *privacy filter*, a higher-order function that runs a computation only if there is sufficient budget for it. This programming pattern provides a separation of concerns: if the privacy analysis of the individual computations is correct, and the filter is correctly implemented, then the entire computation is private. This allows us to verify the different components (privacy filter and mechanisms) modularly.

```
let privacy_filter ε_budget =
  let ε_rem = ref ε_budget in
  let try_run ε_cost f =
    if !ε_rem < ε_cost then
      None
    else
      ε_rem ← !ε_rem − ε_cost ;
      Some (f ())
  in try_run
```

```
let adaptive_count ε_coarse ε_precise T ε_budget predicates db =
  let try_run = privacy_filter ε_budget in
  List.map (λ pred. let n_exact = List.count pred db in
                    let g _ = Laplace ε_precise n_exact in
                    let f _ = let n_coarse = Laplace ε_coarse n_exact in
                              let n_precise = if T < n_coarse
                                then try_run ε_precise g
                                else None in
                              (n_coarse, n_precise)
    in try_run ε_coarse f)
    predicates
```

Fig. 10. Implementations of a Privacy Filter and Adaptive Count.

The implementation privacy_filter in Figure 10 works as follows. Upon initialization it allocates a reference that tracks the remaining privacy budget and returns a closure try_run that can be used to run private computations in an interactive manner. If a client tries to run a computation with cost exceeding the remaining budget, try_run does not run the computation, otherwise it decreases the budget by the cost, runs the computation, and returns the result.

For the sake of simplicity we present a privacy filter that only tracks the ε-budget, but the method directly generalizes to (ε, δ)-privacy filters.

4.2.1 Proving Privacy of Privacy Filters. The high-level intuition for the privacy filter is that it should never exceed the budget that was initially set up so long as any client that calls try_run $\varepsilon_{\text{cost}}$ f ensures that f is indeed $\varepsilon_{\text{cost}}$-dp. This intuition is captured by the following specification.

$$\forall\, \varepsilon_{\text{budget}}\,. \quad \begin{array}{l} \left\{ \lightning^{\times}\left(\varepsilon_{\text{budget}}\right) \right\} \\ \quad \mathsf{privacy\_filter}\ \varepsilon_{\text{budget}} \precsim \mathsf{privacy\_filter}\ \varepsilon_{\text{budget}} \\ \{\mathsf{try\_run}\ \mathsf{try\_run}'\,.\, \exists\ \mathit{iPF}.\ \mathit{iPF} \ * \ \mathsf{try\_run}\text{-}\mathit{spec}\} \end{array} \tag{8}$$

where try_run-*spec* is defined as

$$\forall\, \varepsilon_{\text{cost}}\ \mathsf{f}\ \mathsf{f}'\ \mathit{Inv}_{\mathsf{f}}\,.\quad \left\{\begin{array}{l} \mathit{iPF} * \mathit{Inv}_{\mathsf{f}} \\ \quad * \left\{ \lightning^{\times}(\varepsilon_{\text{cost}}) \ * \ \mathit{iPF} * \mathit{Inv}_{\mathsf{f}} \right\} \mathsf{f}\ ()\ \precsim\ \mathsf{f}'\ ()\ \{v\ v'\,.\, v = v' * \mathit{iPF} * \mathit{Inv}_{\mathsf{f}}\} \end{array}\right\} \\ \quad \mathsf{try\_run}\ \varepsilon_{\text{cost}}\ \mathsf{f} \precsim \mathsf{try\_run}'\ \varepsilon_{\text{cost}}\ \mathsf{f}' \\ \quad \{b\ b'\,.\, b = b' * \mathit{iPF} * \mathit{Inv}_{\mathsf{f}}\} \qquad (9)$$

The existentially quantified *iPF* token in the postcondition of (8) represents a client's ability to execute computations privately via try_run. The specification (9) defines the behavior of $\mathsf{try\_run}\ \varepsilon_{\text{cost}}\ \mathsf{f}$. Assuming that for a privacy cost of $\varepsilon_{\text{cost}}$ the functions f and f′ produce equal results and maintain an invariant $\mathit{Inv}_{\mathsf{f}}$, calling them through try_run (and try_run′ respectively) satisfies the same invariant regardless of whether there actually is enough budget left to execute f. Since f may itself contain calls to try_run, the specification for (f, f′) has access to the *iPF* token. We give a simple application of this expressivity in the form of nested calls to the privacy filter in §4.2.2.

The proof of the specification in Clutch-DP is straightforward. We define the token *iPF* as

$$\exists \varepsilon.\ \lightning^{\times}(\varepsilon) \ * \ \varepsilon_{\text{rem}} \mapsto \varepsilon \ * \ \varepsilon'_{\text{rem}} \mapsto_{\mathsf{s}} \varepsilon\ .$$

The link between the logical resource representing ownership of the error budget and the program state tracking the remaining budget allows us to conclude that the call to Some (f ()) in the definition of try_run is only executed when sufficient privacy budget remains to satisfy the precondition of (f, f′) in (9), and hence the invariant $\mathit{Inv}_{\mathsf{f}}$ is satisfied. In case try_run decides that the budget is insufficient for f, the invariant is trivially preserved.

The power of this specification for privacy_filter lies in the fact that a client does not have to perform *any* privacy accounting or reasoning whatsoever for $\mathsf{try\_run}\ \varepsilon_{\text{cost}}\ \mathsf{f}$! We can conveniently combine library functions to build a private f and run it without having to worry whether we still have enough budget: the filter ensures that the initial budget is never exceeded.

4.2.2 Client: Adaptive Counting. Another advantage of implementing DP through privacy filters is that it allows the data analyst to decide *dynamically* where the privacy budget should be spent, *i.e.*, the way the budget is spent can adapt to the results of prior analyses. This is especially useful in exploratory data analysis when it is unclear, a priori, what values the dataset ranges over.

The example adaptive_count in Figure 10 employs a form of adaptivity to privately count the number of elements of db that satisfy each of the tests in the list of predicates. First, a cheaper but less precise count is performed, consuming $\varepsilon_{\text{coarse}}$ privacy credits. Only if this yields a promising result that exceeds a threshold T, a more precise analysis is performed for an additional larger budget of $\varepsilon_{\text{precise}}$. The result of adaptive_count is thus a numeric estimate for each predicate, with a more precise value for a few "important" candidates.

A conservative privacy analysis would have to assume a worst-case cost of $\mathsf{length}(\mathsf{predicates}) \cdot (\varepsilon_{\text{coarse}} + \varepsilon_{\text{precise}})$ even if many of the coarse counts may in practice not exceed the threshold. By employing a privacy filter, we can instead fix a budget $\varepsilon_{\text{budget}}$ that we want to allocate to this analysis task and try to run the analysis so long as the filter has enough budget left. If only a few of the data entries exceed T, this allows to count many more predicates than the conservative analysis without a privacy filter.

In Clutch-DP, we can prove that adaptive_count is $\varepsilon_{\text{budget}}$-dp from the specification (8) because f and g meet the precondition of try_run, as they consume a budget of $\varepsilon_{\text{coarse}}$ and $\varepsilon_{\text{precise}}$ respectively.

4.3 Caching Techniques for DP

Interactive analysis with data-dependent queries is common in real-world workloads for DP. This poses a challenge for DP frameworks because it makes it impossible to statically avoid repeated

evaluation of certain queries, say, by refactoring code that requires the same result, and hence repeated queries inflate the privacy cost unnecessarily. This problem can be solved with a query cache that memoizes the results of a query on first execution and reuses this result upon repetition. One would hope that reusing a noisy result in a repeated query should be "for free" and consume no privacy budget. However, the privacy analysis of such a memoization method is subtle, because the privacy cost of a query depends on the history of queries, which is highly non-local information.

```
let mk_query_cache add_noise db =
  let cache = Map.init () in
  let run_cache q =
    match Map.get cache q with
    | Some x ⇒ x
    | None   ⇒ let x = add_noise q db in
               Map.set cache q x;
               x
  in run_cache
```

```
let map_cache add_noise qs db =
  let run_cache =
    mk_query_cache add_noise db in
  List.map run_cache qs
```

Fig. 11. Implementations of a cache and a client.

In this section, we reason about the privacy of the caching method introduced in Figure 3, which we implement in RandML through the algorithm shown in Figure 11. Our formalization crucially relies on our logic being able to support higher-order functions, local state, and a resource-based representation of the privacy budget.

4.3.1 Cache Spec: Repeated Queries are Free. Upon initialization, mk_query_cache allocates a mutable map cache and returns a closure run_cache that stores and looks up noisy query results in cache. This is reflected in (10) as the existentially quantified $\mathsf{iC}(M_{cache})$ resource. Initially, the map M_{cache} is empty, but it can be updated via run_cache as we will see next.

$$\begin{array}{l}\{\mathsf{Adj}(\mathsf{db}, \mathsf{db}')\} \\ \quad \mathsf{mk\_query\_cache\ add\_noise\ db} \preceq \mathsf{mk\_query\_cache\ add\_noise\ db}' \\ \{\mathsf{run\_cache}\ \mathsf{run\_cache}'.\ \exists \mathsf{iC}.\ \mathsf{iC}(\mathsf{Map.empty}) * \textit{spec-cached} * \textit{spec-fresh}\}\end{array} \tag{10}$$

where *spec-fresh* is defined as

$$\begin{array}{ll} & \left\{\mathsf{q} \notin \mathsf{dom}(M_{cache}) * \lightning^{\times}(\varepsilon) * \lightning^{+}(\delta) * (\varepsilon,\delta)\text{-}\mathsf{iDP}(\mathsf{add\_noise\ q}) * \mathsf{iC}(M_{cache})\right\} \\ \forall\, M_{cache}\ \mathsf{q}\,. & \quad \mathsf{run\_cache\ q} \preceq \mathsf{run\_cache}'\ \mathsf{q} \\ & \{v\ v'\,.\ v = v' * \mathsf{iC}(M_{cache}[\mathsf{q} \mapsto v])\}\end{array} \tag{11}$$

and *spec-cached* is defined as

$$\begin{array}{ll} & \{\mathsf{q} \in \mathsf{dom}(M_{cache}) * \mathsf{iC}(M_{cache})\} \\ \forall\, M_{cache}\ \mathsf{q}\,. & \quad \mathsf{run\_cache\ q} \preceq \mathsf{run\_cache}'\ \mathsf{q} \\ & \{v\ v'\,.\ v = v' * \mathsf{iC}(M_{cache}) * M_{cache}[\mathsf{q}] = v\}\end{array} \tag{12}$$

The specification (11) describes the behavior of (run_cache q) on a query that has not been memoized yet. It requires that add_noise should indeed run q under (ε, δ)-iDP and assumes ownership of enough privacy credits to pay for this execution. Furthermore, it requires ownership of $\mathsf{iC}(M_{cache})$ for the current internal state of the cache. In the postcondition, we recover iC where M_{cache} is updated with the result of the noisy query.

The intuition that repeated queries should be free is formalized in (12): if q is in the cache then no privacy credits are consumed for executing it under run_cache.

4.3.2 A Cache Client. A simple application is map_cache (Figure 11), which employs the cache to run add_noise on a list of queries qs. We can prove that the privacy cost of map_cache is $(k\varepsilon, k\delta)$ where (ε, δ) is the privacy cost of the add_noise mechanism and $k = |\{\mathsf{q} \in \mathsf{qs}\}|$ is the number of unique queries in qs. The proof of (13) follows directly from the abstract specification (10).

$$(\forall \mathsf{q} \in \mathsf{qs}.\ (\varepsilon, \delta)\text{-iDP}(\mathsf{add\_noise\ q})) \twoheadrightarrow (k\varepsilon, k\delta)\text{-iDP}(\mathsf{map\_cache\ add\_noise\ qs}) \tag{13}$$

Without caching, the privacy cost would have to be multiplied by $\mathsf{List.length}(\mathsf{qs})$ instead of k.

5 Soundness: A Model of Clutch-DP

In this section we give an overview of the model behind Clutch-DP and its adequacy theorem. A detailed account can be found in the Appendix of [26].

Our program logic is based around the notion of (ε, δ)-approximate coupling [9, 46]:

Definition 5.1. *Let A, B be countable types, and $\Phi \subseteq A \times B$ a relation. Given two real-valued parameters $0 \leq \varepsilon, \delta$, we say that there is an (ε, δ)-approximate Φ−coupling between distributions $\mu_1 \colon \mathcal{D}(A), \mu_2 \colon \mathcal{D}(B)$ if, for any real-valued random variables $f : A \to [0,1], g : B \to [0,1]$ such that $\forall (a,b) \in \Phi, f(a) \leq g(b)$, the following holds : $\mathbb{E}_{\mu_1}[f] \leq \exp(\varepsilon) \cdot \mathbb{E}_{\mu_2}[g] + \delta$. We denote the existence of such a coupling by $\mu_1\ \Phi^{(\varepsilon,\delta)}\ \mu_2$.*

The model is similar in spirit to that of Approxis [27], which can be seen as based on a notion of $(0, \delta)$-approximate coupling. Crucially, we prove a novel *choice composition* theorem, that is key to some of our proof rules, in particular Laplace-choice. This is heavily inspired by choice couplings [5], but it really shines in our setting, since we have a resourceful treatment of the privacy budget.

Theorem 5.2. *Let $\mu_1 \colon \mathcal{D}(A), \mu_2 \colon \mathcal{D}(B), f \colon A \to \mathcal{D}(A'), g \colon B \to \mathcal{D}(B')$. Assume we have a predicate $\Xi \subseteq A$, and $\Phi_1, \Phi_2 \subseteq A \times B, \Psi \subseteq A' \times B'$, with Φ_1, Φ_2 disjoint in the sense that, $\forall a, a', b. a \in \Xi \wedge a' \notin \Xi \Rightarrow ((a,b) \notin \Phi_1 \vee (a',b) \notin \Phi_2)$. Assume further that:*

(i) $\mu_1\ {\Phi_1}^{(\varepsilon_1,\delta_1)}\ \mu_2$
(ii) $\mu_1\ {\Phi_2}^{(\varepsilon_2,\delta_2)}\ \mu_2$
(iii) For all a, b such that $a \in \Xi$ and $(a,b) \in \Phi_1$, $f\ a\ \Psi^{(\varepsilon_1',\delta_1')}\ g\ b$.
(iv) For all a, b such that $a \notin \Xi$ and $(a,b) \in \Phi_2$, $f\ a\ \Psi^{(\varepsilon_2',\delta_2')}\ g\ b$.

Then, $(\mu_1 \gg\!= f)\ \Psi^{(\varepsilon,\delta)}\ (\mu_2 \gg\!= g)$, where $\varepsilon \triangleq \max(\varepsilon_1 + \varepsilon_1', \varepsilon_2 + \varepsilon_2')$ and $\delta \triangleq \delta_1 + \delta_2 + \max(\delta_1', \delta_2')$

This statement might appear cryptic at first. The idea is that, when proving a coupling for the composed computations $\mu_1 \gg\!= f, \mu_2 \gg\!= g$, one can choose between two different couplings for μ_1, μ_2, each with a different cost ε_1 or ε_2, and then also spend a different privacy cost ε_1' or ε_2' for the continuations depending on whether the value sampled from μ falls inside or outside Ξ (note however, that the additive cost $\delta_1 + \delta_2$ for the first step must be paid in any case).

For instance, the proof of Laplace-choice for parameters ε, T, m, m' essentially uses Theorem 5.2 with $\Xi(z) \triangleq (T \leq z)$, $\varepsilon_1 \triangleq 2\varepsilon$, $\varepsilon_1' \triangleq 0$, $\varepsilon_2 \triangleq 0$, and $\varepsilon_2' \triangleq 2\varepsilon$. The coupling (i) comes from Laplace-shift with $k = 1$, which (together with $z \in \Xi$) ensures the condition $\Phi_1 \triangleq (T \leq z \wedge T + 1 \leq z')$ in the postcondition of Laplace-choice for a privacy cost of $\varepsilon_1 = 2\varepsilon$ since $|k + m - m'| \leq 2$. The coupling (ii) comes from Laplace-shift with $k \triangleq m' - m$, which (together with $z \notin \Xi$) ensures the condition $\Phi_2 \triangleq (z < T \wedge z' < T + 1)$ in the postcondition of Laplace-choice for a privacy cost of $\varepsilon_2 = 0$. Since the coupling for (ii) does not consume any privacy credits, we recover $\varepsilon_2' = 2\varepsilon$ in the second branch of the postcondition of Laplace-choice.

In the model of Clutch-DP, this composition is possible at every execution step, and realized through the way separation logic resources tracking the budgets are threaded through the execution.

Table 1. Comparison of DP systems. "Beyond comp." = support for mechanisms whose DP goes beyond composition laws (*e.g.*, SVT, RNM).

	Methodology	State (◐ global, ● local)	Higher-order functions (△ first-order, ■ HO)	DP definition	Verified	Beyond comp.	Adaptive	Interactive
Fuzz [44]	lin. types	○	●	ε	○	○	○	○
Fuzz$^{\epsilon\delta}$ [16]	lin. typ., path adj.	○	●	(ε, δ)	○	○	○	○
DFuzz [21]	lin. dep. types	○	●	ε	○	○	○	○
AFuzz [50]	dyn. typ. + priv. filter	○	●	(ε, δ)	○	○	●	○
Fuzzi [52]	Fuzz-like + apRHL	○	○	(ε, δ)	○	●	○	○
Duet [42]	sens. + priv. types	○	●	(ε, δ), RDP, (z/t)CDP	○	○	○	○
Jazz [49]	ctxt'l sens. + priv. typ.	○	●	(ε, δ), RDP, zCDP	○	○	○	○
DPella [34]	eDSL dep. types	○	●	ε	○	○	○	○
Solo [2]	eDSL dep. types	○	●	(ε, δ), RDP	○	○	○	○
Spar [35]	eDSL dep. types	○	●	ε	○	○	○	○
HOARe^2 [10]	rel. refinement types	○	●	(ε, δ)	○	○	○	○
LightDP [51]	types + annot.	◐ △	○	ε	○	●	○	○
SampCert [17]	semantic	○	●	(ε, δ), zCDP	●	●	○	○
DP/Isabelle [48]	semantic	○	○	(ε, δ)	●	●	○	○
apRHL+ [8, 9, 12]	rel. prob. Hoare logic	◐ △	○	(ε, δ)	○	●	○	◐
HO-RPL [3]	rel. prob. Hoare logic	◐ △	●	(ε, δ)	○	○	○	◐
Clutch-DP	rel. prob. sep. logic	● ■	●	(ε, δ)	●	●	●	●

There is a tight connection between couplings and DP: a function $f : \mathrm{DB} \to \mathcal{D}(X)$ is (ε, δ)-dp iff for any adjacent inputs $b, b' : \mathrm{DB}$, we have $f\ b\ (=)^{(\varepsilon,\delta)}\ f\ b'$. With this connection in mind, we can now state the adequacy theorem of Clutch-DP.

Theorem 5.3. *Let f, f' be two RandML functions and $\Phi, \Psi : Val \times Val \to Prop$. If*

$$\left\{\Phi(w, w') * \lightning^{+}(\delta) * \lightning^{\times}(\varepsilon)\right\}\ f\ w \precsim f'\ w'\ \{\Psi\}$$

holds in Clutch-DP then, for any initial states σ, σ', we have $\mathrm{exec}(f\ w, \sigma)\ \Psi^{(\varepsilon,\delta)}\ \mathrm{exec}(f'\ w', \sigma')$.

By instantiating Φ with adjacency and Ψ with equality, we recover Theorem 3.2 as a corollary.

Internally, the Hoare quadruples are defined in terms of a primitive, unary notion of weakest precondition (WP), where the right-hand side program is represented as a separation logic resource [20, 24, 27]. Validity of the WP is defined by guarded induction on the program execution, establishing an approximate coupling at each step, and finally composing all the couplings into a coupling for the full execution. Each program logic rule is then proven sound w.r.t. the definition of the WP. In particular, all standard separation logic rules for the deterministic fragment of RandML can be re-established. We refer the reader to the supplementary material for more details.

6 Related Work

Types for DP. A wide range of type systems ensuring DP have been developed. Fuzz [44] and its variants track function sensitivity via linear types and rely on metric composition to statically ensure pure DP for a probabilistic λ-calculus. DFuzz [21] integrated linear and dependent types to improve sensitivity analysis. Fuzz$^{\epsilon\delta}$ [16] extends Fuzz with support for (ε, δ)-DP. The two-level type system of Adaptive Fuzz [50] enhances static typing by integrating a trusted privacy filter into the language runtime. The system allows programming with adaptive composition by dynamically type-checking programs during execution and composing them according to the privacy filter. In Clutch-DP, privacy filters are just regular programs that can be verified in their own right.

Duet [42] is a linear type system supporting various notions of DP by a separation of the language into a sensitivity and a privacy layer which interact through bespoke composition laws that restrict rescaling, which limits the kinds of higher-order functions that can be type-checked. Jazz [49] lifts some of these restrictions by introducing latent contextual effect types. HOARe^2 [10] encodes sensitivity and privacy information in relational refinement types for a pure calculus.

The DPella system [34], Solo [2], and Spar [35] leverage extensions to Haskell's type system to encode sensitivity (or distance) information via dependent types instead of linear types.

None of these systems support mutable state, or the verification of programs whose privacy requires advanced budget management instead of following directly from composition laws.

LightDP [51] employs a dependent relational type system to bound distances in program variables. With SMT-backed type inference, LightDP can verify DP for some mechanisms beyond composition laws (*e.g.*, non-interactive SVT), but the method does not extend to advanced language features or flexible privacy budget analysis as used, *e.g.*, in Report Noisy Max.

Logics for DP. The apRHL(+) program logics [8, 9, 12] can prove (ε, δ)-DP for programs written in a first-order while-language. Reasoning about basic mechanisms is well supported in apRHL, and it has been applied to advanced mechanisms such as SVT. Verification of interactive DP is supported through a specialized rule for adversaries that can interact with mechanisms through fixed patterns. In apRHL, adversaries are programs subject to a number of side-conditions; in contrast, our model of interactive computations via higher-order functions does not come with syntactic restrictions and can be composed modularly. For instance, we can recover the apRHL model of SVT used by Barthe et al. [9, Fig. 1] through our streaming SVT (§4.1.3) by instantiating the stream of queries QS with the adversary $\mathcal{A}$ of *loc. cit*. The privacy budget in apRHL is handled via a grading on judgments which offers less flexibility compared to Clutch-DP's privacy credits. On the other hand, working with a concrete denotational model allowed the authors of [9] to extend apRHL with support for "advanced composition" in the sense of [19, §3.5.2] and derive the corresponding bounds for SVT, which is not possible in Clutch-DP.

A variant of the EasyCrypt prover supports apRHL but the implementation of the apRHL rules are part of the trusted code base, whereas the rules of Clutch-DP are proven sound in a proof assistant. During the development of our case studies we found a bug in the Laplace sampling rule of EasyCrypt[7] which has accidentally been exploited in a user-contributed privacy proof of Report Noisy Max. Our foundational approach would have prevented us from introducing such an erroneous rule. Proof search for coupling-based proofs of DP for first-order programs was studied in [5]. Their definition of choice couplings inspired our Laplace-choice rule.

Fuzzi [52] integrates an apRHL-style logic with a Fuzz-inspired sensitivity- and privacy-logic; working at the intersection of the languages of the two systems, it does not support mutable state or higher-order functions.

The HO-RPL logic extends the ideas from apRHL to support higher-order functions and continuous distributions by giving a denotational semantics of programs in Quasi-Borel Spaces, but it is not known how to extend this approach to other language features that our challenge problems require such as dynamic allocation or higher-order store.

The Isabelle/HOL formalization of DP [48] develops the mathematical theory of DP in the continuous setting. Working directly in a measure-theoretic semantics, they prove privacy of Report Noisy Max, stating however that "the formal proof is quite long". It is unclear how to scale their approach to a language like RandML.

The SampCert project [17] formalized DP in the Lean prover by interpreting a shallow embedding of a while-like language in the same kind of denotational semantics that apRHL is based on. Rather

[7] We have disclosed the bug to the EasyCrypt team.

than assuming that the language has a primitive that samples from the Laplacian, SampCert proves that an efficient implementation of a sampler realizes the Laplace distribution. Their approach is particularly well-suited to carrying out the low-level probabilistic reasoning and focuses less on building modular systems. SampCert formalizes non-interactive variants of AT and SVT.

Separation Logic. Several probabilistic separation logics exist, but only Clutch-DP supports reasoning about DP. The Approxis [27] relational separation logic supports reasoning about approximate program equivalence. Their notion of α-approximate equivalence amounts to $(0, \alpha)$-dp in our setting, *i.e.*, our additive privacy credits correspond to their "error credits". Approxis can prove cryptographic security or correctness of samplers but cannot express DP. Approxis includes specific provisions for reasoning about samplers such as the SPEC-COUPL-ERR-1 clause and "expectation-preserving composition" that are not present in our model. Approxis also develops a logical relation model for contextual equivalence. We believe that Clutch-DP can be extended to support these features, and effectively subsume Approxis for additive error credits. Whether a expectation-preserving composition law can be formulated for the full generality of Clutch-DP is an open question. Bluebell [7] encodes coupling-based relational reasoning via a conditioning modality, but only supports exact program equivalences for terminating first-order programs.

(Non-) Termination. When verifying DP, we have to decide whether non-termination is considered an observable behavior that can leak information about the presence of an individual in a database. This is a well-known consideration in the context of DP (*e.g.*, [44, §3.5], [12, p.11]), and the most common solution to this problem is to rule out non-terminating programs a priori, *e.g.*, by restricting the programming language syntactically, by including side-conditions in the program logic, or simply by not supporting reasoning about not-obviously-terminating programs. Clutch-DP is a partial correctness logic: it neither assumes nor guarantees termination, and instead follows a more flexible approach by allowing verification of DP for an arbitrary RandML program f. If we separately also prove that f is terminating on all inputs, we recover termination-sensitive DP. Standard techniques for ensuring termination of probabilistic programs apply [14, 38], and termination for RandML-like languages can be handled via separation logics for reasoning about termination probabilities [4, 23] or expected runtime [28, 36].

7 Conclusion and Future Work

We have developed Clutch-DP, a probabilistic higher-order separation logic for DP. To demonstrate how Clutch-DP enables modular verification of DP libraries, we addressed three representative challenges and verified a wide range of case studies involving interactive mechanisms, privacy filters, and memoization. Clutch-DP is proven sound as a library for Iris in the Rocq Prover.

In future work, we would like to extend Clutch-DP to model concurrency to reason about local DP in a distributed setting. Clutch-DP supports reasoning about generic mechanisms and about Laplace distributions, but it should be possible to extend the operational semantics with, say, Gaussian distributions. This should not invalidate any rules of the logic since Hoare quadruples are defined for a general probabilistic operational semantics and our key lemmas such as choice-composition are valid for arbitrary distributions. To prove approximate DP of programs based on the Gaussian, new rules would be required. A challenging problem would be to extend Clutch-DP with support for further reasoning principles such as advanced composition in the sense of DP or privacy amplification by subsampling [1, 6]. We would also like to integrate verified sampling mechanisms as developed by, *e.g.*, de Medeiros et al. [17], to provide end-to-end DP guarantees by adapting the techniques of Aguirre et al. [4] to verify rejection samplers to the relational setting. Finally, it would be interesting to integrate other divergences [47] with relational separation logics to model, *e.g.*, Rényi-DP.

Data Availability Statement

The Rocq formalization accompanying this work is available on Zenodo [25] and on GitHub at github.com/logsem/clutch/tree/pldi26-clutch-dp.

Acknowledgments

This work was supported in part by the National Science Foundation, grant no. 2338317, the Carlsberg Foundation, grant no. CF23-0791, Villum Investigator grants, no. VIL25804 and no. VIL73403, Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation, and the European Union (ERC, CHORDS, 101096090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 308–318. https://doi.org/10.1145/2976749.2978318

[2] Chiké Abuah, David Darais, and Joseph P. Near. 2022. Solo: A Lightweight Static Analysis for Differential Privacy. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 150 (Oct. 2022). https://doi.org/10.1145/3563313

[3] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, Shin-ya Katsumata, and Tetsuya Sato. 2021. Higher-order probabilistic adversarial computations: categorical semantics and program logics. *Proc. ACM Program. Lang.* 5, ICFP, Article 93 (Aug 2021), 30 pages. https://doi.org/10.1145/3473598

[4] Alejandro Aguirre, Philipp G. Haselwarter, Markus de Medeiros, Kwing Hei Li, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2024. Error Credits: Resourceful Reasoning about Error Bounds for Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 8, ICFP, Article 246 (Aug 2024), 33 pages. https://doi.org/10.1145/3674635

[5] Aws Albarghouthi and Justin Hsu. 2017. Synthesizing Coupling Proofs of Differential Privacy. *Proceedings of the ACM on Programming Languages* 2, POPL (Dec. 2017), 58:1–58:30. https://doi.org/10.1145/3158146

[6] Borja Balle, Gilles Barthe, and Marco Gaboardi. 2018. Privacy amplification by subsampling: tight analyses via couplings and divergences. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6280–6290.

[7] Jialu Bao, Emanuele D'Osualdo, and Azadeh Farzan. 2025. Bluebell: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning. *Proc. ACM Program. Lang.* 9, POPL, Article 58 (Jan. 2025), 31 pages. https://doi.org/10.1145/3704894

[8] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational Reasoning via Probabilistic Coupling. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*. https://doi.org/10.1007/978-3-662-48899-7_27

[9] Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Advanced Probabilistic Couplings for Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016 (CCS '16)*. 55–67. https://doi.org/10.1145/2976749.2978391

[10] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. 2015. Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 55–68. https://doi.org/10.1145/2676726.2677000

[11] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Proving Differential Privacy via Probabilistic Couplings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science* (New York, NY, USA) *(LICS '16)*. Association for Computing Machinery, New York, NY, USA, 749–758. https://doi.org/10.1145/2933575.2934554

[12] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. *SIGPLAN Not.* 47, 1 (Jan 2012), 97–110. https://doi.org/10.1145/2103621.2103670

[13] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. 2020. The Discrete Gaussian for Differential Privacy. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, 15676–15688.

[14] Krishnendu Chatterjee, Hongfei Fu, and Petr Novotný. 2020. Termination Analysis of Probabilistic Programs with Martingales. In *Foundations of Probabilistic Programming*, Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva (Eds.). Cambridge University Press, 221–258. https://doi.org/10.1017/9781108770750.008

[15] Ethan Cowan, Michael Shoemate, and Mayana Pereira. 2024. *Hands-on Differential Privacy: Introduction to the Theory and Practice Using OpenDP* (first edition ed.). O'Reilly Media, Inc, Sebastopol, CA.

[16] Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2021. Probabilistic Relational Reasoning via Metrics. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '19)*. IEEE Press, Vancouver, Canada, 1–19.

[17] Markus de Medeiros, Muhammad Naveed, Tancrède Lepoint, Temesghen Kahsai, Tristan Ravitch, Stefan Zetzsche, Anjali Joshi, Joseph Tassarotti, Aws Albarghouthi, and Jean-Baptiste Tristan. 2025. Verified Foundations for Differential Privacy. *Artifact for Verified Foundations for Differential Privacy* 9, PLDI (June 2025), 191:1094–191:1118. https://doi.org/10.1145/3729294

[18] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

[19] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (Aug. 2014), 211–407. https://doi.org/10.1561/0400000042

[20] Dan Frumin, Robbert Krebbers, and Lars Birkedal. 2021. ReLoC Reloaded: A Mechanized Relational Logic for Fine-Grained Concurrency and Logical Atomicity. *Log. Methods Comput. Sci.* 17, 3 (2021). https://doi.org/10.46298/lmcs-17(3:9)2021

[21] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. 2013. Linear Dependent Types for Differential Privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*. Association for Computing Machinery, New York, NY, USA, 357–370. https://doi.org/10.1145/2429069.2429113

[22] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. 2012. Universally Utility-maximizing Privacy Mechanisms. *SIAM J. Comput.* 41, 6 (Jan. 2012), 1673–1693. https://doi.org/10.1137/09076828X

[23] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Almost-Sure Termination by Guarded Refinement. *Proc. ACM Program. Lang.* 8, ICFP, Article 243 (Aug. 2024), 31 pages. https://doi.org/10.1145/3674632

[24] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous Probabilistic Couplings in Higher-Order Separation Logic. *Proc. ACM Program. Lang.* 8, POPL (2024), 753–784. https://doi.org/10.1145/3632868

[25] Philipp G. Haselwarter, Alejandro Aguirre, Simon Oddershede Gregersen, Kwing Hei Li, Joseph Tassarotti, and Lars Birkedal. 2026. *Modular Verification of Differential Privacy in Probabilistic Higher-Order Separation Logic - Artifact.* https://doi.org/10.5281/zenodo.19089094

[26] Philipp G. Haselwarter, Alejandro Aguirre, Simon Oddershede Gregersen, Kwing Hei Li, Joseph Tassarotti, and Lars Birkedal. 2026. Modular Verification of Differential Privacy in Probabilistic Higher-Order Separation Logic (Extended Version). arXiv:2604.12713 [cs.PL] https://arxiv.org/abs/2604.12713

[27] Philipp G. Haselwarter, Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2025. Approximate Relational Reasoning for Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 9, POPL, Article 41 (Jan. 2025), 31 pages. https://doi.org/10.1145/3704877

[28] Philipp G. Haselwarter, Kwing Hei Li, Markus de Medeiros, Simon Oddershede Gregersen, Alejandro Aguirre, Joseph Tassarotti, and Lars Birkedal. 2024. Tachis: Higher-Order Separation Logic with Credits for Expected Costs. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 313 (Oct. 2024), 30 pages. https://doi.org/10.1145/3689753

[29] Justin Hsu. 2017. *Probabilistic Couplings for Probabilistic Reasoning.* Ph. D. Dissertation. University of Pennsylvania. arXiv:1710.09951

[30] Seidu Inusah and Tomasz J. Kozubowski. 2006. A Discrete Analogue of the Laplace Distribution. *Journal of Statistical Planning and Inference* 136, 3 (March 2006), 1090–1102. https://doi.org/10.1016/j.jspi.2004.08.014

[31] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

[32] Kelly Kostopoulou, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. 2023. Turbo: Effective Caching in Differentially-Private Databases. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 579–594. https://doi.org/10.1145/3600006.3613174

[33] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. *Proc. VLDB Endow.* 12, 11 (July 2019), 1371–1384. https://doi.org/10.14778/3342263.3342274

[34] Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi. 2020. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. In *2020 IEEE Symposium on Security and Privacy (SP)*. 411–428. https://doi.org/10.1109/SP40000.2020.00086

[35] Elisabet Lobo-Vesga, Alejandro Russo, Marco Gaboardi, and Carlos Tomé Cortiñas. 2024. Sensitivity by Parametricity. *Paper Artifact: Sensitivity by Parametricity* 8, OOPSLA2 (Oct. 2024), 286:415–286:441. https://doi.org/10.1145/3689726

[36] Janine Lohse and Deepak Garg. 2024. An Iris for Expected Cost Analysis. arXiv:2406.00884 [cs.PL]

[37] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. *Proc. VLDB Endow.* 10, 6 (Feb. 2017), 637–648. https://doi.org/10.14778/3055330.3055331

[38] Rupak Majumdar and V.R. Sathiyanarayana. 2025. Sound and Complete Proof Rules for Probabilistic Termination. *Proc. ACM Program. Lang.* 9, POPL, Article 63 (Jan. 2025), 32 pages. https://doi.org/10.1145/3704899

[39] Miti Mazmudar, Thomas Humphries, Jiaxiang Liu, Matthew Rafuse, and Xi He. 2022. Cache Me If You Can: Accuracy-Aware Inference Engine for Differentially Private Data Exploration. *Proc. VLDB Endow.* 16, 4 (Dec. 2022), 574–586. https://doi.org/10.14778/3574245.3574246

[40] Ilya Mironov. 2012. On Significance of the Least Significant Bits for Differential Privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. Association for Computing Machinery, New York, NY, USA, 650–661. https://doi.org/10.1145/2382196.2382264

[41] Joseph P. Near and Chiké Abuah. 2021. *Programming Differential Privacy*. Vol. 1. https://programming-dp.com/

[42] Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. 2019. Duet: An Expressive Higher-Order Language and Linear Type System for Statically Enforcing Differential Privacy. *Proc. ACM Program. Lang.* 3, OOPSLA (Oct. 2019). https://doi.org/10.1145/3360598

[43] Shangfu Peng, Yin Yang, Zhenjie Zhang, Marianne Winslett, and Yong Yu. 2013. Query Optimization for Differentially Private Data Management Systems. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 1093–1104. https://doi.org/10.1109/ICDE.2013.6544900

[44] Jason Reed and Benjamin C. Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming (ICFP '10)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/1863543.1863568

[45] Ryan M Rogers, Aaron Roth, Jonathan Ullman, and Salil Vadhan. 2016. Privacy Odometers and Filters: Pay-as-you-Go Composition. In *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc.

[46] Tetsuya Sato. 2016. Approximate Relational Hoare Logic for Continuous Random Samplings. In *The Thirty-second Conference on the Mathematical Foundations of Programming Semantics, MFPS 2016*. https://doi.org/10.1016/J.ENTCS.2016.09.043

[47] Tetsuya Sato, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Approximate Span Liftings: Compositional Semantics for Relaxations of Differential Privacy. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–14. https://doi.org/10.1109/LICS.2019.8785668

[48] Tetsuya Sato and Yasuhiko Minamide. 2025. Formalization of Differential Privacy in Isabelle/HOL. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '25)*. Association for Computing Machinery, New York, NY, USA, 67–82. https://doi.org/10.1145/3703595.3705875

[49] Matías Toro, David Darais, Chike Abuah, Joseph P. Near, Damián Árquez, Federico Olmedo, and Éric Tanter. 2023. Contextual Linear Types for Differential Privacy. *ACM Trans. Program. Lang. Syst.* 45, 2 (May 2023), 8:1–8:69. https://doi.org/10.1145/3589207

[50] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. 2017. A Framework for Adaptive Differential Privacy. *Proc. ACM Program. Lang.* 1, ICFP (Aug. 2017), 10:1–10:29. https://doi.org/10.1145/3110254

[51] Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards Automating Differential Privacy Proofs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL '17)*. Association for Computing Machinery, New York, NY, USA, 888–901. https://doi.org/10.1145/3009837.3009884

[52] Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C. Pierce, and Aaron Roth. 2019. Fuzzi: A Three-Level Logic for Differential Privacy. *Prototype Implementation of Fuzzi: A Three-Level Logic for Differential Privacy* 3, ICFP (July 2019), 93:1–93:28. https://doi.org/10.1145/3341697

Received 2025-11-13; accepted 2026-04-03