Synthesizing Tight Privacy and Accuracy Bounds via Weighted Model Counting

Lisa Oakley
Khoury College of Computer Sciences
Northeastern University
Boston, USA
oakley.l@northeastern.edu

Steven Holtzen
Khoury College of Computer Sciences
Northeastern University
Boston, USA
s.holtzen@northeastern.edu

Alina Oprea
Khoury College of Computer Sciences
Northeastern University
Boston, USA
a.oprea@northeastern.edu

***Abstract*—Programmatically generating tight differential privacy (DP) bounds is a hard problem. Two core challenges are (1) finding expressive, compact, and efficient encodings of the distributions of DP algorithms, and (2) state space explosion stemming from the multiple quantifiers and relational properties of the DP definition.**

We address the first challenge by developing a method for tight privacy and accuracy bound synthesis using weighted model counting on binary decision diagrams, a state of the art technique from the artificial intelligence and automated reasoning communities for exactly computing probability distributions. We address the second challenge by developing a framework for leveraging inherent symmetries in DP algorithms. Our solution benefits from ongoing research in probabilistic programming languages, allowing us to succinctly and expressively represent different DP algorithms with approachable language syntax that can be used by non-experts.

We provide a detailed case study of our solution on the binary randomized response algorithm. We also evaluate an implementation of our solution using the Dice probabilistic programming language for the randomized response and truncated geometric above threshold algorithms. We compare to prior work on exact DP verification using Markov chain probabilistic model checking and the decision procedure DiPC. Very few existing works consider mechanized analysis of accuracy guarantees for DP algorithms. We additionally provide a detailed analysis using our technique for finding tight accuracy bounds for DP algorithms.

***Index Terms*—Differential Privacy, Weighted Model Counting, Probabilistic Programming**

I. Introduction

Differential privacy (DP) [1] is an important property for randomized algorithms that can ensure a balance between preserving user privacy and allowing systems to draw meaningful conclusions from their data. Crafting algorithms which satisfy meaningful differential privacy bounds while maintaining useful accuracy guarantees is no small feat, and the design and analysis of these differentially private algorithms is extensive and highly technical. As is often the case with complicated, technical fields, as the landscape of differential privacy research has grown, so too has the tendency for bugs in the theory and implementations of these algorithms [2]–[4]. It is therefore vital for algorithm designers to have tools and frameworks to help formally and mechanically analyze their algorithms both in the design phase, and to validate their theoretical results and implementations. Furthermore, it is important that these tools be as accessible and automated as possible.

Most current methods for formally verifying differential privacy properties involve manually mechanizing existing pen-and-paper privacy proofs using proof assistants like EasyCrypt [5]. This method of deductive verification is useful for mechanically validating asymptotic privacy bounds [5]–[9]. This method, however, requires almost entirely manual work and often hinges on first having an existing proof written by hand. Importantly, it also requires a technician who understands both theoretical differential privacy and proof assistants well enough to translate the pen-and-paper proof into the language of the proof assistant. Therefore, these methods can be extremely useful, but are not practical for keeping up with the pace of ongoing differential privacy research, and are not as helpful in the algorithmic design phase where there is incomplete pen-and-paper analysis.

There is also active research on statistically verifying differential privacy in practice, discovering lower bounds by finding counterexamples to differential privacy, or programmatically tightening existing bounds [10]–[13]. In addition, privacy auditing provides a set of statistical techniques for estimating privacy leakage of machine learning (ML) algorithms empirically and determining lower bounds on privacy [14]–[19]. While these methods are useful for automatically analyzing algorithms over large data sets, they often rely on statistical procedures and are evaluated mostly through experimental methods on specific data sets.

Existing methods here also focus almost entirely on verifying privacy, and do not analyze accuracy bounds. This is a problem because incorrect or non-existent accuracy bounds forces a reliance on proxy functions for accuracy that can be flawed and result in algorithm designers adding too much noise, or adding noise in non-optimal parts of the algorithm. Furthermore, without full accuracy analysis, applications like optimization-based synthesis for differentially private algorithms [20] must use these flawed accuracy proxy functions and end up finding potentially non-optimal solutions.

Our goal is to develop a technique for exactly solving the tight differential privacy bound and tight accuracy bound synthesis problems. In other words, given a randomized algorithm, find the parameters which tightly bound its privacy

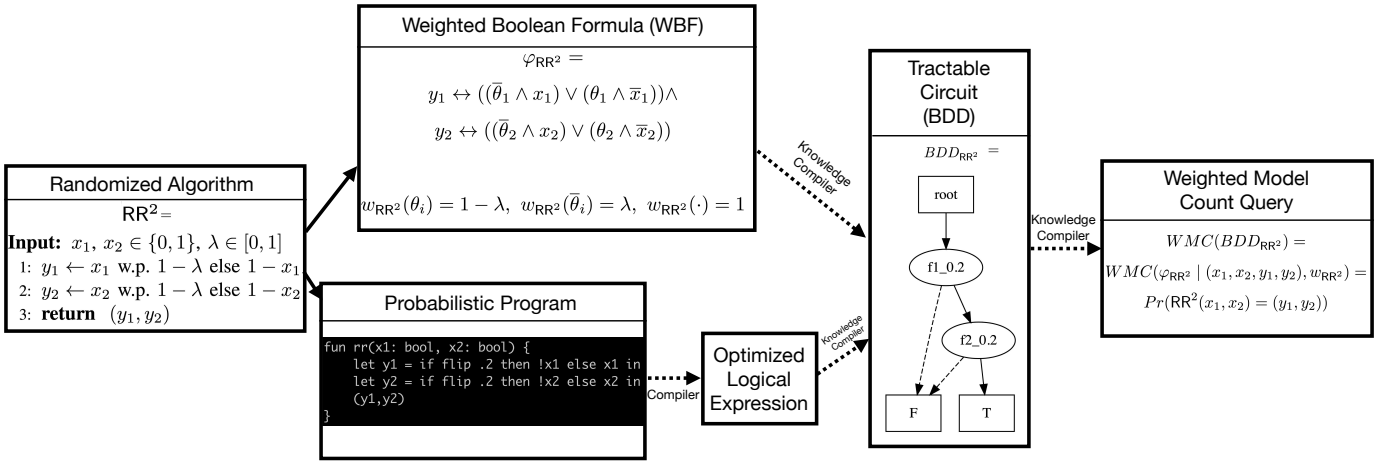


Fig. 1: We represent the probability distribution of a randomized algorithm as a binary decision diagram (BDD) so that we can efficiently compute the weighted model count to perform probabilistic inference (compute the probabilities of certain events occurring). Starting from a randomized algorithm, there are many ways to compile to a BDD. The top path shows a more manual process where a user crafts a weighted boolean formula by hand and uses a *knowledge compiler*, a tool for efficiently representing and querying logical formulas, to compile this into a BDD. The bottom path exhibits an example of the flow of an expressive probabilistic programming languages which allow the user to define an easily readable program defined similarly to the pseudocode. From here, the programming language can compile an optimized logical expression and pass it to the knowledge compiler. In this way, there is very little manual effort, and the resulting logical expression and BDD are optimized for WMC queries.

and accuracy guarantees. Our technique will provide more theoretical grounding to the analysis of differentially private mechanisms than the statistical methods, and provide more automation than the traditional proof mechanizing process described in prior work. It will also critically fill a gap in mechanized analysis for accuracy bounds.

Beyond its utility in finding tight privacy and accuracy bounds, another benefit of our technique is that it allows us to identify which input, neighbor, and output assignments lead to these bounds. In other words, we can identify the worst-case assignments. We provide empirical examples of using our technique to find the top worst-case assignments for accuracy, which highlights how our framework can help an algorithm designer determine which outliers are most impactful on privacy and accuracy bounds.

Very few attempts have been made at exactly synthesizing tight privacy bounds, and those that exist are extremely limited in the kinds of algorithms they can analyze [21]–[23], and in Section VI, we show that our method outperforms these techniques for randomized response. As far as we are aware, no one has attempted to mechanically solve for exact accuracy bounds of a given differentially private algorithm.

There are two main reasons this is the case. Firstly, finding an easily computable encoding of the algorithm on which to do probabilistic inference is hard. Previous work on exact DP verification has required an explicit, hand-crafted Markov chain as input [22], [23]. Secondly, exact verification by quantifying over all neighboring inputs and outputs is intractable as the data sets increase in size. Our approach addresses both of these key challenges, and the resulting technique improves the feasibility of exactly solving the privacy and accuracy bound synthesis problems.

To tackle the first challenge, we use a method called weighted model counting (WMC) for computing the probabilities of certain events (otherwise known as performing *probabilistic inference*). WMC is an established state-of-the-art strategy for performing discrete probabilistic inference in the artificial intelligence and automated reasoning community [24], [25]. In order to leverage WMC, one must encode their problem into a *weighted Boolean formula* (WBF). We reduce the problem of computing accuracy and privacy bounds of randomized algorithms to computing the WMC of a particular set of Boolean formulas. This technique has many benefits. Firstly, we can rely on the many decades of algorithmic advances in WMC: for instance, high-performance data-structures like binary decision diagrams (BDDs) can efficiently represent the distributions of the randomized algorithms and efficiently query for exact probabilities. Secondly, there exist expressive probabilistic programming languages that can efficiently compile into these optimized representations [26]. This makes it simple to encode DP algorithms which often invoke probability distributions as programs, and allows for a usable strategy even for non-experts in probabilistic programming. In Fig. 1, we present an example of this pipeline for randomized response with two inputs.

To tackle the second challenge of state space explosion due to multiple for-all quantifiers on the input and output space, we introduce the concepts of inference, privacy, and

accuracy sets, and provide algorithms for synthesizing bounds with respect to these restricted state spaces. These sets allow us to define which inputs and outputs are sufficient for finding privacy and accuracy bounds of DP algorithms. These sets provide a simple framework for utilizing inherent symmetries in DP algorithms to reduce the space of the exact synthesis problems. Despite being simple to define, a limitation of our approach is that there is some manual work which goes into finding these symmetry sets. We provide a detailed analysis of an example of soundly defining the symmetry sets for a randomized response case study.

Our contributions are (1) a technique for using probabilistic programming languages and WMC for efficient probabilistic inference for DP algorithms, (2) a framework for leveraging symmetries to combat the state space explosion problem for both the tight privacy and accuracy bound synthesis problems, (3) a detailed theoretical analysis of the symmetry sets for a randomized response case study, (4) examples of using our framework for automated analysis of the randomized response and geometric above threshold algorithms, and (5) comparison to existing techniques of probabilistic model checking using Markov chains and the decision procedure, DiPC, from [21].

The paper is organized as follows. In Section II we provide relevant background. In Section III we introduce our tight privacy and accuracy bound synthesis problems and explain the benefits of efficient WMC. In Section IV we introduce a framework for leveraging symmetries. In Section V we provide a detailed case study using our framework for randomized response. In Section VI we present experimental results on implementations of our solution using the Dice probabilistic programming language. In Section VII we outline related work and in Section VIII we conclude and outline future work.

II. Preliminaries

We start by reviewing some important definitions.

A. Differential Privacy

Differential privacy is a notion of privacy which ensures that, on two closely related data sets, the output distribution of a randomized algorithm is similar. In other words, an adversary who receives the output of a differentially private algorithm has trouble distinguishing whether a specific entry is in the data set.

Definition 1 (ε-Differential Privacy)**.** *A randomized algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ is ε-differentially private for size n data sets if, for every pair of neighboring data sets $\mathbf{x}, \mathbf{x}'$, for all $E \subseteq \mathcal{Y}$,*

$$\frac{\Pr(A(\mathbf{x}) \in E)}{\Pr(A(\mathbf{x}') \in E)} \leq e^{\varepsilon}. \tag{1}$$

Where neighboring data sets are data sets that differ in the value of exactly one entry. An important property of ε-Differential Privacy is that for all possible outputs y in $\mathcal{Y}$,

$$\frac{\Pr(A(\mathbf{x}) = y)}{\Pr(A(\mathbf{x}') = y)} \leq e^{\varepsilon} \implies \frac{\Pr(A(\mathbf{x}) \in E)}{\Pr(A(\mathbf{x}') \in E)} \leq e^{\varepsilon} \tag{2}$$

In other words, for ε-DP, it is sufficient to look at all singleton sets in the event space. In subsequent sections we will directly use the definition of differential privacy which is quantified over the singleton sets, and refer to the quantity $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$ as the *likelihood ratio*. We will also limit our method to algorithms with discrete, finite inputs and outputs (i.e. $\mathcal{X}^n$ and $\mathcal{Y}$ are discrete and finite).

B. Accuracy

We consider a widely-used notion of (α, β)-accuracy from the differential privacy literature [27]. Intuitively, this measures the probability that the algorithm's output is within an α ball around the "true" or "target" value for the associated input.

Definition 2 ((α, β)-Accuracy)**.** *Given $\alpha \geq 0$ and $\beta \in [0, 1]$, a randomized algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ is (α, β)-accurate if for all $\mathbf{x} \in \mathcal{X}^n$,*

$$\Pr(|A(\mathbf{x}) - \mathcal{V}_{\mathbf{x}}| \leq \alpha) \geq 1 - \beta \tag{3}$$

where $\mathcal{V}_{\mathbf{x}}$ correspond to the target outputs for each input $\mathbf{x}$.

C. Weighted Model Counting

1) Model Counting: Let φ be a Boolean formula over a set of variables. The model count of φ is the number of solutions to φ, written as $|\{m \models \varphi\}|$ where $\{m \models \varphi\}$ is described as "the set of models m that entail φ". For example, $\varphi = a \vee b$ where φ is defined over $\{a, b\} \implies |\{m \models \varphi\}| = 3$ since there are three satisfying assignments to φ, where an assignment is a mapping from variables to boolean values.

2) Weighted Boolean Formula (WBF): A *weighted Boolean formula* is a tuple (φ, w) where φ is a Boolean formula over literals in L and $w : L \to \mathbb{R}$ is its weighting function that maps literals in φ to weights, where literals are a set of variables and their negations.

3) Weighted Model Counting (WMC): We find the *weighted model count* (WMC) of WBF (φ, w) by computing

$$WMC(\varphi_w) = \sum_{m \models \varphi} \prod_{\ell \in m} w(\ell) \tag{4}$$

where m is the set of models, or satisfying assignments, of φ, and ℓ represents the set of literals in m.

For example, we can find the weighted model count of $\varphi = a \vee b$, where φ is defined over variables a and b, by defining the weight function w over the literals as $w(a) = 1/3$, $w(b) = 3/4$, $w(\overline{a}) = 2/3$, and $w(\overline{b}) = 1/4$ and computing

$$WMC(\varphi_w) = 1/3 \cdot 3/4 + 1/3 \cdot 1/4 + 2/3 \cdot 3/4. \tag{5}$$

4) WMC for Probabilistic Algorithms: We can define a WBF for any finite, discrete probabilistic algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ by ensuring that the probabilities of each WBF assignment correspond with probabilities of input/output pairs of the algorithm. More formally,

Definition 3 (WBF of a probabilistic algorithm)**.** *Given a randomized algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ with discrete, finite $\mathcal{X}^n$, $\mathcal{Y}$, we say that (φ, w) is a WBF of A if $\forall (\mathbf{x}, y) \in \mathcal{X}^n \times \mathcal{Y}$,*

there exists exactly one assignment $assn$ to (φ, w) such that $WMC((\varphi \mid assn), w) = \Pr(A(\mathbf{x}) = y)$

We use a shorthand $(\varphi \mid (\mathbf{x}, y))$ to indicate the WBF instantiated with the assignment a for which $WMC((\varphi \mid assn), w) = \Pr(A(\mathbf{x}) = y)$.

For example, we consider binary randomized response for $n = 2$ as described in Fig. 1. In the randomized response algorithm, clients report a bit message (represented by x_1 and x_2 here) with probability $1 - \lambda$, and flip this bit with probability λ for some coin-flip parameter λ. More formally, $\mathsf{RR}^2(x_1, x_2) = (y_1, y_2)$ such that $y_1 = x_1$ *w.p.* $(1 - \lambda)$ else $1 - x_1$ and $y_2 = x_2$ *w.p.* $(1 - \lambda)$ else $1 - x_2$. In this case, the WBF for randomized response is the tuple $(\varphi_{\mathsf{RR}^2}, w_{\mathsf{RR}^2})$ where $\varphi_{\mathsf{RR}^2} = y_1 \leftrightarrow ((\overline{\theta}_1 \wedge x_1) \vee (\theta_1 \wedge \overline{x}_1)) \wedge y_2 \leftrightarrow ((\overline{\theta}_2 \wedge x_2) \vee (\theta_2 \wedge \overline{x}_2))$ defined over $x_1, x_2, y_1, y_2, \theta_1, \theta_2$. We define weight function $w(\theta_i) = 1 - \lambda$, or the probability that the value of x_i does not flip, and $w(\overline{\theta}_i) = \lambda$ as the probability that x_i does flip for $i \in \{1, 2\}$. For all other literals, we set $w(\cdot) = 1$.

We can see that for all (x_1, x_2, y_1, y_2), $WMC(\varphi_{\mathsf{RR}^2} \mid (x_1, x_2, y_1, y_2), w_{\mathsf{RR}^2}) = \Pr(\mathsf{RR}^2(x_1, x_2) = (y_1, y_2))$. For example, for $(x_1, x_2, y_1, y_2) = (0, 0, 1, 0)$, $WMC(\varphi_{\mathsf{RR}^2} \mid (0, 0, 1, 0), w_{\mathsf{RR}^2}) = w_{RR^2}(\theta_1) \cdot w_{RR^2}(\theta_2) = \lambda \cdot (1 - \lambda) = \Pr(\mathsf{RR}^2(0, 0) = (1, 0))$.

III. Weighted Model Counting for Synthesizing Tight Privacy and Accuracy Bounds

We start by introducing the two main synthesis problems we will address in this paper. Then, we describe how weighted model counting can be used to efficiently perform probabilistic inference in an exhaustive solution. In Section IV, we will outline a framework for leveraging symmetries in the DP algorithms to further improve on this solution.

A. Synthesis Problems

The first problem is the *privacy bound synthesis problem*. We want to find the triple of neighboring input and output values which maximize the likelihood ratio.

Problem 1 (Privacy Bound Synthesis)**.** *Given a randomized algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ with discrete, finite $\mathcal{X}^n$, $\mathcal{Y}$, find the triple $(\mathbf{x}, \mathbf{x}', y)$ that maximizes $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$, where x, x' are neighboring inputs.*

If we take solution $(\mathbf{x}, \mathbf{x}', y)$ to the privacy bound synthesis problem to find $e^{\varepsilon} = \frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$, we have a tight ε-DP bound for algorithm A.

The second synthesis problem is the *tight accuracy bound synthesis problem*. Here we want to find the input to A which minimizes the accuracy probability with respect to a given α bound.

Problem 2 (Accuracy Bound Synthesis)**.** *Given a randomized algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ with discrete, finite $\mathcal{X}^n$, $\mathcal{Y}$, and $\alpha \geq 0$, find $\mathbf{x}$ which minimizes $\Pr(|A(\mathbf{x}) - \mathcal{V}_{\mathbf{x}}| \leq \alpha)$ where $\mathcal{V}_{\mathbf{x}}$ is the target output value for input $\mathbf{x}$.*

If we take solution $\mathbf{x}$ from the Accuracy Bound Synthesis problem to find $1 - \beta = \Pr(|A(\mathbf{x}) - \mathcal{V}_{\mathbf{x}}| \leq \alpha)$, we have a tight (α, β)-accuracy bound for algorithm A.

B. WMC and Probabilistic Programming For Privacy Bound Synthesis

We start by looking at the most straightforward solution to the privacy bound synthesis problem, which is to iterate through every input/neighbor/output triple and compute the likelihood ratio for that set. In other words, we consider a solution where, for $A : \mathcal{X}^n \to \mathcal{Y}$ we exhaustively compute $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$ for each $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$ where $\mathbf{x}, \mathbf{x}'$ are neighbors. We will refine this solution further in Section IV, but for now we use this simple solution to illustrate the utility of WMC for probabilistic inference.

If we consider even this simple exhaustive solution, it is immediately apparent that for a complex randomized algorithm, A, it is necessary to perform probabilistic inference (i.e. compute the probabilities $\Pr(A(\mathbf{x}) = y)$) for each input and output pair.

The power of our technique comes from performing probabilistic inference (computing the probability distributions of the algorithm) by compiling a representation of the probabilistic algorithm into a *tractable circuit* for which computing the weighted model count can be performed efficiently [24], [26].

We consider our target data structure to be a *binary decision diagram* (BDD). A BDD is a directed acyclic graph that represents boolean formulas. We prefer to use this data structure for WMC computation because BDDs can leverage shared structure in the encoding to simplify a complicated function with many free variables into a compact, efficiently queryable circuit. As we see in Fig. 1, a program with two inputs, two outputs, and two weighted coin flip parameters can be encoded into a BDD with only two internal nodes when the input is fixed. When we compare this to other popular methods for exact probabilistic inference such as Markov chains, we see in Section VI that a BDD is able to scale much more efficiently with the state size on static systems such as DP algorithms.

For example, prior work has used Markov chain model checking for exact DP verification [22], [23]. This technique requires an explicit Markov chain model of the probability distribution, which is optimized for dynamic models with small state space which model long-running services. In this case, computation becomes inefficient quickly as state size increases, as is the case in DP algorithms [28].

Using BDDs for WMC computation is therefore more efficient for our problem space than in prior attempts, but we also desire an expressive encoding method for the randomized algorithm. To use WMC to perform inference on the randomized algorithm, we must encode the algorithm as a weighted Boolean formula with free variables that represent the inputs, outputs, and coin flips/distributions, and craft a weighting function to represent the randomness of the algorithm. For some algorithms, this is straightforward and little manual work is required to discover the right formula and weighting function. We provide an example of this manual

process for randomized response in Section V. However, for many algorithms, this manual process can be difficult, and the resulting weighted Boolean formula can be unnecessarily large or otherwise inefficient to work with.

In recent years, there has been a significant amount of work on developing probabilistic programming languages that can efficiently compile weighted Boolean formulas from probabilistic programs [25]. Importantly, these weighted Boolean formulas are optimized to efficiently compile into tractable data structures such as BDDs that can be efficiently queried to find the weighted model count. This means that we can write our randomized algorithm as a probabilistic program in these languages and to easily compile it into a WBF and perform efficient WMC. We show an example of this full pipeline in Fig. 1.

In summary, using WMC for performing probabilistic inference is a method of efficient computation of the probabilities needed to find tight bounds. Furthermore, recently developed probabilistic programming languages provide approachable and intuitive methods for encoding DP algorithms. This is a big improvement over prior work which required that the DP algorithm be explicitly encoded as a Markov chain [22], [23].

C. WMC for Exhaustive Accuracy Bound Synthesis

Similarly to our exhaustive approach for solving the privacy bound synthesis problem, we can also leverage the benefits of WMC in an exhaustive approach for solving the accuracy bound synthesis problem. Because the target value of the DP algorithm (i.e. the value of the query with no added noise) is non-probabilistic, we can formulate an optimization which will directly use probabilistic inference to solve the accuracy bound synthesis problem.

Theorem 1. *Given randomized $A : \mathcal{X}^n \to \mathcal{Y}$, with discrete, finite $\mathcal{X}^n$ and $\mathcal{Y}$, set of target values $\mathcal{V}$, and $\alpha \geq 0$, the $\mathbf{x}$ which minimizes $\sum_{y \in [\mathcal{V}_\mathbf{x} - \alpha, \mathcal{V}_\mathbf{x} + \alpha]} \Pr(A(\mathbf{x}) = y)$ also minimizes $\Pr(|A(\mathbf{x}) - \mathcal{V}_\mathbf{x}| \leq \alpha)$*

Proof.

$$\begin{aligned}
&\Pr(|A(\mathbf{x}) - \mathcal{V}_\mathbf{x}| \leq \alpha) \\
&= \Pr(\mathcal{V}_\mathbf{x} - \alpha \leq A(\mathbf{x}) \leq \mathcal{V}_\mathbf{x} + \alpha) \\
&= \sum_{y \in [\mathcal{V}_\mathbf{x} - \alpha, \mathcal{V}_\mathbf{x} + \alpha]} \Pr(A(\mathbf{x}) = y).
\end{aligned}$$

□

We can therefore also use weighted model counting for the simple exhaustive approach for $A : \mathcal{X}^n \to \mathcal{Y}$ where we compute $\sum_{y \in [\mathcal{V}_\mathbf{x} - \alpha, \mathcal{V}_\mathbf{x} + \alpha]} \Pr(A(\mathbf{x}) = y)$ for each $\mathbf{x} \in \mathcal{X}^n$. Again, in Section IV, we will show a further refinement of this exhaustive algorithm.

IV. Framework for Leveraging Symmetries

Arguably the hardest problem when attempting to find bounds for DP algorithms – explored in related works on counterexample generation and proof synthesis – is handling the multiple quantifiers over inputs, neighbors, and outputs. Approaches like StatDP [10] and DP-Sniper [29] use a manually chosen heuristic to divide the input search space, then use hypothesis testing or classifiers to find candidate outputs which are likely DP violations. These techniques result in approximate solutions with little to no theoretical guarantees on their correctness. On the other hand, all prior work which attempts to find provably exact bounds has exhaustively searched the space of inputs, neighbors, and outputs [21]–[23].

Because we want to provide theoretical evaluations of our solution, we cannot use these manual heuristics for finding candidate counterexamples. However, we also want to make progress toward a solution that is more tractable than the exhaustive search methods from prior work. Fortunately, many differentially private algorithms have inherent symmetry. We provide a framework for leveraging these symmetries to make our WMC solutions more tractable by introducing algorithms ExactDP and ExactAcc for computing the privacy and accuracy bounds which use modular *inference*, *privacy*, and *accuracy sets* for limiting the search space while maintaining provable exactness of our solution.

While this step requires a manual process of finding these symmetry sets, we believe it is still valuable to define this framework so that we can explore this problem from a fresh perspective of finding symmetries to reduce the search space. This framework provides proof obligations for determining a valid symmetry set, which can set guidelines for future automated analysis. Automatically finding these symmetry sets is an interesting open problem that we hope to explore in future work by leveraging techniques for exploiting symmetry in satisfiability or probabilistic inference [26], [30]–[32].

A. Inference Algorithm

Our refined solutions for privacy and accuracy bound synthesis both use probabilistic inference via weighted model counting as a subprocess. We provide an algorithm for precomputation of necessary probabilities to help with modularization and simplification of the algorithm and analysis.

In Algorithm 1, we describe how we compute a matrix of assignment probabilities for a given restricted *inference set*. An *inference set* $\mathcal{I}_A \subseteq \mathcal{X}^n \times \mathcal{Y}$ for $A : \mathcal{X}^n \to \mathcal{Y}$ is a set of input/output pairs on which to do probabilistic inference.

Algorithm 1 INFERENCE

Input: WBF (φ, w) of $A : \mathcal{X}^n \to \mathcal{Y}$ and inf. set $\mathcal{I}$
Output: M a matrix of probabilities

```
Initialize M matrix
for (x, y) in I do
    M(x, y) ← WMC(φ | (x, y))
end for
return M
```

Algorithm 1 runs in $O(\mathsf{WMC} \cdot |\mathcal{I}|)$ where WMC is the complexity of the WMC computation. We will show in Section V that for some algorithms, we can reduce the size of the inference set such that the complexity is linear in the length

		input		
		[0,0]	[1,0]	[1,1]
output	[0,0]	$(1-\lambda)^2$	$\lambda \cdot (1-\lambda)$	λ^2

TABLE I: Output of Algorithm 1 for $A = \mathsf{RR}^2$, $\mathcal{I}_{\mathsf{RR}^2} = \{([0,0],[0,0]),\ ([1,0],[0,0]),\ ([1,1],[0,0])\}$

of the input vector, multiplied by the complexity of the WMC computation.

Returning to our example of binary randomized response, an inference set for RR^2 could be $\mathcal{I}_{\mathsf{RR}^2} = \{([0,0],[0,0]),\ ([1,0],[0,0]),\ ([1,1],[0,0])\}$. Table I shows the output of this computation. We show in Section V that this table grows linearly in n, where n is the length of the input list for randomized response.

B. Finding Tight Privacy Bound

Once we have computed the necessary probabilities using weighted model counting, we can use these probabilities to find the maximum likelihood ratio. Because we consider only neighboring inputs, and due to the symmetries in differential privacy algorithms, we do not have to enumerate all pairs of inputs and all output events to cover all sufficient ratios. We therefore define our algorithm over a set of inputs and outputs which are a sufficient subset of all computable likelihood ratios.

Definition 4 (Privacy Set)**.** *A* privacy set $\mathcal{C}$ *for* $A : \mathcal{X}^n \to \mathcal{Y}$ *is a set of tuples of the form* $(\mathbf{x}_\mathcal{C}, \mathbf{x}'_\mathcal{C}, y_\mathcal{C})$ *such that*

1) $(\mathbf{x}_\mathcal{C}, \mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$
2) $\forall (\mathbf{x}, \mathbf{x}', y) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$ *where* $\mathbf{x}, \mathbf{x}'$ *are neighbors, there exists* $(\mathbf{x}_\mathcal{C}, \mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{C}$ *such that*

$$\frac{\Pr(A(\mathbf{x}) = y)}{\Pr(A(\mathbf{x}') = y)} = \frac{\Pr(A(\mathbf{x}_\mathcal{C}) = y_\mathcal{C})}{\Pr(A(\mathbf{x}'_\mathcal{C}) = y_\mathcal{C})}.$$

We present the ExactDP algorithm for computing worst case privacy bounds in Algorithm 2. We use the INFERENCE algorithm (Algorithm 1) as a subprocess to compute exact probabilistic inference via weighted model counting.

Algorithm 2 ExactDP

Input: WBF (φ, w) of $A : \mathcal{X}^n \to \mathcal{Y}$, privacy set $\mathcal{C}$, and inference set $\mathcal{I}$ such that $\forall (\mathbf{x}_\mathcal{C}, \mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{C}$, $(\mathbf{x}_\mathcal{C}, y_\mathcal{C}) \in \mathcal{I}$ and $(\mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{I}$
Output: Maximum likelihood ratio p and worst case assignments $c = (\mathbf{x}, \mathbf{x}', y)$

```
M ← INFERENCE(φ, I)
p ← 0, c ← ∅
for (x, x', y) in C do
    if M(x,y)/M(x',y) > p then
        p ← M(x,y)/M(x',y)
        c ← (x, x', y)
    end if
end for
return p, c
```

Returning to our binary randomized response example, we can use $\mathcal{I}_{\mathsf{RR}^2} = \{([0,0],[0,0]),\ ([1,0],[0,0]),\ ([1,1],[0,0])\}$, and $\mathcal{C}_{\mathsf{RR}^2} = \{([0,0],[1,0],[0,0]),\ ([1,0],[0,0],[0,0]),\ ([1,0],[1,1],[0,0]),\ ([1,1],[1,0],[0,0])\}$ and set $\lambda = 0.2$. In this case, the output of Algorithm 2 would be $c = ([0,0],[1,0],[0,0])$, $p = 4$. This means that $([0,0],[1,0],[0,0])$ is the worst case assignment for RR^2 and $e^\varepsilon = 4$ is a tight privacy bound.

1) Correctness and Complexity of Privacy Bound Synthesis Algorithm: To verify correctness of ExactDP, we must show that the maximal likelihood ratio found using the potentially restricted inference and privacy set is the same as the maximal likelihood ratio using exhaustive inference and privacy sets (i.e. $\mathcal{I} = \mathcal{X}^n \times \mathcal{Y}$ and $\mathcal{C} = \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$ where all x, x' are neighbors).

Theorem 2. *The output* c *of Algorithm 2 is a solution to the Privacy Bound Synthesis problem.*

Proof. Let WBF (φ, w) be the WBF of algorithm $A : \mathcal{X}^n \to \mathcal{Y}$ with inference and privacy sets $\mathcal{I}$, $\mathcal{C}$. From the definition of Algorithm 1, $\forall (\mathbf{x}, y) \in \mathcal{I}$, $M(\mathbf{x}, y) = \Pr(A(\mathbf{x}, y))$. Therefore, by the definition of Algorithm 2, output c maximizes $\frac{M(\mathbf{x}_\mathcal{C}, y_\mathcal{C})}{M(\mathbf{x}'_\mathcal{C}, y_\mathcal{C})}$ over all $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{C}$. By the definition of a privacy set, for all $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$, where $\mathbf{x}$, $\mathbf{x}'$ are neighbors, there exists a $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{C}$ such that $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)} = \frac{\Pr(A(\mathbf{x}_\mathcal{C})=y_\mathcal{C})}{\Pr(A(\mathbf{x}'_\mathcal{C})=y_\mathcal{C})}$. Therefore, c maximizes $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$ over all $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$ where $\mathbf{x}$, $\mathbf{x}'$ are neighbors and is therefore a solution to the Privacy Bound Synthesis problem. □

Algorithm 2 runs in $O(\mathsf{WMC} \cdot |\mathcal{I}| + |\mathcal{C}|)$. Again, in Section V, we will show a case where the privacy set can be restricted such that $|\mathcal{I}|$ and $|\mathcal{C}|$ are linear in n.

When we concretely define the inference set $\mathcal{I}$ and privacy set $\mathcal{C}$ for a specific algorithm, we must prove two things:

1) $\mathcal{C}$ satisfies the definition of a valid privacy set in Def. 4, and
2) All the neighboring assignments in $\mathcal{C}$ are present in $\mathcal{I}$, i.e. $\forall (\mathbf{x}_\mathcal{C}, \mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{C}$, $(\mathbf{x}_\mathcal{C}, y_\mathcal{C}) \in \mathcal{I}$ and $(\mathbf{x}'_\mathcal{C}, y_\mathcal{C}) \in \mathcal{I}$.

We provide an example of one such instantiation for randomized response in Section V.

C. Finding Tight Accuracy Bound

In the case of accuracy bound synthesis, we can also limit the number of values we need to compute to leverage symmetries in the problem to reduce the runtime of the accuracy computation. We introduce the accuracy set as follows.

Definition 5 (Accuracy Set)**.** *An* accuracy set $\mathcal{A}$ *for* $A : \mathcal{X}^n \to \mathcal{Y}$ *is a set* $\mathcal{A}$ *such that*

1) $\mathcal{A} \subseteq \mathcal{X}^n$
2) $\forall \mathbf{x} \in \mathcal{X}^n$*, there exists* $\mathbf{x}_\mathcal{A} \in \mathcal{A}$ *such that*

$$\sum_{y \in [\mathcal{V}[\mathbf{x}]-\alpha, \mathcal{V}[\mathbf{x}]+\alpha]} \Pr(A(\mathbf{x}) = y) =$$

$$\sum_{y \in [\mathcal{V}[\mathbf{x}_\mathcal{A}]-\alpha, \mathcal{V}[\mathbf{x}_\mathcal{A}]+\alpha]} \Pr(A(\mathbf{x}_\mathcal{A}) = y).$$

where $\mathcal{V}[\mathbf{x}]$ is the target output for input $\mathbf{x}$.

We present our solution in Algorithm 3. Again, use the INFERENCE algorithm (Algorithm 1) as a subprocess to compute exact probabilistic inference via WMC.

Algorithm 3 ExactAcc

Input: WBF (φ, w) of $A : \mathcal{X}^n \to \mathcal{Y}$, accuracy set $\mathcal{A}$, vector of target outputs $\mathcal{V}$, accuracy parameter α, and inference set $\mathcal{I}$ that contains all $(\mathbf{x}, y)$ pairs such that $x \in \mathcal{A}$ and $y \in [\mathcal{V}_\mathbf{x} - \alpha, \mathcal{V}_\mathbf{x} + \alpha]$

Output: Minimal probability p and worst case input $\mathbf{acc}$

```
M ← INFERENCE(φ, I)
p ← ∞, acc ← ∅
for x in (I_{X^n}) do
    if Σ_{y∈[V_x−α, V_x+α]} M[x, y] < p then
        p ← Σ_{y∈[V_x−α, V_x+α]} M[x, y]
        acc ← x
    end if
end for
return p, acc
```

Again, we return to our binary randomized response example. Since we are considering accuracy, we have to wrap the raw output to compute some quantifiable query. For example, we can add a counting query wrapper, such that the output of $\mathsf{RRcount}^2(x_1, x_2) = sum(\mathsf{RR}^2(x_1, x_2))$. Here we set $\alpha = 1,\ \lambda = 0.2,\ \mathcal{I}_{\mathsf{RRcount}^2} = \{([0,0],0), ([1,1],0), ([0,0],1), ([1,0],1), ([1,1],1), ([1,0],2), ([1,1],2)\}$, and $\mathcal{A}_{\mathsf{RRcount}^2} = \{[0,0],[1,0],[1,1]\}$ where $\mathcal{V}_{[0,0]} = 0, \mathcal{V}_{[1,0]} = 1, \mathcal{V}_{[1,1]} = 2$. In this case, the output of Algorithm 3 would be $\mathbf{acc} = [0,1]$, $p = .96$. This means that $[1,0]$ is the worst case accuracy assignment for $\mathsf{RRcount}^2$ and $1 - \beta = .96$ is a tight accuracy bound.

1) Correctness and Complexity of Accuracy Bound Synthesis: To verify correctness of ExactAcc, we must show that the minimal accuracy probability found using the potentially restricted inference and accuracy set is the same as the minimal accuracy probability using exhaustive inference and accuracy sets.

Theorem 3. *The output* $\mathbf{acc}$ *of Algorithm 3 is a solution to the Accuracy Bound Synthesis problem.*

Proof. Let WBF (φ, w) be the WBF of $A : \mathcal{X}^n \to \mathcal{Y}$, $\mathcal{V}$ be a vector of target outputs of A, $\alpha \geq 0$ and let $\mathcal{I}$, $\mathcal{A}$ be inference and accuracy sets of A such that $\mathcal{I}$ contains all $(\mathbf{x}_\mathcal{A}, y)$ where $\mathbf{x}_\mathcal{A} \in \mathcal{A}$ and $y \in [\mathcal{V}[\mathbf{x}_\mathcal{A}] - \alpha, \mathcal{V}[\mathbf{x}_\mathcal{A}] + \alpha]$. We know that $\mathbf{acc}$ minimizes $\sum_{y \in [\mathcal{V}[\mathbf{x}_\mathcal{A}]-\alpha, \mathcal{V}[\mathbf{x}_\mathcal{A}]+\alpha]} M[\mathbf{x}_\mathcal{A}, y]$ by the definition of Algorithm 3, and that $M[\mathbf{x}_\mathcal{A}, y] = \Pr(A(\mathbf{x}_\mathcal{A}) = y)$ by the definition of Algorithm 1. Therefore, $\mathbf{acc}$ minimizes $\Pr(A(\mathbf{x}_\mathcal{A}) = y)$ over $\mathbf{x}_\mathcal{A} \in \mathcal{A}$. By the definition of an accuracy set, for all $\mathbf{x} \in \mathcal{X}^n$, there exists a $\mathbf{x} \in \mathcal{A}$ such that $\sum_{y \in [\mathcal{V}[\mathbf{x}]-\alpha, \mathcal{V}[\mathbf{x}]+\alpha]} \Pr(A(\mathbf{x}) = y) = \sum_{y \in [\mathcal{V}[\mathbf{x}_\mathcal{A}]-\alpha, \mathcal{V}[\mathbf{x}_\mathcal{A}]+\alpha]} \Pr(A(\mathbf{x}_\mathcal{A}) = y)$. Therefore, c maximizes $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)}$ over all $(\mathbf{x}, \mathbf{x}', y) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$. By Theorem 1, this means that $\mathbf{acc}$ minimizes $\Pr(|A(\mathbf{x}) - \mathcal{V}_\mathbf{x}| \leq \alpha)$ over all $\mathbf{x} \in \mathcal{X}^n$ and is therefore a solution to the Accuracy Bound Synthesis problem. □

Algorithm 3 runs in $O(\mathsf{WMC} \cdot |\mathcal{I}| + |\mathcal{A}| \cdot \alpha)$.

Like with the privacy algorithm, there are two main proof requirements when we design a concrete inference set $\mathcal{I}$ and accuracy set $\mathcal{A}$ to use in the accuracy bound synthesis algorithm. These requirements are:

1) $\mathcal{A}$ satisfies the definition of a valid accuracy set in Definition 5 and
2) all necessary input/output assignments are present in $\mathcal{I}$, i.e. $\forall x \in \mathcal{A}$ and $y \in [\mathcal{V}[\mathbf{x}] - \alpha, \mathcal{V}[\mathbf{x}] + \alpha]$, $(\mathbf{x}, y) \in \mathcal{I}$.

V. Randomized Response Case Study

We have been using an example of binary randomized response for $n = 2$. In this section, we generalize this solution to any n and provide a detailed explanation of how to utilize these tools for a tractable solution to the tight privacy and accuracy bound problem.

Algorithm 4 Randomized Response (RR)

Input: Bit array $\mathbf{x}$ of true client messages.

Output: Bit array $\mathbf{y}$ of randomized client messages.

```
Initialize bit array y of length |x|
for i ∈ {1, ..., |x|} do
    y[i] ← 1 − x[i] with probability λ, otherwise x[i].
end for
return y
```

In this section, we provide a WBF for RR and show the complexity of ExactDP and ExactAcc for RR with exhaustive search space. We then analyze the correctness of ExactDP with restricted search space that leverages the inherent symmetries in RR that runs linearly in n (multiplied by WMC). We also provide an instantiation of ExactAcc with restricted search space that runs in time quadratic in n (again multiplied by WMC).

A. Weighted Boolean Formula for RR

We can manually craft a WBF (φ, w) for randomized response where n is the number of clients. We set

$$\varphi = \bigwedge_{i=1}^{n} y_i \leftrightarrow ((\overline{\theta}_i \wedge x_i) \vee (\theta_i \wedge \overline{x}_i)) \tag{6}$$

where each x_i corresponds to the bit value in the input vector of RR, and the y_i's likewise correspond with the output vectors. The θ values correspond with the coin flips in the randomized portion of the algorithm.

We set the weighting function to be $w(\theta_i) = 1 - \lambda$, or the probability that the value of x_i does not flip, and $w(\overline{\theta}_i) = \lambda$ as the probability that x_i does flip. For all other literals, we set $w(\cdot) = 1$.

Theorem 4. (φ, w) *is a valid WBF of* RR.

Proof. Let $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^n \times \mathcal{Y}^n$. For each $i \in \{1, \ldots, n\}$ assign the x_i and y_i literals in φ the values of $\mathbf{x}[i]$ and $\mathbf{y}[i]$, and

$\overline{x}_i = 1 - \mathbf{x}[i]$. If $\mathbf{x}[i] = \mathbf{y}[i]$, then for $y_i \leftrightarrow ((\overline{\theta}_i \wedge x_i) \vee (\theta_i \wedge \overline{x}_i))$ to be satisfied, $\theta_i = 0$. If $\mathbf{x}[i] \neq \mathbf{y}[i]$, then for $y_i \leftrightarrow ((\overline{\theta}_i \wedge x_i) \vee (\theta_i \wedge \overline{x}_i))$ to be satisfied, $\theta_i = 1$.

This is the only satisfying assignment of φ for $x,\ y$ assigned as described, therefore there is only one model, and the weighted model count is therefore $\prod_{\ell \in m} w(\ell)$.

The product of weights of literals is $w(x_i) \cdot w(y_i) \cdot w(\overline{\theta}_i) = 1 \cdot 1 \cdot (1 - \lambda)$ for i s.t. $\mathbf{x}[i] = \mathbf{y}[i]$ and $w(x_j) \cdot w(y_j) \cdot w(\theta_j) = 1 \cdot 1 \cdot \lambda$ for i s.t. $\mathbf{x}[i] \neq \mathbf{y}[i]$, so $WMC((\varphi, w)) = (1 - \lambda)^{\#i \text{ s.t. } \mathbf{x}[i]=\mathbf{y}[i]} \lambda^{\#i \text{ s.t. } \mathbf{x}[i]\neq\mathbf{y}[i]} = \Pr(\mathsf{RR}(\mathbf{x}) = \mathbf{y})$. □

We provide this analysis to demonstrate what a valid WBF for RR would look, however, as discussed in Section III, this is not necessarily the best WBF for computing the WMC of RR. In Section VI we use a probabilistic programming language to find the WBF, compile to a compact BDD, and efficiently query the WMC.

B. Exhaustive Solution

It is evident that for ExactDP and ExactAcc, setting $\mathcal{I}_{\mathsf{RR}} = \mathcal{X}^n \times \mathcal{Y}$, $\mathcal{C}_{\mathsf{RR}} = \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$, and $\mathcal{A}_{\mathsf{RR}} = \mathcal{X}^n$ gives the correct bounds for both privacy and accuracy because we are optimizing the bounds over all possible inputs and outputs. However, because $|\mathcal{X}^n| = 2^n$ and $|\mathcal{Y}| = 2^n$ for randomized response, when we analyze this solution, we see that the runtime of the inference algorithm is $O(\mathsf{WMC} \cdot |\mathcal{I}_{\mathsf{RR}}|) = O(\mathsf{WMC} \cdot 4^n)$, the runtime of ExactDP is $O(\mathsf{WMC} \cdot |\mathcal{I}_{\mathsf{RR}}| \cdot |\mathcal{C}|) = O(\mathsf{WMC} \cdot 4^n + 16^n)$ and the runtime of ExactAcc is $O(\mathsf{WMC} \cdot |\mathcal{I}_{\mathsf{RR}}| + |\mathcal{A}_{\mathsf{RR}}| \cdot \alpha) = O(\mathsf{WMC} \cdot 4^n + 2^n \cdot \alpha)$. We therefore need to find a way to improve this runtime for randomized response.

C. Leveraging Symmetries for Finding Privacy Bound

Because of the independence properties between different clients responses, there are many inherent symmetries in this algorithm. In this section, we identify an inference and privacy set which satisfy the coverage properties from the previous section, and reduce the complexity by orders of magnitude.

1) Counting Flips: We identify one significant symmetry in RR which is the number of clients whose bit flips. In other words, we can find a representative input/output pair for each number of bit flips that occurs.

Lemma 1. *Given $i \in \{1, \ldots, n\}$, $\forall \mathbf{x} \in \mathcal{X}^n$ and $\mathbf{y} \in \mathcal{Y}^n$ such that $count(\mathbf{x} \oplus \mathbf{y}) = i$, $\Pr(A(\mathbf{x}) = \mathbf{y}) = \Pr(A(1^i 0^{n-i}) = 0^n)$.*

Proof. Let $\mathbf{x} \in \mathcal{X}^n$ and $\mathbf{y} \in \mathcal{Y}^n$ such that $count(\mathbf{x} \oplus \mathbf{y}) = i$. This means that there are i entries such that $\mathbf{x}[i] \neq \mathbf{y}[i]$. Therefore, $\Pr(\mathsf{RR}(\mathbf{x}) = \mathbf{y}) = (1-\lambda)^{\#i \text{ s.t. } \mathbf{x}[i]=\mathbf{y}[i]} \lambda^{\#i \text{ s.t. } \mathbf{x}[i]\neq\mathbf{y}[i]} = \Pr(A(1^i 0^{n-i}) = 0^n)$. □

We also identify a key fact about the relationship between the number of bit flips in neighboring inputs, specifically that a neighboring input has either one more or one less bit flip with respect to the output.

Lemma 2. *For neighboring inputs $\mathbf{x}, \mathbf{x}' \in \mathcal{X}^n$, $count(\mathbf{x}' \oplus y) = count(\mathbf{x} \oplus \mathbf{y}) + 1$ or $count(\mathbf{x} \oplus \mathbf{y}) - 1$.*

Lemma 2 follows directly from the definition of neighboring inputs.

2) Constructing Privacy Set: Because we have these inherent symmetries in the number of flips between each input/output pair, we can define the privacy set to be

$$\mathcal{C}_{\mathsf{RR}} = \{(1^i 0^{n-i}, 1^{i+1} 0^{n-(i+1)}, 0^n)\}_{i \in \{0,\ldots,n-1\}} \cup \quad (7)$$

$$\{(1^i 0^{n-i}, 1^{i-1} 0^{n-(i-1)}, 0^n)\}_{i \in \{1,\ldots,n\}}. \quad (8)$$

We note that $|\mathcal{C}_{\mathsf{RR}}| = 2n$.

Theorem 5. *$\mathcal{C}_{\mathsf{RR}}$ is a valid privacy set for* RR.

Proof. Let $(\mathbf{x}, \mathbf{x}', \mathbf{y}) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}^n$ such that $\mathbf{x}, \mathbf{x}'$ are neighbors.

If $count(\mathbf{x}' \oplus \mathbf{y}) = count(\mathbf{x} \oplus \mathbf{y}) + 1$, then by Lemma 1 with $count(\mathbf{x} \oplus \mathbf{y}) = i$ and $count(\mathbf{x}' \oplus \mathbf{y}) = i + 1$, $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)} = \frac{\Pr(A(1^i 0^{n-i})=0^n)}{\Pr(A(1^{i+1} 0^{n-(i+1)})=0^n)}$. By definition, $(1^i 0^{n-i}, 1^{i+1} 0^{n-(i+1)}, 0^n)$ is in $\mathcal{C}_{\mathsf{RR}}$.

If $count(\mathbf{x}' \oplus \mathbf{y}) = count(\mathbf{x} \oplus \mathbf{y}) - 1$, then by Lemma 1 with $count(\mathbf{x} \oplus \mathbf{y}) = i$ and $count(\mathbf{x}' \oplus \mathbf{y}) = i - 1$, $\frac{\Pr(A(\mathbf{x})=y)}{\Pr(A(\mathbf{x}')=y)} = \frac{\Pr(A(1^i 0^{n-i})=0^n)}{\Pr(A(1^{i-1} 0^{n-(i-1)})=0^n)}$. By definition, $(1^i 0^{n-i}, 1^{i-1} 0^{n-(i-1)}, 0^n)$ is in $\mathcal{C}_{\mathsf{RR}}$.

By Lemma 2, this covers all possible cases, and by definition of $\mathcal{C}_{\mathsf{RR}}$, $\forall (\mathbf{x}, \mathbf{x}', \mathbf{y}) \in \mathcal{C}_{\mathsf{RR}}$, $(\mathbf{x}, \mathbf{x}', \mathbf{y}) \in \mathcal{X}^n \times \mathcal{X}^n \times \mathcal{Y}$. Therefore $\mathcal{C}_{\mathsf{RR}}$ is a valid privacy set. □

We can now build a smaller inference set that computes all necessary probabilities that are used in the privacy set. In this case we define

$$\mathcal{I}_{\mathsf{RR}} = \{(1^i 0^{n-i}, 0^n)\}_{i \in \{0,\ldots,n\}}. \quad (9)$$

Theorem 6. *$\forall (\mathbf{x}_{\mathcal{C}_{\mathsf{RR}}}, \mathbf{x}'_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{C}_{\mathsf{RR}}$, $(\mathbf{x}_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{I}_{\mathsf{RR}}$ and $(\mathbf{x}'_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{I}_{\mathsf{RR}}$*

Proof. Let $(\mathbf{x}_{\mathcal{C}_{\mathsf{RR}}}, \mathbf{x}'_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{C}_{\mathsf{RR}}$ and $i \in \{0, 1, \ldots, n\}$. By the definition of $\mathcal{C}_{\mathsf{RR}}$, $\mathbf{x}$ is $(1^i 0^{n-i}, 1^{i+1} 0^{n-(i+1)}, 0^n)$ or $(1^i 0^{n-i}, 1^{i-1} 0^{n-(i-1)}, 0^n)$. Therefore, $(\mathbf{x}_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{I}_{\mathsf{RR}}$ and $(\mathbf{x}'_{\mathcal{C}_{\mathsf{RR}}}, y_{\mathcal{C}_{\mathsf{RR}}}) \in \mathcal{I}_{\mathsf{RR}}$ as desired. □

By the previous theorems, ExactDP is correct for $\mathcal{I}_{\mathsf{RR}}$ and $\mathcal{C}_{\mathsf{RR}}$.

3) Complexity: Since $|\mathcal{I}_{\mathsf{RR}}| = n + 1$ and $|\mathcal{C}_{\mathsf{RR}}| = 2n$, the inference algorithm (Algorithm 1) runs in $O(\mathsf{WMC} \cdot n)$ time and ExactDP (Algorithm 2) runs in $O(\mathsf{WMC} \cdot n)$ time as well.

D. Computing Accuracy

For accuracy computation, we consider the counting query on RR. Here, we take the output vector $\mathbf{y}_i$ of RR and compute the number of 1's in that vector. In this case, the input space is $\mathcal{X}^n = \{0, 1\}^n$ and output space is $\{0, 1, \ldots, n\}$. We refer to this counting version of RR as RRcount. Here, the set $\mathcal{V}$ is equivalent to the counting function over Booleans, $count : \{0, 1\}^n \to \{0, \ldots, n\}$.

Here too we have a key lemma about the symmetries.

Lemma 3. *Given $i \in \{0, \ldots, n\}$, $\forall y \in \{0, \ldots, n\}$, for any $\mathbf{x}$ such that $count(\mathbf{x}) = i$, it is true that $\Pr(\mathsf{RRcount}(\mathbf{x}) = y) = \Pr(\mathsf{RRcount}(1^i 0^{n-i}) = y)$.*

Proof. Let $\mathbf{x} \in \mathcal{X}^n$, $y \in \mathcal{Y}$ such that $count(\mathbf{x}) = i$. By the definition of RRcount, $\Pr(\mathsf{RRcount}(\mathbf{x}) = y) = \sum_{y_{\mathsf{RR}} \text{ s.t. } count(y_{\mathsf{RR}})=y} \Pr(\mathsf{RR}(x) = y_{\mathsf{RR}}) = \Pr(\mathsf{RRcount}(1^i 0^{n-i}) = y)$. □

1) Constructing $\mathcal{A}$: We define the accuracy set to be

$$\mathcal{A}_{\mathsf{RRcount}} = \{1^i 0^{n-i}\}_{i \in \{0,\ldots,n\}} \tag{10}$$

Theorem 7. *$\mathcal{A}_{\mathsf{RRcount}}$ is an accuracy set for* RRcount.

Proof. Let $\mathbf{x} \in \mathcal{X}^n$ and $count(\mathbf{x}) = i$. By Lemma 3, $\Pr(\mathsf{RRcount}(\mathbf{x}) = y) = \Pr(\mathsf{RRcount}(1^i 0^{n-i}) = y)$. Since $count(\mathbf{x}) = count(1^i 0^{n-i})$, $\sum_{y \in [count(\mathbf{x})-\alpha, count(\mathbf{x})+\alpha]} \Pr(\mathsf{RRcount}(\mathbf{x}) = y) = \sum_{y \in [count(1^i 0^{n-i})-\alpha, count(1^i 0^{n-i})+\alpha]} \Pr(\mathsf{RRcount}(\mathbf{x}_{\mathcal{A}}) = y)$.
By definition, $\forall \mathbf{x} \in \mathcal{A}_{\mathsf{RRcount}}$, $\mathbf{x} \in \mathcal{X}^n$. Therefore $\mathcal{C}_{\mathsf{RRcount}}$ is a valid accuracy set. □

We define the inference set to accommodate all necessary values.

$$\mathcal{I}_{\mathsf{RRcount}} = \bigcup_{i \in [0,n]} \bigcup_{j \in [i-\alpha, i+\alpha]} \{(1^i 0^{n-i}, j)\} \tag{11}$$

Theorem 8. *$\mathcal{I}_{\mathsf{RRcount}}$ contains all $(\mathbf{x}, y)$ pairs such that $\mathbf{x} \in \mathcal{A}_{\mathsf{RRcount}}$ and $y \in [\mathcal{V}[\mathbf{x}] - \alpha, \mathcal{V}[\mathbf{x}] + \alpha]$*

Proof. Let $\mathbf{x} \in \mathcal{A}_{\mathsf{RRcount}}$ such that $count(\mathbf{x}) = i$. By definition of $\mathcal{A}_{\mathsf{RRcount}}$, $\mathbf{x} = 1^i 0^{n-i}$. By definition of $\mathcal{I}_{\mathsf{RRcount}}$, $\forall y \in [i - \alpha, i + \alpha]$, $(1^i 0^{n-i}, y)$ as desired. □

By these theorems, ExactAcc with this inference and accuracy set is correct.

2) Complexity: We have $|\mathcal{I}_{\mathsf{RRcount}}| = n \cdot 2\alpha$ and $|\mathcal{A}_{\mathsf{RRcount}}| = (n+1) \cdot \alpha$. Therefore, inference (Algorithm 1) and accuracy bound synthesis (Algorithm 3) both run in $O(\mathsf{WMC} \cdot n\alpha)$ (or $O(\mathsf{WMC} \cdot n^2)$ time since $2\alpha \leq n$).

VI. Evaluation

We have shown that our method is theoretically sound. In this section we demonstrate how our implemented method far outperforms prior methods even for the exhaustive case. We also show how the expressiveness of probabilistic programming can be leveraged to easily implement new algorithms with more complex input and output spaces, a feat which would be difficult using prior techniques for exact verification. We then go on to discuss how our method computes accuracy and demonstrate the importance of also having access to accuracy analysis when evaluating private algorithms.

We implement the weighted model counting solution with BDDs by running programs in the Dice probabilistic programming language [25]. We compare with a discrete time Markov chain (DTMC) model checking solution using the Storm probabilistic model checker [33] and with DiPC, an implementation of the decision procedure described in [21]. All experiments are written in Python and use the solver (Dice or Storm) as a subprocess. We implement INFERENCE (Algorithm 1), ExactDP (Algorithm 2), and ExactAcc (Algorithm 3) in Python. In the case of DiPC, we use the published code without modification. To easily compare performance on different parameters, we also develop a tool to generate Dice programs and Storm models for different DP algorithms, input/output spaces, and randomization parameters. Experiments are run on a 16 core AMD EPYC with 64GB of RAM and code for the experiments can be found here: https://github.com/lisaoakley/wmc_for_privacy_and_accuracy.

A. Comparison to Markov Chain Model Checking

The heart of our method is using weighted model counting on tractable circuits to vastly improve both the size of the models and the implementation runtime. In our experiments, we use the Dice probabilistic programming language to model and compute tight privacy bounds for the randomized response algorithm as explained in Section V for variable number of clients n.

Prior work on inference for exact DP verification uses model checking on discrete time Markov chains (DTMCs) to perform exact inference [22], [23]. The examples in these papers are limited and require adaptation to solve our synthesis problem. We compare our solution with a similar Markov chain model checking technique and confirm that our WMC solution outperforms the Markov chain solution in model size, number of solver runs, and inference time.

In Table II, we see that, for various n, ExactDP finds an extremely compact BDD for randomized response, where the model size is $n + 2$. This contrasts with the Markov chain solution where the models are size 7^n. This is because every state in the Markov chain represents a possible setting of every free variable in the model. With each client added in randomized response, we multiply the number of free variables in the Markov chain, which causes an exponential increase in the number of states. These tools were developed to handle dynamic programs for small state spaces, and are therefore not intended to handle problems with large numbers of free variables and little to no notion of dynamics over time. BDDs, however, are exactly optimized for this kind of problem.

The Markov chain solution also requires a manually crafted Markov chain which means that the Markov chains used in our experiments might not be the optimal encoding of this algorithm. However, finding an optimal encoding is a hard problem and requires technical knowledge on the internals of the model checking process. A benefit of our solution is that the Dice program is almost identical to the randomized response pseudocode, and the optimization of the BDD size is handled automatically.

Another benefit of inference via WMC on BDDs demonstrated in Table II is that Dice is able to compute the entire output distribution of the program for a given input in one run, whereas the Markov chain model checking solution requires a run for each input/output pair. As shown in Table II we have to run Storm 4^n times in the exhaustive case, but even in the exhaustive case we only have to run Dice 2^n times. We leverage the symmetries in the randomized response algorithm such that for the Dice restricted solution, we only have to run

Method	n	BDD Size	# States	# Trans.	# Solver Runs	Inf. Time (s)	Synth. Time (s)	Build Time (s)
DTMC (Exhaustive)	2	-	49	100	16	0.0104	< 0.0001	0.0001
	4	-	2401	8488	256	0.8877	0.0007	0.0005
	6	-	TO	TO	TO	TO	TO	TO
ExactDP (Exhaustive)	2	4	-	-	4	0.152	0.0001	0.0005
	4	6	-	-	16	0.4128	0.0011	0.0015
	6	8	-	-	64	1.5683	0.0226	0.008
	8	10	-	-	256	8.097	0.5285	0.024
	10	12	-	-	1024	63.1988	11.016	0.0917
DTMC (Restricted)	5	-	16807	73054	32	1.371	0.0001	0.0001
	10	-	TO	TO	TO	TO	TO	TO
ExactDP (Restricted)	5	7	-	-	1	0.0239	< 0.0001	0.0001
	10	12	-	-	1	0.0518	0.0001	0.0001
	15	17	-	-	1	1.2864	0.0001	0.0001
	20	22	-	-	1	53.9282	0.0002	0.0001

TABLE II: Comparison of state space and runtimes for a BDD weighted model counting solution (ExactDP) and a Markov chain model checking solution (DTMC). Model size for ExactDP is the number of nodes in the BDD (BDD size) and for DTMC is the number of states and transitions in the Markov chain. TO means that the experiment timed out on probabilistic inference. For ExactDP and DTMC we provide results for both the exhaustive solution and restricted solution which uses symmetry sets as described in Section V. Model size is independent of the symmetry sets, e.g. the model size for $n = 10$ is 12 for both ExactDP exhaustive and ExactDP restricted. # Solver Runs indicates the number of calls to the solver required to compute the necessary probabilities. Inf. and Synth. time are the the total times in seconds for running implementations of the Inference and Privacy Bound Synthesis algorithms including subprocess times. Build time is the time it takes to build the program or model sent to the solver.

the solver once, compared with Storm where we still have to run it 2^n times.

The outcome of these factors is that the Storm solution times out for $n = 6$ in the exhaustive case and $n = 10$ in the restricted case, while our solution with a Dice solver can find a tight bound in around a minute for $n = 10$ in the exhaustive case, and less than a minute for $n = 20$ in the restricted case as can be seen in Table II.

B. Comparison to DiPC

In [21], Barthe *et. al* provide a decision procedure for exactly verifying DP for a class of algorithms that can be encoded as DTMCs. The implementation of their decision procedure, DiPC, exhaustively computes the likelihood ratio by generating Wolfram Mathematica® scripts for a subset of the class of programs on which they prove decidability.

In Table III, we provide an empirical comparison of ExactDP vs. DiPC for randomized response with varying numbers of clients, n. DiPC is primarily an implementation to illustrate the decision procedure, and therefore is not optimized for fast probabilistic inference. As expected, we see that our tool outperforms DiPC as the number of clients increases.

n	DiPC [21]	ExactDP (Exhaustive)	ExactDP (Restricted)
2	0s	0s	0s
4	0s	0s	0s
6	1s	2s	0s
8	7s	9s	0s
10	149s	74s	0s
12	2841s	1965s	0s

TABLE III: Runtime comparison between DiPC [21] and ExactDP with exhaustive and restricted search spaces (rounding to the nearest second to match the granularity reported by DiPC). For $n > 8$, ExactDP outperforms DiPC even without using any symmetry optimizations.

C. Expressiveness and the Geometric Above Threshold

We have shown that our method outperforms Markov chain model checking for computing tight privacy bounds both for the exhaustive and restricted solutions in the case of binary randomized response. We now demonstrate the expressiveness of our solution using the truncated geometric above threshold mechanism.

The geometric above threshold takes as input a length n list of integers in $\{0, 1, \ldots, k\}$, and outputs an integer representing the index of the first value in the list that exceeds the threshold. We provide the complete pseudocode and an example Dice program for the randomized (private) above threshold algorithm in Appendix A. For methods like model checking with Markov chains, this would be a very difficult algorithm to encode as it has multiple invocations of the geometric mechanism and multiple growth dimensions which would require many explicit states and transitions to encode as a Markov chain.

We encode this algorithm as a simple $2n + 3$ line Dice program and run our exhaustive privacy verification bound algorithm on it. We see in Fig. 2 that, even with the exhaustive solution, we are able to easily encode compact, computable BDDs for a variety of list lengths and max int values.

Not only does Dice provide a simple encoding, but we see that the optimized BDDs are extremely reasonably sized. Even

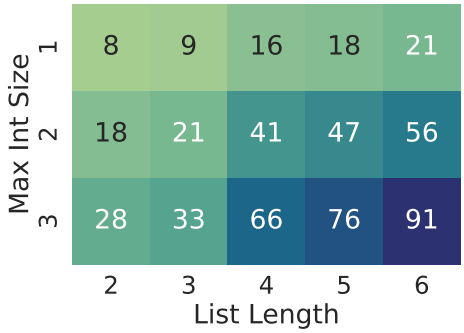
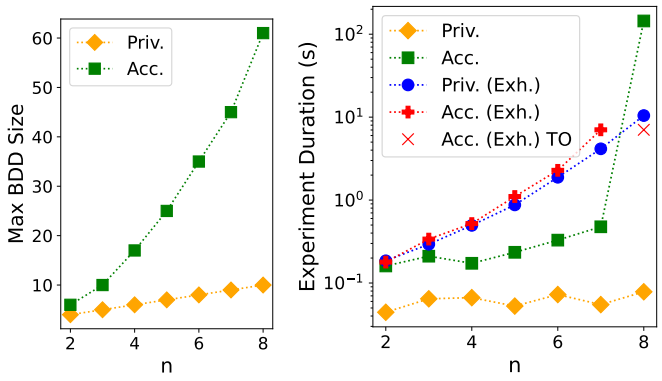


Fig. 2: BDD sizes for the truncated geometric above threshold algorithm for various maximum integer sizes (k) and list lengths (n).



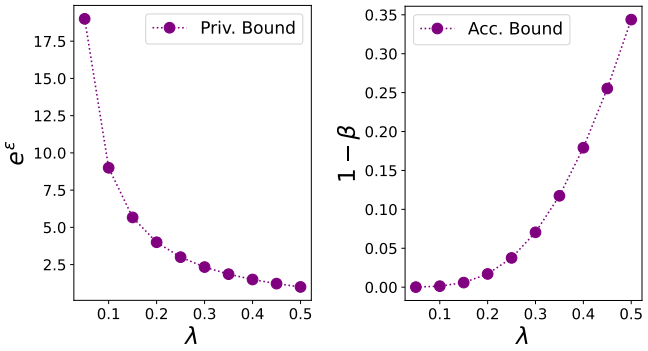
(a) BDD Sizes (b) Experiment Durations

Fig. 3: BDD sizes and experiment durations for accuracy bound synthesis with ExactAcc for randomized response with counting over various numbers of clients n and $\lambda = 0.2$.

for integers between 0 and 3 and list lengths up to 6 (which means $6^4 = 1296$ possible inputs and two invocations of the geometric distribution over 4 values each), we see that the generated BDD is able to represent this with only 91 nodes. This is still less than the number of states in the Markov chain for the much simpler randomized response algorithm for $n = 3$ when looking at the Storm solution. This means that exact inference, even for more complex DP algorithms can be orders of magnitude more efficient than previously believed.

D. Accuracy

A key benefit of our method is its ability to compute tight accuracy bounds. We see in Fig. 3a that having large integral output space (for example an output space of $\{1 \dots n+1\}$ as is the case for the randomized response algorithm with a counting wrapper) causes the BDD sizes to grow more quickly than in privacy computation. As we noted in the theoretical portion of the paper, finding accuracy bounds also has larger time complexity due to the range of outputs bounded by α. However, we are still able to compute accuracy bounds for small n in under 4 minutes, as shown in Fig. 3b.



(a) Privacy Bound (RR) (b) Accuracy (RR counting)

Fig. 4: Tight Accuracy and privacy bounds over varying coin flip parameters, λ, for binary randomized response with $n = 8$. Generated using the ExactDP and ExactAccimplementations with restricted search space.

Input	$\mathbf{1-\beta}$
00000000	0.9437184
11111111	0.9437184
01111111	0.9723904
00000001	0.9723904

TABLE IV: 4 inputs for which $1-\beta$ is lowest for randomized response counting with $n = 8$ and $\lambda = 0.2$. Generated using ExactAccimplementation with restricted search space.

Even though we are only able to perform tight accuracy bound synthesis on small examples, we begin to see the utility of our program as a tool in the algorithm design process. For example, in Fig. 4 we use our tool to find tight privacy and accuracy bounds for noise parameters. We see that noise is not linearly related to privacy or accuracy, as often implicitly assumed when algorithm designers minimize noise as a proxy for accuracy, as is done for differentially private synthesis as in [20]. Using such a proxy algorithm will result in an inexact outcome, and it is important to have a more detailed view of accuracy, even if n is small.

Our method goes beyond bound synthesis. We can run a variation of ExactAcc and get each example for which there is a unique $1-\beta$, sorted by their values. We show a result generated by our tool of 4 inputs for which $1-\beta$ is lowest for randomized response with $n = 8$ and $\lambda = 0.2$ in Table IV. An algorithm designer can use this information to inform which examples have the greatest impact on accuracy.

VII. Related Works

A. Exact Methods

There are a few closely related works on exact differential privacy verification. In [22], [23], Liu *et. al* provide a probabilistic model checking approach for exactly solving the differential privacy verification problem. We provide a comparison with this approach in Section VI.

In [21] Barthe *et. al*, provide theoretical grounding for a class of algorithms for which exact DP verification is decidable. This class of programs, namely DiPWhile programs, are looping programs in which each real and integer variable can be assigned a bounded number of times during program execution. Here, the authors propose a similar decision procedure to our exhaustive algorithm, however, they do not discuss the method for performing exact probabilistic inference except to prove that the probabilities are computable because DiPWhile programs can be encoded as DTMCs (Lemma 7.1). In contrast, our work has a focus on the computational gains we can achieve if we use weighted model counting for this inference step.

Along with theoretical analysis, [21] presents an implementation of their decision procedure which solves the differential privacy verification problem for a subset of DiPWhile programs, including randomized response, by generating Wolfram Mathematica® scripts to perform probabilistic inference. As shown in Section VI, our tool outperforms DiPC for randomized response, even in the exhaustive case where we do not restrict the search space at all.

In [21], the authors also present results for solving the (ε, δ)-DP problem for a very small example. Our technique could also be immediately applied to verify (ε, δ)-DP by checking every subset of output events rather than each individual output. This straightforward, exhaustive solution has an even larger computation explosion than the exhaustive solution for ε-DP, as you have to run inference and validation for the power set of the output space. As is mentioned in [21], this is not practical for exhaustive automated techniques. Further exploration into symmetry-finding for the (ε, δ)-DP case remains an interesting open problem.

B. Finding Proofs and Counterexamples

Other works which are closely related to ours are works on finding counterexamples to differential privacy or finding proofs of differential privacy. In [10], Ding *et. al* develop a statistical procedure for finding a candidate counterexample to differential privacy in probabilistic algorithms. In [29], Bichsel *et. al* present a similar technique which uses the same heuristic for limiting the search space as in [10], but replace hypothesis testing with trained classifiers.

Implicitly, our method for finding symmetries is similar to the technique of Ding *et. al* [10]. They manually choose a heuristic for segmenting the input space, for example using a representative pair of neighboring inputs that represent a class of "one below" pairs, where x' has a single entry which is one unit smaller than its corresponding entry in x (e.g. $x = [1, 1, 1, 1, 1]$ and $x' = [0, 1, 1, 1, 1]$). In our paper, however, we formalize this logic and provide proof obligations to show that these representative inputs cover the whole search space, which allows us to introduce more formality and make provable statements about the exactness of our solutions.

In [11], Wang *et. al* develop a tool for finding counterexamples or proofs of validation of privacy in implementations of DP algorithms. In [12], Bichsel *et. al* use a sampling approach to find counterexamples to differential privacy. In [13], Zhang *et. al* use interpreters to find counterexamples to differential privacy.

C. Type Systems and Program Synthesis

There is significant work on developing type systems and program logics for differential privacy algorithms. This work begins with [34] in which Reed *et. al* develop a type system for differential privacy. Following this, [35]–[39] present other type systems and program logics for differential privacy. There is a related line of work which develops and implements a Hoare logic approach and uses probabilistic couplings to mechanize proofs of differential privacy via the apRHL logic and EasyCrypt [5]–[9]. In [40], Albarghouthi *et. al* propose a technique to automate these proofs. In [41], Barthe *et. al* develop a probabilistic programming framework which uses inference for writing verifiable DP programs.

There are also works on program synthesis for DP programs [20], [42], [43]. Our techniques could be used in these synthesis approaches for verifying intermediate programs, and as a more exact method of ensuring high accuracy while satisfying DP properties.

In a more theoretical approach, Gaboardi *et. al* [44] find complexity bounds for exact verification of DP in non-looping programs. In follow-up work to this, Bun *et. al* [45] extend this analysis to looping programs.

D. Privacy Auditing

Privacy auditing provides empirical methods to estimate the privacy leakage of an ML algorithm by mounting privacy attacks. Privacy auditing can be performed with membership inference attacks [46] or data poisoning [14], [15], but initial techniques developed for auditing require training thousands of ML models to provide confidence intervals for the estimated lower bound on the privacy parameter. Recent techniques showed how to rigorously perform estimation of the privacy parameter while using multiple randomized canaries [18] and eventually training a single ML model [16], [19].

E. Accuracy Verification

To our knowledge, there is very little work on theoretical or applied formal verification or bound synthesis for accuracy of DP algorithms. In [47], Barthe *et. al* provide theoretical analysis on the decidability of accuracy different classes of probabilistic computations. In [48], Lobo-Vesga *et. al* develop a programming framework for estimating accuracy bounds.

VIII. Conclusion and Future Work

We believe that our novel approach for synthesizing tight privacy and accuracy bounds has a lot of promise for developing techniques for computing accuracy and privacy properties in a wide range of differential privacy applications. Our approach uses state-of-the-art techniques for probabilistic inference via weighted model counting, and can benefit from ongoing advances in artificial intelligence and automated reasoning.

Though we are able to vastly improve automation from prior work by leveraging expressive probabilistic programming languages, one limitation is the manual step of identifying symmetries to define the inference, comparison, and accuracy sets. While there is room for improvement, in our experiments we demonstrate how the power of our WMC solution provides advances and utility, even for the exhaustive approach. In future work, we plan to investigate techniques from SAT solving and symmetry breaking to improve automation.

In future work, we can also extend this framework to other definitions of differential privacy including Rényi and approximate (or (ε, δ)) differential privacy [27]. Additionally, we can look at more complex systems with more complicated probability distributions such as randomized response with amplification by shuffling [49].

Our work opens the door for further automation and accessibility of using probabilistic programming languages techniques for verification and synthesis of DP algorithms using state-of-the-art inference tools.

Acknowledgment

We would like to thank Professors Marco Gaboardi and Jon Ullman for their technical guidance and input on differential privacy case studies. Thanks also to Lydia Zakynthinou, Konstantina Bairaktari, Ludmila Glinskih, Minsung Cho, and LaKyah Tyner for discussions about approaches and theoretical analysis. This work has been supported by NSF grant CNS-2247484.

References

[1] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," *Journal of Privacy and Confidentiality*, vol. 7, no. 3, 2016.

[2] I. Mironov, "On significance of the least significant bits for differential privacy," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 650–661.

[3] F. Tramer, A. Terzis, T. Steinke, S. Song, M. Jagielski, and N. Carlini, "Debugging differential privacy: A case study for privacy auditing," *arXiv preprint arXiv:2202.12219*, 2022.

[4] T. Stevens, I. C. Ngong, D. Darais, C. Hirsch, D. Slater, and J. P. Near, "Backpropagation clipping for deep learning with differential privacy," *arXiv preprint arXiv:2202.05089*, 2022.

[5] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella-Beguelin, "Verified computational differential privacy with applications to smart metering," in *2013 IEEE 26th Computer Security Foundations Symposium*. IEEE, 2013, pp. 287–301.

[6] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella Beguelin, "Probabilistic relational reasoning for differential privacy," in *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2012, pp. 97–110.

[7] G. Barthe, M. Gaboardi, E. J. G. Arias, J. Hsu, C. Kunz, and P.-Y. Strub, "Proving differential privacy in hoare logic," in *2014 IEEE 27th Computer Security Foundations Symposium*. IEEE, 2014, pp. 411–424.

[8] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P.-Y. Strub, "Proving differential privacy via probabilistic couplings," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, 2016, pp. 749–758.

[9] J. Hsu, *Probabilistic couplings for probabilistic reasoning*. University of Pennsylvania, 2017.

[10] Z. Ding, Y. Wang, G. Wang, D. Zhang, and D. Kifer, "Detecting violations of differential privacy," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 475–489.

[11] Y. Wang, Z. Ding, D. Kifer, and D. Zhang, "Checkdp: An automated and integrated approach for proving differential privacy or finding precise counterexamples," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 919–938.

[12] B. Bichsel, T. Gehr, D. Drachsler-Cohen, P. Tsankov, and M. Vechev, "Dp-finder: Finding differential privacy violations by sampling and optimization," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 508–524.

[13] H. Zhang, E. Roth, A. Haeberlen, B. C. Pierce, and A. Roth, "Testing differential privacy with dual interpreters," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–26, 2020.

[14] M. Jagielski, J. Ullman, and A. Oprea, "Auditing differentially private machine learning: How private is private SGD?" in *Proceedings of Advances in Neural Information Processing Systems*, ser. NeurIPS, vol. 33, 2020, pp. 22 205–22 216. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/fc4ddc15f9f4b4b06ef7844d6bb53abf-Paper.pdf

[15] M. Nasr, S. Song, A. Thakurta, N. Papernot, and N. Carlini, "Adversary instantiation: Lower bounds for differentially private machine learning," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 866–882. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00069

[16] G. Andrew, P. Kairouz, S. Oh, A. Oprea, H. B. McMahan, and V. Suriyakumar, "One-shot empirical privacy estimation for federated learning," *CoRR*, vol. abs/2302.03098, 2023.

[17] M. Nasr, J. Hayes, T. Steinke, B. Balle, F. Tramèr, M. Jagielski, N. Carlini, and A. Terzis, "Tight auditing of differentially private machine learning," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023.

[18] K. Pillutla, G. Andrew, P. Kairouz, H. B. McMahan, A. Oprea, and S. Oh, "Unleashing the power of randomization in auditing differentially private ML," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=mlbes5TAAg

[19] M. J. Thomas Steinke, Milad Nasr, "Privacy auditing with one (1) training run," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=mlbes5TAAg

[20] S. Roy, J. Hsu, and A. Albarghouthi, "Learning differentially private mechanisms," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 852–865.

[21] G. Barthe, R. Chadha, V. Jagannath, A. P. Sistla, and M. Viswanathan, "Deciding differential privacy for programs with finite inputs and outputs," in *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2020, pp. 141–154.

[22] D. Liu, B.-Y. Wang, and L. Zhang, "Model checking differentially private properties," in *Asian Symposium on Programming Languages and Systems*. Springer, 2018, pp. 394–414.

[23] ——, "Verifying pufferfish privacy in hidden markov models," in *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 2022, pp. 174–196.

[24] M. Chavira and A. Darwiche, "On probabilistic inference by weighted model counting," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 772–799, 2008.

[25] S. Holtzen, G. Van den Broeck, and T. Millstein, "Scaling exact inference for discrete probabilistic programs," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–31, 2020.

[26] S. Holtzen, T. Millstein, and G. Van den Broeck, "Generating and sampling orbits for lifted probabilistic inference," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 985–994.

[27] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[28] S. Holtzen, S. Junges, M. Vazquez-Chanlatte, T. Millstein, S. A. Seshia, and G. Van den Broeck, "Model checking finite-horizon markov chains with probabilistic inference," in *Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV)*, July 2021.

[29] B. Bichsel, S. Steffen, I. Bogunovic, and M. Vechev, "Dp-sniper: Black-box discovery of differential privacy violations using classifiers," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 391–409.

[30] G. Van den Broeck, K. Kersting, S. Natarajan, and D. Poole, *An Introduction to Lifted Probabilistic Inference*. MIT Press, 2021.

[31] A. Sabharwal, "Symchaff: exploiting symmetry in a structure-aware satisfiability solver," *Constraints*, vol. 14, pp. 478–505, 2009.
[32] F. A. Aloul, K. A. Sakallah, and I. L. Markov, "Efficient symmetry breaking for boolean satisfiability," *IEEE Transactions on Computers*, vol. 55, no. 5, pp. 549–558, 2006.
[33] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, "The probabilistic model checker storm," *International Journal on Software Tools for Technology Transfer*, pp. 1–22, 2021.
[34] J. Reed and B. C. Pierce, "Distance makes the types grow stronger: a calculus for differential privacy," in *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, 2010, pp. 157–168.
[35] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce, "Linear dependent types for differential privacy," in *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2013, pp. 357–370.
[36] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub, "Higher-order approximate relational refinement types for mechanism design and differential privacy," *ACM SIGPLAN Notices*, vol. 50, no. 1, pp. 55–68, 2015.
[37] D. Zhang and D. Kifer, "Lightdp: Towards automating differential privacy proofs," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 2017, pp. 888–901.
[38] J. P. Near, D. Darais, C. Abuah, T. Stevens, P. Gaddamadugu, L. Wang, N. Somani, M. Zhang, N. Sharma, A. Shan *et al.*, "Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–30, 2019.
[39] M. Fredrikson and S. Jha, "Satisfiability modulo counting: A new approach for analyzing privacy properties," in *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2014, pp. 1–10.
[40] A. Albarghouthi and J. Hsu, "Synthesizing coupling proofs of differential privacy," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, pp. 1–30, 2017.
[41] G. Barthe, G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P.-Y. Strub, "Differentially private bayesian programming," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 68–79.
[42] C. Smith and A. Albarghouthi, "Synthesizing differentially private programs," *Proceedings of the ACM on Programming Languages*, vol. 3, no. ICFP, pp. 1–29, 2019.
[43] Y. Wang, Z. Ding, Y. Xiao, D. Kifer, and D. Zhang, "Dpgen: Automated program synthesis for differential privacy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 393–411.
[44] M. Gaboardi, K. Nissim, and D. Purser, "The complexity of verifying loop-free programs as differentially private," *arXiv preprint arXiv:1911.03272*, 2019.
[45] M. Bun, M. Gaboardi, and L. Glinskih, "The complexity of verifying boolean programs as differentially private," in *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*. IEEE, 2022, pp. 396–411.
[46] S. Zanella-Beguelin, L. Wutschitz, S. Tople, A. Salem, V. Rühle, A. Paverd, M. Naseri, B. Köpf, and D. Jones, "Bayesian estimation of differential privacy," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 40 624–40 636. [Online]. Available: https://proceedings.mlr.press/v202/zanella-beguelin23a.html
[47] G. Barthe, R. Chadha, P. Krogmeier, A. P. Sistla, and M. Viswanathan, "Deciding accuracy of differential privacy schemes," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–30, 2021.
[48] E. Lobo-Vesga, A. Russo, and M. Gaboardi, "A programming framework for differential privacy with accuracy concentration bounds," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 411–428.
[49] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*. Springer, 2019, pp. 375–403.
[50] V. Balcer and S. Vadhan, "Differential privacy on finite computers," *arXiv preprint arXiv:1709.05396*, 2017.

APPENDIX A
TRUNCATED GEOMETRIC ABOVE THRESHOLD

The above threshold algorithm takes as input an array of values and a thresholding value T and outputs the index of the first value in the input array that exceeds T. The DP version of this algorithm adds noise both to each entry and to the threshold T [27].

Algorithm 5 Truncated Geometric Above Threshold

Input: Array of input integers X, threshold value T, randomness parameters λ_1 and λ_2
Output: Index i

$\hat{T} \leftarrow T + Geom(\lambda_1)$
for $x_i \in X$ **do**
  $r_i \leftarrow Geom(\lambda_2)$
  if $x_i + r_i \geq \hat{T}$ **then**
    return i
  end if
end for
return y

We use a discrete, finite variation of this DP algorithm which is defined over integer-valued input arrays, integer-valued T, and uses the truncated geometric mechanism for the added noise [50]. We provide the pseudocode for this variation in Algorithm 5, and an example of the Dice program generated by our ExactDP tool in Fig. 5. In our experiments, we set randomness parameters $\lambda_1 = \lambda_2$ for simplicity.

```
let t_noisy = discrete(0.006692850924284843,0.9866142981514303,0.006692850924284843) in
let x0_noisy = discrete(4.5096074800597685e-05,0.006647754849484246,0.9933071490757152) in
let x1_noisy = discrete(0.9933071490757152,0.006647754849484246,4.5096074800597685e-05) in
let x2_noisy = discrete(0.006692850924284843,0.9866142981514303,0.006692850924284843) in
let x3_noisy = discrete(4.5096074800597685e-05,0.006647754849484246,0.9933071490757152) in
let answ = if x0_noisy > t_noisy then int(3,0) else int(3,4) in
let answ = if answ == int(3,4) && x0_noisy > t_noisy then int(3,0) else answ in
let answ = if answ == int(3,4) && x1_noisy > t_noisy then int(3,1) else answ in
let answ = if answ == int(3,4) && x2_noisy > t_noisy then int(3,2) else answ in
let answ = if answ == int(3,4) && x3_noisy > t_noisy then int(3,3) else answ in
```

Fig. 5: Sample Dice program generated from ExactDP implementation for above threshold for lists of length 4 and maximum integer value 2 with $X = [2, 0, 1, 2]$ and $T = 1$.