The Complexity of Verifying Boolean Programs as Differentially Private

Mark Bun, Marco Gaboardi, and Ludmila Glinskih
Boston University, MA, USA

Abstract

We study the complexity of the problem of verifying differential privacy for while-like programs working over boolean values and making probabilistic choices. Programs in this class can be interpreted into finite-state discrete-time Markov Chains (DTMC). We show that the problem of deciding whether a program is differentially private for specific values of the privacy parameters is **PSPACE**-complete. To show that this problem is in **PSPACE**, we adapt classical results about computing hitting probabilities for DTMC. To show **PSPACE**-hardness we use a reduction from the problem of checking whether a program almost surely terminates or not. We also show that the problem of approximating the privacy parameters that a program provides is **PSPACE**-hard. Moreover, we investigate the complexity of similar problems also for several relaxations of differential privacy: Rényi differential privacy, concentrated differential privacy, and truncated concentrated differential privacy. For these notions, we consider gap-versions of the problem of deciding whether a program is private or not and we show that all of them are PSPACE-complete.

1 Introduction

Differential privacy [20] provides a formal framework for guaranteeing that programs respect the privacy of the individuals contributing their data as input. The idea at the heart of differential privacy is to use carefully calibrated random noise to guarantee that an individual's data has a limited influence on the result of a data analysis. The literature on differential privacy shows how this can be done for numerous tasks across statistics, optimization, machine learning, and more. However, showing that a program satisfies differential privacy can be difficult, subtle, and error prone [40, 37]. For this reason, several techniques have been proposed in order to verify or find violations in differential privacy programs, e.g. [45, 8, 25, 53, 18, 9, 4].

Despite tremendous progress in the development of methods and tools to support the deployment of differential privacy, there are fundamental open questions about the complexity of the problems these tools address. In this paper, we focus on one of these problems:

Approximate-DP: Given a Boolean program and parameters e^ε, δ, decide whether a program is (ε, δ)-differentially private or not.

Barthe et al. [4] showed that a version of this problem, for probabilistic while-like programs using both finite and infinite data, is undecidable. However, it becomes decidable when a restriction is imposed on the way infinite data are used in while loops. Gaboardi et al. [26] showed that, for probabilistic programs over finite data domains and without loops, when the parameters are rational, this problem is $\mathbf{coNP^{\#P}}$-complete for $(\varepsilon, 0)$-differential privacy and even harder for (ε, δ)-differential privacy. In this work we consider the case where programs can contain loops and work over finite data, and the parameters are given as dyadic numbers (rational numbers whose denominator is a power of two). We show that adding loops and maintaining the restriction on finite data preserves decidability but significantly increases the complexity of the problem, even for just $(\varepsilon, 0)$-differential privacy.

Our contributions

We consider programs from a simple probabilistic while-like programming language over boolean data, where randomness is represented as probabilistic choice. We call this language BPWhile. This language can be seen as a low-level target language for differential privacy implementations which are intrinsically over finite data types [41, 27, 3, 33].

As a first step, we show **PSPACE**-hardness for **Approximate-DP** over this language, with respect to the size of the program. We show this result by using a reduction from the problem of deciding *almost sure termination* for programs in BPWhile. Programs in this language can be seen as discrete-time recursive Markov chains for which almost sure termination has been shown **PSPACE**-complete [22]. Intuitively, the hardness of verifying whether a program is differentially private comes from the fact that we need to compare distributions on outputs for neighboring pairs of inputs. Understanding such distributions essentially gives us a way to check whether a program terminates with probability 1 or not. We use this idea in all **PSPACE**-hardness proofs in this work.

We then present an algorithm for **Approximate-DP** which uses polynomial space, completing our proof of **PSPACE**-completeness for **Approximate-DP**. Our algorithm is based on classical results showing that computing hitting probabilities in discrete-time Markov chains can be done in a space efficient way. Our proof of **PSPACE**-completeness even holds in the case where the privacy parameter δ is zero—this setting is usually called *pure differential privacy*.

Similarly to [26], we also consider a related problem concerning the approximation of privacy parameters. In particular, we study the following gap-promise variant of the problem.

Distinguish (ε, δ)**-DP**: Given a program that is promised to either be $(0, 0)$-differentially private or not (ε, δ)-differentially private, decide which is the case. Here, ε, δ may be fixed constants independent of the input.

2

We show that this problem is also **PSPACE**-hard via another reduction from the problem of deciding almost sure termination. At first glance, the statement seems specific to $(0,0)$-differentially privacy, but it implies more generally that it is hard to distinguish between (ε, δ)-differentially private programs and programs which fail to be $(\varepsilon + \alpha, \delta + \beta)$-differentially private for positive constants α, β. In particular, it is hard even to approximate the best ε and δ parameters for which a program guarantees differential privacy.

Further, we consider several relaxations of the definition of differential privacy which have recently appeared in the literature. Specifically, we consider deciding Rényi-differential privacy (RDP) [42], concentrated differential privacy (CDP) [12], and truncated concentrated differential privacy (tCDP) [11]. For each of these privacy notions we define a gap version of the problem of deciding whether a program is private or not and we show that all of them are **PSPACE**-complete.

To show membership in **PSPACE** we use similar approach as in the **PSPACE**-algorithm for **Decide** (ε, δ)-DP. The main difference is that definitions of RDP, CDP, and tCDP involve computations of Rényi divergences and, as we are working with probabilities that can have exponentially long descriptions, we carefully apply known uniform families of polylogarithmic depth circuits to perform these calculations. We prove our lower bounds using reductions from **Distinguish** (ε, δ)**-DP**.

To summarize, our contributions are:

1. We give a proof of **PSPACE**-hardness for the problem of deciding (ε, δ)-differential privacy (by showing a polynomial time reduction from the language of almost surely terminating programs, Section 5.4).

2. We show a **PSPACE** algorithm for deciding (ε, δ)-differential privacy (Section 5.3).

3. We show **PSPACE**-hardness for the problem of approximating the privacy parameters (Section 6).

4. We show **PSPACE** algorithms for deciding Rényi-differential privacy (Section 7.1), concentrated differential privacy (Section 7.2), and truncated differential privacy (Section 7.3).

5. We also give a proof of **PSPACE**-hardness for deciding Rényi-differential privacy (Theorem 7.3), concentrated differential privacy (Theorem 7.9), and truncated concentrated differential privacy (Theorem 7.13) (via reductions from the problem of approximating privacy parameters, Section 6).

2   Related work

Verification tools for differential privacy. Several tools have been developed with the goal of supporting programmers in their effort to write code that is guaranteed to be

differentially private, including type systems [45, 25, 6, 53, 44], program logics [8, 5, 7], and other program analyzers [49, 24, 2, 39, 15]. Other tools help programmers find violations in differentially private implementations [18, 9, 54]. Finally, several recent tools address both problems at the same time [51, 4, 23]. Most of these tools are capable of analyzing complex examples corresponding to the state of the art in differential privacy algorithm design [19, 36].

Implementations on finite computers.   Several works have studied how to implement differentially private algorithms using finite arithmetics. Mironov [41] showed that naïve implementations of the Laplace distribution using floating point numbers are actually not private. Gazeau et al. [27] showed that similar problems as the one identified by Mironov are not only due to the non-uniformity of floating points but they are actually intrinsically due to the use of finite precision arithmetic. Ilvento [33] showed that similar considerations can be applied also to algorithms that are in principle discrete, such as the exponential mechanism. Balcer and Vadhan [3] showed how to implement several important differentially private algorithms in an efficient way on finite precision machines.

Related results in complexity.   Murtagh and Vadhan [43] studied the complexity of finding the best privacy parameters for the composition of multiple differentially private mechanisms and showed it to be #**P**-complete. This work, in part, led to the development of several variants of differential privacy, most of which we consider here, with better composition properties. Barthe et al. [4] showed that deciding differential privacy for probabilistic while-like programs using both finite and infinite data is undecidable, but it becomes decidable when a restriction is imposed on the way infinite data are used in while loops. However, they do not study the computational complexity of this problem. Gaboardi et al. [26] showed **coNP**$^{\#\mathbf{P}}$-completeness for the problem of deciding $(\varepsilon, 0)$-differential privacy for probabilistic programs over finite data domains and without loops. They also studied this problem and approximate versions of it for (ε, δ)-differential privacy. Chadha et al. [14] recently showed that deciding differential privacy for a class of automata that can be used to describe classical examples from the differential privacy literature can be done in linear time in the size of the automata. This class of automata includes computations over unbounded input data, such as real numbers. Chistikov et al. [15, 16] studied several complexity problems concerning differential privacy in the setting of labeled Markov chains. They showed that the threshold problem for a computable bisimilarity distance giving a sound technique to reason about differential privacy is in **NP** [15]. Further, they proved that another distance, based on total variation, which can be used to more precisely reason about differential privacy is undecidable in general, and the problem of approximating it is #**P**-hard, and in **PSPACE** [16].

There are also other related results from the program verification and privacy literatures. Courcoubetis and Yannakakis [17] studied the complexity of several verification

4

problems for probabilistic programs. Etessami and Yannakakis [22] studied the complexity of several problems for recursive Markov chains. Notably, they showed that deciding almost sure termination for this computational model is **PSPACE**-complete. Kaminski et al. [35] studied the arithmetic complexity of almost sure termination for general probabilistic programs with unbounded data types. Chadha et al. [13] showed **PSPACE**-completeness for the problem of bounding quantitative information flow for boolean programs with loops and probabilistic choice. A bound on pure differential privacy entails a bound on quantitative information flow, but not the other way around, and hence their result does not directly apply in our context. Gilbert and McMillan [29] studied the query complexity of verifying differential privacy programs modeled as black boxes.

3   Preliminaries

3.1   Boolean Programs with Loops and Random Assignments

In this paper we consider a simple while-like language working over booleans, extended with probabilistic choice. This language, which we call BPWhile, can be seen as a probabilistic extension of the language for input/output bounded boolean programs studied in [30]. The syntax of the language is defined by the following grammar.

$$
\begin{array}{rcl}
b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{random} \mid x \mid b \wedge b \mid b \vee b \mid !b \\
c & ::= & \texttt{skip} \mid x := b \mid c; c \mid \texttt{if } b \texttt{ then } c \texttt{ else } c \mid \texttt{while } b \texttt{ then } c \\
C & ::= & \texttt{input}(x, \ldots, x); c; \texttt{return}(x, \ldots, x)
\end{array}
$$

All of the constructs are standard. The expression `random` represents a random fair coin, which with probability $1/2$ evaluates to true and with probability $1/2$ evaluates to false. The semantics for BPWhile programs is also standard and we omit it here. However, notice that program may fail to terminate, and we also have to consider this when analyzing probabilities. To mark non-termination we will use the symbol $\perp$. We also remark that a given BPWhile program operates only on boolean inputs of a single fixed length n, specified (implicitly) in the program description.

Our language is very similar to the one studied in [26]. The main difference is that we have an additional loop construction **while** b **then** c. Without loops, programs in this language can be interpreted into boolean circuits of roughly the same size. However, this cannot be done in presence of loops, as the straightforward approach of unfolding loops gives a circuit of size exponential in the program length. To avoid analyzing boolean circuits of exponential size, we will instead analyze programs as discrete-time Markov chains, in a manner similar to [4]. This is possible because BPWhile programs use a bounded amount of memory (that is at most linear in the size of the input program), corresponding to an exponential, in the size of the input, number of states in the resulting Markov chain. The precise translation will be given in Theorem 5.3.

5

Similarly to [26] we measure the complexity of the problems we are interested in as functions of the size of the input program, rather than, e.g., the number of bits the input program itself takes as input.

Language expressivity. We use booleans as our basic data type to keep our proofs simple. However, all of the results we show also hold for programs where values are from a fixed finite domain. In fact, the language we use here can be thought as a low-level language which could be the target of implementations of differential privacy primitives. As shown in several works, one has to be very careful when implementing differentially private primitives [41, 27, 3, 33]. One way to guarantee correctness for this process could be to give a translation into BPWhile and then decide whether the given program is differentially private or not. We illustrate how this process could work with an example.

Using $1 + n + m$ boolean values we can represent arbitrary positive and negative fixed-point numbers with range $(-2^n + 1, 2^n - 1)$ and precision 2^{-m}, and perform standard arithmetic operations and comparison over them. We can then think about working with blocks of variables of size $1 + n + m$, which we denote using vector notation, e.g. $\vec{x}, \vec{y}, \ldots$. Notice that using this representation we can also easily encode a uniform sampling operation for elements in a range (v,w], which we denote $\texttt{uniform}(v, w)$. We can, for example, implement the bounded Geometric Mechanism from [28], using this encoding and the implementation in finite precision arithmetic provided in [3]. Given a positive integer n and a private positive integer value $c \leq n$, this discrete mechanism selects an integer element z from the range $[0, n]$ with probability proportional to $e^{\frac{-\varepsilon|z-c|}{2}}$. Essentially, the mechanism implements inverse transform sampling based on the inverse CDF of the output distribution. Given c, n and ε, this mechanism can be described in BPWhile as in Figure 1.

All the operations in this piece of code are assumed to work on blocks of variables and of booleans that are long enough to avoid overflow and approximations. This algorithm samples from a uniform distribution (line 3) for a value of $\vec{d}$ large enough and uses a while loop to go through the integers in the range $[0, n]$ to find the right element to return. A faster implementation could be based on binary search. The nested conditionals (lines 7-14) implement the checks required for the inverse transform sampling to identify the right element to return.

We gave this example to show that the language is expressive enough to implement a real-world mechanism. However, we also chose this example because identifying the privacy guarantee provided by this algorithm is non-trivial. Balcer and Vadhan [3] showed this algorithm to be $(\tilde{\varepsilon}, 0)$-differentially private when $\tilde{\varepsilon} = \ln(1 + 2^{-\lceil \log(2/\varepsilon) \rceil})$ and $\tilde{\varepsilon} \in (2/9\varepsilon, \varepsilon/2]$, where the complexity in the expression for $\tilde{\varepsilon}$ comes from the implementation. This example shows why several works have designed methods to decide differential privacy, and why it is important to understand the complexity of this problem.

```
0.    input($\vec{c}, \varepsilon$);
1.    $\vec{k} := \lceil \log(2/\varepsilon) \rceil$;
2.    $\vec{d} := (2^{\vec{k}+1} + 1)(2^{\vec{k}} + 1)^{n-1}$;
3.    $\vec{u} := $ uniform$(0, \vec{d}]$;
4.    $\vec{z} := 0$;
5.    $\vec{r} := n$;
6.    while $\vec{z} < \vec{n} \wedge \vec{r} = n$ then
7.      if $\vec{z} < \vec{c}$ then
8.        if $\vec{u} \leq 2^{\vec{k}(\vec{c}-\vec{z})}(2^{\vec{k}} + 1)^{n-(\vec{c}-\vec{z})}$
9.        then $\vec{r} := \vec{z}$
10.       else skip
11.     else
12.       if $\vec{u} \leq d - 2^{\vec{k}(\vec{z}-\vec{c}+1)}(2^{\vec{k}} + 1)^{n-1-(\vec{z}-\vec{c})}$
13.       then $\vec{r} := \vec{z}$
14.       else skip
15.     $\vec{z} = \vec{z} + 1$;
16.   return($\vec{z}$);
```

Figure 1: Example: Bounded Geometric Mechanism in finite precision arithmetic

3.2 Almost Sure Termination and Configuration Graph

Our approach will rely on the hardness of the problem of deciding *almost sure termination* for probabilistic boolean programs (Lemma 4.3). Almost sure termination is a natural probabilistic extension of the concept of termination.

Definition 3.1. *A program C* almost surely terminates *if on all inputs it terminates with probability 1.*

Deciding almost sure termination for general probabilistic programs on unbounded data types is known to be Π_2^0-complete [35] while for programs representing recursive Markov chains it is known to be **PSPACE**-complete [22].

In the following, it will be convenient to analyze BPWhile programs using their configuration graph. To do this, we assume that the code of a program comes with lines of code associated to each command, in a way similar to the code in Figure 1.

Definition 3.2. *Consider a BPWhile program C with l lines of code and v Boolean variables. A state s of C is a pair (m, i) where $m \in \{0, 1\}^v$ represents a potential value of the memory, i.e. values for all the variables, and $i \in [l]$ is a line of code. The* size *of the program is the number of symbols in the description of the program.*

Note that as the description of each variable, input value and line in the program requires at least one symbol, we get that l, v, and the size of the input of the program are

7

always at most the size of the program. Throughout this paper we measure complexity of the verifying procedures based on the size of the program.

Definition 3.3. *The configuration graph $G = (V, E)$ of a BPWhile program C on input x has a vertex for every possible state of the program and a directed edge $((m, i), (m', i')) \in E$ if the probability of getting the memory m' starting from the memory m and executing the command at line i' is strictly greater than 0.*

We will also sometimes use the term *state graph* to refer to the configuration graph. The starting state of a program is the state at the beginning of C's execution on x, where the input variables are set to x and the index of the execution line is 0. A final state is any state following the execution of the last line of code (the final return command). We denote the set of final states in a configuration graph by $V_f \subseteq V$.

3.3 Markov Chains

To analyze the probability that a boolean program C on input x outputs a specific value, we need to associate probabilities to each transition in the configuration graph. By doing this, we turn a configuration graph into a discrete time Markov chain.

Definition 3.4 ([31, 38, 50])**.** *A discrete-time Markov chain $M = (V, E, \{p_{uw} \mid (u, w) \in E\}, \{p_0(v) | v \in V\})$ consists of a set of states V, a set $E \subseteq V \times V$ of transitions between states, a list p_{uw} of positive probabilities for all transitions (u, w) such that for each state $u \in V$, we have $\sum_{w \in V} p_{uw} = 1$, and an initial probability distribution p_0 on states in V.*

Following [17] we view a Markov chain as a directed graph (V, E), with weights p_{uw} on all edges (u, w). Moreover, as in a configuration graph, we associate each vertex in the graph to a state of a BPWhile program, and a transition between states to one possible execution step of the program. As an initial probability distribution we use a unit distribution that places weight 1 on the unique start state of the program.

To verify whether a BPWhile program is differentially private, as we will see in the next section, we need to compare the probabilities of outputting the same output on neighboring inputs. We will do this by computing hitting probabilities of final states with a fixed output values.

Definition 3.5. *The hitting probability of a state $s \in V$ in a Markov chain $M = (V, E, p, p_0)$ is the probability of reaching s in M starting with a initial probability distribution p_0 after an arbitrary number of steps.*

3.4 Differential Privacy

Differential privacy is a property of a program that can be expressed in terms of a neighboring relation over possible program inputs. Here we view an input as a sensitive

dataset, and say that two inputs are neighboring if they differ in one individual's information. As our focus in this paper is on boolean programs, we define two datasets to be neighboring when they differ in a single bit.

Definition 3.6. *Two boolean vectors of the same length are said to be* neighboring *if their Hamming distance (the number of positions in which these vectors differ) equals* 1.

Notice that this is a strong notion of neighboring, which makes our hardness results stronger. That is, our hardness results extend naturally to other more involved notions of neighboring. Moreover, our upper bound arguments apply to any neighboring relation between boolean vectors (or more generally, vectors over any fixed finite data domain) as long as that relation can be checked in polynomial space. Using the notion of neighboring we introduced above we can now formulate differential privacy.

Differential privacy guarantees that a change of any one data in the input will not change much the observed output of the program. More formally, differential privacy guarantees that the distributions of outputs of a program when run on neighboring datasets are close.

Definition 3.7 (Differential Privacy [20])**.** *A boolean program C with inputs of length n and producing outputs of length l is (ε, δ)-differentially private if for every pair of neighboring inputs $x, x' \in \{0,1\}^n$ and for every set of possible outputs $O \subseteq \{0,1\}^l \cup \{\bot\}$:*

$$\Pr[C(x) \in O] \leq e^{\varepsilon} \cdot \Pr[C(x') \in O] + \delta. \tag{1}$$

This version of differential privacy is often called *approximate differential privacy* to distinguish it from *pure* differential privacy, which is the special case where $\delta = 0$. We will denote the latter by ε-differential privacy.

In the following, it will be convenient at times to work with the following reformulation of differential privacy.

Lemma 3.1 (Pointwise differential privacy [7])**.** *A program C is (ε, δ)-differentially private if and only if for all neighboring inputs $x, x' \in \{0,1\}^n$,*

$$\sum_{o \in \{0,1\}^l \cup \{\bot\}} \delta_{x,x'}(o) \leq \delta,$$

where $\delta_{x,x'}(o) = \max(\Pr[C(x) = o] - e^{\varepsilon}(\Pr[C(x') = o]), 0)$.

4 Complexity of Checking Almost Sure Termination

In this section we give intuition for the hardness of deciding differential privacy by discussing the complexity of almost sure termination. While it is known that almost sure termination for Markov chains is **PSPACE**-complete [22], we believe it is instructive to understand where this complexity comes from. We start with a helpful characterization of almost sure termination of a BPWhile program in terms of reachability in the program's configuration graph.

Theorem 4.1. *A program C terminates almost surely if and only if for every input x and every vertex v in the configuration graph of $C(x)$ that is reachable from the start state, there is a path from v to one of the final states.*

Proof. For the "if" direction, suppose x is an input such that for every reachable vertex in the state graph $G = (V, E)$ of $C(x)$, there is a path from v to one of the final states. Let $m = |V|$ be the number of vertices. Since every simple path in G has at most m edges, we have that for every v, the probability of reaching a final state after at most m additional steps of computation starting from v is least 2^{-m}. Therefore, for any $k \geq 1$, the probability that the program fails to terminate on input x after km steps is at most $(1 - 2^{-m})^k$. Taking $k \to \infty$, we see that that the program fails to terminate with probability 0. Therefore, C terminates almost surely on input x.

For the "only if" direction, suppose there is an input x and a vertex v in the state graph of $C(x)$ that is reachable from the start state but cannot reach any final state. Then on $C(x)$ reaches state v with probability at least 2^{-m} by following the simple path from the start state to v. Once the program has reached v, it is impossible to terminate. So the program terminates with probability at most $1 - 2^{-m} < 1$. □

The main intuition of this theorem is that the only way for a program to fail to terminate with probability 1 is if there is some positive probability that it enters an infinite loop from which it cannot exit. This is possible if and only if there exists a state that is reachable from the start state, but from which we cannot reach any of the final states.

Theorem 4.1 immediately suggests a simple exponential-time (and exponential-space) algorithm for checking almost sure termination. For each possible input to the program, we can construct the configuration graph of the program on that input. Using breadth-first search, we can mark which states are reachable from the start state, and for each such state we check whether any of the final states are reachable. If there exists an input and a state in its configuration graph that is reachable from the start state but cannot reach a final state, then by Theorem 4.1 we get that the program doesn't terminate almost surely. If for every input, there is no such state, then the program almost surely terminates.

Constructing the configuration graph explicitly and running breadth-first search uses exponential space. In what follows, we describe how to reduce the space complexity to polynomial.

We can improve the previous algorithm by avoiding storing the whole configuration graph, and instead providing implicit access to any edge in the graph.

This requires us to re-compute on-the-fly information about the set of reachable states from any given vertex, but fortunately, this can still be done in polynomial space.

Theorem 4.2. *There is a deterministic algorithm for checking almost sure termination of a BPWhile program using space polynomial in the size of the program.*

To show **PSPACE**-hardness of checking almost sure termination we reduce from the **PSPACE**-complete true quantified boolean formula (TQBF) problem. This is the prob-

lem of deciding whether a fully quantified propositional boolean formula is true. For a formula ϕ with t quantifiers, we define a BPWhile program with t nested while-loops to evaluate the formula. The reduction is similar to the reduction in [30] from TQBF to the reachability problem for extended hierarchical state machines. We prove the following theorem in Appendix B.

Theorem 4.3. *The problem of checking whether a BPWhile terminates almost surely is* **PSPACE**-*hard.*

5 PSPACE-Completeness for Pure and Approximate Differential Privacy

We reason about differential privacy in a manner similar to almost sure termination. In particular, we use a Markov chain interpretation of a program C.

We first give a formal definition of the verification problems we consider. Then we give an inefficient (exponential-time) but simple algorithm (Section 5.1), followed by a **PSPACE**-algorithm for verifying whether a program is pure (Section 5.2) or approximate differentially private (Section 5.3).

Definition 5.1. *In the* PURE-DP *problem, an instance (C, e^ε) consists of a BPWhile program C, and a dyadic rational number e^ε. The problem is to distinguish whether for all neighboring inputs x, x' and for every set of possible outputs O we have*

$$\Pr[C(x) \in O] \leq e^\varepsilon \cdot \Pr[C(x) \in O].$$

Definition 5.2. *In the* APPROXIMATE-DP *problem, an instance $(C, e^\varepsilon, \delta)$ consists of a BPWhile program C, and two dyadic rational numbers e^ε, δ. The problem is to distinguish whether for all neighboring inputs x, x' and for every set of possible outputs O we have*

$$\Pr[C(x) \in O] \leq e^\varepsilon \cdot \Pr[C(x) \in O] + \delta.$$

5.1 Exponential-Time Algorithm for Checking $(\varepsilon, 0)$-Differential Privacy

To give an exponential-time algorithm for checking $(\varepsilon, 0)$-differential privacy, we first review the algorithm for computing the probability of reaching any given final state s_f in a Markov chain from [4]:

1. For each state v, initialize a variable q_v representing the probability of reaching s_f from this state.

2. Set $q_{s_f} = 1$ for the final state s_f.

3. For each state v from which s_f is not reachable set $q_v = 0$.

4. For any state v for which we do not yet have an equation, we introduce the equation $q_v = \sum_{u \in V} q_u \cdot p_{vu}$, where p_{vu} is the probability of transitioning from v to u in one step. If there is no transition from v to u, then $p_{vu} = 0$.

5. The previous steps give us a set of equations, one for each possible state of the Markov chain. The number of variables equals the number of equations. This linear system can be solved unambiguously by using any polynomial-time algorithm for solving systems of linear equations.

We can now state our exponential time algorithm for deciding differential privacy for BP-While programs.

Theorem 5.1. Pure-DP *problem is solvable by a deterministic algorithm using time exponential in the size of a program.*

Proof. By Lemma 3.1 for checking $(\varepsilon, 0)$-Differential Privacy for a program C it is sufficient to compare for every pair of neighboring dataset the output distributions on every possible value. Using this approach, we get the following simple algorithm:

1. For neighboring inputs x, x' of size n and a program C of size N construct two Markov chains, with one start-state in each, set these start-states to x and x', respectively.

2. Find the probabilities of each final state in each Markov chain.

3. Compare the probability of the same states in the two Markov chains. If there is at least one output c such that $P[C(x) = c] > e^\varepsilon P[C(x') = c]$, terminate and output "Not $(\varepsilon, 0)$-DP". Otherwise continue.

4. If the checks were successful for all pairs, terminate with an output: "$(\varepsilon, 0)$-DP".

This algorithm explicitly store probabilities of reaching all of up to 2^N final states, as well as a system of linear equations of size exponential in N. As the input of the algorithm is a program C of size N, we get that this algorithm requires exponential space and time in the size of its input. $\square$

5.2 PSPACE Algorithm for Checking $(\varepsilon, 0)$-Differential Privacy

A classic line of work [48, 10, 34] showed that computing the hitting probabilities of final states can be done efficiently in space. This is what we need to design a **PSPACE** algorithm to check differential privacy. In designing this algorithm we use the work of Simon [48] who showed that given a Markov chain of size M, the hitting probability of any state can be computed in space $O((\log M)^6)$. Subsequent work [10, 34] improved this result by showing that $O((\log M)^2)$ is enough. Nevertheless, we focus our exposition on Simon's algorithm as its presentation simplifies the description of our algorithm, and improving the polynomial does not affect membership of our problem in **PSPACE**.

Simon's result can be formally stated as follow:

Lemma 5.2. *[48] Let M be a Markov chain (represented by its transition matrix) with at most 2^L states, an initial distribution placing all mass on one state s, a set of final states F each with only one self-transition, and the property that every state not in F each outgoing transition probability is either 0 or $1/2$. There is an $O(L^6)$-space deterministic algorithm that computes the hitting probabilities of every state in F.*

To apply the algorithm from the previous lemma we need to do an extra pre-processing step to remove all non-final recurrent states of a Markov chain.

Definition 5.3. *A recurrent state in a Markov chain is a state such that, after reaching it once, the probability of reaching it again is 1.*

A similar pre-processing step appears in Simon's paper, and we describe our removal process below in our proof of Theorem 5.3.

Now we are ready to show that $(\varepsilon, 0)$-differential privacy for BPWhile programs can be decided in polynomial space.

Theorem 5.3. *The PURE-DP problem is solvable by a deterministic algorithm using space polynomial in the size of the program.*

Proof. To apply the algorithm from Lemma 5.2 and conclude that polynomial space is sufficient in order to compute the final probabilities, we need to be able to compute the probability of each transition in the Markov chain using polynomial space. We cannot explicitly store the whole Markov chain using space that is polynomial in the size of a program. Instead, we can construct an algorithm working in polynomial space which gets as input a description of the BPWhile program C, the program input x, and two states u, v of the Markov chain corresponding to $C(x)$. It outputs the transition probability of edge (u, v) (the probability that $C(x)$ gets from state u to state v in one step).

We need to find the probability of hitting each reachable final state of the Markov chain of $C(x)$. Note that these probabilities can be as small as $1/2^{2^{p(N)}}$ for some polynomial $p(N)$, where N is the size of the input program. This is because a Markov chain for a program of size N has a number of states which is at most exponential in N, and as each transition probability is either 0, or 1/2, or 1, there can be a simple path in the Markov chain from the start state to the final state that goes through all the states with probability $1/2^{2^{p(N)}}$.

Storing these values requires exponential space, so the **PSPACE** algorithm described further only provides implicit access to these probabilities, i.e., the ability to compute any desired bit of a probability.

Here are the conditions that the Markov chain we construct needs to satisfy in order to apply Lemma 5.2:

- The transition probability between every two states in the Markov chain of size $O(2^{\text{poly}(N)})$ should be computable in polynomial space. Every final state has a self-transition with probability 1.

- Each transition in the Markov chain for all non-final states has to have weight either $1/2$ or 0, and the graph underlying the Markov chain shouldn't contain multiple edges. This can be done by duplicating every state, except the start state, increasing the number of vertices by a factor of 2. Every duplicate final state is also marked as a final state. Let a and b be vertices in the original Markov chain of the program that are transformed to two pairs of vertices a_1, a_2 and b_1, b_2 respectively. Then we re-assign the weight of edge e from a to b in the original Markov chain as follows:

    - If the original weight of e is $1/2$, then we add two edges $(a_1, b_1), (a_2, b_2)$ each of weight $1/2$ to the new Markov chain.
    - If the weight of e is 1 we add four edges $(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_2, b_1)$ each of weight $1/2$.
    - If the weight of e is 0, we do not add any edges between vertices a_1, a_2 and b_1, b_2.

    Therefore, for each original edge we add at most 4 new edges, so we do not increase the size of the Markov chain by more than a factor of 4. Moreover, for every pair of vertices in the new Markov chain, we can recompute the weight of the edge based on the the weight of the edge in the original Markov chain in linear time. Overall, this transformation is computable in the space polynomial in the size of the input BPWhile program and it guarantees that the probability of getting from one vertex to any other in one step is either 0 or $1/2$.

- All recurrent states except the final states should be deleted. We simulate this deletion as follows. Whenever our algorithm reads the probability on an edge (u, v), we check whether either u or v are recurrent and zero out this probability if so. This check is similar to the one that we discussed earlier for almost sure termination. We consider the graph underlying the Markov chain of the program. To check whether state u is recurrent, we run a search algorithm checking whether there is at least one path through edges with non-zero weight to at least one of the final states. We can use Savitch's algorithm [47] to do this check in space polynomial in the size of the program.

To verify whether a program C is ε-differentially private we can now enumerate all pairs x, x' of neighboring inputs, and all possible outcomes o. For each outcome o, we compute $\Pr[C(x) = o]$ by summing the hitting probabilities of reaching final states in the configuration graph of C on x that result in outputting o. Finally, we compare $\Pr[C(x) = o]$ to $e^\varepsilon \Pr[C(x') = o]$. Note that if e^ε is a rational number with numerator a and denominator b, then we can avoid division by comparing $b \cdot \Pr[C(x) = o]$ to $a \cdot \Pr[C(x') = o]$.

We remark that the necessary arithmetic operations on exponentially long (implicitly represented) numbers can be carried out in polynomial space (though exponential time

is still required) using classic logspace algorithms for addition and multiplication.[1]   In particular, this works even if e^ε is an exponentially long rational number provided as input to the problem. $\square$

5.3   PSPACE Algorithm for Checking (ε, δ)-Differential Privacy

Now, using the pointwise definition of differential privacy from Lemma 3.1 and using similar ideas to the algorithm in Section 5.2 we can construct a **PSPACE**-algorithm for checking (ε, δ)-differential privacy of BPWhile programs.

Theorem 5.4. APPROXIMATE-DP *is solvable by a deterministic algorithm using space polynomial in the size of the input program.*

Proof. Let $e^\varepsilon = a/b$ for natural numbers a, b. As in the algorithm in Section 5.2 we iterate through all pairs of neighboring inputs (x, x'), and for each of them compute

$$b\delta_{x,x'}(o) = \max(b\Pr[A(x) = o] - a\Pr[A(x') = o], 0),$$

using the algorithm from Theorem 5.2. Then we add this value to the sum

$$\sum_{o \in \{0,1\}^l \cup \{\bot\}} b\delta_{x,x'}(o),$$

until we have iterated over all possible inputs, or until the partial sum is greater than $b\delta$. In the former case we terminate with the output "not DP", otherwise we do not terminate until checking the last output, and output "DP".

Again, the necessary arithmetic computations (maximum, addition, subtraction, and multiplication) on exponentially long rational numbers can be done in polynomial space. $\square$

5.4   PSPACE-Hardness

To show **PSPACE**-hardness of checking whether a BPWhile program is differentially private, we reduce from the problem of checking almost sure termination. All of our hardness results have a similar structure: for a program C we construct another program C' that is differentially private (with some parameters) if and only if program C terminates almost surely. We show such reductions for the problems of PURE-DP, APPROXIMATE-DP, and for DISTINGUISH (ε, δ)-DP that hold even when the parameters e^ε and δ are fixed.

Lemma 5.5. *For a fixed rational $e^\varepsilon > 1$, the problem of checking almost sure termination for BPWhile Boolean programs is poly-time Karp-reducible to the problem of checking $(\varepsilon, 0)$-differential privacy for those programs.*

---

[1]We can construct uniform NC^1 and NC^2 circuits for these operations. Simple constructions are described in [52].

Proof. Let $C(x)$ be a BPWhile program for which we want to check almost sure termination. We construct a new program C' that will receive an input x and one additional bit of input b, and runs C as a subroutine. The BPWhile language doesn't support procedure calls, but we can encode the same behavior using the following code representing a template for the code of the program C'.

In this reduction we consider two inputs to a program C' as neighboring if they disagree only in one bit.

Here is the template code for C':

$$\texttt{input}(x, b); \texttt{if } b == 1 \texttt{ then } C(x) \texttt{ else skip}; \texttt{return}(1)$$

Notice that the return statement is executed only if $C(x)$ halts or $b == 0$.

As we add constant number of extra lines to the original program C, it takes linear time to construct C'. Hence the reduction takes linear time.

To show correctness of the reduction we need to check that it maps yes-instances of the almost sure termination problem to yes-instances of DECIDE $(\varepsilon,0)$-DP problem, and no-instances to no-instances. If a program C almost surely terminates on all inputs, then for all possible values of bit b we get that $C'(x, b)$ outputs 1 with probability 1. Hence this program is $(\varepsilon, 0)$-differentially private for every $\varepsilon \geq 0$.

If the program C is not almost surely terminating, then there exists an input x such that program $C(x)$ fails to terminate with some probability $\rho > 0$. Hence, we get that:

$$\Pr[C'(x, 1) \text{ doesn't halt}] = \rho > 0.$$

On the other hand, on the neighboring input $(x, 0)$ we get

$$\Pr[C'(x, 0) \text{ doesn't halt}] = 0.$$

Therefore C' is not ε-DP for any ε. $\qquad\square$

Lemma 5.6. *For any fixed rational e^{ε} and dyadic $\delta \in (0, 1)$, the problem of checking almost sure termination for BPWhile Boolean programs is poly-time Karp-reducible to the problem of checking (ε, δ)-differential privacy for those programs.*

Proof. As in the proof of Lemma 5.5, let $C(x)$ be a BPWhile program for which we want to check almost sure termination. We construct a new program C' that will receive an input x and one additional bit of input b, that runs C as a subroutine. We denote by $\texttt{delta\_rand}$ a subroutine that outputs 1 with probability $1 - \delta$, and outputs 0 with probability δ. For any dyadic rational constant $\delta = a/2^m$, this can be constructed using m calls to the the random operator. Note that the length of the program computing this subroutine is a constant independent of the length of the input program C. The following is a template for the code of C':

```
1.   input(x, b);
2.   if b == 1 then
3.     C(x);
4.     r = delta_rand();
5.     if r == 0 then
6.       while true then
7.          skip;
8.     else skip;
9.   else skip;
10.  return(1)
```
Notice that the while-loop in line 6-7 is potentially infinite. As delta_rand can be computed by a program of constant size, this reduction takes linear time as in the analysis of Lemma 5.5. Now to analyze the correctness of the reduction, first assume that C almost surely terminates. Then $C'(x, b)$ either outputs 1 with probability 1, or it outputs 1 with probability $1 - \delta$ and doesn't halt with probability δ. For every pair of input $(x, b), (x', b')$, the statistical distance between the possible distributions on outputs is at most δ. Hence $C'(x, b)$ is $(0, \delta)$-DP, hence (ε, δ)-DP.

If $C(x)$ doesn't almost surely terminate, then there exists some $\alpha > 0$ such that on some input x program $C(x)$ enters an infinite loop with probability α. Hence overall we get that $C'(x, 1)$ enters an infinite loop with probability at least $\delta(1 - \alpha) + \alpha > \delta$, but $C'(x, 0)$ terminates and outputs 1 with probability 1. Hence we get that

$$\Pr[C'(x, 1) \text{ doesn't halt}] > \delta = e^\varepsilon \Pr[C'(x, 0) \text{ doesn't halt}] + \delta,$$

and therefore $C'(x, b)$ is not (ε, δ)-DP. □

Combining the algorithms from Theorem 5.3 and Theorem 5.4 with the fact that the problems PURE-DP and APPROXIMATE-DP are **PSPACE**-hard even for fixed values of the privacy parameters, we conclude that corresponding verification problems are **PSPACE**-complete.

Corollary 5.7. *For any rational ε and dyadic $\delta \in (0, 1)$ the problems of checking whether a BPWhile program is ε-DP or whether a BPWhile program is (ε, δ)-DP are both* **PSPACE**-*complete.*

6 Hardness of Approximation of Privacy

In this section, we show a strong sense in which the privacy parameters of a BPWhile program are hard even to approximate. We do this by showing that for any constant parameters ε, δ, it is **PSPACE**-hard even to distinguish between the case where a program is $(0, 0)$-DP or whether it fails to be (ε, δ)-DP. This, for example, implies that the privacy parameters of a program are hard to approximate up to an additive $(\varepsilon/2, \delta/2)$.

17

Lemma 6.1. *For any rational constants $\varepsilon, \delta \in (0,1)$ the problem of checking almost sure termination for BPWhile programs is Karp-reducible to the promise problem of determining whether BPWhile program is $(0,0)$-differentially private or it is not (ε, δ)-differentially private.*

Proof. Our reduction consists of two parts:

1. Given a BPWhile program C, we construct a new BPWhile program C' such that if C almost surely terminates, then C' almost surely terminates too. Meanwhile, if C doesn't almost surely terminate, then C' terminates with probability at most $\frac{1}{2}$.

2. As in the reductions in the proofs of Lemma 5.5 and Lemma 5.6 we construct a program C'' that calls C' with the property that C'' is $(0,0)$-DP if C' is almost sure terminating, and C'' is not (ε, δ)-DP if C' halts with probability at most $\frac{1}{2}$.

For the first step, we use the following claim that we prove in Appendix C.

Claim 6.2. *If C is a BPWhile program, then we can construct in polynomial-time a new program C' that almost surely terminates if C almost surely terminates, and terminates with probability at most $1/2$ if C is not almost surely terminating.*

As in the reductions in Lemma 5.5 and Lemma 5.6, we now construct a new program C'' that receives an input x and one additional bit of input b, and runs C' as a subprogram. We repeat the execution of $C'(x)$ a total of m times, where $e^\varepsilon \cdot 2^{-m} + \delta < 1$. Note that m depends only on the privacy parameters ε, δ and not on the program C'. Now we construct the following program C''.

```
1.  input(x, b);
2.  if b == 1 then
3.    C'(x);          # run C'(x) m times
4.    ...
5.    C'(x);
6.  else skip;
7.  return(1)
```

The time complexity of constructing $C''(x, b)$ is linear in the size of C' as m is a constant. To show correctness, assume that C, and hence C' terminates almost surely. Then $C''(x, b)$ outputs 1 with probability 1 on all inputs. Hence $C''(x, b)$ is $(0,0)$-DP.

If $C(x)$ does not terminate almost surely then there exist an input x and some $\alpha > 1/2$ such that $C'(x)$ fails to halt with probability α. As we chose the number of repetitions m in such way that

$$e^\varepsilon \cdot \Pr[m \text{ sequential runs of } C'(x) \text{ halt}] + \delta < 1,$$

we get that $C''(x, b)$ is not differentially private on neighboring inputs $(x, 0)$ and $(x, 1)$, since

$$e^\varepsilon \cdot \Pr[C''(x, 1) \text{ halts}] + \delta < 1 = \Pr[C''(x, 0) \text{ halts}].$$

18

Therefore, if the original program $C(x)$ is not almost surely terminating, we transformed it via the intermediate program $C'(x)$ to a program $C''(x, b)$ that is not (ε, δ)-differentially private. $\qquad\square$

Corollary 6.3. *For any rational constants e^ε, δ the problem* DISTINGUISH (ε, δ)-DP *is* **PSPACE**-*hard.*

7 Other Definitions of Differential Privacy

Pure and approximate differential privacy degrade smoothly under composition: the overall privacy guarantee of a sequence of DP algorithms remains DP. However, in the worst case it is #**P**-hard to compute the best possible parameters achievable by a composition of approximate differentially private algorithms [43]. Other variants of differential privacy, such as Rényi [42], concentrated [12, 21], and truncated concentrated differential privacy [11], were introduced, in part, to address this problem. All of these notions lead to efficiently computable optimal composition bounds.

We show **PSPACE**-completeness for each of the problems of verifying (up to a precision parameter given as input) whether a BPWhile program is Rényi differentially private, concentrated differentially private, or truncated concentrated differentially private.

7.1 Rényi Differential Privacy

Definition 7.1. *Let $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$ be probability distributions over $1, \ldots, n$. For $\alpha > 1$, the Rényi divergence of P from Q is*

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \log \left(\sum_{i=1}^{n} \frac{p_i^\alpha}{q_i^{\alpha-1}} \right).$$

Definition 7.2. *[42] A program C is $(\alpha, \rho\alpha)$-Rényi-DP if for all neighboring inputs x, x',*

$$D_\alpha(C(x)\|C(x')) \le \rho\alpha.$$

We can check whether a BPWhile program C is $(\alpha, \rho\alpha)$-Rényi-DP using an algorithm similar to the one for checking (ε, δ)-DP from Section 5.3. A technical issue that arises here is that when computing Rényi divergences, we need to exponentiate possibly exponentially long numbers to exponentially large degrees α and $\alpha - 1$. We do not have the space to perform such computations exactly, so instead we consider a "gappped promise" version of the problem which takes an additional precision parameter η as input, and distinguishes between the case where the program is $(\rho, \rho\alpha)$-RDP and the case where it fails to be $(\rho, \rho\alpha + 2^{-\eta})$-RDP. The inclusion of this precision parameter allows us to approximately compute Rényi divergences via additions of logarithms of exponentially long numbers to at most exponential precision.

Definition 7.3. *In the* GAP-RÉNYI-DP *problem, an instance* (C, α, ρ, η) *consists of a BPWhile program C, two dyadic rational numbers α and ρ, and a binary integer parameter η. The problem is to distinguish between the following two cases:*

1. *Yes instances: for all neighboring inputs x, x' we have $D_\alpha(C(x)\|C(x')) \leq \rho\alpha$,*

2. *No instances: there exists a pair of neighboring inputs x, x' such that $D_\alpha(C(x)\|C(x')) \geq \rho\alpha + \frac{1}{2^\eta}$.*

Theorem 7.1. *The gap problem* GAP-RÉNYI-DP *is solvable by a deterministic algorithm using space polynomial in the size of the instance.*

Proof. Consider an instance (C, α, ρ, η) of the problem GAP-RÉNYI-DP. Following the definition of the problem, we iterate through all pairs of neighboring inputs (x, x'), and check that Rényi divergence is smaller than $\rho\alpha$. If the length of an instance is at most n, then, as the length of the program C is bounded by the length of the instance, we have at most 2^n possible output values. Denote the set of output values (including the non-termination outcome $\perp$) as O. We need to check the following condition:

$$\log \sum_{o \in O} \frac{\Pr[C(x) = o]^\alpha}{\Pr[C(x') = o]^{\alpha-1}} \leq \rho\alpha(\alpha - 1),$$

which we can rewrite as

$$\sum_{o \in O} \frac{\Pr[C(x) = o]^\alpha}{\Pr[C(x') = o]^{\alpha-1}} \leq 2^{\rho\alpha(\alpha-1)}.$$

First of all, we observe that for any pair of neighboring inputs x, x', if for some outcome o we have $\Pr[C(x) = o] > 0$ but $\Pr[C(x') = o] = 0$, then we automatically have a no-instance of the problem. This is because for every possible ρ and α we would get

$$\frac{\Pr[C(x) = o]}{\Pr[C(x') = o]} > 2^{\rho\alpha(\alpha-1)},$$

and so such C is not Rényi differentially private for these parameters. So for each pair of neighboring inputs and for each potential outcome, we first check whether at least one of the probabilities is equal to zero, and output "no-instance" if the second probability is non-zero. As all probabilities are finite and represented by numerators and denominators of at most exponential length, this can be performed in polynomial space.

Since α and ρ are given as part of the input, the lengths of α and ρ are at most n. Hence $\alpha, \rho \leq 2^n$. Therefore, $|\rho\alpha(\alpha - 1)| \leq 2^{3n}$ and we can compute $2^{\rho\alpha(\alpha-1)}$ in polynomial space.[2]

---

[2] We can print '1' followed by $2^{\rho\alpha(\alpha-1)}$ zeros by using a counter up to $\rho\alpha(1 - \alpha)$ to output the correct number of zeros.

Taking base-2 logarithms of $\Pr[C(x) = o]$ and $\Pr[C(x) = o]$, our goal is to (approximately) determine whether

$$\sum_{o \in O} 2^{\log \Pr[C(x)=o]\cdot\alpha - \log \Pr[C(x')=o]\cdot(\alpha-1)} \leq 2^{\rho\alpha(\alpha-1)}.$$

As this is a comparison between a sum of at most 2^n numbers and a number of length 2^{3n}, it suffices to compute $2^{3n} + \frac{\alpha-1}{2^\eta} + n$ bits of the quantity

$$2^{\log \Pr[C(x)=o]\cdot\alpha - \log \Pr[C(x')=o]\cdot(\alpha-1)}$$

for each $o \in O$ to determine which of the two cases we are in:

$$1) \sum_{o \in O} 2^{\log \Pr[C(x)=o]\cdot\alpha - \log \Pr[C(x')=o]\cdot(\alpha-1)} \leq 2^{\rho\alpha(\alpha-1)}, \text{ or}$$

$$2) \sum_{o \in O} 2^{\log \Pr[C(x)=o]\cdot\alpha - \log \Pr[C(x')=o]\cdot(\alpha-1)} \geq 2^{\rho\alpha(\alpha-1)+\frac{\alpha-1}{2^\eta}}.$$

As in our previous **PSPACE**-algorithms, we cannot explicitly store the value of each partial sum in the memory, as each has exponential length. So below, when we say that we "compute" an exponentially long number, we mean that we provide a polynomial space procedure that computes every bit of the number if its index is at most $2^{q(n)}$, where $q(n)$ is a fixed polynomial.

Again as in the algorithm for DECIDE (ε, δ) in Section 5.3, our goal is to compute a sum of 2^n numbers. But now each of this numbers have more complicated form $2^{\log \frac{\Pr_a}{\Pr_b}\cdot\alpha + \log \Pr_b}$, where $\Pr_a$ and $\Pr_b$ are exponentially long numbers, and α is a dyadic rational number of length at most n. For each element of the sum we need only to compute the $2^{3n} + \frac{\alpha-1}{2^\eta} + n$ most significant bits to guarantee that we underestimate each element of the sum by at most $2^{-(\alpha-1)/2^\eta - n}$. This yields an overall underestimate of the sum of all 2^n elements is at most $2^{-(\alpha-1)/2^\eta}$. Therefore, we underestimate the logarithm of this sum by at most $-(\alpha-1)/2^\eta$. Hence we always distinguish case 1 from case 2, by performing a comparison[3] to determine whether $2^{\log \frac{\Pr_a}{\Pr_b}\cdot\alpha + \log \Pr_b}$ is greater than $2^{\rho\alpha(\alpha-1)+\frac{\alpha-1}{2^\eta}}$ or smaller than $2^{\rho\alpha(\alpha-1)}$.

All that remains is to show that we can compute (i.e., give implicit access to each bit of) each term of the form $2^{\log \frac{\Pr_a}{\Pr_b}\cdot\alpha + \log \Pr_b}$ in polynomial space. Every bit of the integer logarithm can be computed using uniform circuits of polylogarithmic depth [46] in the length of the input integer and the index of the requested bit, so we can compute numbers of the form $\log(\Pr_a / \Pr_b)$ and $\log \Pr_b$. Further, using space-efficient algorithms for addition and multiplication of exponentially long numbers, as in Theorem 5.2, we can compute $\log \frac{\Pr_a}{\Pr_b} \cdot \alpha + \log \Pr_b$. Therefore, we can implicitly compute $\log \frac{\Pr_a}{\Pr_b} \cdot \alpha + \log \Pr_b$ with a

---

[3]Exponentially long numbers can be compared in polynomial space by finding the most significant bit on which they differ.

polynomial space algorithm. Finally, the exponential function has a representation as power series, and such power series can be computed by uniform families of logarithmic-depth circuits [46, Corollary 2.2]. So we can exponentiate 2 to a dyadic rational degree using an algorithm that runs in space logarithmic in the length of the exponent. Combining polynomial space computations we obtain a polynomial space algorithm for computing $2^{\log \frac{\Pr_a}{\Pr_b} \cdot \alpha + \log \Pr_b}$. $\square$

To show **PSPACE**-hardness, we use Theorem 6.1, and the following fact to reduce from DISTINGUISH (ε, δ)-DP to GAP-RENYI-DP:

Theorem 7.2 ([42]). *If C is an $(\alpha, \rho\alpha)$-RDP program, it also satisfies $(\rho\alpha + \frac{\log 1/\delta}{\alpha - 1}, \delta)$-differential privacy for any $\delta \in (0, 1)$.*

Combining this with Theorem 6.1, which states that it is **PSPACE**-hard to determine whether a BPWhile program is $(0, 0)$-differentially private or not (ε, δ)-differentially private, we obtain:

Theorem 7.3. GAP-RÉNYI-DP *is* **PSPACE**-*hard.*

Proof. Fix two dyadic rational numbers $\varepsilon, \delta \in (0, 1)$. Let η, ρ and α be positive numbers with finite binary representations such that

$$0 < \rho\alpha + \frac{\log (1/\delta)}{\alpha - 1} + \frac{1}{2^\eta} < \varepsilon.$$

To reduce from DISTINGUISH (ε, δ)-DP to the GAP-RÉNYI-DP problem, we map an instance C of DISTINGUISH (ε, δ)-DP to the instance (C, α, ρ, η) in deterministic linear time.

To show correctness of this reduction, first consider the case where C is a yes-instance of DISTINGUISH (ε, δ)-DP. That means that C is $(0, 0)$-DP. Then the distributions on the outputs of C are identical for every pair of neighboring inputs. Hence C is also $(\alpha, \rho\alpha)$-Rényi-DP, so (C, α, ρ, η) is a yes-instance of GAP-RÉNYI-DP.

Now, consider the case where C is a no-instance of DISTINGUISH (ε, δ)-DP. We need to show that (C, α, ρ, η) is a no-instance of GAP-RÉNYI-DP. We do this by contraposition: If (C, α, ρ, η) is not a no-instance, then for all neighboring inputs x, x' it holds that $D_\alpha(C(x)\|C(x')) \leq \rho\alpha + \frac{1}{2^\eta}$. Then for $\rho' = \rho + \frac{1}{\alpha 2^\eta}$, we have that C is $(\alpha, \rho'\alpha)$-Rényi-DP. Then by Theorem 7.2, C is $(\rho'\alpha + \frac{\log (1/\delta)}{\alpha - 1}, \delta)$-DP. But by our choice of the parameters, $\rho'\alpha + \frac{\log (1/\delta)}{\alpha - 1} = \rho\alpha + \frac{1}{2^\eta} + \frac{\log (1/\delta)}{\alpha - 1} < \varepsilon$, hence C is (ε, δ)-DP. This implies that C is not a no-instance of DISTINGUISH (ε, δ)-DP.

Since we showed in Theorem 6.1 we showed that DISTINGUISH (ε, δ)-DP is **PSPACE**-hard, it follows that GAP-RÉNYI-DP is **PSPACE**-hard. $\square$

Combining the results of Theorem 7.1 and Theorem 7.3 we obtain:

Corollary 7.4. GAP-RÉNYI-DP *is* **PSPACE**-*complete.*

7.2 Concentrated Differential Privacy

Definition 7.4. *[12, 21] A program C is ρ-Concentrated-DP if for every neighboring inputs x, x' and every $\alpha \in (1, +\infty)$*

$$D_\alpha(C(x)\|C(x')) \leq \rho\alpha.$$

As in Section 7.1 we consider a gapped version of the problem for an integer precision parameter η provided as input.

Definition 7.5. *An instance of the* Gap-Concentrated-DP *problem* (C, ρ, η) *consists of a BPWhile program C, a dyadic rational number ρ, and a binary integer precision parameter η. The goal is to distinguish between the following two cases:*

1. *Yes-instances: for all neighboring inputs x, x' and for all $\alpha \in (1, +\infty)$, we have $D_\alpha(C(x)\|C(x')) \leq \rho\alpha$,*

2. *No-instances: there exists a pair of neighboring inputs x, x' and $\alpha \in (1, +\infty)$ such that $D_\alpha(C(x)\|C(x')) \geq \rho\alpha + \frac{1}{2^\eta}$.*

In order to verify whether an instance (C, ρ, η) is a yes-instance of Gap-Concentrated-DP we should verify whether the inequality $D_\alpha(C(x)\|C(x')) \leq \rho\alpha$ holds not only for all pairs (x, x'), but also for all $\alpha > 1$. This is equivalent to verifying that C is $(\alpha, \rho\alpha)$-Rényi-DP for every $\alpha > 1$. But as there is an unbounded continuum of possible α to consider, we do not immediately obtain an algorithm by attempting to exhaustively check them. The following lemma shows that to solve the gapped version of the problem, we need only to consider finitely many α within a finite range.

Lemma 7.5. *Suppose (C, ρ, η) is a no-instance of the* Gap-Concentrated-DP *problem. Then there exists a polynomial $p(n)$, neighboring inputs x, x', and $\alpha \in (1, 1 + 2^{p(n)}/\rho)$ an integer multiple of $2^{-\eta-1}/\rho$ such that $D_\alpha(C(x)\|C(x')) \geq \rho\alpha + \frac{1}{2^{\eta+1}}$.*

Proof. The problem we are interested in is as follows. Given implicit descriptions of two finite probability distributions $d_1(i)$ and $d_2(i)$, where each probability is discretized to $1/2^{2^{p(n)}}$, and a rational parameter ρ, determine whether

$$\frac{1}{\alpha}D_\alpha(d_1\|d_2) = \frac{1}{\alpha(\alpha-1)}\log\sum_i (d_1(i))^\alpha (d_2(i))^{1-\alpha} \leq \rho,$$

for all dyadic rational $\alpha \in (1, \infty)$ with precision $1/2^{p(|x|)}$, or whether for at least one dyadic rational $\alpha \in (1, \infty)$ with precision $1/2^{p(|x|)}$ it holds that

$$\frac{1}{(\alpha-1)}\log\sum_i (d_1(i))^\alpha (d_2(i))^{1-\alpha} \geq \rho\alpha + \frac{1}{2^\eta}.$$

Let $m = 2^{2^{p(n)}}$, so each probability in $d_1(i)$ and $d_2(i)$ are discretized to $1/m$. We claim that it suffices to check this condition for all $\alpha < 1 + \log m/\rho$. Either

23

1. There exists i in the probability space such that $d_1(i) > 0$ and $d_2(i) = 0$, in which case $D_\alpha(d_1 \| d_2)$ is infinite for every $\alpha > 1$; or

2. For every outcome i, the value m is an upper bound on the ratio $d_1(i)/d_2(i)$. In this case, the quantity $\frac{1}{\alpha} D_\alpha(d_1 \| d_2)$ we are interested in is at most

$$\frac{1}{\alpha(\alpha-1)} \log \sum_i m^\alpha \cdot d_2(i) \leq \frac{1}{\alpha(\alpha-1)} \log(m^\alpha) \leq \frac{\log m}{\alpha-1} \quad ,$$

which is at most ρ for $\alpha \geq 1 + \frac{\log m}{\rho}$.

In both cases we need to check values of α within the interval $(1, 1 + \log m/\rho)$.

That still leaves us with the infinite number of values of α we need to check. To finish the proof of the lemma we show that it is enough to consider values of α discretized to $\frac{2^{-\eta}}{\rho}$:

Claim 7.6. *Fix distributions P and Q and $\rho > 0$. If $D_\alpha(P\|Q) \leq \rho\alpha + 2^{-\eta}$ for every $\alpha > 1$ that is discretized to an integer multiple of $2^{-\eta}/\rho$, then $D_\alpha(P\|Q) < \rho\alpha + 2^{-\eta+1}$ for every $\alpha > 1$.*

Proof. It suffices to show that if $0 < \alpha_1 < \alpha_2 = \alpha_1 + 2^{-\eta}/\rho$ are such that $D_{\alpha_1}(P\|Q) \leq \rho\alpha_1 + 2^{-\eta}$ and $D_{\alpha_2}(P\|Q) \leq \rho\alpha_2 + 2^{-\eta}$, then for every α' with $\alpha_1 < \alpha' < \alpha_2$ we have $D_{\alpha'}(P\|Q) < \rho\alpha' + 2^{-\eta+1}$. As the Rényi divergence between two distributions increases monotonically as a function of α, we have

$$D_{\alpha'}(P\|Q) \leq D_{\alpha_2}(P\|Q) \leq \rho\alpha_2 + 2^{-\eta}$$
$$= \rho(\alpha' + (\alpha_2 - \alpha')) + 2^{-\eta} = \rho\alpha' + \rho(\alpha_2 - \alpha') + 2^{-\eta}.$$

From the facts that $\alpha_2 - \alpha_1 = 2^{-\eta}/\rho$ and $\alpha_1 < \alpha' < \alpha_2$ we get $(\alpha_2 - \alpha') < 2^{-\eta}/\rho$. Hence $D_{\alpha'}(P\|Q) < \rho\alpha' + 2^{-\eta+1}$. $\square$

This completes the proof of the lemma, as we showed that we can consider values of α discretized to $2^{-\eta}/\rho$ in a bounded range. $\square$

Combining this lemma with the algorithm from Theorem 7.1 we get the following theorem:

Theorem 7.7. *The gap problem* GAP-CONCENTRATED-DP *is solvable by a deterministic algorithm using space polynomial in the size of the instance.*

Similarly to the proof of Theorem 7.3 we can use the fact that concentrated DP implies approximate DP to give a reduction from DISTINGUISH (ε, δ)-DP to GAP-CONCENTRATED-DP.

Theorem 7.8 ([12])**.** *If C is an ρ-CDP program, it also satisfies $(\rho + 2\sqrt{\rho \log(1/\delta)}, \delta)$-differential privacy for any $\delta \in (0, 1)$.*

24

Theorem 7.9. GAP-CONCENTRATED-DP *is* **PSPACE**-*hard.*

Combining the results of Theorem 7.7 and Theorem 7.9 we get the corollary:

Corollary 7.10. GAP-CONCENTRATED-DP *is* **PSPACE**-*complete.*

7.3 Truncated Concentrated Differential Privacy

Definition 7.6. *[11] A program C is ω-Truncated ρ-Concentrated-DP if for every neighboring inputs x, x' and every $\alpha \in (1, \omega)$*

$$D_\alpha(C(x)\|C(x')) \leq \rho\alpha.$$

Again, we introduce a promise version of the problem. This allows us to consider quantities of fixed precision parameterized by η:

Definition 7.7. *In the* GAP-TRUNCATED-CONCENTRATED-DP *problem, an instance (C, ρ, ω, η) consists of a BPWhile program C, two dyadic rational numbers ρ and ω, and a binary integer precision parameter η. The goal is to distinguish between the following two cases:*

1. *Yes-instances: for all neighboring inputs x, x' and for all $\alpha \in (1, \omega)$,*

$$D_\alpha(C(x)\|C(x')) \leq \rho\alpha,$$

2. *No-instances: there exists a pair of neighboring inputs x, x' and $\alpha \in (1, \omega)$ such that*

$$D_\alpha(C(x)\|C(x')) \geq \rho\alpha + \frac{1}{2^\eta}.$$

As we need to verify a bounded range of values of the parameter α the verification procedure is analogous to the algorithm for verifying concentrated differential privacy. Therefore, we get the following theorem:

Theorem 7.11. *The promise problem* GAP-TRUNCATED-CONCENTRATED-DP *is solvable by a deterministic algorithm using space polynomial in the size of the instance.*

Similarly to the proof of Theorem 7.3 and Theorem 7.9 we use an existing result connecting the parameters of approximate and truncated concentrated differential privacy to show reduction from DISTINGUISH (ε, δ)-DP to GAP-TRUNCATED-CONCENTRATED-DP.

Theorem 7.12 ([11])**.** *If C is an (ρ, ω)-TCDP mechanism, then it also satisfies $(\rho + 2\sqrt{\rho \log(1/\delta)}, \delta)$-differential privacy for any $\delta \in (0, 1)$ that satisfies $\log(1/\delta) \leq (\omega - 1)^2\rho$.*

The proof of hardness is then similar to the hardness result for GAP-RÉNYI-DP.

Theorem 7.13. GAP-TRUNCATED-CONCENTRATED-DP *is* **PSPACE**-*hard.*

Combining the results of Theorem 7.11 and Theorem 7.13 we get the corollary:

Corollary 7.14. GAP-TRUNCATED-CONCENTRATED-DP *is* **PSPACE**-*complete.*

8 Termination-sensitive vs termination-insensitive differential privacy

In the definition of differential privacy we have considered in this paper, Definition 3.7, we considered sets of outcomes over $\{0,1\}^l \cup \{\bot\}$. This definition corresponds to a "termination-senstive" differential privacy model where the adversary can observe the program's termination behavior. As in information flow control, one can also study a "termination-insensitive" model where we require Condition 1, in Definition 3.7, to hold only for sets of outcomes $O \subseteq \{0,1\}^\ell$ and where the probabilities are conditioned on the program C terminating. It is easy to see that, up to a factor of 2, termination-sensitive (pure) ε-differential privacy implies termination-insensitive (pure) ε-differential privacy. Indeed, for all neighbors x, x' and $O \subseteq \{0,1\}^\ell$ we have:

$$\Pr[C(x) \in O \mid C(x) \neq \bot] = \frac{\Pr[C(x) \in O]}{\Pr[C(x) \neq \bot]}$$
$$\leq \frac{e^\varepsilon \Pr[C(x') \in O]}{e^{-\varepsilon} \Pr[C(x') \neq \bot]} = e^{2\varepsilon} \Pr[C(x') \in O | C(x') \neq \bot].$$

This argument does not work for (ε, δ)-differential privacy. Indeed, a program that given a boolean input b returns b with probability δ and $\bot$ with probability $1 - \delta$ is termination-sensitive $(0, \delta)$-differentially private, but is not termination-insensitive (ε, δ)-differentially private for any $\varepsilon < \infty$ or $\delta < 1$.

Meanwhile, even termination-insensitive pure differential privacy does not imply termination-sensitive differential privacy. For example, a program that on input 0 returns 0 with probability 1, and on input 1 returns 0 with probability 0.01, and $\bot$ with probability 0.99 is termination-insensitive 0-differentially private, but is not termination-sensitive (ε, δ)-differentially private for any $\delta < 0.99$.

In this work, we focused on the termination-sensitive model because it is the most natural from a probabilistic perspective and because it is less susceptible to timing side-channel attack such as the one illustrated in the latter example above. Nevertheless, all of our results can be adapted to work for the termination-insensitive model as well. Our algorithmic results can be adapted by explicitly normalizing the computed probabilities by the probability of termination, which can be computed in **PSPACE**. Meanwhile, our lower bounds hold by replacing all steps where we explicitly enter an infinite loop with steps where we output a special failure symbol. This modification should be made to both the specification and reduction we consider for almost-sure termination, as well as for our reductions from almost-sure termination to privacy verification problems.

9 Conclusions and Future work

In this paper we have shown that the problem of deciding a probabilistic boolean program to be differentially private, for several notions of differential privacy, is **PSPACE**-complete. In addition we have shown that also an approximate version of this problem is **PSPACE**-hard. These results can help identify the limitations of automated verification methods. One direction that our results point to is the use of QBF solvers [1] for reasoning about differential privacy and almost sure termination for BPWhile programs. But first, to apply a QBF solver to verify whether an input BPWhile program is differentially private, the program should be efficiently converted to a quntatified boolean formula. In current work we only showed reduction in the opposite direction, namely we converted a QBF formula to a BPWhile program in Lemma B. We leave the discussion of the applicability of QBF solvers for future work.

Our proofs of the **PSPACE**-hardness results for Gap-Rényi-DP, Gap-Concentrated-DP, and Gap-Truncated-Concentrated-DP uses simple reduction from **PSPACE**-hardness of Distinguish (ε, δ)-DP that doesn't rely on any specific properties of BPWhile language. The similar reduction can be used to show, for example, **NP**- and **coNP**-hardness of Gap-Rényi-DP, Gap-Concentrated-DP, and Gap-Truncated-Concentrated-DP for loops-free boolean language from [26].

Results that we discuss show that a problem of checking various properties of probabilistic boolean programs with while loops require exactly polynomial space. A problem that generalizes all these results is the problem of checking whether a probability distributions on the outputs of the program satisfy a property expressed by a polynomial-space computation. The **PSPACE**-hardness of verification of such generalized property is implied by the almost sure termination. That is so as we have an easily verifiable property of the output distribution, we just need to check whether a sum of the probabilities of the outputs is 1 or not. But can we show that every property of the output distribution can be computed by a polynomial space algorithm, if we are given an implicit access to the distribution through the algorithm that outputs every requested bit of the probability of requested output?

Acknowledgments

References

[1] The quantified boolean formulas satisfiability library. http://www.qbflib.org.

[2] Aws Albarghouthi and Justin Hsu. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.*, 2(POPL):58:1–58:30, 2018. `doi:10.1145/3158146`.

[3] Victor Balcer and Salil P. Vadhan. Differential privacy on finite computers. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 43:1–43:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ITCS.2018.43`.

[4] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. Deciding differential privacy for programs with finite inputs and outputs. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 141–154. ACM, 2020.

[5] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, and Pierre-Yves Strub. Proving differential privacy in hoare logic. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 411–424. IEEE Computer Society, 2014. `doi:10.1109/CSF.2014.36`.

[6] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *POPL*, pages 55–68, 2015.

[7] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. In *LICS '16*, pages 749–758, New York, NY, USA, 2016. ACM.

[8] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Beguelin. Probabilistic relational reasoning for differential privacy. *ACM SIGPLAN Notices*, 47(1):97–110, 2012.

[9] Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin Vechev. Dp-finder: Finding differential privacy violations by sampling and optimization. In *CCS '18*, pages 508–524, 2018.

[10] Allan Borodin, Stephen A. Cook, and Nicholas Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Inf. Control.*, 58(1-3):113–136, 1983. `doi:10.1016/S0019-9958(83)80060-6`.

[11] Mark Bun, Cynthia Dwork, Guy N. Rothblum, and Thomas Steinke. Composable and versatile privacy via truncated CDP. In Ilias Diakonikolas, David Kempe, and

Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 74–86. ACM, 2018. `doi:10.1145/3188745.3188946`.

[12] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 635–658, 2016. `doi:10.1007/978-3-662-53641-4\_24`.

[13] Rohit Chadha, Dileep Kini, and Mahesh Viswanathan. Quantitative information flow in boolean programs. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2014. `doi:10.1007/978-3-642-54792-8\_6`.

[14] Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. On linear time decidability of differential privacy for programs with unbounded inputs. In *LICS*, pages 1–13. IEEE, 2021.

[15] Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Bisimilarity distances for approximate differential privacy. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2018. `doi:10.1007/978-3-030-01090-4_12`.

[16] Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Asymmetric distances for approximate differential privacy. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.10`.

[17] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995. `doi:10.1145/210332.210339`.

[18] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. Detecting violations of differential privacy. In *CCS 2018*, pages 475–489, 2018.

[19] Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Dan Kifer. Free gap information from the differentially private sparse vector and noisy max mechanisms. *Proc. VLDB Endow.*, 13(3):293–306, 2019. `doi:10.14778/3368289.3368295`.

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[21] Cynthia Dwork and Guy N. Rothblum. Concentrated differential privacy. *CoRR*, abs/1603.01887, 2016. `arXiv:1603.01887`.

[22] Kousha Etessami and Mihalis Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, 2009. `doi:10.1145/1462153.1462154`.

[23] Gian Pietro Farina, Stephen Chong, and Marco Gaboardi. Coupled relational symbolic execution for differential privacy. In *ESOP*, volume 12648 of *Lecture Notes in Computer Science*, pages 207–233. Springer, 2021.

[24] Matthew Fredrikson and Somesh Jha. Satisfiability modulo counting: a new approach for analyzing privacy properties. In Thomas A. Henzinger and Dale Miller, editors, *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), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 42:1–42:10. ACM, 2014. `doi:10.1145/2603088.2603097`.

[25] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *POPL*, pages 357–370, 2013.

[26] Marco Gaboardi, Kobbi Nissim, and David Purser. The complexity of verifying loop-free programs as differentially private. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 129:1–129:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[27] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. Preserving differential privacy under finite-precision semantics. *Theor. Comput. Sci.*, 655:92–108, 2016. `doi:10.1016/j.tcs.2016.01.015`.

[28] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM J. Comput.*, 41(6):1673–1693, 2012. `doi:10.1137/09076828X`.

[29] Anna C. Gilbert and Audra McMillan. Property testing for differential privacy. In *56th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2018, Monticello, IL, USA, October 2-5, 2018*, pages 249–258. IEEE, 2018. `doi:10.1109/ALLERTON.2018.8636068`.

[30] Patrice Godefroid and Mihalis Yannakakis. Analysis of boolean programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2013.

[31] Sergiu Hart and Micha Sharir. Probabilistic temporal logics for finite and bounded models. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 1–13. ACM, 1984. `doi:10.1145/800057.808660`.

[32] Juris Hartmanis and Janos Simon. On the structure of feasible computations. *Adv. Comput.*, 14:1–43, 1976. `doi:10.1016/S0065-2458(08)60449-0`.

[33] Christina Ilvento. Implementing the exponential mechanism with base-2 differential privacy. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 717–742. ACM, 2020. `doi:10.1145/3372297.3417269`.

[34] H. Jung. Relationships between probabilistic and deterministic tape complexity. In Jozef Gruska and Michal Chytil, editors, *Mathematical Foundations of Computer Science 1981, Strbske Pleso, Czechoslovakia, August 31 - September 4, 1981, Proceedings*, volume 118 of *Lecture Notes in Computer Science*, pages 339–346. Springer, 1981. `doi:10.1007/3-540-10856-4\_101`.

[35] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Informatica*, 56(3):255–285, 2019. `doi:10.1007/s00236-018-0321-1`.

[36] Haim Kaplan, Yishay Mansour, and Uri Stemmer. The sparse vector technique, revisited. In *COLT*, volume 134 of *Proceedings of Machine Learning Research*, pages 2747–2776. PMLR, 2021.

[37] Daniel Kifer, Solomon Messing, Aaron Roth, Abhradeep Thakurta, and Danfeng Zhang. Guidelines for implementing and auditing differentially private systems. *CoRR*, abs/2002.04049, 2020. `arXiv:2002.04049`.

[38] Daniel Lehmann and Saharon Shelah. Reasoning with time and chance. *Inf. Control.*, 53(3):165–198, 1982. `doi:10.1016/S0019-9958(82)91022-1`.

[39] Depeng Liu, Bow-Yaw Wang, and Lijun Zhang. Model checking differentially private properties. In Sukyoung Ryu, editor, *Programming Languages and Systems - 16th*

Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings, volume 11275 of *Lecture Notes in Computer Science*, pages 394–414. Springer, 2018. `doi:10.1007/978-3-030-02768-1\_21`.

[40] Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, February 2017.

[41] Ilya Mironov. On significance of the least significant bits for differential privacy. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 650–661. ACM, 2012. `doi:10.1145/2382196.2382264`.

[42] Ilya Mironov. Rényi differential privacy. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 263–275. IEEE Computer Society, 2017. `doi:10.1109/CSF.2017.11`.

[43] Jack Murtagh and Salil P. Vadhan. The complexity of computing the optimal composition of differential privacy. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 157–175. Springer, 2016. `doi:10.1007/978-3-662-49096-9\_7`.

[44] Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proc. ACM Program. Lang.*, 3(OOPSLA):172:1–172:30, 2019.

[45] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *ICFP 2010*, pages 157–168. ACM, 2010.

[46] John H. Reif. Logarithmic depth circuits for algebraic functions. *SIAM J. Comput.*, 15(1):231–242, 1986. `doi:10.1137/0215017`.

[47] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. `doi:10.1016/S0022-0000(70)80006-X`.

[48] Janos Simon. On tape-bounded probabilistic turing machine acceptors. *Theor. Comput. Sci.*, 16:75–91, 1981.

[49] Michael Carl Tschantz, Dilsun Kirli Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). In Michael W. Mislove and Joël Ouaknine, editors, *Twenty-seventh Conference on the Mathematical*

Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier, 2011. `doi:10.1016/j.entcs.2011.09.015`.

[50] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 327–338. IEEE Computer Society, 1985. `doi:10.1109/SFCS.1985.12`.

[51] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng 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. To appear.

[52] Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.

[53] Danfeng Zhang and Daniel Kifer. Lightdp: towards automating differential privacy proofs. In *POPL 2017*, pages 888–901. ACM, 2017.

[54] Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C. Pierce, and Aaron Roth. Testing differential privacy with dual interpreters. *Proc. ACM Program. Lang.*, 4(OOPSLA):165:1–165:26, 2020.

A   Computing Hitting Probabilities in Polylogarithmic Space

Here we give a brief overview of Simon's algorithm [48] for computing the hitting probabilities of a Markov chain with n states using $O(\log^6 n)$ space.

The original application in Simon's paper was to show that unbounded-error probabilistic Turing machines can be simulated in deterministic polynomial space. That is, he showed that one can determine in **PSPACE** whether the accept configuration in the configuration graph of a probabilistic TM is reached with probability strictly greater than $1/2$. This in turn is accomplished by interpreting the configuration graph as a Markov chain and exactly computing the hitting probability of the accept configuration.

We now describe the algorithm for computing hitting probabilities captured in Lemma 5.2. Recall that we are given a Markov chain $M = (V, E, p, p_0)$ with 2^L states. The Markov chain is represented by its transition matrix (an object of size $2^{O(L)}$), so each entry can be addressed using $O(L)$ space. We assume that p_0 is supported on a single start state, that all non-final states are non-recurrent (i.e., upon leaving a non-final state, the probability the Markov chain returns to it is less than 1) and that for all non-final states, every outgoing transition has probability either 0 or $1/2$.

Simon first described an algorithm using $O(L^3)$ time in the *random access machine with multiplication* (MRAM) model – a model of parallel computation with unit-cost multipli-

cation. This implies an $O(L^6)$-space algorithm on a deterministic TM using a generic simulation of time $T(n)$ MRAM algorithms by space $O(T^2(n))$-space deterministic TMs [32].

The MRAM algorithm works as follows. Let P denote the transition matrix of the Markov chain M. Let Q be the submatrix of P corresponding to the non-final states. For a given final state f, let v_f be the column of P corresponding to state f, but restricted to the entries corresponding to non-final states. Let v_s^T be the row vector with a 1 in the entry corresponding to the start state and 0's elsewhere. Then letting $Q_\infty = Q + Q^2 + Q^3 + \ldots$, we have that the probability of reaching the final state f from the start state s is $a = v_s^T Q_\infty V_f$. The goal now becomes to compute this matrix-vector product.

The key idea is that since Q consists only of non-recurrent states, then Q_∞ is well-defined and $Q_\infty = (I - Q)^{-1} - I$. Matrix inversion (more precisely, computing the numerators and denominators of the resulting entries separately) can be performed on an MRAM in time $O(L^3)$ using a variant of Csanksy's algorithm. This dominates the runtime of the algorithm, which just has to perform the matrix-vector product.

B Reduction from TQBF to Almost Sure Termination

Suppose we have a fully quantified Boolean formula

$$\psi = \forall x_1 \in \{0,1\} \exists x_2 \in \{0,1\} \ldots \forall x_t \in \{0,1\} \phi(x_1, \ldots, x_t)$$

in prenex normal form. We wish to check whether $\psi \in$ TQBF. We create a BPWhile program, the template for which we give below, such that the program terminates almost surely iff ψ is true. As in previous reductions, for the sake of readability we use a few extra constructions that BPWhile doesn't formally support, such as variables that take on constant-size integer values.

```
A:
input(b); # dummy input bit that
# the program ignores
c1 = 0;
x1 = 0;
while x1 <= 1   then
    c2 = 0;
    x2 = 0;
    while x2 <= 1 then
        c3 = 0;
        x3 = 0;
        while x3 <= 1 then
        ...
            while xt <= 1 then
                if phi(x1,..., xt)==1 then
```

34

```
                    ct++;
                xt++;
            if  ct == 2 then
                    c(t-1)++;
        ...
        if  c3 == 2 then
            c2++;
        x2++;
    if  c2 >= 1 then
        c1++;
    x1++;
if  (c1 < 2) then   #psi  is  false
    while  true  then  #enter  infinite  loop
        skip;
return (1)
```

The first part of the program uses t nested while loops to evaluate the QBF formula ψ. Each loop corresponding to a universal quantifier checks that both assignments to its variable return 1. Meanwhile, each loop corresponding to an existential quantifier checks that at least one of the assignments returns 1.

After evaluating the entire formula, the program enters an infinite loop if it evaluates to false, and otherwise terminates with probability 1.

Hence this construction produces a BPWhile program that terminates with probability 1 iff the formula ψ is true. The construction of the program takes time polynomial in the size of ψ, so checking almost sure termination is **PSPACE**-hard.

C Additional Proofs from Section 6

Proof of Claim 6.2. Suppose P is not almost surely terminating and that x is an input on which the program is not terminating with some positive probability. Consider the Markov chain corresponding to the execution of $P(x)$. This Markov chain has a reachable, recurrent non-final state. Since a program of size N has at most $2^{\text{poly}(N)}$ states, this recurrent state is reachable within $2^{\text{poly}(N)}$ transitions. Moreover, since each transition has probability either $0, 1/2$, or 1, the probability of reaching this recurrent state is at least $2^{-2^{\text{poly}(N)}}$.

The program P' will amplify the probability of reaching this recurrent state (i.e., entering this infinite loop) by repeating P many times. Below we describe how to encode this number of repetitions succinctly. We provide a code template where we operate with two vectors of $m+1$ boolean variables $Counter$ and B, that we use in this program to represent integers in the range $[0, 2^{m+1})$. We compare and increment these variables, and both of these operations can be encoded as simple procedure of polynomial size in the length of m with boolean variables only.

```
1.    input(x);
2.    while true then
3.       B = 0;
4.       Counter = 0;
5.       while (Counter < 2^m) then
6.          increment(Counter);
7.          a = random;
8.          if a = 1 then
9.             increment(B);
10.            if B < 2^m then
11.               P(x);
12.            else
13.               return(1)
```

In short, this program terminates if and only if out of 2^m coin tosses in the inner while loop, we get 2^m 1's. As the probability of this event is $1/2^{2^m}$, we get that we need approximately 2^{2^m} iterations of the external loop to finally get exactly 2^m ones in 2^m coin tosses. As in each iteration of the external loop we run the program P that with probability at least $1/2^{2^m}$ enters an infinite loop, overall we enter this loop with constant probability when it exists.

We now analyze the guarantee of P' more formally. Let X be the number of rounds in which the outer loop runs, and $q = 2^{-2^m}$ be the probability of getting $B = 2^m$ after the inner loop run. Then we get that X is distributed as a geometric random variable

$$\Pr[X = 1] = q, \Pr[X = 2] = q(1 - q),$$

$$\Pr[X = 3] = q(1 - q)^2, \ldots$$

Then we can estimate the probability that P' halts as:

$$\Pr[P' \text{ halts}] \leq \sum_{k=1}^{\infty} q(1 - q)^{k-1} \cdot (1 - q)^k = q \sum_{k=1}^{\infty} (1 - q)^{2k-1}$$

$$= \frac{q}{1 - q} \cdot \sum_{k=1}^{\infty} (1 - q)^{2k} = \frac{q}{1 - q} \cdot \frac{(1 - q)^2}{1 - (1 - q)^2}$$

$$= \frac{q \cdot (1 - q)}{1 - (1 - q)^2} = \frac{q(1 - q)}{q(2 - q)} = \frac{1 - q}{2 - q} = 1/2 - \frac{0.5q}{2 - q}.$$

Hence, if $0 < q < 1$ we get that $\Pr[P' \text{ halts}] < 1/2$. □

D   Additional Proofs from Section 7

Proof of Theorem 7.7. Let C be a BPWhile program of length n, ρ be a dyadic rational number, and η be a binary integer precision parameter. Let $p(n)$ be a polynomial such

that all probabilities of reaching the final state in the Markov chain of P are discretized to $2^{-2^{p(n)}}$.

By Lemma 7.5, in order to solve an instance (C, ρ, η) GAP-CONCENTRATED-DP, it suffices to solve instances (C, α, ρ, η) of GAP-RÉNYI-DP for every α in range $(1, 1 + \frac{2^{p(n)}}{\rho})$ discretized to precision $2^{-\eta-1}/\rho$. That will also imply that we never have an event where x, x' are neighboring inputs, $\Pr[C(x) = o] = 0$ and $\Pr[C(x') = o] > 0$. Note that the largest value of α is $1 + 2^{p(n)}/\rho$, and all these operations for computing each bit of α can be done by uniform families of polylogarithmic-depth circuits. Therefore, for any α in the considered interval we can recompute $2^n \cdot (1 + 2^{p(n)}/\rho)^2 + \frac{\alpha-1}{2^\eta} + n$ bits of the quantity

$$\sum_{o \in O} 2^{\log \Pr[C(x)=o] \cdot \alpha - \log \Pr[C(x')=o] \cdot (\alpha-1)}$$

and check whether it is less than $2^{\rho\alpha(\alpha-1)}$ or greater than $2^{\rho\alpha(\alpha-1)+\frac{\alpha-1}{2^\eta}}$. As was shown in Theorem 7.1 all these operations are computable in polynomial space. Hence, using the algorithm from Theorem 7.1 for every α we can verify whether (C, ρ, η) is a yes-instance of GAP-CONCENTRATED-DP or whether it is a no-instance.

As by Lemma 7.5 we can consider values of α discretized to $\frac{2^{-\eta-1}}{\rho}$ we get that there are at most $\frac{\rho}{2^{-\eta-1}} \cdot (1 + \frac{2^{p(n)}}{\rho})$ values of α to consider, and each such value has representation of polynomial length. Therefore, we run the algorithm for checking $(\alpha, \alpha\rho)$-Rényi-DP at most $(p(n)+\rho) \cdot 2^{\eta+1}$ times. As iterating over exponentially many elements keeps us in **PSPACE** if we re-use the space on each iteration, we get a **PSPACE**-algorithm for checking whether C is ρ-concentrated differentially private. $\square$

Proof of Theorem 7.9. Fix two dyadic rational numbers $\varepsilon, \delta \in (0, 1)$. Let η and ρ be positive numbers with finite binary representations such that

$$0 < \rho + 2\sqrt{\left(\rho + \frac{1}{2^\eta}\right) \log(1/\delta)} + \frac{1}{2^\eta} < \varepsilon.$$

We show a reduction from DISTINGUISH (ε, δ)-DP to GAP-CONCENTRATED-DP problem. We map each instance C of DISTINGUISH (ε, δ)-DP into an instance (C, ρ, η).

As in the proof of Theorem 7.3 when C is a yes-instance of DISTINGUISH (ε, δ)-DP, C is also ρ-CDP. Therefore, (C, ρ, η) is a yes-instance of GAP-CONCENTRATED-DP.

In the case where C is a no-instance of DISTINGUISH (ε, δ)-DP we need to show that then (C, ρ, η) is a no-instance of GAP-CONCENTRATED-DP. Assume the opposite: for all neighboring inputs x, x', and for all $\alpha > 1$ it holds $D_\alpha(C(x) \| C(x')) \le \rho\alpha + \frac{1}{2^\eta}$. Then, for $\rho' = \rho + \frac{1}{2^\eta}$, C is $(\alpha, \rho'\alpha)$-RDP for all $\alpha > 1$, as

$$D_\alpha(C(x) \| C(x')) \le \rho\alpha + \frac{1}{2^\eta} \le \left(\rho + \frac{1}{\alpha 2^\eta}\right)\alpha \le \left(\rho + \frac{1}{2^\eta}\right)\alpha.$$

Therefore, C is (ρ')-CDP. Hence by Theorem 7.8, C is $(\rho' + 2\sqrt{\rho' \log{(1/\delta)}}, \delta)$-DP. But by our choice of the parameters, $\rho' + 2\sqrt{\rho' \log{(1/\delta)}} = \rho\alpha + 2\sqrt{(\rho + \frac{1}{2^\eta})\log{(1/\delta)}} + \frac{1}{2^\eta} < \varepsilon$, hence C is (ε, δ)-DP. But that contradicts our assumption that C is a no-instance of DISTINGUISH (ε, δ)-DP. Therefore, (C, ρ, η) is a no instance of GAP-CONCENTRATED-DP.

We showed that the reduction is correct. It runs in linear time because the parameters ρ, η are computed from ε, δ independent of the instance.

Since Theorem 6.1 showed that DISTINGUISH (ε, δ)-DP is **PSPACE**-hard, we get that GAP-CONCENTRATED-DP is **PSPACE**-hard. $\square$

Proof of Theorem 7.11. The algorithm is the same as in the proof of Theorem 7.7, with only one change: instead of considering α in the range $(1, 1 + \log m/\rho)$, where m depends on a discretization of probabilities in the Markov chain of the program C, we consider the range $(1, \omega)$. Although the parameter $\omega > 1 + \log m/\rho$, we again need to iterate over at most exponentially many in the length of the input values of α. Therefore, getting a **PSPACE**-algorithm for GAP-TRUNCATED-CONCENTRATED-DP. $\square$

Proof of Theorem 7.13. Fix two dyadic rational numbers $\varepsilon, \delta \in (0, 1)$. Let η, ω, and ρ be positive numbers with finite binary representation, such that

$$0 < \rho + 2\sqrt{(\rho + \frac{1}{2^\eta})\log{(1/\delta)}} + \frac{1}{2^\eta} < \varepsilon, \text{ and}$$

$$\log 1/\delta < (\omega - 1)^2 \rho.$$

Similarly to the proof of Theorem 7.9 we show a reduction from DISTINGUISH (ε, δ)-DP to GAP-CONCENTRATED-DP problem. We map each instance C of DISTINGUISH (ε, δ)-DP into an instance (C, ρ, ω, η).

As in the proof of Theorem 7.3 and Theorem 7.9 when C is a yes-instance of DISTINGUISH (ε, δ)-DP, C is also (ρ)-CDP. Therefore, (C, ρ, η) is a yes-instance of GAP-CONCENTRATED-TRUNCATED-DP.

In case of C is a no-instance of DISTINGUISH (ε, δ)-DP we need to show that then (C, ρ, ω, η) is a no-instance of GAP-CONCENTRATED-DP. By way of contraposition, suppose that for all neighboring inputs x, x', and for all $1 < \alpha < \omega$ holds $D_\alpha(C(x)\|C(x')) \le \rho\alpha + \frac{1}{2^\eta}$. Then, for $\rho' = \rho + \frac{1}{2^\eta}$, C is $(\alpha, \rho'\alpha)$-RDP for all $\alpha > 1$, as

$$D_\alpha(C(x)\|C(x')) \le \rho\alpha + \frac{1}{2^\eta} \le (\rho + \frac{1}{\alpha 2^\eta})\alpha \le (\rho + \frac{1}{2^\eta})\alpha.$$

Therefore, C is (ρ', ω)-TCDP. Hence by Theorem 7.12, C is $(\rho' + 2\sqrt{(\rho' \log{(1/\delta)}}, \delta)$-DP. But by our choice of the parameters, $\rho' + 2\sqrt{(\rho' \log{(1/\delta)}} = \rho\alpha + 2\sqrt{(\rho + \frac{1}{2^\eta})\log{(1/\delta)}} + \frac{1}{2^\eta} < \varepsilon$, hence C is (ε, δ)-DP. This means that P is not a no-instance of DISTINGUISH (ε, δ)-DP.

We showed that the reduction is correct. As the parameters ρ, ω, and η depend only on the constants eps, δ, overall we get a linear-time deterministic algorithm for such reduction. Hence, GAP-TRUNCATED-CONCENTRATED-DP is **PSPACE**-hard. $\square$