A Framework for Differential Privacy Against Timing Attacks

Zachary Ratliff* Salil Vadhan†

September 2024

Abstract

The standard definition of differential privacy (DP) ensures that a mechanism's *output* distribution on adjacent datasets is indistinguishable. However, real-world implementations of DP can, and often do, reveal information through their *runtime* distributions, making them susceptible to timing attacks. In this work, we establish a general framework for ensuring differential privacy in the presence of timing side channels. We define a new notion of *timing privacy*, which captures programs that remain differentially private to an adversary that observes the program's runtime in addition to the output. Our framework enables chaining together component programs that are *timing-stable* followed by a random delay to obtain DP programs that achieve timing privacy. Importantly, our definitions allow for measuring timing privacy and output privacy using different privacy measures. We illustrate how to instantiate our framework by giving programs for standard DP computations in the RAM and Word RAM models of computation. Furthermore, we show how our framework can be realized in code through a natural extension of the OpenDP Programming Framework.

*Harvard University & OpenDP. Email: zacharyratliff@g.harvard.edu. Supported in part by Cooperative Agreement CB20ADR0160001 with the Census Bureau, and in part by Salil Vadhan's Simons Investigator Award

†Harvard University & OpenDP. Email: salil_vadhan@harvard.edu

Contents

1 Introduction

The framework of differential privacy (DP) [DMNS06] is used extensively for computing privacy-preserving statistics over sensitive data. A differentially private algorithm has the property that *close* inputs map to *indistinguishable* output distributions. Here, "close" and "indistinguishable" are often given by various metrics and probability distance measures respectively (§2). For example, algorithms that compute dataset statistics commonly require that adding or removing any individual's data does not change the probability of outputting any given value by more than a constant factor. More formally, the definition of differential privacy is given as follows.

Definition 1 (Differential Privacy [DMNS06, DKM+06])**.** *Let $M : \mathcal{X} \to \mathcal{Y}$ be a randomized function. We say M is* (ε, δ)-differentially private *if for every pair of adjacent datasets x and x' and every subset $S \subseteq \mathcal{Y}$*

$$\Pr[M(x) \in S] \leq e^{\varepsilon} \cdot \Pr[M(x') \in S] + \delta$$

When $\delta = 0$ we say that M satisfies *pure* differential privacy. Intuitively, this property guarantees that the result of a statistical analysis is essentially the same regardless of the presence or absence of any individual's data. Notable examples of differential privacy in practice include the US Census Bureau [Abo18, MKA+08], Apple [G+16], Facebook [MDH+20], and Google [ABC+20] who have deployed DP to expand access to sensitive data while protecting individual privacy. DP has also been applied to problems that are not explicitly about privacy-preserving statistics such as private machine learning [ACG+16, PSM+], adaptive data analysis [BNS+16, DFH+15, NR18], anonymous messaging [LGZ18, TGL+17, VDHLZZ15], and distributed analytics [RNM+21, RZHP20].

Unfortunately, implementing differential privacy faithfully is a tricky business. Algorithms that are proven on paper to achieve differential privacy are eventually implemented and ran on hardware with various resource constraints. These constraints can lead to discrepancies between the computational model used in the privacy proof and the actual implementation, potentially invalidating the algorithm's privacy guarantees. For example, many mathematical proofs in the DP literature assume exact arithmetic over the real numbers while their implementations instead use finite-precision floating-point or integer arithmetic. Such inconsistencies have led to faulty implementations of textbook mechanisms that do not actually achieve differential privacy [CSVW22, JMRO22, Mir12], and prompted new research on differential privacy in finite models of computation [BV18, CKS20]. In a similar vein, real-world implementations of differential privacy are often exposed to observable *side channels* that are not modeled in the definition of differential privacy. For instance, in the online query setting, where an analyst submits a query and receives a differentially private response, the analyst observes not only the query's output but also the query's execution time. Leveraging the timing of operations to learn otherwise protected information is known as a *timing attack*, and prior work has demonstrated that such attacks can be used to violate differential privacy. For example, Haeberlen, Pierce, and Narayan showed how user-defined queries can be used to leak the presence of records in a sensitive dataset via their running time [HPN11]. More recently, Jin, McMurtry, Rubinstein, and Ohrimenko discovered that the runtime of discrete sampling algorithms (specifically, the Discrete Laplace [GRS12] and Discrete Gaussian [CKS20]) can reveal the magnitude of their sampled noise values [JMRO22]. Leaking the sampled noise value can result in a violation of DP since it can be used to remove the noise and hence the privacy protection from the output of the algorithm.

Despite some growing awareness of timing attacks on differential privacy, prior work has focused on ad hoc mitigation strategies that typically apply to a single class of differentially private

mechanisms[1] or database system [HPN11, BV18, AR23]. The most commonly suggested strategy has been to enforce constant-time program execution. However, enforcing constant-time execution inherently limits the amount of data that the program can process. The program must either truncate the input (e.g., by performing subsampling) or reveal an upper bound on the accepted dataset size. Constant-time programs also come with a significant performance cost, as fast computations must be padded to take the same amount of time as the slowest possible computation. However, constant-time program execution is overkill. It prevents *all* information leakage from the runtime, whereas differential privacy only requires that executions over neighboring inputs are indistinguishable.

To avoid enforcing constant-time program execution, Haeberlen et al. suggested making the program's runtime differentially private by adding random delay before returning the output [HPN11]. This approach mimics that of adding noise to the program's output to achieve differential privacy. However, this proposal is incomplete because it does not specify

(1) What it means for a program's runtime to be differentially private? and

(2) How much delay, and from what distribution, is necessary to achieve that definition of privacy?

New definitions and theorems are needed to precisely reason about the privacy guarantees of such programs in the presence of timing side channels.

1.1 Our Results

To reason about timing side channels, we need to reason about programs rather than functions. The behavior of programs includes both the program's output and runtime, which can depend on the program's execution environment. This might include, for example, the current state of cache memory or any resource contention caused by concurrent program execution (see Section 2.1 below for more on execution environments). For this reason, we explicitly generalize the definition of differential privacy to this setting.

Definition 2 ((ε, δ)-Differentially Private Programs)**.** *Let $\mathcal{E}$ be the set of possible execution environments for a fixed computational model and let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a randomized program. We say P is* (ε, δ)-differentially private *if for every pair of adjacent datasets x and x', every pair of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, and every subset $S \subseteq \mathcal{Y}$*

$$\Pr[\texttt{out}(P(x, \texttt{env})) \in S] \leq e^{\varepsilon} \cdot \Pr[\texttt{out}(P(x', \texttt{env}')) \in S] + \delta$$

where $\texttt{out}(P(x, \texttt{env}))$ indicates P's output value $y \in \mathcal{Y}$.

Given this more generalized definition, we define a new notion of privacy with respect to timing attacks. Specifically, our definition asks that the running time of a program is differentially private *conditioned on the output of the program.* For the special case of pure DP, our definition is as follows:

Definition 3 (ε-Timing Privacy, special case of Def. 31)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a (possibly) randomized program. Then we say that P is* ε-timing-private *if for all adjacent $x, x' \in \mathcal{X}$, all*

[1]The differential privacy literature uses the word "mechanism" to refer to a randomized algorithm. Since our work focuses primarily on actual implementations of such mechanisms within a computational model, we will instead use the term "programs" throughout the paper.

pairs of input-compatible execution environments $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$*, all* $y \in$ *supp*$(\mathtt{out}(P(x, \mathtt{env}))) \cap$ *supp*$(\mathtt{out}(P(x', \mathtt{env}')))$*, and all* $S \subseteq \mathcal{T}$

$$\Pr[T_P(x, \mathtt{env}) \in S | \mathtt{out}(P(x, \mathtt{env})) = y] \leq e^{\varepsilon} \cdot \Pr[T_P(x', \mathtt{env}') \in S | \mathtt{out}(P(x', \mathtt{env}')) = y]$$

where $T_P(x, \mathtt{env})$ *denotes the running time of* P *on input* x *in execution environment* $\mathtt{env}$*,* $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ *represents the units in which we measure time (e.g.,* $\mathcal{T} = \mathbb{N}$ *if we count the number of executed instructions), and* $\mathtt{out}(P(x, \mathtt{env}))$ *indicates the program's output.*

Intuitively, this definitional approach provides several benefits, one of which is that it enables measuring timing privacy using a different privacy measure than that used to measure output privacy. For example, one can be measured by approximate DP and the other with Rényi DP. This feature is useful since one generally compromises on utility to achieve output privacy, while one can instead compromise on efficiency (longer runtimes) to achieve timing privacy. In contrast, a previous definition of Ben Dov, David, Naor, and Tzalik [BDDNT23] (which we rename[2]) couples the leakage of the output and the runtime:

Definition 4 ((ε, δ)-Joint Output/Timing Privacy [BDDNT23], special case of Def. 34)**.** *Let* $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ *be a (possibly) randomized program. Then we say that* P *is* ε-jointly output/timing-private *if for all adjacent* $x, x' \in \mathcal{X}$*, all pairs of input-compatible execution environments* $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$*, and for all* $S \subseteq \mathcal{Y} \times \mathcal{T}$

$$\Pr[(\mathtt{out}(P(x, \mathtt{env})), T_P(x, \mathtt{env})) \in S] \leq e^{\varepsilon} \cdot \Pr[(\mathtt{out}(P(x', \mathtt{env}')), T_P(x', \mathtt{env}')) \in S] + \delta$$

This is the standard definition of DP applied to the joint random variable of the program's output and running time. We show that in the case of pure-DP ($\delta = 0$), joint output/timing privacy is equivalent to output privacy together with our definition of timing privacy up to a constant factor in ε. More generally, our definition together with output privacy implies joint output/timing privacy when both output and timing privacy are measured using the same privacy measure (Lemma 36).

We establish a general framework for chaining together *timing-stable programs* (programs where close inputs map to close runtime distributions) with *timing-private delay programs* (programs that add random delay before releasing their output) such that the entire execution is *timing-private*. DP software libraries such as OpenDP [GHV20, SVM+] and Tumult Core [BBD+22] support the concatenation of multiple data operations, such as clamping, summation, and adding noise from a distribution, into a unified mechanism. Our framework enables examining how adjacent datasets impact the execution time of modular components within such a chain. A single timing delay can then be applied before returning the mechanism's output to safeguard the entire chain against timing attacks.

To illustrate how our framework can be instantiated, we provide concrete examples of RAM and Word RAM programs that satisfy our notion of timing privacy when we treat each RAM instruction as a single time step. These include mechanisms for computing a randomized response, private sums, and private means (Section 6). Notably, our mechanisms are often much more efficient than constant-time constructions. We also describe mechanisms that achieve timing privacy in the unbounded DP setting and have accuracy that matches their non-timing-private counterparts.

[2]Ben Dov et al. described mechanisms that meet this criteria as *differentially private time oblivious mechanisms*. We will instead use the more informative term *jointly output/timing-private*.

To show the compatibility of our framework with existing libraries for differential privacy, we implement a proof-of-concept timing-privacy framework within the OpenDP library [SVM+]. Our implementation supports the tracking of timing stability across dataset transformations, and includes an implementation of a timing-private delay program for delaying the output release to achieve timing-privacy. This proof-of-concept is only meant to show how our framework can be easily implemented on top of existing frameworks, not to provide actual guarantees of privacy against an adversary who can measure physical time, which we leave as an important direction for future work.

1.2 Future Work

We have shown how one can rigorously reason about timing privacy in idealized computational models (like the RAM and Word-RAM models) where every instruction is assumed to take the same amount of time. On physical CPUs, this assumption does not hold and the varying time for different operations has been used as the basis for past timing attacks [AKM+15]. Nevertheless, we believe that our *framework* can be fruitfully applied to physical computations with suitably constrained execution environments. Importantly, our notion of timing stability does not require precise estimates of computing time, but only an *upper bound* on the time difference that one individual's data can make on a computation, which seems feasible to estimate in practice.

Another direction for future work is to design timing-private and timing-stable programs for a wider array of mechanisms and transformations from the differential privacy literature. Some examples of interest would be the Discrete Gaussian Mechanism [CKS20] and forms of the Exponential Mechanism [MT07], as well providing user-level privacy on datasets where a user may contribute many records, and node-level privacy on graphs. Getting timing privacy for DP algorithms that involve superlinear-time operations such as sorting (e.g. approximate medians via the exponential mechanism [MT07]) or pairwise comparisons (e.g. the DP Theil-Sen linear regression algorithm [AMS+22]) are also an intriguing challenge, because timing-stable programs must have $O(n)$ runtime (Lemma 30).

Finally, from a theoretical point of view, it is interesting to explore the extent to which *pure* timing privacy is achievable. Our examples of timing-private programs mostly come from adding random delays to timing-stable programs, which seems to inherently yield approximate DP (i.e. $\delta > 0$). Ben-Dov et al. [BDDNT23] show some limitations of what can be achieved with pure DP, but their result does not rule out many useful DP computations (such as a RAM program that randomly samples $O(1)$ records from a dataset and then carries out a constant-time DP computation on the subsample, which achieves perfect timing privacy even on unbounded-sized datasets).

2 Preliminaries

2.1 Models of Computation

The runtime of a program is determined by its computational model. For instance, within the idealized RAM model, runtime is typically measured by the number of executed instructions until halting. Conversely, for a C program running on an x86 architecture, runtime might be quantified in CPU cycles, though the same program on identical hardware can exhibit varied runtimes due to differences in microcode patches [Sta03]. Moreover, runtime can be affected by non-deterministic factors like cache interference from concurrent executions on modern CPUs. Thus, we allow a program's output and runtime to depend on an execution environment $\texttt{env} \in \mathcal{E}$, where an execution environment is defined to be any state stored by the computational model that affects program

execution (e.g., the contents of uninitialized memory, the program's input, the initial values stored in CPU registers, etc.). We consider the program's input to be part of the program's execution environment and say that $\text{supp}(P(x, \texttt{env})) = \emptyset$ if the input and environment are *incompatible*.

Definition 5 (Input/Environment Compatibility)**.** *Our computational models come with an input/environment compatibility relation, and when an environment* $\texttt{env} \in \mathcal{E}$ *is incompatible with an input* $x \in \mathcal{X}$ *for program* P*, we define supp*$(P(x, \texttt{env})) = \emptyset$*.*

The compatibility of a given $(x, \texttt{env})$ pair depends on the computational model. For example, the calling convention for C functions on modern x86 hardware require the EBP and ESP registers, which indicate the base and top of the stack, to specify the function's stack frame in memory. If these registers point to conflicting memory locations or do not contain the correct setup for the function's input arguments, the execution environment would be incompatible with the function's input. We give further examples of such incompatible input/environment pairs below when discussing the RAM model below.

Definition 6 (Program Execution)**.** *A randomized program* $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ *takes in an input* $x \in \mathcal{X}$*, and an execution environment* $\texttt{env} \in \mathcal{E}$*, and outputs a value* $y \in \mathcal{Y}$ *denoted by the random variable* $\texttt{out}(P(x, \texttt{env}))$*. The execution of* P *may also induce changes on the environment, and we indicate the new state of the environment as the random variable* $\texttt{outenv}(P(x, \texttt{env}))$*.*

Definition 7 (Program Runtime)**.** *Suppose that our computational model's runtime is measured in units* $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ *(e.g.,* $\mathcal{T} = \mathbb{N}$ *if runtime measures the number of executed instructions). Then we denote the* runtime *of program* $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ *within execution environment* $\texttt{env}$ *on input* x *to be the random variable* $T_P(x, \texttt{env}) \in \mathcal{T}$*.*

We can drop the environment $\texttt{env}$ from our notation when a program's output and runtime is independent of the environment. We call such programs *pure* to follow standard programming languages terminology, but not to be confused with pure DP.

Definition 8 (Output-Pure Programs)**.** *A program* $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ *is* output-pure *if there exists a (possibly randomized) function* $f : \mathcal{X} \to \mathcal{Y}$ *such that for all* $x \in \mathcal{X}$ *and all input-compatible* $\texttt{env} \in \mathcal{E}$*,* $f(x)$ *is identically distributed to* $\texttt{out}(P(x, \texttt{env}))$*.*

Definition 9 (Timing-Pure Programs)**.** *A program* $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ *is* timing-pure *if there exists a (possibly randomized) function* $f : \mathcal{X} \to \mathcal{T}$ *such that for all* $x \in \mathcal{X}$*, and all input-compatible* $\texttt{env} \in \mathcal{E}$*,* $f(x)$ *is identically distributed to* $T_P(x, \texttt{env})$*.*

We remark that the standard definition of differential privacy implicitly assumes that programs are *output-pure*, and in fact, its composition and post-processing theorems fail if that is not the case.

RAM Model. In this work, we will use the idealized RAM and Word RAM models of computation to give proof-of-concept results on how our framework can be used. Applying our framework to physical hardware implementations is an important challenge for future work. The RAM model includes an infinite sequence of memory cells, each capable of holding an arbitrarily large natural number. The model stores variables in registers and supports a set of basic operations for performing arithmetic, logical, and memory operations. Programs also include instructions for conditional jumps (**if** CONDITIONAL **goto** LINE) which allow simulation of the standard **if**, **else**, and **while** expressions which we freely use.[3] Additionally, many of our RAM programs are randomized and use

[3]We define condition matching in branching expressions to take 1 instruction.

a $\texttt{RAND}(n)$ instruction to sample a uniform random integer in $\{0, \ldots, n\}$. We define the runtime of a RAM program to be the number of basic instructions executed before halting, which is indicated by a `HALT` command. That is, the space of runtime values $\mathcal{T}$ is equal to the natural numbers $\mathbb{N}$. The model includes a built-in variable `input_len` that indicates the length $|x|$ of the program's input x, as well as a built-in variable `input_ptr` that indicates where in memory the input resides. Additionally, every program writes their output to some location in memory indexed by the variable `output_ptr` and sets a variable `output_len` to indicate the length of the output.

RAM programs should not make assumptions about the values loaded in memory locations outside the locations where the input resides, and are responsible for initializing any such memory that they use. This specification aligns with the behavior observed in, for instance, C programs, wherein it cannot be presumed that newly allocated memory through a `malloc`[4] instruction is initialized to zero. We remark that this specification introduces an element of non-determinism to our model since a program may read and use uninitialized memory values. The space of environments $\mathcal{E}$ for a RAM program is therefore the set of possible memory configurations as well as the initial values stored by the built-in variables `input_ptr`, `input_len`, `output_ptr`, and `output_len`. Some environments may not be relevant for a given input, e.g., the environment corresponding to all memory locations being uninitialized is not possible for programs whose input lengths are greater than zero. Similarly, compatible $(x, \texttt{env})$ pairs have the property that the memory locations of `env` indicated by the variables `input_ptr` and `input_len` should contain the input x. For all such incompatible pairs $(x, \texttt{env})$, $P(x, \texttt{env})$ is undefined and so we say that $\text{supp}(P(x, \texttt{env})) = \emptyset$ in Definition 3 and elsewhere. Finally, most of our RAM programs are *pure* and will not interact with uninitialized memory, and therefore both the program's output and running time will be independent of its execution environment for all compatible $(x, \texttt{env})$ pairs.

Definition 10 (RAM Environment)**.** *The* environment `env` *of a RAM program is the infinite sequence* $(v_0, v_1, \ldots,)$ *such that* $M[i] = v_i$ *for all* i *along with the values stored by the built-in variables* ***`input_ptr`****,* ***`input_len`****,* ***`output_ptr`****, and* ***`output_len`****.*

Remark 11. *A program's environment includes the program's input* x *by definition, and similarly, a program's output environment* $\texttt{outenv}(P(x, \texttt{env}))$ *includes the program's output* $\texttt{out}(P(x, \texttt{env}))$*.*

Word RAM. We will use a more realistic ω-bit Word RAM model for describing programs that operate on bounded-length values. The word size ω corresponds to the bitlength of values held in memory and variables, effectively capping the total addressable memory to 2^ω. This limitation arises because memory access, denoted by $M[\texttt{var}]$, relies on the size of `var`, which cannot exceed $2^\omega - 1$. We fix the word size ω up front which therefore bounds the worst-case dataset size. Therefore the model does not allow inputs to grow unboundedly. This constraint is principally driven by the fact that computers in the real world are equipped with finite memory and a pre-determined word size. Furthermore, allowing the word size to vary with the length of the input (e.g., it is standard in the algorithms literature $\omega = \Theta(\log n)$ and allow $n \to \infty$) introduces an additional side channel within the computational framework. In particular, the output itself is given as a sequence of ω-bit words, which reveals information about the input length. For these reasons, we treat the word size as a separate public parameter which implies an upper bound on the input length.

[4] `malloc` is the standard C library function for dynamic memory allocation.

2.2 Datasets, Distance Metrics, and Privacy Measures

Differential privacy (and timing privacy, Definition 3) is defined with respect to a dataset space $\mathcal{X}$. Typically, datasets consist of $n \geq 0$ records drawn from a row-domain $\mathcal{D}$, i.e., $\mathcal{X} = \bigcup_{n=0}^{\infty} \mathcal{D}^n$. We take elements of $\mathcal{D}$ as consisting of an individual's data. As such, $\mathcal{D}$ often consists of d-dimensional entries, for example, in tabular data where each individual has d attributes. Thus, in the RAM model we take $\mathcal{D} = \mathbb{N}^d$ and in the Word RAM model $\mathcal{D} = \{0, \ldots, 2^\omega - 1\}^d$. Since input lengths are bounded by 2^ω in Word RAM, we only consider datasets of size at most some $n_{\max} < 2^{\omega/d}$.

Following the OpenDP Programming Framework [GHV20], which is also the basis of Tumult Core [BBD+22], our timing privacy and stability definitions work with arbitrary metrics on input data(sets) and arbitrary measures of privacy. The dataset metrics are arbitrary, user-defined metrics that are not required to satisfy the standard properties of non-negativity, symmetry, and triangle inequality as in the standard mathematical definition of a metric. Two common choices for dataset distance metrics are the *Hamming distance* and *insert-delete distance*.

Definition 12 (Hamming Distance)**.** *Let $x, x' \in \mathcal{D}^*$ be datasets. The* Hamming distance *denoted d_{Ham} of x and x' is*

$$d_{Ham}(x, x') = \begin{cases} \#\{i : x_i \neq x'_i\} & \text{if } |x| = |x'| \\ \infty & \text{otherwise} \end{cases}$$

Definition 13 (Insert-Delete Distance)**.** *For $x \in \mathcal{D}^*$, an insertion to x is an addition of an element z to a location in x resulting in a new dataset $x' = [x_1, \ldots, x_i, z, x_{i+1}, \ldots, x_n]$. Likewise, a deletion from x is the removal of an element from some location within x, resulting in a new dataset $x' = [x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$. The* insert-delete distance *denoted d_{ID} of datasets $x, x' \in \mathcal{D}^*$ is the minimum number of insertion and deletion operations needed to change x into x'.*

We focus our attention on ordered distance metrics since the running time of a program often depends on the ordering of its input data, e.g., if the algorithm starts by sorting its input.

We say that two datasets x and x' are *adjacent* with respect to dataset distance metric $d_{\mathcal{X}}$ if $d_{\mathcal{X}}(x, x') \leq 1$. When $d_{\mathcal{X}} = d_{\text{Ham}}$, this notion of adjacency captures "bounded differential privacy," where the dataset size n is known and public. When $d_{\mathcal{X}} = d_{\text{ID}}$, it captures for "unbounded differential privacy," where the dataset size itself may be unknown. We will also use what we call "upper-bounded differential privacy," where we assume a known and public upper bound $n_{\max}$ on the dataset size, i.e., $\mathcal{X} = \mathcal{D}^{\leq n_{\max}} = \bigcup_{n=0}^{n_{\max}} \mathcal{D}^n$ with metric $d_{\mathcal{X}} = d_{\text{ID}}$ so that the exact dataset size is unknown and needs to be protected with DP. This, for example, arises in the ω-bit Word RAM model of computation where $n_{\max} \leq 2^{\omega/d}$ for d-dimensional datasets. We note that these metrics are appropriate for tabular datasets where each record corresponds to one individual's data. Our framework can also be instantiated for other data domains and metrics (e.g. user-level DP in a dataset of events or graph DP with node privacy), but we leave designing algorithms for these settings as future work.

A core feature of several DP systems and libraries is the ability to combine various component functions into more complex differentially private mechanisms [McS09, HPN11, GHV20]. Analyzing the *stability* of the various component functions is a useful method for understanding the privacy implications of arbitrarily combining the functions [McS09]. The stability of a deterministic program characterizes the relationship between its input and output distances. Executing the program on "close" inputs will result in outputs that are also "close." To define stability for randomized programs, we'll need the notion of couplings.

Definition 14 (Coupling)**.** *A* coupling *of two random variables r and r' taking values in sets $\mathcal{R}$ and $\mathcal{R}'$ respectively, is a joint random variable $(\tilde{r}, \tilde{r}')$ taking values in $\mathcal{R} \times \mathcal{R}'$ such that $\tilde{r}$ has the same marginal distribution as r and $\tilde{r}'$ has the same marginal distribution as r'.*

Couplings allow us to extend the notion of stability to randomized programs by measuring stability in terms of distributions rather than fixed outcomes. Specifically, the stability of a randomized program can be interpreted as a worst-case analogue of the Wasserstein[5] distance between the program's output distributions on close inputs.

We now generalize the notion of stability to programs.

Definition 15 (Output-Stable Programs)**.** *A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_{in} \mapsto d_{out})$*-output stable *with respect to input metric $d_{\mathcal{X}}$ and output metric $d_{\mathcal{Y}}$ if $\forall x, x' \in \mathcal{X}$ such that $d_{\mathcal{X}}(x, x') \leq d_{in}$, and all pairs of input-compatible $\texttt{env}, \texttt{env}' \in \mathcal{E}$, there exists a coupling $(\tilde{y}, \tilde{y}')$ of the random variables $(\texttt{out}(P(x, \texttt{env})), \texttt{out}(P(x', \texttt{env}')))$ such that $d_{\mathcal{Y}}(\tilde{y}, \tilde{y}') \leq d_{out}$ with probability 1.*

As an example, consider an output-pure program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ that takes an input dataset $x \in \{0, 1\}^n$ and outputs the sum $y = \sum_{i=1}^{n} x_i$. Such a program would be $(1 \mapsto 1)$-*output stable* under the input distance metric d_{Ham} and output distance metric $d_{\mathbb{N}}(y, y') = |y - y'|$, since changing a row's value in the input can affect the output sum by at most 1. The program P can therefore be made differentially private (Definition 21) by chaining it with another program that adds Laplace noise with scale $1/\varepsilon$ to the output (see Section 6 for a formal definition of program chaining and Lemma 54 for a RAM program that computes a DP sum).

We now introduce *privacy measures* which are arbitrary distances between probability distributions.

Definition 16 (Generalized Privacy Measures)**.** *A* privacy measure *is a tuple $(\mathcal{M}, \leq, M)$ where $(\mathcal{M}, \leq)$ is a partially-ordered set, and M is a mapping of two random variables X and X' over the same measurable space to an element $M(X, X') \in \mathcal{M}$ satisfying:*

1. *(*Post-processing*). For every function g, $M(g(X), g(X')) \leq M(X, X')$*
2. *(*Joint convexity*). For any collection of random variables $(X_i, X_i')_{i \in \mathcal{I}}$ and a random variable $I \sim \mathcal{I}$, if $M(X_i, X_i') \leq c$ for all i, then $M(X_I, X_I') \leq c$.*

We will frequently refer to a privacy measure only by the mapping M, where M and $\leq$ are implicit. Some useful examples of privacy measures are *pure* and *approximate* DP which use the *max-divergence* and *smoothed max-divergence* as their respective distance mappings.

Definition 17 (Max-Divergence)**.** *The* max-divergence *between two random variables Y and Z taking values from the same domain is defined to be:*

$$D_\infty(Y||Z) = \max_{S \subseteq supp(Y)} \left[\ln \frac{\Pr[Y \in S]}{\Pr[Z \in S]} \right]$$

Lemma 18. *A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is ε-differentially private if and only if for every pair of adjacent datasets x and x', and every pair of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, $D_\infty(\texttt{out}(P(x, \texttt{env}))||\texttt{out}(P(x', \texttt{env}'))) \leq \varepsilon$ and $D_\infty(\texttt{out}(P(x', \texttt{env}'))||\texttt{out}(P(x, \texttt{env}))) \leq \varepsilon$.*

[5]The Wasserstein distance between probability distributions Q and S on metric space $(\mathcal{Y}, d_{\mathcal{Y}})$ is defined as:

$$W_1(Q, S) = \inf_{\gamma \in \Gamma} \mathbb{E}_{(q,s) \sim \gamma} d_{\mathcal{Y}}(q, s)$$

where Γ is the set of all couplings of Q and S.

Definition 19 (Smoothed Max-Divergence)**.** *The* smoothed max-divergence *between Y and Z is defined to be:*

$$D_\infty^\delta(Y||Z) = \max_{S \subseteq supp(Y): \Pr[Y \in S] \geq \delta} \left[\ln \frac{\Pr[Y \in S] - \delta}{\Pr[Z \in S]} \right]$$

Lemma 20. *A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is (ϵ, δ)-differentially private if and only if for every pair of adjacent datasets x and x', and every pair of input-compatible execution environments $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$,*

$$D_\infty^\delta(\mathtt{out}(P(x, \mathtt{env}))||\mathtt{out}(P(x', \mathtt{env}'))) \leq \varepsilon$$

and

$$D_\infty^\delta(\mathtt{out}(P(x', \mathtt{env}'))||\mathtt{out}(P(x, \mathtt{env}))) \leq \varepsilon$$

Other examples of privacy measures that can be used in our framework are concentrated DP [DR16, BS16], Rényi DP [Mir17], and f-DP [DRS22]. We give a general definition of differential privacy for arbitrary input metrics and privacy measures.

Definition 21. *A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_{in} \mapsto d_{out})$-differentially private with respect to input metric $d_{\mathcal{X}}$ and privacy measure M if for every pair of datasets $x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$ and every pair of input-compatible execution environments $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$, $\mathtt{out}(P(x', \mathtt{env}')) \leq d_{out}$.*

Definition 2 is a special case of Definition 21 by replacing setting the privacy measure M to be the smoothed max-divergence (Definition 19), $d_{out} = (\varepsilon, \delta)$, $d_{in} = 1$, and $d_{\mathcal{X}}$ is a dataset distance metric such as d_{ID} or d_{Ham}.

Finally, we remark that (ε, δ)-differentially private algorithms have the following properties.

Lemma 22 (Post-processing [DMNS06])**.** *Let $M : \mathcal{X} \to \mathcal{Y}$ be an (ε, δ)-differentially private function and $f : \mathcal{Y} \to \mathcal{Z}$ be a (possibly) randomized function. Then $f \circ M : \mathcal{X} \to \mathcal{Z}$ is (ε, δ)-differentially private.*

Lemma 23 (Composition [DKM+06])**.** *Let $M_1 : \mathcal{X} \to \mathcal{Y}_1$ be a $(\varepsilon_1, \delta_1)$-differentially-private function and $M_2 : \mathcal{X} \to \mathcal{Y}_2$ be $(\varepsilon_2, \delta_2)$-differentially-private function. Then $(M_1 \otimes M_2)(x) = (M_1(x), M_2(x))$ is a $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-differentially-private function.*

3 Timing-Stable Programs

In this section, we introduce new stability definitions for program runtime. Analogous to how global sensitivity determines an amount of noise that suffices for achieving differential privacy in randomized functions, we present a similar notion that determines an added runtime delay that suffices for ensuring timing privacy in randomized programs.

3.1 Timing Stability

Definition 24 (Timing Stability)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a (possibly randomized) program and $d_{\mathcal{X}}$ a metric on $\mathcal{X}$. Then we say that P is* $(d_{in} \mapsto t_{out})$-timing-stable *with respect to $d_{\mathcal{X}}$ if $\forall x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, $\forall \mathtt{env}, \mathtt{env}' \in \mathcal{E}$, $\exists$ a coupling $(\tilde{r}, \tilde{r}')$ of $T_P(x, \mathtt{env})$ and $T_P(x', \mathtt{env}')$ such that $|\tilde{r} - \tilde{r}'| \leq t_{out}$ with probability 1.*

Program 1
A RAM program for randomized response.

Input: A bit $x \in \{0, 1\}$ at memory location $M[\texttt{input\_ptr}]$
Output: A uniform random bit

```
1: x = M[input_ptr];
2: output_len = 1;
3: output_ptr = input_ptr;
4: b = RAND(1); {flip coin}
5: if b == 1 then
6:     M[output_ptr] = 1 − x; {flip bit}
7: HALT;
```

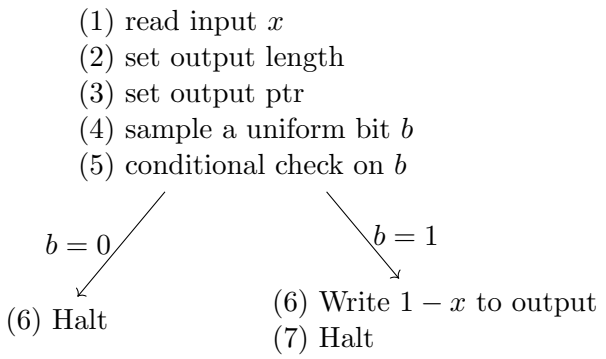

Figure 1: A Boolean randomized response RAM program and its corresponding execution tree. The program executes 1 additional instruction when it outputs $1 - x$ versus x.

Programs that exhibit timing stability ensure that changes in their inputs only cause bounded changes in their runtime distributions. However, this property is less useful for addressing privacy concerns when the program's output is also made available. For instance, consider the execution tree of a program that returns a randomized response to a Boolean input, as shown in Figure 1. Each leaf in the tree signifies an output, with the depth of the leaf indicating its runtime. An analysis of the tree reveals that the runtime does not depend on the input bit x. Programs with such input-independent runtime are categorized as 0-timing-stable (see Appendix, Lemma 70 for proof). Furthermore, the program outputs an independent Bernoulli random variable with $p = 1/2$ and therefore achieves 0-DP. This might suggest that the program cannot leak any information about the input. However, the runtime exposes the program's internal randomness, indicating whether the output is x or $1 - x$. Thus, the input bit can be precisely determined by the combination of the program's output and runtime, despite the program being both 0-timing stable and 0-DP. This demonstrates the need for a refined definition of timing stability that considers the program's output, which we provide in the next section.

3.2 Output-Conditional Timing Stability

Definition 25 (Output-Conditional Timing Stability)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be (possibly randomized) program and $d_{\mathcal{X}}$ a metric on $\mathcal{X}$. Then we say that P is $(d_{in} \mapsto t_{out})$*-OC timing-stable *with respect to $d_{\mathcal{X}}$ if $\forall x, x'$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, $\forall$ environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, and $\forall y \in supp(P(x, \texttt{env})) \cap supp(P(x', \texttt{env}'))$, there exists a coupling $(\tilde{r}, \tilde{r}')$ of the conditional runtime random variables $T_P(x, \texttt{env})|_{\texttt{out}(P(x,\texttt{env}))=y}$ and $T_P(x', \texttt{env}')|_{\texttt{out}(P(x',\texttt{env}'))=y}$ such that $|\tilde{r} - \tilde{r}'| \leq t_{out}$ with probability* 1*.*

Output-conditional timing stability imposes stricter requirements than basic timing stability when the support sets of $\texttt{out}(P(x, \texttt{env}))$ and $\texttt{out}(P(x', \texttt{env}'))$ are similar. For any output y that has a non-zero probability of being produced by P for inputs x and x', as is implied if P's output is differentially private (e.g., for both pure and Rényi DP, the supports must be identical), output-conditional timing stability ensures that the conditional runtime distributions $T_P(x, \texttt{env})|_{\texttt{out}(P(x,\texttt{env}))=y}$ and $T_P(x', \texttt{env}')|_{\texttt{out}(P(x',\texttt{env}'))=y}$ are close.

We now analyze the program depicted in Figure 1 through the perspective of output-conditional timing stability.

Lemma 26. *The Boolean randomized response program P shown in Figure 1 is* $(1 \mapsto 1)$-OC-timing stable *with respect to the Hamming distance metric* d_{Ham}.

Proof. $T_P(0, \texttt{env}_0)$ is always 6 conditioned on $\texttt{out}(P(0, \texttt{env}_0)) = 0$, and $T_P(1, \texttt{env}_1)$ is always 7 conditioned on $\texttt{out}(P(1, \texttt{env}_1)) = 0$. Thus a point coupling of the conditional random variables $T_P(0, \texttt{env})|_{\texttt{out}(P(0,\texttt{env}_0))=0}$ and $T_P(1, \texttt{env}_1)|_{\texttt{out}(P(1,\texttt{env}_1))=0}$) as $(\tilde{r}, \tilde{r}') = (6, 7)$ satisfies $\Pr[|\tilde{r} - \tilde{r}'| \leq 1] = 1$. Similarly, we can take the point coupling of $(T_P(0, \texttt{env}_0)|_{\texttt{out}(P(0,\texttt{env}_0))=1}, T_P(1, \texttt{env}_1)|_{\texttt{out}(P(1,\texttt{env}_1))=1})$ as $(\tilde{r}, \tilde{r}') = (7, 6)$ which also satisfies $\Pr[|\tilde{r} - \tilde{r}'| \leq 1] = 1$. □

Constant-time programs are $(d_{in} \mapsto 0)$-OC timing stable for all d_{in} (Lemma 73, proof in Appendix). Additionally, any program that has deterministic output and is $(d_{in} \mapsto t_{out})$-timing-stable is also $(d_{in} \mapsto t_{out})$-OC timing stable (Lemma 72, proof in Appendix).

3.3 Jointly-Output/Timing Stable Programs

We consider another natural definition of runtime stability, called *joint output/timing stability*, which considers the joint distribution of a program's output and its runtime. This notion becomes particularly useful when chaining programs together to create more complex functionality (§6). In particular, joint output/timing stability ensures that close inputs simultaneously produce close outputs and runtimes. This property becomes useful when reasoning about the output-conditional timing stability of programs whose inputs come from the output of another program (§6). In particular, an output-conditional timing-stable program P_2, bounds the change in runtime conditioned on a given output value for all d_{in}-close inputs. However, when constructing a chained program $P_2 \circ P_1$, where P_2 receives its input from the output of P_1, it's necessary to jointly bound the output and runtime stability of P_1 to guarantee output-conditional timing stability for the chained program $P_2 \circ P_1$.

Definition 27 (Joint Output/Timing Stability)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a program and $d_{\mathcal{X}}$ a metric on $\mathcal{X}$. Then we say that P is* $(d_{in} \mapsto \{d_{out}, t_{out}\})$-jointly output/timing stable *with respect to distance metrics $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$, if $\forall x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, and all pairs of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, $\exists$ a coupling $((\tilde{u}, \tilde{v}), (\tilde{u}', \tilde{v}'))$ of $(\texttt{out}(P(x, \texttt{env})), T_P(x, \texttt{env}))$ and $(\texttt{out}(P(x', \texttt{env}')), T_P(x', \texttt{env}'))$ such that $d_{\mathcal{Y}}(\tilde{u}, \tilde{u}') \leq d_{out}$ and $|\tilde{v} - \tilde{v}'| \leq t_{out}$ with probability 1.*

For programs with deterministic[6] outputs, if the program simultaneously satisfies $(d_1 \mapsto d_2)$-output stability and $(d_1 \mapsto t_1)$-timing stability, then the program will also satisfy $(d_1 \mapsto \{d_2, t_1\})$-joint output/timing stability.

Lemma 28. *If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is a deterministic (in its output) program that is $(d_1 \mapsto d_2)$ output stable and $(d_1 \mapsto t_1)$-timing stable, then P is $(d_1 \mapsto \{d_2, t_1\})$-jointly output/timing stable.*

Proof. Let $y = \texttt{out}(P(x, \texttt{env}))$ and $y' = \texttt{out}(P(x', \texttt{env}'))$ and choose the coupling to be $((y, r), (y', r'))$ where (r, r') is sampled from the coupling $(\tilde{r}, \tilde{r}')$ associated with the timing stability of P. Then $d_{\mathcal{Y}}(y, y') \leq d_2$ by output stability and $|r - r'| \leq t_1$ by timing stability. Since P is deterministic, it follows that (y, r) and (y', r') are identically distributed to $(\texttt{out}(P(x, \texttt{env})), T_P(x, \texttt{env}))$ and $(\texttt{out}(P(x', \texttt{env}')), T_P(x', \texttt{env}'))$ respectively and the claim is satisfied. □

[6] A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is *deterministic in its output* if for all $x \in \mathcal{X}$, $\exists y \in \mathcal{Y}$, such that for all input-compatible $\texttt{env} \in \mathcal{E}$, $\Pr[\texttt{out}(P(x, \texttt{env})) = y] = 1$. Therefore, deterministic programs are also output-pure programs.

Program 2 Sum Program

Input: A dataset $x \in \{0, \ldots, \Delta\}^n$ located at $M[\texttt{input\_ptr}], \ldots, M[\texttt{input\_ptr} + \texttt{input\_len} - 1]$. We require that $\Delta < 2^\omega$ and $n \leq 2^\omega - 1$ in the ω-bit Word RAM model.

Output: $\sum M[i]$ for $i = \texttt{input\_ptr}, \ldots, (\texttt{input\_ptr} + \texttt{input\_len} - 1)$. The program outputs $\min\{\sum M[i], 2^\omega - 1\}$ in the Word RAM model.

```
1: output_len = 1;
2: idx = input_ptr;
3: n = input_ptr + input_len;
4: sum = 0;
5: while idx < n do
6:    sum = M[idx] + sum;
7:    idx = idx + 1;
8: output_ptr = 0;
9: M[output_ptr] = sum;
10: HALT;
```

A useful example of a jointly output/timing stable RAM program is one that sums over the dataset.

Lemma 29. *The Sum Word RAM program $P : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ (Program 2) is $(1 \mapsto \{\Delta, 3\})$-jointly output/timing stable under the insert-delete input distance metric d_{ID} and the output distance metric $d_{\mathbb{N}}$ defined as $d_{\mathbb{N}}(y, y') = |y - y'|$.*

Proof. Inserting or deleting an input record changes the runtime by one loop iteration, which consists of 3 instructions. Therefore the program is $(1 \mapsto 3)$-timing stable under the input distance metric d_{ID}. Additionally, the program is $(1 \mapsto \Delta)$-output stable under d_{ID} since adding or removing an input record changes the sum by at most Δ (the maximum value of an input record). Since the program is deterministic in its output, the program satisfies $(1 \mapsto \{\Delta, 3\})$-joint output/timing stability (Lemma 28). The proof follows similarly if P is instead a RAM program. □

We now show that the class of timing-stable programs is restricted to programs that have a worst-case runtime that is at most linear in their input length. This implies that certain programs, e.g., those that perform superlinear-time sorting, cannot achieve timing stability on inputs of unbounded length. It is an interesting challenge for future work to see if non-trivial timing-private programs can be designed that incorporate such operations.

Lemma 30 (Timing-stable programs have $O(n)$ runtime)**.** *Let $\mathcal{X} = \mathcal{D}^*$ for a row-domain $\mathcal{D}$. A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ on inputs of unbounded length that is $(d_{in} \mapsto t_{out})$-timing-stable with respect to distance metric $d_{\mathcal{X}}(x, x') = ||x| - |x'||$ has linear worst-case runtime. Specifically, for all x and all $t \in \mathcal{T}$, we have*

$$\Pr[T_P(x) \leq |x| \cdot t_{out} + t_0] \geq \Pr[T_P(\lambda) \leq t_0]$$

where λ is the empty dataset of size 0.

Note that for every $p < 1$, there exists a t_0 such that $\Pr[T_P(\lambda) \leq t_0] > p$. Therefore the above lemma says that, with high probability, the runtime is at most linear.

Proof. Consider a sequence of datasets $(x_0, x_1, \ldots, x_n)$ such that $d_{\mathcal{X}}(x_i, x_{i+1}) \leq d_{in}$ and $x_0 = \lambda$. Then for all x_i, x_{i+1}, and all pairs of input-compatible execution environments $\mathtt{env}_i, \mathtt{env}_{i+1} \in \mathcal{E}$, there exists a coupling $(\tilde{r}_i, \tilde{r}_{i+1})$ of $T_P(x_i, \mathtt{env}_i)$ and $T_P(x_{i+1}, \mathtt{env}_{i+1})$ such that $|\tilde{r}_i - \tilde{r}_{i+1}| \leq t_{out}$ by the $(d_{in} \mapsto t_{out})$-timing-stability of P. Using the Gluing Lemma for couplings [V+09], we can construct a big coupling $(\tilde{r}_0, \tilde{r}_1, \ldots, \tilde{r}_n)$ such that $\Pr[|\tilde{r}_i - \tilde{r}_{i+1}| \leq t_{out}] = 1$ for all i. Then, by an application of the triangle inequality, for all pairs of execution environments $\mathtt{env}_0, \mathtt{env}_n \in \mathcal{E}$, there exists a coupling $(\tilde{r}_0, \tilde{r}_n)$ of $T_P(x_0, \mathtt{env}_0)$ and $T_P(x_n, \mathtt{env}_n)$ such that $|\tilde{r}_0 - \tilde{r}_n| \leq n \cdot t_{out}$ with probability 1. Therefore

$$\begin{aligned}
&\Pr[T_P(x_n, \mathtt{env}_n) \leq |x_n| \cdot t_{out} + t_0] \\
&= \Pr[T_P(x_n, \mathtt{env}_n) \leq n \cdot t_{out} + t_0] \\
&\geq \Pr[\tilde{r}_0 \leq t_0] \\
&= \Pr[T_P(\lambda, \mathtt{env}_0) \leq t_0]
\end{aligned}$$

□

The result also holds for jointly output/timing-stable programs since jointly output/timing-stable programs are also timing-stable programs (Lemma 74). However, a similar statement for OC-timing-stable programs does not hold, since it does not guarantee much about distributions of the runtimes on adjacent inputs when the output distributions on those inputs are very different. (In an extreme case, when the output distributions have disjoint supports, OC-timing stability says nothing).

4 Timing-Private Programs

In this section, we introduce a notion of privacy with respect to timing attacks. Intuitively, we require that *timing-private* programs should not leak much more information about their input than what is already revealed by their output distributions.

4.1 Timing Privacy

We generalize Definition 3 to work with arbitrary dataset distance metrics and privacy measures.

Definition 31 (Timing Privacy)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a program, $d_{\mathcal{X}}$ a metric on $\mathcal{X}$, and $\mathcal{M}$ a distance measure between probability distributions. Then we say that P is* $(d_{in} \mapsto d_{out})$-timing-private *with respect to $d_{\mathcal{X}}$ and $\mathcal{M}$ if for all x, x' satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, all pairs of environments $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$, and all $y \in supp(\mathtt{out}(P(x, \mathtt{env}))) \cap supp(\mathtt{out}(P(x', \mathtt{env}')))$*

$$\mathcal{M}(T_P(x, \mathtt{env})|_{\mathtt{out}(P(x,\mathtt{env}))=y}, T_P(x', \mathtt{env}')|_{\mathtt{out}(P(x',\mathtt{env}'))=y}) \leq d_{out}$$

We also provide an alternative and equivalent simulation-based definition of timing privacy.

Definition 32 (Sim-Timing Privacy)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a program, $d_{\mathcal{X}}$ a metric on $\mathcal{X}$, and $\mathcal{M}$ a distance measure between probability distributions. Then we say that P is $(d_{in} \mapsto d_{out})$-timing private with respect to $d_{\mathcal{X}}$ and $\mathcal{M}$ if $\forall x, x'$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, all environments $\mathtt{env}, \mathtt{env}' \in \mathcal{E}$, $\exists$ a simulator $S : \{(x, \mathtt{env}), (x', \mathtt{env}')\} \times \mathcal{Y} \to \mathcal{T}$ such that:*

1. *$\forall y \in supp(\mathtt{out}(P(x, \mathtt{env})))$, we have $S(x, \mathtt{env}, y) \equiv T_P(x, \mathtt{env})|_{\mathtt{out}(P(x,\mathtt{env}))=y}$*
2. *$\forall y \in supp(\mathtt{out}(P(x', \mathtt{env}')))$, we have $S(x', \mathtt{env}', y) \equiv T_P(x', \mathtt{env}')|_{\mathtt{out}(P(x',\mathtt{env}'))=y}$*

3. $\forall y \in \mathcal{Y}, \mathcal{M}(S(x, \texttt{env}, y), S(x', \texttt{env}', y)) \leq d_{out}$

where $Y \equiv Z$ denotes that the random variables Y and Z are identically distributed.

The alternative definition is equivalent to Definition 31, but it lends itself more naturally to a computational analogue of timing privacy when we add the assumption that S runs in polynomial time. Requirements (1) and (2) that S is identically distributed to the conditional runtime random variable gets relaxed to computational indistinguishability, and requirement (3) gets relaxed to computational differential privacy [MPRV09]. Importantly, such a definition is meaningful even if $\text{supp}(\texttt{out}(P(x, \texttt{env}))) \cap \text{supp}(\texttt{out}(P(x', \texttt{env}'))) = \emptyset$, in contrast to Definition 31. The supports on adjacent inputs may be disjoint, but a polynomial-time simulator S will not be able to detect this.

Lemma 33. *Timing privacy (Definition 31) and Sim-Timing Privacy (Definition 32) are equivalent.*

Proof. We first show that Definition 31 $\implies$ Definition 32. On every pair of inputs x, x' satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, all pairs of input-compatible environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, and every output $y \in \text{supp}(Y) \cap \text{supp}(Y')$, where $Y = \texttt{out}(P(x, \texttt{env}))$ and $Y' = \texttt{out}(P(x', \texttt{env}'))$, we have that

$$\mathcal{M}(T_P(x, \texttt{env})|_{Y=y}, T_P(x', \texttt{env}')|_{Y'=y}) \leq d_{out}$$

by timing privacy. Thus, let

$$S(x, \texttt{env}, y) = \begin{cases} t \sim T_P(x, \texttt{env})|_{Y=y} & \text{if } y \in \text{supp}(Y) \\ t \sim T_P(x', \texttt{env}')|_{Y'=y} & \text{if } y \in \text{supp}(Y') \setminus \text{supp}(Y) \\ 0 & \text{otherwise} \end{cases}$$

defined similarly for $S(x', \texttt{env}', y)$. We now make the following claims.

(1) $\forall y \in \text{supp}(Y)$

$$S(x, \texttt{env}, y) \equiv T_P(x, \texttt{env})|_{Y=y}$$

(2) $\forall y \in \text{supp}(Y')$

$$S(x', \texttt{env}', y) \equiv T_P(x', \texttt{env}')|_{Y'=y}$$

(3) $\forall y \in \mathcal{Y}, \mathcal{M}(S(x, \texttt{env}, y), S(x', \texttt{env}', y)) \leq d_{out}$

Claims (1) and (2) are trivially true by the definition of S. The proof of claim (3) follows from the fact that for all $y \notin \text{supp}(Y) \cup \text{supp}(Y')$, the simulator outputs 0 and therefore $S(x, \texttt{env}, y) \equiv S(x', \texttt{env}', y)$. Similarly, for all $y \in \text{supp}(Y) \cap \text{supp}(Y')$, the simulator outputs either $t \sim T_P(x, \texttt{env})$ or $t \sim T_P(x', \texttt{env}')$ according to its input. For all such y, $M(T_P(x, \texttt{env}), T_P(x', \texttt{env}')) \leq d_{out}$ by timing privacy. The only remaining case is when $S(x, \texttt{env}, y)$ is given a value $y \in \text{supp}(Y') \setminus \text{supp}(Y)$. In this scenario, S outputs $t \sim T_P(x', \texttt{env}')$ which will be distributed identically to $S(x', \texttt{env}', y)$ by definition (and $S(x', \texttt{env}', y)$ behaves similarly for $y \in \text{supp}(Y) \setminus \text{supp}(Y')$). Therefore $S(x, \texttt{env}, y) \equiv S(x', \texttt{env}', y)$ for all such y and the claim is proven.

Now we show that Definition 32 $\implies$ Definition 31. For every pair of datasets x, x' satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, and for all input-compatible environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, there exists a simulator $S : \{(x, \texttt{env}), (x', \texttt{env}')\} \times \mathcal{Y} \to \mathcal{T}$ such that

1. $\forall y \in \text{supp}(Y), S(x, \texttt{env}, y) \equiv T_P(x, \texttt{env})|_{Y=y}$
2. $\forall y \in \text{supp}(Y'),\ S(x', \texttt{env}', y) \equiv T_P(x', \texttt{env}')|_{Y'=y}$
3. $\forall y \in \mathcal{Y}, \mathcal{M}(S(x, \texttt{env}, y), S(x', \texttt{env}', y)) \leq d_{out}$

by sim-timing privacy. It follows that P is $(d_{in} \mapsto t_{out})$-timing private since for all $y \in \text{supp}(Y)$, $S(x, \texttt{env}, y)$ is identically distributed to the conditional runtime random variable $T_P(x, \texttt{env})|_{Y=y}$ (and similarly for $S(x', \texttt{env}', y) \equiv T_P(x', \texttt{env}')|_{Y'=y}$ for all $y \in \text{supp}(Y')$). Additionally, since

$$\forall y \in \mathcal{Y}, \mathcal{M}(S(x, \texttt{env}, y), S(x', \texttt{env}', y)) \leq d_{out}$$

it follows that

$$\mathcal{M}(T_P(x, \texttt{env})|_{Y=y}, T_P(x', \texttt{env}')|_{Y'=y}) \leq d_{out}$$

which is what we wanted to show. □

4.2 Joint Output/Timing Privacy

We now give a more general notion of *joint output/timing privacy* for arbitrary dataset metrics and privacy measures, generalizing the approximate DP version in Definition 4, which is from [BDDNT23].

Definition 34 (Joint Output/Timing Privacy)**.** *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a (possibly) randomized program. Then we say that P is* $(d_{in} \mapsto d_{out})$-jointly output/timing-private *with respect to distance metric $d_{\mathcal{X}}$ and privacy measure M, if for all adjacent $x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, and all pairs of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$*

$$M(\texttt{out}(P(x, \texttt{env})), T_P(x, \texttt{env})), (\texttt{out}(P(x', \texttt{env}')), T_P(x', \texttt{env}'))) \leq d_{out}$$

In contrast to timing privacy (Definition 31), *joint output/timing privacy* applies the standard DP definition to the joint random variable containing the program's output and runtime. In the case of pure-DP, this is equivalent to the program being both $\Theta(\varepsilon)$-differentially private and $\Theta(\varepsilon)$-timing private (Lemma 35). However, we remark that the result does not extend to (ε, δ)-timing privacy. For example, a (ε, δ)-jointly output/timing-private program may, with probability δ, output a special constant y^* and completely encode the input in its running time. On the other hand, such a program would not satisfy timing privacy, which would require that the runtime be DP even conditioned on this rare, special output y^*.

Lemma 35. *If a program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is ε-jointly output/timing private then it is also ε-DP and 2ε-timing-private.*

Proof. Let $Y = \texttt{out}(P(x, \texttt{env}))$ and $Y' = \texttt{out}(P(x', \texttt{env}'))$. We start with the fact that

$$\begin{aligned}
&\Pr[(Y, T_P(x, \texttt{env})) \in \{y\} \times S_2] \\
&\quad = \Pr[T_P(x, \texttt{env}) \in S_2 | Y = y] \cdot \Pr[Y = y] \\
&\quad \leq e^{\varepsilon} \cdot \Pr[T_P(x', \texttt{env}') \in S_2 | Y' = y] \cdot \Pr[Y' = y]
\end{aligned}$$

by joint output/timing privacy. Note that if

$$\Pr[(Y, T_P(x, \texttt{env})) \in \{y\} \times S_2] \leq e^{\varepsilon} \cdot \Pr[(Y', T_P(x', \texttt{env}')) \in \{y\} \times S_2]$$

then

$$\Pr[Y = y] \leq e^{\varepsilon} \cdot \Pr[Y' = y]$$

by post-processing (Lemma 22). Thus,

$$\begin{aligned}\Pr[T_P(x, \texttt{env}) \in S_2 | Y = y] &= \frac{\Pr[(Y, T_P(x, \texttt{env})) \in \{y\} \times S_2]}{\Pr[Y = y]} \\ &\leq \frac{e^{\varepsilon} \cdot \Pr[(Y', T_P(x', \texttt{env}')) \in \{y\} \times S_2]}{e^{-\varepsilon} \cdot \Pr[Y' = y]} \\ &= e^{2\varepsilon} \cdot \Pr[T_P(x', \texttt{env}') \in S_2 | Y' \in S_1]\end{aligned}$$

□

Lemma 36. *If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(\varepsilon_1, \delta_1)$-differentially private and $(\varepsilon_2, \delta_2)$-timing-private, then P achieves $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-joint output/timing privacy.*

Proof. The proof follows as an application of the composition theorem (Lemma 23). Since the program is $(\varepsilon_1, \delta_1)$-DP, we have that for any pair of d_{in}-close inputs x, x', all input-compatible $\texttt{env}, \texttt{env}' \in \mathcal{E}$, and for all $S \subseteq \mathcal{Y}$

$$\Pr[Y \in S] \leq e^{\varepsilon_1} \cdot \Pr[Y' \in S] + \delta_1$$

where $Y = \texttt{out}(P(x, \texttt{env}))$ and $Y' = \texttt{out}(P(x', \texttt{env}'))$.

By timing privacy, for any pair of d_{in}-close inputs x, x', and all input-compatible $\texttt{env}, \texttt{env}' \in \mathcal{E}$, there exists a simulator S satisfying for all $y \in \text{supp}(Y)$

$$S(x, \texttt{env}, y) \equiv T_P(x, \texttt{env})|_{Y=y}$$

and similarly for and all $y \in \text{supp}(Y')$,

$$S(x', \texttt{env}', y) \equiv T_P(x', \texttt{env}')|_{Y'=y}$$

Finally, for all $y \in \mathcal{Y}$, and all $Q \subseteq \mathcal{T}$ we have that

$$\Pr[S(x, \texttt{env}, y) \in Q] \leq e_2^{\varepsilon} \cdot \Pr[S(x', \texttt{env}', y) \in Q] + \delta_2$$

Thus, the random variable $(Y, S(x, \texttt{env}, Y)) \equiv (Y, T_P(x, \texttt{env})|_Y)$ is equivalent to the composition of an $(\varepsilon_1, \delta_1)$-DP and $(\varepsilon_2, \delta_2)$-DP mechanism. □

In contrast to timing privacy (Definition 31), *joint output/timing privacy* applies the standard DP definition to the joint random variable containing the program's output and runtime. In the case of pure-DP, this is equivalent to the program being both $\Theta(\varepsilon)$-differentially private and $\Theta(\varepsilon)$-timing private (see Appendix, Lemma 35). However, we remark that the result does not extend to (ε, δ)-timing privacy. For example, a (ε, δ)-jointly output/timing-private program may, with probability δ, output a special constant y^* and completely encode the input in its running time. On the other hand, such a program would not satisfy timing privacy, which would require that the runtime be DP even conditioned on this rare, special output y^*.

5 Timing-Private Delay Programs

We now discuss a core component for building timing private programs in our framework. We introduce *timing-private delay programs* which operate similarly to additive noise mechanisms for output privacy. Such programs are used to delay the execution of a program before releasing its output. When applied to timing-stable programs, the addition of such delay can transform these programs into timing-private ones.

5.1 Timing-Private Delays

Definition 37 (Timing-Private Delays)**.** *A distribution Φ on a time domain $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ satisfying* $\Pr[\Phi < 0] = 0$ *is a* $(t_{in} \to d_{out})$-timing-private delay distribution *under privacy measure $\mathcal{M}$ if* $\forall t \in \mathcal{T}$, $t \leq t_{in}$, *if* $T \sim \Phi$, *then*

$$\mathcal{M}(T, T + t) \leq d_{out}$$

Remark 38. *The requirement that Φ has zero probability mass on values less than* 0 *enforces the physical reality that time cannot be subtracted.*

We now give an example of a distribution Φ that is a $(t_{in} \to (\varepsilon, \delta))$-*timing-private delay distribution* under the smoothed max-divergence privacy measure D_∞^δ (Definition 19).

Recall the Discrete Laplace distribution with shift μ and scale s has a probability density function:

$$f(x|\mu, s) = \frac{e^{1/s} - 1}{e^{1/s} + 1} \cdot e^{-|x-\mu|/s}$$

and CDF:

$$F(x|\mu, s) = \begin{cases} \frac{e^{1/s}}{e^{1/s}+1} \cdot e^{-(\mu - x)/s}, & \text{if } x \leq \mu \\ 1 - \frac{1}{e^{1/s}+1} \cdot e^{-(x-\mu)/s}, & \text{otherwise} \end{cases}$$

Lemma 39. *For $\mu \geq t_{in}$, $B \geq 2\mu$, $s = t_{in}/\varepsilon$ with $t_{in}, \varepsilon > 0$, the* censored *Discrete Laplace distribution $\Phi = \min\{\max\{T, 0\}, B\} + c$, for a constant c where T is sampled from a Discrete Laplace distribution with parameters μ and s, is a* $(t_{in} \to (\varepsilon, \delta))$-timing private delay distribution *under the smoothed max-divergence privacy measure D_∞^δ with $\delta = 2 \cdot e^{-\varepsilon(\mu - t_{in})/t_{in}}$.*

Proof. We take $t_1 \leq t_{in}$ and let $\phi \sim \Phi$. For all $t_1 < t \leq B + c$, we have that:

$$\frac{\Pr[\phi + t_1 = t]}{\Pr[\phi = t]} = e^{\frac{\varepsilon|t - t_1 - \mu| - |t - \mu|}{t_{in}}} \leq e^{\frac{\varepsilon|t_1|}{t_{in}}} \leq e^{\varepsilon}$$

Additionally, for all $t < c$ it follows that $\Pr[\phi + t_1 = t] = \Pr[\phi = t] = 0$. Similarly, for all $t > B + c + t_1$ it follows that $\Pr[\phi + t_1 = t] = \Pr[\phi = t] = 0$. However, for all $c \leq t < t_1$ we have that $\Pr[\phi + t_1 = t] = 0$ and $\Pr[\phi = t] > 0$ and therefore the multiplicative distance between the distributions is unbounded on this interval. This event only happens when $\phi \leq t_1$ yielding

$$\begin{aligned} \delta = F(t_1|\mu, s) \leq F(t_{in}|\mu, s) &= \frac{e^{\varepsilon/t_{in}}}{e^{\varepsilon/t_{in}} + 1} \cdot e^{-\varepsilon(\mu - t_{in})/t_{in}} \\ &\leq e^{-\varepsilon(\mu - t_{in})/t_{in}} \end{aligned}$$

by the CDF of the Discrete Laplace distribution. Similarly, for $B + c < t$, we have that $\Pr[\phi + t_1 = t] > 0$ and $\Pr[\phi = t] = 0$. This event only happens when $\phi > B - t_1$:

$$\begin{aligned}\Pr[\phi > B - t_1] &\leq 1 - F(B - t_{in}|\mu, s) \\ &= \frac{1}{e^{\varepsilon/t_{in}} + 1} \cdot \left(e^{-\varepsilon(B - t_{in} - \mu)/t_{in}}\right) \\ &\leq e^{-\varepsilon(\mu - t_{in})/t_{in}}\end{aligned}$$

It follows that $D_\infty^\delta(\phi + t_1||\phi) \leq \varepsilon$ and $D_\infty^\delta(\phi||\phi + t_1) \leq \varepsilon$ for $\delta = 2e^{-\varepsilon(\mu - t_{in})/t_{in}}$. Finally, we note that $\Pr[\phi < 0] = 0$. □

Definition 40 (Timing-Private Delay Program)**.** *A $(t_{in} \mapsto d_{out})$-timing-private delay program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{X} \times \mathcal{E}$ with respect to time domain $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ and probability distance measure $\mathcal{M}$ implements the following functionality:*

1. ***Identity function**. For all $x \in \mathcal{X}$, and all $\texttt{env} \in \mathcal{E}$*

$$\Pr[\texttt{out}(P(x, \texttt{env})) = x] = 1$$

2. ***Timing-private delay distributed runtime**. For all $x \in \mathcal{X}$, all $\texttt{env} \in \mathcal{E}$, $T_P(x, \texttt{env})$ is distributed according to a $(t_{in} \mapsto d_{out})$-timing-private delay distribution on $\mathcal{T}$.*

Lemma 41. *Let $P : \mathcal{X} \times \mathcal{E} \to \mathcal{X} \times \mathcal{E}$ be a timing-private delay program. Then P is output-pure.*

Proof. Since P implements the identity function by definition, for all $x \in \mathcal{X}$, the random variables $\texttt{out}(P(x, \texttt{env}))$ and $\texttt{out}(P(x, \texttt{env}'))$ are identically distributed for all pairs of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$. □

Program 3 is an example of a timing-private delay program in the RAM and Word RAM models. The program has the property that it runs in time identically distributed to a *censored* Discrete Laplace distribution. Thus, we have:

Lemma 42. *The timing-private delay Word RAM program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{X} \times \mathcal{E}$ (Program 3) with $\texttt{shift} = \mu$ and scale parameter $s = 1/\ln(\texttt{b}/\texttt{a})$ satisfying $\ln(\texttt{b}/\texttt{a}) = \varepsilon/t_{in}$ is a $(t_{in} \mapsto (\varepsilon, \delta))$-timing-private delay program on time domain $\mathbb{N}$ for $\delta = 2 \cdot e^{-\varepsilon(\mu - t_{in})/t_{in}}$.*

Proof. We will show that Program 3 runs in time distributed exactly to $\min\{\max\{0, T\}, B\} + c$ where $c = 16 + 7 \cdot \texttt{bound}$ and T is the Discrete Laplace distribution with parameter $\mu = \texttt{shift}$, $s = t_{in}/\varepsilon = 1/\ln(\texttt{b}/\texttt{a})$, and $B \geq 2 \cdot \mu$. The claim then follows from Lemma 39 and the fact that the program implements the identity function.

The program implements the following functionality. First, the program samples a uniform random bit that is assigned to the variable `sign` (line 6). Next, it flips a biased coin with probability $p = (\texttt{b} - \texttt{a})/(\texttt{b} + \texttt{a})$ (lines 7-10) and sets $\texttt{sample} = 0$ if the trial is successful (line 11). Lines 1-14 take exactly 12 instructions to execute independent of the branching condition due to the `NOP` instructions. The program then loops `bound` times, where each loop takes exactly 7 instructions to execute. During this loop, if $\texttt{set} = 0$ (which happens conditioned on $\texttt{sample} \neq 0$), the program sets `sample` to be a value drawn from a *censored* Geometric distribution with $p = \frac{\texttt{b}-\texttt{a}}{\texttt{b}}$ and a bound of $t = \texttt{bound}$, where the censored Geometric distribution has probability mass:

$$\text{CensGeo}(x|p,t) = \begin{cases} p \cdot (1-p)^{x-1} & \text{if } x < t \\ (1-p)^{t-1} & x = t \\ 0 & \text{otherwise} \end{cases}$$

for $x \geq 1$. The distribution is censored since the program exits the loop when $\texttt{count} = \texttt{bound}$ (line 15) regardless of whether or not a success has been observed. In this case, the program sets $\texttt{sample} = \texttt{bound}$ (lines 26-29), which always executes in exactly 2 instructions.

The program then delays execution by the value $\texttt{sleep} = \texttt{shift} \pm \texttt{sample}$ (lines 30 - 33) where `sample` is subtracted or added to `shift` depending on the value of `sign`. Lines 30-33 always execute exactly 2 instructions. Therefore, up to the final $\texttt{NOP}(\texttt{sleep})$ instruction (line 34), the program executes in exactly $c = 12 + 7 \cdot \texttt{bound} + 2 + 2 = 16 + 7 \cdot \texttt{bound}$ instructions. What remains to be shown is that `sleep` is distributed exactly to $\Phi = \min\{\max\{0, T\}, B\}$.

We start with the case that $\texttt{sleep} = \mu$. Let Z be the random variable associated with the coin flip at lines 7-10. Then $\Pr[Z = 1] = \frac{\texttt{b}-\texttt{a}}{\texttt{b}+\texttt{a}}$. Observe that for the PMF f of the Discrete Laplace distribution:

$$\begin{aligned} f(x = \texttt{shift} | \mu = \texttt{shift}, s = 1/\ln(\texttt{b}/\texttt{a})) &= \frac{e^{\ln(\texttt{b}/\texttt{a})} - 1}{e^{\ln(\texttt{b}/\texttt{a})} + 1} \\ &= \frac{\texttt{b} - \texttt{a}}{\texttt{b} + \texttt{a}} \\ &= \Pr[Z = 1] \end{aligned}$$

and therefore the program delays for $\texttt{shift} = \mu$ instructions with probability mass according to DiscreteLaplace(μ, s).

We now consider the case where $\texttt{sleep} \neq \mu$, i.e., $Z = 0$. In this case, the program samples another value from a *censored* Geometric distribution with $p = \frac{\texttt{b}-\texttt{a}}{\texttt{b}}$ (lines 15-29). Thus, for all values $0 < y < B$, $y \neq \mu$

$$\begin{aligned} \Pr[\texttt{sleep} = y] &= \frac{1}{2} \cdot (1 - \Pr[Z = 1]) \cdot \Pr[\text{CensGeo}(p, t) = |y - \mu|] \\ &= \frac{1}{2} \cdot \big(1 - \frac{\texttt{b} - \texttt{a}}{\texttt{b} + \texttt{a}}\big) \cdot \big(\frac{\texttt{a}}{\texttt{b}}\big)^{|y-\mu|-1} \cdot \big(1 - \frac{\texttt{a}}{\texttt{b}}\big) \\ &= \frac{1}{2} \cdot (1 - \frac{e^{1/s} - 1}{e^{1/s} + 1}) \cdot (e^{-1/s})^{(|y-\mu|-1)} \cdot (1 - e^{-1/s}) \\ &= \frac{1}{2} \cdot (\frac{e^{1/s} + 1 - e^{1/s} + 1}{e^{1/s} + 1}) \cdot (e^{-1/s})^{(|y-\mu|-1)} \cdot (1 - e^{-1/s}) \\ &= \frac{1}{e^{1/s} + 1} \cdot (e^{-1/s})^{(|y-\mu|-1)} \cdot (1 - e^{-1/s}) \\ &= \frac{e^{(-(|y-\mu|)+1)/s} - e^{-(|y-\mu|)/s}}{e^{1/s} + 1} \\ &= \frac{e^{1/s} - 1}{e^{1/s} + 1} \cdot e^{-|y-\mu|/s} \end{aligned}$$

which equals the PMF of the Discrete Laplace distribution for all $0 < y < B, y \neq \mu$.

We now consider the edge cases. We show that $\Pr[\texttt{sleep} = 0] = F(0|\mu, s)$ where F is the CDF of the Discrete Laplace distribution with shift μ and with scale parameter $s = 1/\ln(\texttt{b}/\texttt{a})$ satisfying $\ln(\texttt{b}/\texttt{a}) = \varepsilon/t_{in}$:

$$\begin{aligned}
\Pr[\texttt{sleep} = 0] &= \frac{1}{2} \cdot (1 - \Pr[Z = 1]) \cdot \Pr[\text{CensGeo}(p, t) \geq \mu - 1] \\
&= \frac{1}{2} \cdot \big(1 - \frac{\texttt{b} - \texttt{a}}{\texttt{b} + \texttt{a}}\big) \cdot \big(\frac{\texttt{a}}{\texttt{b}}\big)^{(\mu-1)} \\
&= \frac{1}{2} \cdot (1 - \frac{e^{\varepsilon/t_{in}} - 1}{e^{\varepsilon/t_{in}} + 1}) \cdot (e^{-\varepsilon/t_{in}})^{(\mu-1)} \\
&= \frac{1}{e^{\varepsilon/t_{in}} + 1} \cdot (e^{-\varepsilon \cdot (\mu-1)/t_{in}}) \\
&= \frac{e^{\varepsilon/t_{in}}}{e^{\varepsilon/t_{in}} + 1} \cdot (e^{-\varepsilon \cdot \mu/t_{in}}) \\
&= F(0|\mu = \mu, s = \varepsilon/t_{in})
\end{aligned}$$

where the leading $\frac{1}{2}$ comes from the probability that the sign is negative.

Finally, we show that $\Pr[\texttt{sleep} = B] = 1 - F(B - 1|\mu, s)$ where F is the CDF of the Discrete Laplace distribution with shift μ and scale parameter $s = 1/\ln(\texttt{b}/\texttt{a})$ satisfying $\ln(\texttt{b}/\texttt{a}) = \varepsilon/t_{in}$ and $B = \mu + \texttt{bound} \geq 2\mu$:

$$\begin{aligned}
\Pr[\texttt{sleep} = B] &= \frac{1}{2} \cdot (1 - \Pr[Z = 1]) \cdot \Pr[\text{CensGeo}(p, t) = B] \\
&= \frac{1}{2} \cdot \big(1 - \frac{\texttt{b} - \texttt{a}}{\texttt{b} + \texttt{a}}\big) \cdot \big(\frac{\texttt{a}}{\texttt{b}}\big)^{B-1} \\
&= \frac{1}{2} \cdot (1 - \frac{e^{\varepsilon/t_{in}} - 1}{e^{\varepsilon/t_{in}} + 1}) \cdot (e^{-\varepsilon/t_{in}})^{B-1} \\
&= \frac{1}{e^{\varepsilon/t_{in}} + 1} \cdot (e^{-\varepsilon \cdot (B-1)/t_{in}}) \\
&= \frac{1}{e^{\varepsilon/t_{in}} + 1} \cdot (e^{-\varepsilon \cdot (B-1)/t_{in}}) \\
&= 1 - F(B - 1|\mu = \mu, s = \varepsilon/t_{in})
\end{aligned}$$

Thus, the program has runtime that is distributed exactly to $\Phi = \min\{\max\{0, T\}, B\} + c$ for $c = 16 + 7 \cdot \texttt{bound}$. The lemma follows from Lemma 39. □

Lemma 43. *The timing-private delay Word RAM program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{X} \times \mathcal{E}$ (Program 3) is both output-pure and timing-pure.*

Proof. The proof follows from the fact that the program never interacts with uninitialized memory. Therefore for all $y \in \mathbb{N}$, the program's output and runtime is identically distributed on all pairs of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$. □

Program 3 Timing-Private Delay Program

Input: An input x located in memory $M[\texttt{input\_ptr}], \ldots, M[\texttt{input\_ptr} + \texttt{input\_len} - 1]$.

Output: The input x, with runtime $\min\{\max\{T, 0\}, \texttt{shift} + \texttt{bound}\} + c$, where T is drawn from a Discrete Laplace distribution with parameters $\mu = \texttt{shift}$ and $s = 1/\ln(\texttt{b}/\texttt{a})$ where `shift`, `a`, $\texttt{b} > \texttt{a}$, and $\texttt{bound} \geq \texttt{shift}$ are hardcoded constants, and $c = 16 + 7 \cdot \texttt{bound}$. For the Word RAM version of the program, we require $s, (\mu + \texttt{bound})$ and $(\texttt{a} + \texttt{b})$ to be less than 2^ω.

```
output_ptr = input_ptr;
output_len = input_len; {compute the identity function}
count = 0;
set = 0;
sample = 0;
sign = RAND(1); {add or subtract from μ}
zeroprobA = b − a; {num. of prob. to sample a zero}
zeroprobB = b + a; {denom. of prob. to sample a zero}
idx = RAND(zeroprobB − 1);

if idx < zeroprobA then
  sample = 0; {With probability (b−a)/(b+a) add μ delay}
  set = 1;
else
  NOP(2);

while count < bound do
  count = count + 1;
  if set == 0 then
    idx = RAND(b − 1);
    if idx < b − a then
      sample = count; {with probability (b−a)/b add μ ± count delay}
      set = 1;
    else
      NOP(2);
  else
    NOP(4);

if set == 0 then
  sample = bound;
else
  NOP(1);

if sign == 0 then
  sleep = shift − sample;
else
  sleep = shift + sample;

NOP(sleep); {Delay for sampled amount of time}
HALT;
```

6 Chaining and Composition

Software libraries such as OpenDP and Tumult Core [GHV20, BBD+22, SVM+] support chaining together multiple algorithms in a modular fashion to create more complex mechanisms with provable stability and privacy guarantees. Motivated by these libraries, we extend our framework to support similar functionality. Specifically, we discuss properties of timing-stable programs that are combined to create more complex functionality such as chaining and composition.

6.1 Chaining Timing-Stable Programs

While timing stability bounds the differences in program execution time for a single program P, we are often interested in chaining together multiple programs, e.g., executing program P_2 on the output of program P_1. For such chaining operations, we use the notation $(P_2 \circ P_1)(x, \texttt{env}_1)$ interchangeably with $P_2(P_1(x, \texttt{env}_1), \texttt{env}_2)$ for some execution environment $\texttt{env}_2$ (see below).

Definition 44 (Chaining-compatible Programs)**.** *We say that programs $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ and $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ are* chaining-compatible *if there exists a program $(P_2 \circ P_1)$ and a constant c such that for inputs $x \in \mathcal{X}$ and environments $\texttt{env}_1 \in \mathcal{E}$, there exists a (possibly random) $\texttt{env}_2 \in \mathcal{E}$ such that we have:*

$$\texttt{out}((P_2 \circ P_1)(x, \texttt{env}_1)) = \texttt{out}(P_2(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2))$$

and

$$T_{(P_2 \circ P_1)}(x, \texttt{env}_1) = T_{P_1}(x, \texttt{env}_1) + T_{P_2}(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2) + c.$$

Here we allow the random variable $\texttt{env}_2$ to be arbitrarily correlated with x, as well as the output and runtime of $P_1(x, \texttt{env}_1)$, but should be independent of the coin tosses of P_2.

Remark 45. *If the construction of $P_2 \circ P_1$ in the computational model does not modify the environment when chaining programs together, then $\texttt{env}_2$ equals $\texttt{outenv}(P_1(x, \texttt{env}_1))$. However, real-world systems often modify the environment before executing sequential programs, e.g., setting up CPU registers.*

RAM and Word RAM programs can easily be made chaining-compatible.

Lemma 46. *Word RAM programs $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ and $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ are chaining compatible.*

Proof. Observe that, after P_1's execution, the program's output $y = \texttt{out}(P_1(x, \texttt{env}_1))$ is located in memory locations

$$M[\texttt{output\_ptr}], \ldots, M[\texttt{output\_len} - 1]$$

To create the chained program $P_2 \circ P_1$, we replace the `HALT` instruction at the end of P_1 with the instructions `input_ptr = output_ptr` and `input_len = output_len`. We then add all of the code from P_2 and adjust the necessary line numbers in any `GOTO` statements in the relevant P_2 code. The resulting program is an execution of P_2 on input y in environment $\texttt{env}_2$ that is the same as $\texttt{outenv}(P_1(x, \texttt{env}_1))$ except for these modifications to `input_ptr` and `input_len`. The total runtime is the time to execute $P_1(x, \texttt{env}_1)$, plus the execution time corresponding to the extra lines for adjusting the input pointer and input length variables, plus the time to execute P_2 on the output from P_1 in environment $\texttt{env}_2$. The proof follows similarly for RAM programs. □

Lemma 47. *Suppose $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_1 \mapsto \{d_2, t_1\})$-jointly output/timing stable under input distance metric $d_{\mathcal{X}}$ and output distance metric $d_{\mathcal{Y}}$. Similarly, suppose $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ is $(d_2 \mapsto \{d_3, t_2\})$-jointly output/timing stable under input distance metric $d_{\mathcal{Y}}$ and output distance metric $d_{\mathcal{Z}}$. Then if P_1 and P_2 are chaining compatible, the chained program $P_2 \circ P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ is $(d_1 \mapsto \{d_3, t_1 + t_2\})$-*jointly output/timing stable.

Proof. We show that for every x, x' satisfying $d_{\mathcal{X}}(x, x') \leq d_1$ under metric $d_{\mathcal{X}}$, every pair of input-compatible execution environments $\mathtt{env}_1, \mathtt{env}_1' \in \mathcal{E}$, there exists a coupling $((\tilde{u}, \tilde{v}), (\tilde{u}', \tilde{v}'))$ of $(\mathtt{out}((P_2 \circ P_1)(x, \mathtt{env}_1)), T_{P_2 \circ P_1}(x, \mathtt{env}_1))$ and $(\mathtt{out}((P_2 \circ P_1)(x', \mathtt{env}_1')), T_{(P_2 \circ P_1)}(x', \mathtt{env}_1'))$ such that $d_{\mathcal{Z}}(\tilde{u}, \tilde{u}') \leq d_3$ and $|\tilde{v} - \tilde{v}'| \leq t_1 + t_2$ with probability 1.

We construct the coupling as follows. Let $((\tilde{u}_1, \tilde{v}_1), (\tilde{u}_1', \tilde{v}_1'))$ be the coupling associated with the joint random variables $(\mathtt{out}(P_1(x, \mathtt{env}_1)), T_{P_1}(x, \mathtt{env}_1))$ and $(\mathtt{out}(P_1(x', \mathtt{env}_1')), T_{P_1}(x', \mathtt{env}_1'))$ satisfying $d_{\mathcal{Y}}(\tilde{u}_1, \tilde{u}_1') \leq d_2$ and $|\tilde{v}_1 - \tilde{v}_1'| \leq t_1$ with probability 1 (by joint output/timing stability of P_1).

Then condition on any fixed $((u_1, v_1), (u_1', v_1')) \in \mathrm{supp}((\tilde{u}_1, \tilde{v}_1), (\tilde{u}_1', \tilde{v}_1'))$ and execution environments $\mathtt{env}_2$ and $\mathtt{env}_2'$ (conditioned on (u_1, v_1) and (u_1', v_1') respectively). Since $d_{\mathcal{Y}}(u_1, u_1') \leq d_2$, then for every pair of execution environments $\mathtt{env}_2, \mathtt{env}_2' \in \mathcal{E}$, there exists a coupling $((\tilde{u}_2, \tilde{v}_2), (\tilde{u}_2', \tilde{v}_2'))$ of the joint random variables $(\mathtt{out}(P_2(u_1, \mathtt{env}_2)), T_{P_2}(u_1, \mathtt{env}_2))$ and $(\mathtt{out}(P_2(u_1', \mathtt{env}_2')), T_{P_2}(u_1', \mathtt{env}_2'))$ satisfying $d_{\mathcal{Z}}(\tilde{u}_2, \tilde{u}_2') \leq d_3$ and $|\tilde{v}_2 - \tilde{v}_2'| \leq t_2$ with probability 1 (by joint output/timing stability of P_2).

We sample $((u_2, v_2), (u_2', v_2')) \sim ((\tilde{u}_2, \tilde{v}_2), (\tilde{u}_2', \tilde{v}_2'))|_{\mathtt{env}_2, \mathtt{env}_2'}$ and set:

$$\begin{aligned}
(\tilde{u}, \tilde{v}) &= (u_2, v_1 + v_2 + c) \\
&\equiv (\mathtt{out}(P_2(u_1, \mathtt{env}_2)), T_{P_1}(x, \mathtt{env}_1) + T_{P_2}(u_1, \mathtt{env}_2) + c) \\
&\equiv (\mathtt{out}(P_2(\mathtt{out}(P_1(x, \mathtt{env}_1)), \mathtt{env}_2)), T_{P_1}(x, \mathtt{env}_1) + T_{P_2}(\mathtt{out}(P_1(x, \mathtt{env}_1)), \mathtt{env}_2) + c) \\
&\equiv ((\mathtt{out}((P_2 \circ P_1)(x, \mathtt{env}_1)), T_{P_2 \circ P_1}(x, \mathtt{env}_1))
\end{aligned}$$

and similarly for $(\tilde{u}', \tilde{v}')$, where c is the chaining constant in Definition 44. It follows that:

$$d_{\mathcal{Z}}(\tilde{u}, \tilde{u}') \leq d_3$$

and

$$\begin{aligned}
|\tilde{v} - \tilde{v}'| &= |v_1 + v_2 + c - v_1' - v_2' - c| \\
&= |v_1 - v_1' + v_2 - v_2'| \\
&\leq |v_1 - v_1'| + |v_2 - v_2'| \\
&\leq t_1 + t_2
\end{aligned}$$

□

Lemma 48. *Suppose $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_1 \mapsto \{d_2, t_1\})$-jointly output/timing stable under input distance metric $d_{\mathcal{X}}$ and output distance metric $d_{\mathcal{Y}}$. Similarly, suppose $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ is $(d_2 \mapsto t_2)$-OC timing stable under $d_{\mathcal{Y}}$. Then if P_1 and P_2 are chaining compatible, the chained program $(P_2 \circ P_1) : \mathcal{X} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ is $(d_1 \mapsto t_1 + t_2)$-*OC timing stable.

Proof. We show that for every x, x' satisfying $d_{\mathcal{X}}(x, x') \leq d_1$ under metric $d_{\mathcal{X}}$, every pair of input-compatible execution environments $\mathtt{env}_1, \mathtt{env}_1' \in \mathcal{E}$, and for every $z \in \mathrm{supp}((P_2 \circ P_1)(x, \mathtt{env}_1)) \cap$

$\text{supp}((P_2 \circ P_1)(x', \texttt{env}_1'))$, there exists a coupling $(\tilde{w}, \tilde{w}')$ of $T_{P_2 \circ P_1}(x, \texttt{env}_1)_{|\texttt{out}((P_2 \circ P_1)(x, \texttt{env}_1))=z}$ and $T_{(P_2 \circ P_1)}(x', \texttt{env}_1')_{|\texttt{out}((P_2 \circ P_1)(x', \texttt{env}_1'))=z}$ such that $|\tilde{w} - \tilde{w}'| \leq t_1 + t_2$ with probability 1.

We construct the coupling as follows. Let $((\tilde{u}, \tilde{v}), (\tilde{u}', \tilde{v}'))$ be the coupling associated with the joint random variables $(\texttt{out}(P_1(x, \texttt{env}_1)), T_{P_1}(x, \texttt{env}_1))$ and $(\texttt{out}(P_1(x', \texttt{env}_1')), T_{P_1}(x', \texttt{env}_1'))$ satisfying $d_{\mathcal{Y}}(\tilde{u}, \tilde{u}') \leq d_2$ and $|\tilde{v} - \tilde{v}'| \leq t_1$ with probability 1 (by joint output/timing stability).

Then condition on any fixed $((u, v), (u', v')) \in \text{supp}((\tilde{u}, \tilde{v}), (\tilde{u}', \tilde{v}'))$ and execution environments $\texttt{env}_2$ and $\texttt{env}_2'$ (conditioned on (u, v) and (u', v') respectively). Since $d_{\mathcal{Y}}(u, u') \leq d_2$, then for every $z \in \text{supp}(\texttt{out}(P_2(u, \texttt{env}_2))) \cap \text{supp}(\texttt{out}(P_2(u', \texttt{env}_2'))))$, there exists a coupling $(\tilde{r}, \tilde{r}')$ of the conditional runtime random variables $T_{P_2}(u, \texttt{env}_2)|_{\texttt{out}(P_2(u, \texttt{env}_2))=z}$ and $T_{P_2}(u', \texttt{env}_2')|_{\texttt{out}(P_2(u', \texttt{env}_2'))=z}$ satisfying $|\tilde{r} - \tilde{r}'| \leq t_2$ (by output-conditional timing stability).

We sample $(r, r') \sim (\tilde{r}, \tilde{r}')|_{\texttt{env}_2, \texttt{env}_2'}$ and set:

$$\begin{aligned}
\tilde{w} &= v + r + c \\
&\equiv T_{P_1}(x, \texttt{env}_1) + T_{P_2}(u, \texttt{env}_2)|_{\texttt{out}(P_2(u, \texttt{env}_2))=z} + c \\
&\equiv T_{P_1}(x, \texttt{env}_1) + T_{P_2}(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2)|_{\texttt{out}(P_2 \circ P_1(x, \texttt{env}_1))=z} + c \\
&\equiv T_{P_2 \circ P_1}(x)|_{P_2(P_1(x))=z}
\end{aligned}$$

and similarly for w', where c is the chaining constant in Definition 44. It follows that

$$\begin{aligned}
|\tilde{w} - \tilde{w}'| &= |u + r + c - u' - r' - c| \\
&= |u - u' + r - r'| \\
&\leq |u - u'| + |r - r'| \\
&\leq t_1 + t_2
\end{aligned}$$

□

Finally, we can chain together OC-timing-stable programs with timing-private delay programs to achieve timing privacy:

Theorem 49. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be $(d_1 \mapsto t_1)$-OC timing stable under input distance metric $d_{\mathcal{X}}$. If $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is a $(t_1 \mapsto d_2)$-timing-private delay program with respect to privacy measure $\mathcal{M}$, then $P_2 \circ P_1$ is $(d_1 \mapsto d_2)$-timing private with respect to $d_{\mathcal{X}}$ and $\mathcal{M}$.*

Proof. We want to show that for all pairs of x, x' such that $d_{\mathcal{X}}(x, x') \leq d_1$, all pairs of input-compatible environments $\texttt{env}_1, \texttt{env}_1'$, and all $y \in \text{supp}(\texttt{out}(P_1(x, \texttt{env}_1))) \cap \text{supp}(\texttt{out}(P_1(x', \texttt{env}_1'))))$,

$$\mathcal{M}(T_{(P_2 \circ P_1)}(x, \texttt{env}_1)_{|\texttt{out}((P_2 \circ P_1)(x, \texttt{env}_1))=y}, T_{(P_2 \circ P_1)}(x', \texttt{env}_1')|_{\texttt{out}((P_2 \circ P_1)(x', \texttt{env}_1'))=y}) \leq d_2$$

Let Φ be the $(t_1 \mapsto d_2)$-timing private delay distribution implemented by P_2, c is the chaining constant used in Definition 44, and $(\tilde{r}, \tilde{r}')$ is the coupling of the conditional random variables $T_{P_1}(x, \texttt{env}_1)_{|\texttt{out}(P_1(x, \texttt{env}_1))=y}$ and $T_{P_1}(x', \texttt{env}_1')_{|\texttt{out}(P_1(x', \texttt{env}_1'))=y}$ satisfying $|\tilde{r} - \tilde{r}'| \leq t_1$ (by output-conditional timing stability). Let $(r, r') \sim (\tilde{r}, \tilde{r}')$ and $\texttt{env}_2$ and $\texttt{env}_2$ be conditioned on (r, y) and (r', y) respectively. Then,

$$\begin{aligned}
r + c + \Phi &\equiv T_{P_1}(x, \texttt{env}_1)_{|\texttt{out}(P_1(x, \texttt{env}_1))=y} + c + \phi \\
&\equiv T_{P_1}(x, \texttt{env}_1)_{|\texttt{out}(P_1(x, \texttt{env}_1))=y} + c + T_{P_2}(y, \texttt{env}_2)_{|\texttt{out}(P_1(x, \texttt{env}_1))=y} \\
&\equiv T_{(P_2 \circ P_1)}(x, \texttt{env}_1)_{|\texttt{out}((P_2 \circ P_1)(x, \texttt{env}_1))=y}
\end{aligned}$$

and similarly $r' + c + \Phi \equiv T_{(P_2 \circ P_1)}(x', \texttt{env}_1')_{|\texttt{out}((P_2 \circ P_1)(x', \texttt{env}_1'))=y}$. Since $\Pr[|r - r'| \leq t_1] = 1$, and Φ is a timing-private delay distribution, it follows that for $T \sim \Phi$:

$$\mathcal{M}(r + c + T, r' + c + T) \leq d_2$$

and therefore

$$\mathcal{M}(T_{(P_2 \circ P_1)}(x, \texttt{env}_1)_{|\texttt{out}((P_2 \circ P_1)(x, \texttt{env}_1))=y}, T_{(P_2 \circ P_1)}(x', \texttt{env}_1')_{|\texttt{out}((P_2 \circ P_1)(x', \texttt{env}_1'))=y}) \leq d_2$$

□

We now give prove a post-processing property for programs.

Lemma 50 (Post-processing Program Outputs)**.** *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be (ε, δ)-differentially private program under input metric $d_{\mathcal{X}}$ and the smoothed max-divergence privacy measure. Let $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ be a (possibly) randomized program that is output-pure (Definition 8). Then if P_2 and P_1 are chaining compatible, the chained program $P_2 \circ P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ is (ε, δ)-differentially private under input metric $d_{\mathcal{X}}$ and the smoothed max-divergence privacy measure D_∞^δ.*

Proof. For all adjacent inputs $x, x' \in \mathcal{X}$ under input metric $d_{\mathcal{X}}$, and all $\texttt{env}_1, \texttt{env}_1' \in \mathcal{E}$, we have that $D_\infty^\delta(\texttt{out}(P_1(x, \texttt{env}_1)) || \texttt{out}(P_1(x', \texttt{env}_1')) \leq \varepsilon$ and $D_\infty^\delta(\texttt{out}(P_1(x', \texttt{env}_1')) || \texttt{out}(P_1(x, \texttt{env}_1)) \leq \varepsilon$ by the fact that P_1 is achieves (ε, δ)-differential privacy (Lemma 20). Since P_2 is output-pure, for all execution environments $\texttt{env}_2 \in \mathcal{E}$, $\texttt{out}(P_2(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2) = f(\texttt{out}(P_1(x, \texttt{env}_1)))$ where $f : \mathcal{Y} \to \mathcal{Z}$ is a (possibly randomized) function. By the post-processing property for approximate DP (Lemma 22), it follows that for all $\texttt{env}_2, \texttt{env}_2' \in \mathcal{E}$:

$$\begin{aligned}
&D_\infty^\delta(\texttt{out}(P_2(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2)) || \texttt{out}(P_2(\texttt{out}(P_1(x', \texttt{env}_1')), \texttt{env}_2'))) \\
&\quad = D_\infty^\delta(f(\texttt{out}(P_1(x, \texttt{env}_1))) || f(\texttt{out}(P_1(x', \texttt{env}_1')))) \\
&\quad \leq D_\infty^\delta(\texttt{out}(P_1(x, \texttt{env}_1)) || \texttt{out}(P_1(x', \texttt{env}_1'))) \\
&\quad \leq \varepsilon
\end{aligned}$$

and similarly for $D_\infty^\delta(\texttt{out}(P_2(\texttt{out}(P_1(x', \texttt{env}_1')), \texttt{env}_2')) || \texttt{out}(P_2(\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{env}_2)))$. □

Lemma 51. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be $(d_{in} \mapsto d_{out})$-differentially private with respect to input distance metric $d_{\mathcal{X}}$ and a privacy measure M. Let $P_2 : \mathcal{Y} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ be a timing-private delay program such that P_1 and P_2 are chaining compatible. Then the chained program $P_2 \circ P_1$ is $(d_{in} \mapsto d_{out})$-differentially private with respect to input metric $d_{\mathcal{X}}$ and privacy measure M.*

Proof. The proof follows from the fact that timing-private delay programs are output pure (Lemma 41) and Lemma 50. □

The post-processing property for DP (Lemma 22) states that arbitrary transformations applied to differentially private outputs produces new outputs that remain differentially private. Unfortunately, our definition of timing privacy is incompatible with post-processing. Timing privacy bounds the *additional* information leakage caused by observing the program's running time after observing the program's output. However, post-processing can potentially destroy information in the program's output and this can result in unbounded additional information leakage in the program's runtime.

To see this effect in action, consider a jointly output/timing-pure program P implementing the identity function $f(x) = x$ with the additional property that the runtime reveals something about the input. That is, $T_P(x) = g(x)$ for a non-constant function g. Then P is $(1 \mapsto 0)$-timing-private with respect to the change-one distance metric d_{CO} and any privacy measure $\mathcal{M}$ (e.g., max divergence). The identity function program satisfies perfect timing privacy despite its running time being directly correlated with its input. This is due to the fact that P's output already reveals the input and so the program's running time reveals no additional information. However, if we apply the simple post-processing step by chaining P with a deterministic, constant-time program P' that sets $\texttt{output\_len} = 0$, then the result is no longer perfectly timing-private since $\texttt{out}((P' \circ P)(x, \texttt{env})) = \lambda$[7] while $T_{P' \circ P}(x, \texttt{env})$ leaks information about the input (everything about it if g is injective).

6.2 A Timing-Private Unbounded Noisy Sum

We illustrate the chaining operations above by chaining the Sum program (Program 2) with the Discrete Laplace Mechanism (Program 4, given below) and our Timing-Private Delay Program (Program 3).

We start by giving a Word RAM program for the Discrete Laplace Mechanism and analyze its timing properties.

Lemma 52. *The Discrete Laplace Word RAM program $P : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ (Program 4) is output-pure and timing-pure.*

Proof. The program only ever reads memory at location $M[\texttt{input\_ptr}]$ (line 3) to obtain the input and later writes the output to $M[0]$. Since the program's execution does not depend on any memory values beyond its input, then for all $y \in \mathbb{N}$, the program's output and runtime is identically distributed on all pairs of input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$. □

Lemma 53. *Let $P : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be the Discrete Laplace program (Program 4) with scale parameter $s = 1/\ln(\mathtt{b}/\mathtt{a}) = d_{in}/\varepsilon$. Then P is $(d_{in} \mapsto \varepsilon)$-DP with respect to input metric $d_{\mathbb{N}}(y, y') = |y - y'|$ and the max-divergence privacy measure.*

Proof. We will show that for all for y, y' satisfying $|y - y'| \leq d_{out}$, and all $S \subseteq \mathbb{N}$, $\Pr[\texttt{out}(P_2(y)) \in S] / \Pr[\texttt{out}(P_2(y')) \in S] \leq e^{\varepsilon}$. We ignore the execution environments in our analysis since P is output-pure (Lemma 52).

The Discrete Laplace program first samples a value from a censored Discrete Laplace distribution parameterized by a shift $\mu = y$ and scale $s = 1/\ln(\mathtt{b}/\mathtt{a}) = d_{out}/\varepsilon$. The program works similarly to Program 3 in how it samples from a Discrete Laplace distribution except that it does not bound the number of trials when sampling from a Geometric distribution (lines 14-18). Since the Word RAM model does not support values < 0 or $> 2^\omega - 1$, the result of $\texttt{noisy\_y} = \texttt{y} - \texttt{noise}$ (line 20) is always rounded to 0 when $\texttt{noise} > \texttt{y}$ and the result of $\texttt{noisy\_y} = \texttt{y} + \texttt{noise}$ (line 22) is always rounded to $2^\omega - 1$. As a result, the program's output is distributed as a *censored* Discrete Laplace random variable with PMF f defined below:

$$f(z|\mu = y, s = 1/\ln(\mathtt{b}/\mathtt{a})) = \begin{cases} \frac{\mathtt{b}-\mathtt{a}}{\mathtt{b}+\mathtt{a}} & z = \mu \\ \frac{1}{2} \cdot \left(1 - \frac{\mathtt{b}-\mathtt{a}}{\mathtt{b}+\mathtt{a}}\right) \cdot \left(\frac{\mathtt{a}}{\mathtt{b}}\right)^{|z-\mu|-1} \cdot \left(1 - \frac{\mathtt{a}}{\mathtt{b}}\right) & 0 < z < 2^\omega, z \neq \mu \\ \frac{1}{2} \cdot \left(1 - \frac{\mathtt{b}-\mathtt{a}}{\mathtt{b}+\mathtt{a}}\right) \cdot \left(\frac{\mathtt{a}}{\mathtt{b}}\right)^{|z-\mu|-1} & z = 0 \text{ or } z = 2^\omega - 1 \\ 0 & \text{otherwise} \end{cases}$$

[7] We use λ to indicate an empty output

Program 4 Discrete Laplace Mechanism

Input: A number $y \in \mathbb{N}$ occupying memory location $M[\texttt{input\_ptr}]$

Output: $\max\{y + \text{DiscreteLaplace}(\mu = 0, s), 0\}$ where DiscreteLaplace is defined as in Section 5.1 and $s = 1/\ln(\texttt{b}/\texttt{a})$. For the Word RAM version of the program, we require $s < 2^\omega$ and $\texttt{a} + \texttt{b} < 2^\omega$ and $\min\{\max\{y + \text{DiscreteLaplace}(\mu = 0, s), 0\}, 2^\omega - 1\}$ is output instead since the Word RAM program outputs values in $[0, 2^\omega)$.

1: $\texttt{output\_len} = 1$;
2: $\texttt{output\_ptr} = 0$;
3: $\texttt{y} = M[\texttt{input\_ptr}]$;
4: $\texttt{noise} = 0$;
5: $\texttt{set} = 0$;
6: $\texttt{sign} = \texttt{RAND}(1)$; {Pick a uniformly random noise direction (positive or negative)}
7: $\texttt{zprobA} = \texttt{b} - \texttt{a}$;
8: $\texttt{zprobB} = \texttt{b} + \texttt{a}$;
9: $\texttt{idx} = \texttt{RAND}(\texttt{zprobB} - 1)$;
10: **if** $\texttt{idx} < \texttt{zprobA}$ **then**
11: $\quad\texttt{set} = 1$; {sample 0 noise with probability $\frac{\texttt{b}-\texttt{a}}{\texttt{b}+\texttt{a}}$}
12: **else**
13: $\quad\texttt{set} = 0$;

14: **while** $\texttt{set} == 0$ **do**
15: $\quad\texttt{noise} = \texttt{noise} + 1$; {sample from a Geometric random variable with $p = \frac{\texttt{b}-\texttt{a}}{\texttt{b}}$}
16: $\quad\texttt{idx} = \texttt{RAND}(\texttt{b} - 1)$;
17: $\quad$**if** $\texttt{idx} < \texttt{b} - \texttt{a}$ **then**
18: $\quad\quad\texttt{set} = 1$;

19: **if** $\texttt{sign} == 0$ **then**
20: $\quad\texttt{noisy\_y} = \texttt{y} - \texttt{noise}$;
21: **else**
22: $\quad\texttt{noisy\_y} = \texttt{y} + \texttt{noise}$;

23: $M[\texttt{output\_ptr}] = \texttt{noisy\_y}$;
24: HALT;

It follows that for all $y \in \mathcal{Y}$:

$$\begin{aligned}
\Pr[\texttt{out}(P_2(y)) = y] &= \frac{\texttt{b} - \texttt{a}}{\texttt{b} + \texttt{a}} \\
&= \frac{e^{\ln(\texttt{b}/\texttt{a})} - 1}{e^{\ln(\texttt{b}/\texttt{a})} + 1} \\
&= \frac{e^{1/s} - 1}{e^{1/s} + 1} \\
&= \frac{e^{1/s} - 1}{e^{1/s} + 1} \cdot e^{-|y-y|/s}
\end{aligned}$$

which matches the probability mass function for a Discrete Laplace distribution. Similarly, for all $y, z \in \mathbb{N}$, $0 < z < 2^\omega - 1, z \neq y$:

$$\begin{aligned}
\Pr[\mathtt{out}(P_2(y)) = z] &= \frac{1}{2} \cdot \big(1 - \frac{\mathtt{b}-\mathtt{a}}{\mathtt{b}+\mathtt{a}}\big) \cdot \big(\frac{\mathtt{a}}{\mathtt{b}}\big)^{|z-y|-1} \cdot (1 - \frac{\mathtt{a}}{\mathtt{b}}) \\
&= \frac{1}{2} \cdot (1 - \frac{e^{\ln(\mathtt{b}/\mathtt{a})} - 1}{e^{\ln(\mathtt{b}/\mathtt{a})} + 1}) \cdot (e^{-\ln(\mathtt{b}/\mathtt{a})})^{(|z-y|-1)} \cdot (1 - e^{-\ln(\mathtt{b}/\mathtt{a})}) \\
&= \frac{1}{2} \cdot (1 - \frac{e^{1/s} - 1}{e^{1/s} + 1}) \cdot (e^{-1/s})^{(|z-y|-1)} \cdot (1 - e^{-1/s}) \\
&= \frac{1}{2} \cdot (\frac{e^{1/s} + 1 - e^{1/s} + 1}{e^{1/s} + 1}) \cdot (e^{-1/s})^{(|z-y|-1)} \cdot (1 - e^{-1/s}) \\
&= \frac{1}{e^{1/s} + 1} \cdot (e^{-1/s})^{(|z-y|-1)} \cdot (1 - e^{-1/s}) \\
&= \frac{e^{(-|z-y|+1)/s} - e^{-(|z-y|)/s}}{e^{1/s} + 1} \\
&= \frac{e^{1/s} - 1}{e^{1/s} + 1} \cdot e^{-|z-y|/s}
\end{aligned}$$

Therefore, for all $y, y' \in \mathbb{N}$ such that $|y - y'| \leq d_{out}$, and all $0 < z < 2^\omega - 1$, we have:

$$\begin{aligned}
\frac{\Pr[\mathtt{out}(P_2(y)) = z]}{\Pr[\mathtt{out}(P_2(y')) = z]} &= \frac{e^{-|z-y|/s}}{e^{-|z-y'|/s}} \\
&= e^{(|z-y'|-|z-y|)/s} \\
&\leq e^{|y-y'|/s} \\
&\leq e^{\varepsilon \cdot |y-y'|/d_{out}} \\
&\leq e^{\varepsilon}
\end{aligned}$$

Finally for $z = 2^\omega - 1$ or $z = 0$,

$$\begin{aligned}
\frac{\Pr[\mathtt{out}(P_2(y)) = z]}{\Pr[\mathtt{out}(P_2(y')) = z]} &= \frac{(\mathtt{a}/\mathtt{b})^{|z-y|-1}}{(\mathtt{a}/\mathtt{b})^{|z-y'|-1}} \\
&= \frac{e^{-\ln(\mathtt{b}/\mathtt{a}) \cdot (|z-y|-1)}}{e^{-\ln(\mathtt{b}/\mathtt{a}) \cdot (|z-y'|-1)}} \\
&= \frac{e^{-(|z-y|-1)/s}}{e^{-(|z-y'|-1)/s}} \\
&\leq e^{(|z-y'|-|z-y|)/s} \\
&\leq e^{\varepsilon \cdot |y-y'|/d_{out}} \\
&\leq e^{\varepsilon}
\end{aligned}$$

Therefore, for all y, y' satisfying $|y - y'| \leq d_{out}$, and all $S \subseteq \mathbb{N}$, we have

$$\frac{\Pr[\mathtt{out}(P_2(y)) \in S]}{\Pr[\mathtt{out}(P_2(y')) \in S]} \leq e^{\varepsilon}$$

and the claim is satisfied. □

Lemma 54. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be the Sum Word RAM program (Program 2) and $P_2 : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be the Discrete Laplace (Program 4) with scale parameter $s = 1/\ln(\mathtt{b}/\mathtt{a})$ satisfying $\ln(\mathtt{b}/\mathtt{a}) = \varepsilon/\Delta$. Then the chained program $P_2 \circ P_1$ is ε-DP under the insert-delete distance metric d_{ID}.*

Proof. The proof follows from the fact that P_1 is $(1 \mapsto \Delta)$-output stable (Lemma 29 and Lemma 75) and therefore $P_2 \circ P_1$ is the Discrete Laplace mechanism executed on Δ-close inputs. The proof follows from Lemma 53. □

Lemma 55. *For any $d_{in} \in \mathbb{N}$, the Discrete Laplace program $P : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ (Program 4) is $(d_{in} \mapsto 5 \cdot d_{in})$-OC timing stable under the input distance metric $d_{\mathbb{N}}$ defined as $d_{\mathbb{N}}(x, x') = |x - x'|$.*

Proof. Conditioned on $\mathtt{out}(P(x, \mathtt{env})) = y$, the program has deterministic runtime. The program's conditional runtime $T_P(x, \mathtt{env})|_{\mathtt{out}(P(x,\mathtt{env}))=y}$ is equal to $15 + (5 \cdot |x - y|)$ where $5 \cdot |x - y|$ comes from the number of loops that are executed in lines (14-18). The other 15 instructions are always executed on every input (lines 1-10, 2 instructions depending on the branching condition in lines 10-13, 2 instructions depending on the branching condition in lines 19-22, and lines 23-24). Therefore, for all inputs $x, x' \in \mathbb{N}$, conditioned on output $y \in \mathbb{N}$,

$$\begin{aligned}
|T_P(x, \mathtt{env})|_{\mathtt{out}(P(x,\mathtt{env}))=y} - T_P(x, \mathtt{env})|_{\mathtt{out}(P(x,\mathtt{env}))=y}| &= |15 + 5 \cdot |x - y| - 15 - 5 \cdot |x' - y|| \\
&= |5 \cdot (|x - y| - |x' - y|)| \\
&\leq 5 \cdot (|x - x'|) \\
&= 5 \cdot d_{in}
\end{aligned}$$

□

Lemma 56. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be the Sum program (Program 2) and $P_2 : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be the Discrete Laplace program (Program 4). Then $P_2 \circ P_1$ is $(1 \mapsto 3 + 5\Delta)$-OC timing stable under the insert-delete distance metric d_{ID} on the input and the output distance metric $d_{\mathbb{N}}$ defined as $d_{\mathbb{N}}(y, y') = |y - y'|$.*

Proof. Since P_1 is $(1 \mapsto \{\Delta, 3\})$-jointly output/timing stable (Lemma 29), P_2 is $(d_{in} \mapsto 5 \cdot d_{in})$-OC timing stable (Lemma 55), and P_1 is chaining compatible with P_2 (by Lemma 46), then by Lemma 48, the chained program $P_2 \circ P_1$ is $(1 \mapsto 3 + 5 \cdot \Delta)$-OC-timing stable. □

Lemma 57. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be Program 2 (Sum), $P_2 : \mathbb{N} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ be Program 4 (Discrete Laplace) with scale parameter $s_1 = \Delta/\varepsilon_1$, and $P_3 : \mathcal{Z} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ be Program 3 (Timing-Private Delay) with scale parameter $s_2 = (3 + 5\Delta)/\varepsilon_2$ and shift μ. Then $P_3 \circ (P_2 \circ P_1)$ is ε_1-differentially private in its output and (ε_2, δ)-timing private for $\delta = 2 \cdot e^{-\varepsilon_2(\mu - (3+5\Delta))/(3+5\Delta)}$.*

Proof. The chained program $P_2 \circ P_1 : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ is ε_1-differentially private by Lemma 54. Since P_3 is a timing-private delay program (Lemma 42) it will compute a post-processing (identity function) on the output of $P_2 \circ P_1$ and by Lemma 51 the chained program $P_3 \circ (P_2 \circ P_1) : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ is ε_1-differentially private.

Since $P_2 \circ P_1$ is $(1 \mapsto 3 + 5\Delta)$-OC timing stable (Lemma 56), and P_3 is a $(3 + 5\Delta \mapsto (\varepsilon_2, \delta))$-timing-private delay program for $\delta = 2 \cdot e^{-\varepsilon_2(\mu - (3+5\Delta))/(3+5\Delta)}$ (Lemma 42) under input distance metric $d_{\mathbb{N}}$ and the smoothed max-divergence privacy measure D_∞^δ, then by Theorem 49, the chained program $P_3 \circ (P_2 \circ P_1)$ is $(1 \mapsto (\varepsilon_2, \delta))$-timing-private with respect to privacy measure D_∞^δ. □

Note that Lemma 57 provides an ε_1-DP mechanism that achieves ε_2-timing privacy in the upper-bounded DP model when the programs are Word RAM programs. Moreover, this construction is much more efficient than the naive approach of padding execution time to the worst-case dataset size $n_{\max} = 2^\omega - 1$, which would result in slow runtimes for datasets with size $n < n_{\max}$. In the RAM model of computation, the construction yields a timing-private DP sum in the unbounded DP model.

6.3 Composition of Timing-Private Programs

We can treat the composition of timing-private mechanisms similarly to how we treat chaining. In particular, our framework supports the composition of timing-private programs that perform intermediate DP computations.

Definition 58 (Composition-compatible Programs)**.** *We say that programs $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ and $P_2 : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ are* composition-compatible *if there is a program $P_2 \otimes P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{Y} \times \mathcal{Z}) \times \mathcal{E}$ and a constant c such that for all inputs $x \in \mathcal{X}$ and all execution environments $\texttt{env}_1 \in \mathcal{E}$, there exists a (possibly random) input-compatible execution environment $\texttt{env}_2 \in \mathcal{E}$ such that we have:*

$$\texttt{out}((P_2 \otimes P_1)(x, \texttt{env}_1)) = (\texttt{out}(P_1(x, \texttt{env}_1)), \texttt{out}(P_2((x, \texttt{out}(P_1(x, \texttt{env}_1))), \texttt{env}_2)))$$

and

$$T_{(P_2 \otimes P_1)}(x, \texttt{env}_1) = T_{P_1}(x, \texttt{env}_1) + T_{P_2}((x, \texttt{out}(P_1(x, \texttt{env}_1))), \texttt{env}_2) + c.$$

Similar to the case of chaining-compatible programs, we allow the random variable $\texttt{env}_2$ to be arbitrarily correlated with the output and runtime of $P_1(x, \texttt{env}_1)$, but it should be independent of the coin tosses of P_2.

Lemma 59 (Composition-compatible Word RAM Programs)**.** *Let program $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ be a RAM program that writes its unmodified input to*

$$M[\texttt{output\_ptr}], \ldots, M[\texttt{output\_ptr} + \texttt{input\_len} - 1]$$

and appends the rest of its output to $M[\texttt{output\_ptr} + \texttt{input\_len}], \ldots, M[\texttt{output\_ptr} + \texttt{output\_len} - 1]$. Then P_1 and P_2 are composition-compatible *for all programs $P_2 : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$.*

Proof. The proof for composition-compatible programs works similarly to that of chaining-compatible programs. We replace the `HALT` at the end of P_1 to set $\texttt{input\_len} = \texttt{output\_len}$ and $\texttt{input\_ptr} = \texttt{output\_ptr}$. The resulting program is an execution of P_2 on input (x, y) in environment $\texttt{env}_2$ that is the same as $\texttt{outenv}(P_1(x, \texttt{env}_1))$ except for these modifications to `input_ptr` and `input_len`. □

Remark 60. *The proof follows for RAM programs also.*

Lemma 61 (Composition of OC-Timing-Stable Programs)**.** *Let $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ be $(d_1 \mapsto t_1)$-OC timing stable with respect to the $\mathcal{Y}$ output coordinate*[8] *and input distance metric $d_{\mathcal{X}}$. Similarly, let $P_2 : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E} \to (\mathcal{Y} \times \mathcal{Z}) \times \mathcal{E}$ be $(d_1 \mapsto t_2)$-OC timing stable with respect to input distance metric $d_{\mathcal{X}}$ on its first input coordinate. If P_1 and P_2 are composition compatible, then the composed program $P_2 \otimes P_1$ is $(d_1 \mapsto t_1 + t_2)$-OC timing stable.*

[8] This means that we condition only on the $y \in \mathcal{Y}$ part of the output when considering OC-timing stability.

Proof. For all $x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_1$, all input-compatible environments $\mathtt{env}_1, \mathtt{env}_1' \in \mathcal{E}$, and all $(y, z) \in \mathrm{supp}(\mathtt{out}((P_2 \otimes P_1)(x, \mathtt{env}_1))) \cap \mathrm{supp}(\mathtt{out}((P_2 \otimes P_1)(x', \mathtt{env}_1')))$, we will construct a coupling $(\tilde{r}, \tilde{r}')$ of $T_{(P_2 \otimes P_1)}(x, \mathtt{env}_1)|_{\mathtt{out}((P_2 \otimes P_1)(x, \mathtt{env}_1))=(y,z)}$ and $T_{(P_2 \otimes P_1)}(x', \mathtt{env}_1')|_{\mathtt{out}((P_2 \otimes P_1)(x', \mathtt{env}_1'))=(y,z)}$ such that $|\tilde{r} - \tilde{r}'| \leq t_1 + t_2$ with probability 1.

Observe that for all $x, x' \in \mathcal{X}$ such that $d_{\mathcal{X}}(x, x') \leq d_1$, all $\mathtt{env}_1, \mathtt{env}_1' \in \mathcal{E}$, and all $(\cdot, y) \in \mathrm{supp}(\mathtt{out}(P_1(x, \mathtt{env}_1))) \cap \mathrm{supp}(\mathtt{out}(P_1(x', \mathtt{env}_1')))$, there exists a coupling $(\tilde{r}_1, \tilde{r}_1')$ of the conditional random variables $T_{P_1}(x, \mathtt{env}_1)|_{\mathtt{out}(P_1(x, \mathtt{env}_1))=(x,y)}$ and $T_{P_1}(x', \mathtt{env}_1')|_{\mathtt{out}(P_1(x', \mathtt{env}_1'))=(x',y)}$ such that $|\tilde{r}_1 - \tilde{r}_1'| \leq t_1$ (by the OC timing stability of P_1).

For all such x, x' and $\mathtt{env}_1, \mathtt{env}_1'$, take $\mathtt{env}_2$ and $\mathtt{env}_2'$ to be the execution environments satisfying:

$$T_{(P_2 \otimes P_1)}(x, \mathtt{env}_1) = T_{P_1}(x, \mathtt{env}_1) + T_{P_2}((x, \mathtt{out}(P_1(x, \mathtt{env}_1))), \mathtt{env}_2) + c$$

and

$$T_{(P_2 \otimes P_1)}(x', \mathtt{env}_1') = T_{P_1}(x', \mathtt{env}_1') + T_{P_2}((x', \mathtt{out}(P_1(x', \mathtt{env}_1'))), \mathtt{env}_2') + c$$

respectively (such an $\mathtt{env}_2$ and $\mathtt{env}_2'$ exist due to P_1 and P_2 being composition-compatible). Now fix $y \in \mathrm{supp}(\mathtt{out}(P_1(x, \mathtt{env}_1))) \cap \mathrm{supp}(\mathtt{out}(P_1(x', \mathtt{env}_1')))$. Then for all z such that

$$(y, z) \in \mathrm{supp}(\mathtt{out}(P_2((x, y), \mathtt{env}_2))) \cap \mathrm{supp}(\mathtt{out}(P_2((x', y), \mathtt{env}_2')))$$

there exists a coupling $(\tilde{r}_2, \tilde{r}_2')$ of the random variables $T_{P_2}((x, y), \mathtt{env}_2)|_{\mathtt{out}(P_2((x,y), \mathtt{env}_2))=(y,z)}$ and $T_{P_2}((x', y), \mathtt{env}_2')|_{\mathtt{out}(P_2((x',y), \mathtt{env}_2'))=(y,z)}$ such that $|\tilde{r}_2 - \tilde{r}_2'| \leq t_2$ (by the OC timing stability of P_2).

We sample $(r_1, r_1') \sim (\tilde{r}_1, \tilde{r}_1')$ and sample $\mathtt{env}_2$ conditioned on (r_1, y) and $\mathtt{env}_2'$ conditioned on (r_1', y). We then samlpe $(r_2, r_2') \sim (\tilde{r}_2, \tilde{r}_2')|_{\mathtt{env}_2, \mathtt{env}_2'}$ and let

$$\begin{aligned}
\tilde{r} &= r_1 + r_2 + c \\
&= T_{P_1}(x, \mathtt{env}_1)|_{\mathtt{out}(P_1(x, \mathtt{env}_1))=y} + T_{P_2}((x, y), \mathtt{env}_2)|_{\mathtt{out}(P_2((x,y), \mathtt{env}_1))=(y,z)} + c \\
&= T_{(P_2 \otimes P_1)}(x, \mathtt{env}_1)|_{\mathtt{out}((P_2 \otimes P_1)(x, \mathtt{env}_1))=(y,z)}
\end{aligned}$$

and similarly for $\tilde{r}'$, where c is the constant in Definition 58. It follows that

$$\begin{aligned}
|\tilde{r} - \tilde{r}'| &= |r_1 + r_2 + c - r_1' - r_2' - c| \\
&= |r_1 - r_1' + r_2 - r_2'| \\
&\leq |r_1 - r_1'| + |r_2 - r_2'| \\
&\leq t_1 + t_2
\end{aligned}$$

□

Lemma 61 shows that we can reason about timing stability when composing DP programs. After analyzing the timing stability of the overall composed program, we can add a single timing delay to protect the release from timing attacks. Alternatively, we can compose timing-private programs such that composed program is also timing-private.

Lemma 62 (Composition of Timing-Private Programs)**.** *Let $\mathcal{C} : (\mathcal{M} \times \mathcal{M}) \to \mathcal{M}$ be a valid composition function for a privacy measure M; meaning that if $M(X_1, X_1') \leq d_1$ and $M(X_2|X_1 = x, X_2'|X_1' = x') \leq d_2$ then $M((X_1, X_2), (X_1', X_2')) \leq C(d_1, d_2)$. Let $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ be $(d_0 \mapsto d_1)$-timing private with respect to its second output coordinate, input metric $d_{\mathcal{X}}$, and privacy*

measure M. Let $P_2 : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E} \to (\mathcal{Y} \times \mathcal{Z}) \times \mathcal{E}$ be $(d_0 \mapsto d_2)$-timing private with respect to its second output coordinate, input metric $d_{\mathcal{X}}$, and privacy measure M. If P_1 and P_2 be composition compatible, then $P_1 \otimes P_2 : \mathcal{X} \times \mathcal{E} \to (\mathcal{Y} \times \mathcal{Z}) \times \mathcal{E}$ is $(d_0 \mapsto C(d_1, d_2))$-timing-private with respect to $d_{\mathcal{X}}$ and privacy measure M.

Proof. By composition-compatibility, for all $x \in \mathcal{X}$ and input-compatible execution environments $\texttt{env}_1$, there exists an input-compatible execution environment $\texttt{env}_2$ such that

$$T_{P_2 \otimes P_1}(x, \texttt{env}_1) = T_{P_1}(x, \texttt{env}_1) + T_{P_2}((x, \texttt{out}(P_1(x, \texttt{env}_1))), \texttt{env}_2) + c$$

We can therefore analyze $T_{P_2 \otimes P_1}(x, \texttt{env}_1)|_{\texttt{out}((P_2 \otimes P_1)(x, \texttt{env}_1) = (y,z)}$ using the joint random variable:

$$(T_{P_1}(x, \texttt{env}_1)|_{\texttt{out}(P_1(x, \texttt{env}_1)) = (x,y)}, T_{P_2}((x, y), \texttt{env}_2)|_{\texttt{out}(P_2((x,y), \texttt{env}_2)) = (y,z)})$$

and similarly for x'. Since P_1 is timing-private with respect to input metric $d_{\mathcal{X}}$ and privacy measure M, it follows that for all d_1-close inputs $x, x' \in \mathcal{X}$, all input-compatible execution environments $\texttt{env}_1, \texttt{env}_1'$, and all $y \in \text{supp}(\texttt{out}(P_1(x, \texttt{env}_1))) \cap \text{supp}(\texttt{out}(P_1(x', \texttt{env}_1')))$

$$M(T_{P_1}(x, \texttt{env}_1)|_{\texttt{out}(P_1(x, \texttt{env}_1)) = (x,y)}, T_{P_1}(x', \texttt{env}_1')|_{\texttt{out}(P_1(x', \texttt{env}_1')) = (x', y)}) \leq d_1$$

Similarly, since P_2 is timing-private with respect to input metric $d_{\mathcal{X}}$ and privacy measure M, it follows that for all inputs $(x, y), (x', y) \in \mathcal{X} \times \mathcal{Y}$ that are d_1-close on their first coordinate, all input-compatible execution environments $\texttt{env}_2, \texttt{env}_2'$, and all $(y, z) \in \text{supp}(\texttt{out}(P_2((x, y), \texttt{env}_2))) \cap \text{supp}(\texttt{out}(P_2((x', y), \texttt{env}_2')))$

$$M(T_{P_2}((x, y), \texttt{env}_2)|_{\texttt{out}(P_2((x,y), \texttt{env}_2)) = (y,z)}, T_{P_2}((x', y), \texttt{env}_2')|_{\texttt{out}(P_2((x', y), \texttt{env}_2')) = (y,z)}) \leq d_2$$

Let

$$\begin{aligned}
Y &= T_{P_1}(x, \texttt{env}_1)|_{\texttt{out}(P_1(x, \texttt{env}_1)) = (x,y)} \\
Y' &= T_{P_1}(x', \texttt{env}_1')|_{\texttt{out}(P_1(x', \texttt{env}_1')) = (x', y)} \\
Z &= T_{P_2}((x, y), \texttt{env}_2)|_{\texttt{out}(P_2((x,y), \texttt{env}_2)) = (y,z)} \\
Z' &= T_{P_2}((x', y), \texttt{env}_2')|_{\texttt{out}(P_2((x', y), \texttt{env}_2')) = (y,z)}
\end{aligned}$$

Then

$$\begin{aligned}
M((Y, Z), (Y', Z')) &\leq C(M(Y, Y'), M(Z, Z')) \\
&\leq C(d_1, d_2)
\end{aligned}$$

which is what we wanted to show. □

The above composition theorem says that we can compose timing-private programs and keep track of the overall timing privacy guarantees. For example, $C((\varepsilon_1, \delta_1), (\varepsilon_2, \delta_2)) = (\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$ is a valid composition function for approximate-DP (Lemma 23). Lemma 62 says that the composition of a program P_1 that is $(\varepsilon_1, \delta_1)$-timing private with a program P_2 that is $(\varepsilon_2, \delta_2)$-timing private, results in a new program that is $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-timing private.

We can use the above fact to construct timing-private programs, for example, that compute DP means from a timing-private DP sum program and a timing-private DP count program. The DP sum and DP count can be interpreted as the numerator and denominator of the mean, respectively. For example, Program 5 accepts as input (x, y) where x is a dataset and y is a DP sum over x. By chaining together Program 5 with a (modified) Program 4, we obtain a program that outputs a DP sum followed by a DP count.

Program 5 Dataset Count Program

Input: A dataset x occupying memory locations $M[0], \ldots, M[\texttt{input\_len} - 2]$ and a value $y \in \mathbb{N}$ at memory location $M[\texttt{input\_len} - 1]$. The value y can be a (possibly DP) sum over the elements $M[0], \ldots, M[\texttt{input\_len} - 1]$, for example.

Output: The value y and the size of the dataset $\texttt{input\_len} - 1$.

1: $\texttt{output\_ptr} = \texttt{input\_len} - 2$;
2: $\texttt{output\_len} = 2$;
3: $M[\texttt{output\_ptr} + 1] = \texttt{input\_len} - 1$;
4: HALT;

6.4 A Timing Private Unbounded Mean

We now give an example of how one can use composition-compatible programs to compute timing-private DP means in the unbounded DP model.

Remark 63. *In the following constructions, we will assume a modified version of the timing-private DP sum program from Lemma 57 that instead writes its output in a composition-compatible way (by appending its output to the end of its input and setting* ***output_ptr*** *and* ***output_len*** *accordingly). This is a straight-forward modification that involves updating Program 2 and Program 4 so that the original input is included in the chained program's output. Similarly, when we chain together Program 4 with Program 5, we will assume a modified version of the Discrete Laplace program that accepts as input (y_1, y_2) and outputs $(y_1, y_2 + \eta)$ where η is the noise drawn from the Discrete Laplace distribution. We note that these changes can be made such that the stability claims for Lemma 55 and Lemma 29 continue to hold by ensuring that the modifications incur some constant additive overhead to the runtime.*

Lemma 64. *The Dataset Count program $P : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ (Program 5) is $(d_{in} \to \{d_{in}, 0\})$-jointly output/timing stable under the output distance metric $d_{\mathbb{N}}$ on the second output coordinate of $(\mathbb{N} \times \mathbb{N})$ and the input distance metric d_{ID}.*

Proof. The program always executes in 4 instructions and is therefore $(d_{in} \mapsto 0)$-timing stable under the input distance metric d_{ID} (Lemma 70). If you add or remove a row from the dataset $x \in \mathcal{X}$, then the count changes by at most d_{in}. Therefore, the program is $(d_{in} \mapsto d_{in})$-output stable (with respect to its second output coordinate) under the output distance metric $d_{\mathbb{N}}$ and input metric d_{ID}. Since the program is deterministic in its output, the program satisfies $(d_{in} \mapsto \{d_{in}, 0\})$-joint output/timing stability (Lemma 28). □

Lemma 65. *Let $P_1 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Dataset Count program (Program 5) and $P_2 : (\mathbb{N} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Discrete Laplace program (Program 4) modified according to Remark 63 with $s = 1/\varepsilon_2 = 1/\ln(\mathtt{b}/\mathtt{a})$. Then $P_2 \circ P_1 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ is ε_2-differentially private under the max-divergence privacy measure with respect to the 2nd coordinate of $(\mathbb{N} \times \mathbb{N})$ and the input distance d_{ID}.*

Proof. P_1 is $(d_{in} \mapsto d_{in})$-output stable with respect to input distance metric d_{ID} and output distance metric $d_{\mathbb{N}}$ on the second output coordinate of $(\mathbb{N} \times \mathbb{N})$ (Lemma 64 and Lemma 75). Therefore the proof follows exactly by Lemma 53 since $P_2 \circ P_1$ is the Discrete Laplace program executed on d_{in}-close inputs under the input distance $d_{\mathbb{N}}(n, n') = |n - n'|$. □

Lemma 66. *Let $P_1 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Dataset Count program (Program 5), and $P_2 : (\mathbb{N} \times \mathbb{N}) \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Discrete Laplace program (Program 4) modified according to Remark 63. Then the chained program $P_2 \circ P_1 : (\mathcal{X} \times \mathbb{N}) \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ is $(d_{in} \mapsto 5 \cdot d_{in})$-OC timing stable under the insert-delete distance metric d_{ID}.*

Proof. The dataset count program is constant-time and therefore $(d_{in} \mapsto \{d_{in}, 0\})$-jointly output/timing stable under d_{ID}. By Lemma 55 and Lemma 48, we have that the chained program is $(d_{in} \mapsto 5 \cdot d_{in})$-OC timing stable under d_{ID}. □

Lemma 67. *Let $P_1 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Dataset Count program (Program 5), and $P_2 : (\mathbb{N} \times \mathbb{N}) \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the Discrete Laplace program (Program 4) modified according to Remark 63, with scale parameter $s_1 = 1/\varepsilon_1$. Let $P_3 : \mathcal{Z} \times \mathcal{E} \to \mathcal{Z} \times \mathcal{E}$ be Program 3 (Timing-Private Delay) with scale parameter $s_2 = 5/\varepsilon_2$ and shift μ. Then $P_3 \circ (P_2 \circ P_1)$ is ε_1-differentially private in its output and (ε_2, δ)-timing private for $\delta = 2 \cdot e^{\varepsilon_2(\mu-5)/5}$.*

Proof. The chained program $P_2 \circ P_1 : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ is ε_1-differentially private by Lemma 65. Since P_3 is a timing-private delay program (Lemma 42) it will compute a post-processing (identity function) on the output of $P_2 \circ P_1$ and by Lemma 51 the chained program $P_3 \circ (P_2 \circ P_1) : \mathcal{X} \times \mathcal{E} \to \mathbb{N} \times \mathcal{E}$ is ε_1-differentially private.

Since $P_2 \circ P_1$ is $(1 \mapsto 5)$-OC timing stable (Lemma 66), and P_3 is a $(5 \mapsto (\varepsilon_2, \delta))$-timing-private delay program for $\delta = 2 \cdot e^{-\varepsilon_2(\mu-5)/5}$ (Lemma 42) under input distance metric $d_{\mathbb{N}}$ and the smoothed max-divergence privacy measure D^{δ}_{∞}, then by Theorem 49, the chained program $P_3 \circ (P_2 \circ P_1)$ is $(1 \mapsto (\varepsilon_2, \delta))$-timing-private with respect to privacy measure D^{δ}_{∞}. □

Lemma 68. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ be the ε_1-DP and (ε_2, δ)-timing-private sum program (Lemma 57) that has been modified according to Remark 63. Let $P_2 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the ε_1-DP and (ε_2, δ)-timing-private dataset count program (Lemma 67). Then P_1 and P_2 are composition-compatible.*

Proof. Observe that, due to the modifications from Remark 63, P_1 writes to its output the original dataset $x \in \mathcal{X}$ followed by the DP sum $y \in \mathbb{N}$. By Lemma 59 the claim follows. □

Lemma 69. *Let $P_1 : \mathcal{X} \times \mathcal{E} \to (\mathcal{X} \times \mathcal{Y}) \times \mathcal{E}$ be the ε_1-DP and (ε_2, δ)-timing-private sum program (Lemma 57) that has been modified according to Remark 63. Let $P_2 : (\mathcal{X} \times \mathbb{N}) \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ be the ε_1-DP and (ε_2, δ)-timing-private dataset count program (Lemma 67). Then $P_2 \otimes P_1 : \mathcal{X} \times \mathcal{E} \to (\mathbb{N} \times \mathbb{N}) \times \mathcal{E}$ is $2\varepsilon_1$-DP and $(2\varepsilon_2, 2\delta)$-timing-private.*

Proof. The first output coordinate of $P_2 \otimes P_1$ is the output of P_1, and the second output coordinate of $P_2 \otimes P_1$ is the output of P_2. Since P_1 is ε_1-DP under input metric d_{ID} and output metric $d_{\mathbb{N}}$ (Lemma 54), and P_2 is ε_1-DP under input metric d_{ID} and output metric $d_{\mathbb{N}}$ (Lemma 65), it follows that for all $x, x' \in \mathcal{X}$ satisfying $d_{\mathrm{ID}}(x, x') \leq 1$, all input-compatible $\mathtt{env}_1, \mathtt{env}'_1, \mathtt{env}_2, \mathtt{env}'_2 \in \mathcal{E}$, and all $(S_1, S_2) \subseteq (\mathbb{N} \times \mathbb{N})$:

$$
\begin{aligned}
&\Pr[\mathtt{out}((P_2 \otimes P_1)(x, \mathtt{env}_1)) \in (S_1, S_2)] \\
&\quad = \Pr[\mathtt{out}(P_1(x, \mathtt{env}_1)) \in S_1] \cdot \Pr[\mathtt{out}(P_2((x, \mathtt{out}(P_1(x, \mathtt{env}_1))), \mathtt{env}_2)) \in S_2] \\
&\quad \leq e^{\varepsilon_1} \cdot \Pr[\mathtt{out}(P_1(x', \mathtt{env}'_1)) \in S_1] \cdot e^{\varepsilon_2} \cdot \Pr[\mathtt{out}(P_2((x', \mathtt{out}(P_1(x', \mathtt{env}'_1))), \mathtt{env}'_2)) \in S_2] \\
&\quad \leq e^{\varepsilon_1 + \varepsilon_2} \cdot \Pr[\mathtt{out}((P_2 \otimes P_1)(x', \mathtt{env}'_1)) \in (S_1, S_2)]
\end{aligned}
$$

Therefore, $P_2 \otimes P_1$ is $2\varepsilon_1$-DP with respect to input metric d_{ID} and the max-divergence privacy measure. The claim that $P_2 \otimes P_1$ is $(2\varepsilon_2, 2\delta)$-timing-private follows from the fact that P_1 is (ε_2, δ)-timing-private, P_2 is (ε_2, δ)-timing-private, and Lemma 62 and letting $C((\varepsilon, \delta), (\varepsilon, \delta)) = (2\varepsilon, 2\delta)$ be the composition function by Lemma 23. □

7 Implementation

A proof-of-concept implementation of our framework on top of the OpenDP library in available in the Github repository [SVM+]. The proof of concept supports defining timing stability maps for transformations and output-conditional timing stability maps for measurements. We additionally implemented a proof-of-concept timing-private delay function that can be chained with arbitrary output-conditional timing stable measurements to delay their output release.

The purpose of the implementation is to illustrate the compatibility of our framework with existing differential privacy libraries, not to claim that the implementation provides timing privacy for physical executions. As discussed in Section 1.2, we leave for future work the problem of instantiating our framework for physical timing channels, which would involve constraining the execution environment, identifying the appropriate units to measure timing, and finding realistic upper bounds on actual timing-stability constants.

8 Acknowledgments

The authors would like to thank Mike Shoemate for his help with implementing the proof-of-concept in the OpenDP library. This work was supported in part by Cooperative Agreement CB20ADR0160001 with the Census Bureau, and in part by Salil Vadhan's Simons Investigator Award.

References

[ABC+20] Ahmet Aktay, Shailesh Bavadekar, Gwen Cossoul, John Davis, Damien Desfontaines, Alex Fabrikant, Evgeniy Gabrilovich, Krishna Gadepalli, Bryant Gipson, Miguel Guevara, et al. Google covid-19 community mobility reports: anonymization process description (version 1.1). *arXiv preprint arXiv:2004.04145*, 2020.

[Abo18] John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2867–2867, 2018.

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[AKM+15] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*, pages 623–639. IEEE, 2015.

[AMS+22] Daniel Alabi, Audra McMillan, Jayshree Sarathy, Adam Smith, and Salil Vadhan. Differentially private simple linear regression. *Proceedings on Privacy Enhancing Technologies*, 2:184–204, 2022.

[AR23] Jordan Awan and Vinayak Rao. Privacy-aware rejection sampling. *Journal of machine learning research*, 24(74):1–32, 2023.

[BBD+22] Skye Berghel, Philip Bohannon, Damien Desfontaines, Charles Estes, Sam Haney, Luke Hartman, Michael Hay, Ashwin Machanavajjhala, Tom Magerlein, Gerome Miklau, et al. Tumult analytics: a robust, easy-to-use, scalable, and expressive framework for differential privacy. *arXiv preprint arXiv:2212.04133*, 2022.

[BDDNT23] Yoav Ben Dov, Liron David, Moni Naor, and Elad Tzalik. Resistance to timing attacks for sampling and privacy preserving schemes. In *4th Symposium on Foundations of Responsible Computing (FORC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[BNS+16] Raef Bassily, Kobbi Nissim, Adam Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. Algorithmic stability for adaptive data analysis. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1046–1059, 2016.

[BS16] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

[BV18] Victor Balcer and Salil Vadhan. Differential privacy on finite computers. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[CKS20] Clément L Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. *Advances in Neural Information Processing Systems*, 33:15676–15688, 2020.

[CSVW22] Sílvia Casacuberta, Michael Shoemate, Salil Vadhan, and Connor Wagaman. Widespread underestimation of sensitivity in differentially private libraries and how to fix it. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 471–484, 2022.

[DFH+15] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 117–126, 2015.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 486–503. Springer, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam 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 3*, pages 265–284. Springer, 2006.

[DR16] Cynthia Dwork and Guy N Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

[DRS22] Jinshuo Dong, Aaron Roth, and Weijie J Su. Gaussian differential privacy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):3–37, 2022.

[G+16] Andy Greenberg et al. Apple's "differential privacy" is about collecting your data–but not your data. *Wired, June*, 13(1), 2016.

[GHV20] Marco Gaboardi, Michael Hay, and Salil Vadhan. A programming framework for opendp. *Manuscript, May*, 2020.

[GRS12] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[HPN11] Andreas Haeberlen, Benjamin C Pierce, and Arjun Narayan. Differential privacy under fire. In *USENIX Security Symposium*, volume 33, page 236, 2011.

[JMRO22] Jiankai Jin, Eleanor McMurtry, Benjamin IP Rubinstein, and Olga Ohrimenko. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 473–488. IEEE, 2022.

[LGZ18] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 711–725, 2018.

[McS09] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.

[MDH+20] Solomon Messing, Christina DeGregorio, Bennett Hillenbrand, Gary King, Saurav Mahanti, Zagreb Mukerjee, Chaya Nayak, Nate Persily, Bogdan State, and Arjun Wilkins. Facebook Privacy-Protected Full URLs Data Set, 2020.

[Mir12] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 650–661, 2012.

[Mir17] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017.

[MKA+08] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *2008 IEEE 24th international conference on data engineering*, pages 277–286. IEEE, 2008.

[MPRV09] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, pages 126–142. Springer, 2009.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103. IEEE, 2007.

[NR18] Seth Neel and Aaron Roth. Mitigating bias in adaptive data gathering via differential privacy. In *International Conference on Machine Learning*, pages 3720–3729. PMLR, 2018.

[PSM+] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. Scalable private learning with pate. In *International Conference on Learning Representations*.

[RNM+21] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 327–343, 2021.

[RZHP20] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C Pierce. Orchard: Differentially private analytics at scale. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 1065–1081, 2020.

[Sta03] William Stallings. *Computer organization and architecture: designing for performance*. Pearson Education India, 2003.

[SVM+] Michael Shoemate, Andrew Vyrros, Chuck McCallum, Raman Prasad, Philip Durbin, Sílvia Casacuberta Puig, Ethan Cowan, Vicki Xu, Zachary Ratliff, Nicolás Berrios, Alex Whitworth, Michael Eliot, Christian Lebeda, Oren Renard, and Claire McKay Bowen. OpenDP Library.

[TGL+17] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[V+09] Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

[VDHLZZ15] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

A Stability Proofs

Lemma 70. *A program $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is* $(d_{in} \mapsto 0)$-timing-stable *if for all pairs of inputs $x, x' \in \mathcal{X}$, and all input-compatible execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, $T_P(x, \texttt{env}) \equiv T_P(x', \texttt{env}')$.*

Proof. For any two inputs x and x', take the coupling $(\tilde{r}, \tilde{r}')$ to be $(T_P(x, \texttt{env}), T_P(x, \texttt{env}))$. Since $T_P(x, \texttt{env}) \equiv T_P(x', \texttt{env}')$, it follows that the marginal distributions of $\tilde{r}$ and $\tilde{r}'$ are identical to $T_P(x, \texttt{env})$ and $T_P(x', \texttt{env}')$, and $|\tilde{r} - \tilde{r}'| = 0$. □

Lemma 71. *If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ satisfies for all $x \in \mathcal{X}$, $\texttt{env} \in \mathcal{E}$, $T_P(x, \texttt{env}) = c$ for some constant c, then P is* $(d_{in} \mapsto 0)$-timing stable.

Proof. Since P is constant time, $T_P(x, \texttt{env}) = c$ for all x and $\texttt{env}$, then for all x, x' and input-compatible $\texttt{env}, \texttt{env}'$, $T_P(x, \texttt{env}) \equiv T_P(x', \texttt{env}')$ and the claim follows from Lemma 70. □

Lemma 72. *If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is deterministic (in its output) and is* $(d_{in} \mapsto t_{out})$-timing stable*, then P is* $(d_{in} \mapsto t_{out})$-OC-timing stable.

Proof. For all x and x' satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$, and any pair of environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, either $\texttt{out}(P(x, \texttt{env})) \neq \texttt{out}(P(x', \texttt{env}'))$ or $\texttt{out}(P(x, \texttt{env})) = \texttt{out}(P(x', \texttt{env}'))$. If $\texttt{out}(P(x, \texttt{env})) \neq \texttt{out}(P(x', \texttt{env}'))$, then there is no requirement on the distributional closeness of $T_P(x, \texttt{env})$ and $T_P(x', \texttt{env}')$ and the claim is satisfied. Now suppose that $\texttt{out}(P(x, \texttt{env})) = \texttt{out}(P(x', \texttt{env}')) = y$. By the timing stability of P there exists a coupling $(\tilde{r}, \tilde{r}')$ of the random variables $T_P(x, \texttt{env})$ and $T_P(x', \texttt{env}')$ such that $|\tilde{r} - \tilde{r}'| \leq t_{out}$. Since P is deterministic it also follows that $\tilde{r}$ and $\tilde{r}'$ have the same marginal distributions as $T_P(x, \texttt{env})_{|\texttt{out}(P(x,\texttt{env}))=y}$ and $T_P(x')_{|\texttt{out}(P(x',\texttt{env}'))=y}$ respectively, and the claim holds. □

Lemma 73. *If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ has constant runtime then P is* $(d_{in} \to 0)$-OC timing stable.

Proof. Since P is constant time, $T_P(x, \texttt{env}) = c$ for all x and $\texttt{env}$. For all $y \in \text{supp}(\texttt{out}(P(x, \texttt{env}))) \cap \text{supp}(\texttt{out}(P(x', \texttt{env}')))$ with $d_{\mathcal{X}}(x, x') \leq d_{in}$, let the coupling $(\tilde{r}, \tilde{r}')$ of $T_P(x, \texttt{env})_{|\texttt{out}(P(x,\texttt{env}))=y}$ and $T_P(x', \texttt{env}')_{|\texttt{out}(P(x',\texttt{env}'))=y}$ take the value (c, c). Then $\tilde{r}$ and $\tilde{r}'$ are identically distributed to $T_P(x, \texttt{env})_{|\texttt{out}(P(x,\texttt{env}))=y}$ and $T_P(x', \texttt{env}')_{|\texttt{out}(P(x',\texttt{env}'))=y}$ respectively, and $|\tilde{r} - \tilde{r}'| = |c - c| = 0$. □

Lemma 74. *Jointly output/timing-stable programs are timing-stable programs.*

Proof. If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_{in} \mapsto \{d_{out}, t_{out}\})$-jointly output/timing stable with respect to input metric $d_{\mathcal{X}}$, then for every $x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$ and all pairs of execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, there exists a coupling $((\tilde{u}, \tilde{r}), (\tilde{u}', \tilde{r}'))$ of the joint random variables $(\texttt{out}(P(x, \texttt{env})), T_P(x, \texttt{env}))$ and $(\texttt{out}(P(x', \texttt{env}')), T_P(x', \texttt{env}'))$ satisfying $\Pr[|\tilde{r} - \tilde{r}'| < t_{out}] = 1$. Take $(\tilde{r}, \tilde{r}')$ to be the coupling of the random variables $T_P(x, \texttt{env})$ and $T_P(x', \texttt{env}')$ and the claim follows. □

Lemma 75. *Jointly output/timing-stable programs are output-stable programs.*

Proof. If $P : \mathcal{X} \times \mathcal{E} \to \mathcal{Y} \times \mathcal{E}$ is $(d_{in} \mapsto \{d_{out}, t_{out}\})$-jointly output/timing stable with respect to input metric $d_{\mathcal{X}}$ and output metric $d_{\mathcal{Y}}$, then for every $x, x' \in \mathcal{X}$ satisfying $d_{\mathcal{X}}(x, x') \leq d_{in}$ and all pairs of execution environments $\texttt{env}, \texttt{env}' \in \mathcal{E}$, there exists a coupling $((\tilde{u}, \tilde{r}), (\tilde{u}', \tilde{r}'))$ of the joint random variables $(\texttt{out}(P(x, \texttt{env})), T_P(x, \texttt{env}))$ and $(\texttt{out}(P(x', \texttt{env}')), T_P(x', \texttt{env}'))$ satisfying $d_{\mathcal{Y}}(\tilde{u}, \tilde{u}') \leq d_{out}$ with probability 1. Take $(\tilde{u}, \tilde{u}')$ to be the coupling of the random variables $\texttt{out}(P(x, \texttt{env}))$ and $\texttt{out}(P(x', \texttt{env}'))$ and the claim follows. □