Resistance to Timing Attacks for Sampling and Privacy Preserving Schemes

Yoav Ben Dov
Weizmann Institute of Science, Rehovot, Israel

Liron David
Weizmann Institute of Science, Rehovot, Israel

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Elad Tzalik
Weizmann Institute of Science, Rehovot, Israel

Abstract

Side channel attacks, and in particular timing attacks, are a fundamental obstacle for secure implementation of algorithms and cryptographic protocols. These attacks and countermeasures have been widely researched for decades. We offer a new perspective on resistance to timing attacks.

We focus on sampling algorithms and their application to differential privacy. We define sampling algorithms that do not reveal information about the sampled output through their running time. More specifically: (1) We characterize the distributions that can be sampled from in a "time oblivious" way, meaning that the running time does not leak any information about the output. We provide an optimal algorithm in terms of randomness used to sample for these distributions. We give an example of an efficient randomized algorithm $\mathcal{A}$ such that there is no subexponential algorithm with the same output as $\mathcal{A}$ that does not reveal information on the output or the input, therefore we show leaking information on either the input or the output is unavoidable. (2) We consider the impact of timing attacks on (pure) differential privacy mechanisms. It turns out that if the range of the mechanism is unbounded, such as counting, then any *time oblivious* pure DP mechanism must give a useless output with constant probability (the constant is mechanism dependent) and must have infinite expected running time. We show that up to this limitations it is possible to transform *any* pure DP mechanism into a time oblivious one.

2012 ACM Subject Classification Mathematics of computing → Random number generation; Theory of computation → Cryptographic primitives; Theory of computation → Generating random combinatorial structures

Keywords and phrases Differential Privacy

Digital Object Identifier 10.4230/LIPIcs.FORC.2023.11

Funding Research supported in part by grants from the Israel Science Foundation (no.2686/20), by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.

1 Introduction

There is always a gap between the way an algorithm is specified and described mathematically and how it is implemented in a physical device and environment. Physical systems often leak information to the environment, for example the power usage, heat radiation, running time and much more. This leakage, in turn, can make systems which are secure in the theoretical model, susceptible to attacks in practice which make them completely insecure. These attacks are called "side channel attacks." In this work we focus on timing attacks, i.e. attacks that exploit the running time leakage.

4th Symposium on Foundations of Responsible Computing (FORC 2023).
Editor: Kunal Talwar; Article No. 11; pp. 11:1–11:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One important field which is sensitive to timing attacks is *Differential Privacy* (DP) [9, 10]. DP deals with analyzing data sets in a way which protects the privacy of an individual contributor to the collection. Informally, for an algorithm to be differentially private it needs to have "close" output distributions on two data sets which differ by a single entity[1]. In this work we consider the question of defining security against timing attacks. To this end, we define new notions of resistance to timing attacks in the realm of sampling and differential privacy, provide resistant constructions, and prove their security.

We formally define *time oblivious* DP mechanisms. We show that time oblivious pure DP mechanisms have some undesirable properties and therefore the recommendation is to use approximate DP in practice. Nevertheless, we show that if one can tolerate those properties in applications, then any pure DP mechanism can be transformed to a time oblivious pure DP mechanism with similar privacy guarantee.

1.1 A Very Brief History of Timing Attacks

Side channel attacks and in particular timing attacks have long history and we would not attempt to survey it (see the companion paper [6] for more details). For instance, an early work by Lipton and Naughton [21] showed a way to exploit timing information to compromise the performance of dictionaries that employ universal hash functions. The work of Kocher [20], showing how the running time of certain implementations of RSA and Diffie Hellman schemes leaks information which can be used to break the systems and put the issue in the forefront of research in the area.

One of the most efficient lattice based digital signature schemes is BLISS, suggested by Ducas, Durmus, Lepoint and Lyubashevsky [7]. This scheme uses a bimodal Gaussian sampler and was shown to be vulnerable to timing attacks, and in particular the *sampling component* is not independent of the secret-key [11, 3], as well as other attacks. These vulnerabilities might be the reason the scheme did not emerge as an option in the Post-Quantum NIST standardization process.

The differential privacy (DP) setting has had its own share of issues with respect to leaky implementations. Starting with Mironov [22] who showed that the problems of finite precision arithmetic imply that pioneering implementations of differentially private databases actually do not satisfy the desired properties. To address this Balcer and Vadahn [2] considered designing DP-algorithms that can be implemented in *strict* polynomial time. More recently, Andrysco et al. [1] showed that various concrete implementations differentially private mechanisms are vulnerable to timing channels. Ilvennto [13] suggested implementing the *Exponential Mechanism* with "Base-2 Differential Privacy," which meant it could be implemented with finite precision, but left open the issue of timing attack resilience.

1.2 Prevention Techniques

The main approach to prevent timing attacks is to use *fixed time algorithms*, often called in the literature "constant time algorithms," meaning algorithms that run the same amount of time on all inputs.

There are two main drawbacks to this solution. First, in order for the algorithm to run in fixed time on all the inputs, we need to know the worst case running time, a task that is often challenging on its own. The second one is that even if we do know the running time,

[1] There are two variants to differential privacy, pure and approximate. In the case of pure DP, the results should be close pointwise.

in many cases there is a very large gap between *best case and worst case running time*, or even *average case and worst case running time*, and by making the algorithm run in the worst case time on all inputs, we create huge overheads. It is also worth mentioning that the second caveat can make many protocols and algorithms impractical and not usable when efficiency is critical.

In addition, the survey in Section 1.1 demonstrates that the task of making an algorithm run in fixed time is more subtle and challenging than meets the eye. Timing information can leak from response times of the server, from I/O calls, from reading RAM memory or cache memory and many more possibilities. In order for the algorithm to be truly and fully fixed time, one must make sure to make everything fixed time, which is often very challenging, and goes against hardware and software optimizations.

A common technique to thwart timing attacks in the public-key context is "blinding," first suggested by Chaum [4] in the context of signatures, where a value v is mapped into a random looking one u prior to the encryption or signature, in a manner that allows to retrieve the desired signature or encryption from the encryption or signature on u. Kocher [20] suggested using blinding to make RSA implementations secure against timing attacks. The blinding works by multiplying the input x by a fresh random element r of the group $\mathbb{Z}_N^*$, i.e. a random element which is co-prime to N. To decode, a multiplication by the group inverse r^{-1} is done at the end of the computation. Note that simply using the same r for many inputs will not work, as the attack suggested by Kocher can recover r over time, and even recover the exponent without knowing r. Hence, fresh r needs to be chosen in each round. This example goes to show that using blinding as a technique to protect against timing attack is often a subtle task, and that if implemented naively or incorrectly can lead to a false sense of security.

A general approach to preventing leakage is to employ techniques from secure multi-party computation, and split the input into various parts where leaking *almost* all of them does not leak the actual values. It was first suggested in Ishai, Sahai and Waters [14] for thwarting probing attacks (see [16] for a survey). This can be thought of as the "moral equivalent" of blinding for a general functions. However, in case of timing, given that what is leaked is a function of *all* parties (at the very least, the sum of their running times), it is not clear that that it solves the problem. Nevertheless, it does point to the issue of the number of random bits used to generate a sample.

1.3 Our Contributions and Technical Overview

Our goal in this work is to investigate the landscape of algorithms and systems that can be implemented in a manner resistant to timing attacks, but we wish to expand the 'Procrustean bed' of fixed time algorithms. We provide foundational treatment to the subject as well as many algorithms and separation results.

In Section 2 We focus on the security against timing attacks in an information theoretic manner and focuses on randomized algorithms and in particular on *sampling algorithms*. We define sampling algorithms secure against timing attacks, "time oblivious" sampling algorithms. Our main result of this section is a charterization of the the distributions that can be sampled without leaking *any* information on the output.

▶ **Theorem 2.3.** *Let $\mathcal{D}$ be a discrete probability distribution. Then $\mathcal{D}$ has a time oblivious generating algorithm if and only if the following holds:*

1. *$\mathcal{D}$ has finite support.*
2. *$\mathcal{D}$ is rational, i.e. all the probabilities of possible outputs are rational.*

We also give an optimal time oblivious algorithm to sample from a rational distribution of finite support, where optimal means that any other time oblivious algorithm has slower running time.

In Section 3 we concentrate on the implication of these results to *Differential Privacy*, i.e. what happens to such mechanisms when their running time (or the number of random bits used) is leaked (recall from Section 1.1 that there is a history of leakage problems in DP implementations). Since many differential privacy mechanisms work by taking the input and adding to it noise generated by some distribution with an infinite support, it is clear from the discussion above that this approach is futile when trying to resist timing leakage.

We formally define *time oblivious DP mechanisms*:

▶ **Definition 3.1.** *Let $\mathcal{M}\colon \mathcal{C} \to \mathcal{R}$ be a randomized algorithm. We say $\mathcal{M}$ is (ε,δ)-* **differentially private** *time oblivious mechanism if for every pair of neighboring datasets D and D', every subset $S \subseteq \mathcal{R} \times \mathbb{N}$*

$$\mathbb{P}[(\mathcal{M}(D), \mathcal{T}(\mathcal{M}(D))) \in S] \leq e^{\varepsilon} \cdot \mathbb{P}[(\mathcal{M}(D'), \mathcal{T}(\mathcal{M}(D'))) \in S] + \delta.$$

We show that the situation is more complex. Some techniques, such as randomized response, work here provided the biased coin flipped is rational. In case the range is unbounded, as in counting in a database whose size is not known, then it is impossible to guarantee *useful* results with very high probability. That is, for any (time oblivious pure DP mechanism with an unbounded range there is a $\rho > 0$ s.t. for most databases the mechanism outputs *useless results with probability at least ρ* (See Claim 3.4). But on the other hand, we can take any DP mechanism and find a pointwise close (for each database) DP mechanism that is time oblivious as we prove in Theorem 3.6:

▶ **Theorem 3.6.** *Let $\mathcal{M}$ be any ε-pure DP mechanism with a discrete range $\mathcal{R}$. For any $\gamma > 0$, $\varepsilon' > \varepsilon$ there is a time oblivious ε' pure DP mechanism $\mathcal{M}_{obl}$ such that $\|\mathcal{M}_{obl}(D) - \mathcal{M}(D)\|_{TV} < \gamma$.*

Finally, we supplement Section 2 with Appendix A which deals with the problem of sampling a satisfying assignment of a DNF formula in a time oblivious way. We show how to convert the well known *non* time oblivious algorithm into a time oblivious algorithm while preserving the run-time. In addition, we show that leaking information on the formula is unavoidable. Specifically, we show that *any* time oblivious algorithm for sampling a satisfying assignment from a DNF formula that "hides the formula" must run in exponential time, and therefore we show an *inherent exponential gap* for hiding the input of a randomized algorithm.

▶ **Theorem A.4.** *Sampling a uniform satisfying assignment of a DNF formula in an input hiding time oblivious way cannot be done efficiently and requires $\Omega(2^n)$ bits in expectation.*

2 Time Oblivious Sampling: Definitions and Characterization

The challenge of designing an algorithm that generates a distribution $\mathcal{D}$ using a sequence of unbiased coins $C_1, C_2, \ldots \sim$ *Bernoulli* $\left(\frac{1}{2}\right)$ was studied in the seminal work of Knuth and Yao [19] in the mid 1970s. They described a greedy algorithm to generate $\mathcal{D}$ and showed that it is optimal in terms of the expected number of coin flips. In addition they discuss properties of the algorithm, such as expected number of coins used, optimality, computational efficiency and more. A detailed discussion about their work can be found in Chapter 15 ("The random bit model") of Devroye [5]

Their work appeared many years before the public discussion on side channel attacks and they did not address this issue. One possible timing attack is to measure the number of coin flips used by the algorithm. To see how this information may be useful, consider the example of sampling from $Geo\left(\frac{1}{2}\right)$ by tossing coins until we get for the first time "heads", and the number of tosses is the output generated. Clearly in this example, an adversary who knows the number of coin flips knows exactly what element was sampled. Such leakage can compromise cryptographic systems which rely on private randomness.

Another scenario where such leakage may be problematic occurs in the context of Differential Privacy (DP) [10]. Informally, the requirement of DP is for neighbouring datasets, to have "close" output distribution. By close we mean up to some multiplicative and additive factor. Removing the additive factor gives a stronger notion of DP, called pure DP. One main motivation for DP is to be able to get meaningful statistics from data, while preserving the privacy of the individual. A very common technique to achieve DP algorithms is to sample from some noise distribution, for example the Laplace distribution, and add it to the actual result.

As pointed by Balcer and Vadhan [2], the runtime of the noise generation can leak information about the noise that was generated, which in turn can make the noisy output not as hiding as well as in the idealized world. This may compromise the DP guarantee of the algorithm, especially if it is a pure DP algorithm.

Therefore, in order for the algorithm to satisfy the DP definition, it is *essential* that the running time of the noise generator will not give information to an adversary regarding the value sampled. This is discussed more thoroughly in Section 3.

Distributed sampling

A case where the number of bits used clearly leaks is when the generation is done distributively, using some sort of multi-party computation (e.g. when creating a root key or in the context of differential privacy [8]). In this case, the amount of communication (in bits) between the parties is directly related to the number of bits needed for the generation, so to keep the value generated hidden we need to make sure that the number of random bits consumed does not leak information.

2.1 Preliminaries, Notation and Definition

Let $\mathcal{D}$ be a discrete distribution on $\mathbb{N}$, with $d_i = \mathbb{P}[\mathcal{D} = i]$. We say $\mathcal{D}$ is rational if each d_i is a rational number, and $\mathcal{D}$ has finite support if: $supp(\mathcal{D}) := \{j \mid d_j > 0\}$ is a finite set.

For a randomized algorithm $\mathcal{A}$ let $\mathcal{R}$ denote its random tape and $\mathcal{R}[i] \sim Bernoulli\left(\frac{1}{2}\right)$ is the i^{th} bit in the tape. Then $\mathcal{R}_n$ is the n-bit random string: $\mathcal{R}[0]\mathcal{R}[1]\ldots\mathcal{R}[n-1]$. We assume a randomized algorithm $\mathcal{A}$ reads the tape sequentially, and after reading $\mathcal{R}_n$ decides deterministically, after a finite amount of steps, whether to return an output or read $\mathcal{R}[n]$ from the tape. The output distribution of $\mathcal{A}$ over random tape $\mathcal{R}$ will be denoted by $\mathcal{O}(\mathcal{A})$, and the number of bits read from the random tape by $\mathcal{A}$ is the random variable (R.V.) $\mathcal{T}(\mathcal{A})$.

A useful way to view the generation of $\mathcal{O}(\mathcal{A})$ by Knuth and Yao is to consider $\mathcal{R}$ as defining a random walk on an infinite binary tree in which going left corresponds to reading a 0 and going right corresponds to reading a 1, for this reason we consider a binary sequence $a_0 \ldots a_n$ as a node. Since $\mathcal{A}$ is deterministic given $\mathcal{R}_{n+1} = a_0 \ldots a_n$, then for the node $a_0 \ldots a_n$ in the tree $\mathcal{A}$ either halt and outputs, or $\mathcal{A}$ "walks" to either $a_0 \ldots a_n 0$ or $a_0 \ldots a_n 1$ with equal probability, $\frac{1}{2}$.

In light of the view above we define the i^{th} level of the tree to be the set of all binary sequences of length i. We will abuse the notation and use $a_0 \dots a_{i-1}$ to denote the corresponding integer associated with the binary sequence. We say that $a_0 \dots a_{i-1}$ precedes $b_0 \dots b_{i-1}$ if it is smaller as an integer. We say that $\mathcal{A}$ outputs on a binary sequence $a_0 \dots a_{i-1}$ if conditioned on $\mathcal{R}_i = a_0 \dots a_{i-1}$ the algorithm reads the first i bits of $\mathcal{R}$ and outputs before reading $\mathcal{R}[i]$. The i^{th} level is called an output level of $\mathcal{A}$ if $\mathcal{A}$ outputs on some sequence $a_0 \dots a_{i-1}$ in that level. The set of all output levels of $\mathcal{A}$ is denote by $\mathcal{L}(\mathcal{A})$. Finally $\mathcal{T}(\mathcal{A})$ will denote the distribution of the number of bits $\mathcal{A}$ read from the random tape $\mathcal{R}$.

When considering the runtime of a randomized algorithms an important resource is *the number of random bits that are read from the random tape.* Having this number be independent of the instance generated is a prerequisite to time obliviousness. Furthermore, once the required number of random bits has been read, the problem is a deterministic computation of a mapping and this can be performed in some worst case time for the given size. So at least in principle there is an independent implementation.

Motivated by the discussion above, we make the following definition:

▶ **Definition 2.1.** *We say $\mathcal{A}$ is a* ***time oblivious generating algorithm*** *if its output distribution and running time distribution are independent, meaning $\mathcal{O}(\mathcal{A})$ and $\mathcal{T}(\mathcal{A})$ are independent random variables.*

2.2 Characterization of Time Oblivious Distributions

A natural question to ask is what distributions can be generated in a time oblivious way? For example, can we generate the distribution $Geo(\frac{1}{2})$ that was discussed at the beginning of the section? What about sampling a biased coin where the probability of '1' is p and '0' otherwise? Can we do it for all values of p? We focus on the exact model, meaning we want to sample from the *exact* distribution and not some approximation.[2] Consider the following "separating bit" algorithm for sampling a biased coin with bias p: Let $p = 0.p_1p_2\dots$ be the binary expansion of p, we toss fair coins $x_1, x_2, \dots$ to generate a number x between 0 and 1, $x = 0.x_1x_2\dots$. We stop in the first index i where $p_i \neq x_i$ we return x_i. We call the algorithm "separating bit" because we toss coins until we find the first index the separates the binary representation of p from the binary representation of the number x we generate.

Notice that:

▷ Claim 2.2. Let p be the bias of the generated coin then:

1. The "separating bit" algorithm is not time oblivious.
2. The expected number of random bits read by the algorithm is 2.

Proof.

1. If the algorithm produced an output after a single coin flip, we know that the result of the coin flip was $1 - p_1$ and therefore the output is $\mathbb{1}_{p_1=1}$.
2. Each coin has probability $\frac{1}{2}$ to be the separating bit, therefore the number of bits read is $Geo(\frac{1}{2})$ and the expected amount of bits read is 2. ◁

While Claim 2.2 shows that the separating bit algorithm is not time oblivious for all p, it does not mean there is no time oblivious algorithm to toss a p-biased coin to some values of $p \neq 1/2$. We show that we can toss a p biased coin iff p is rational. More generally, we will show:

[2] For an example of previous work on exact sampling, consider Feldman et al. [12] who addressed the issue of how many different types of coins one needs in order to generate a die roll in an exact manner.

▶ **Theorem 2.3.** *Let $\mathcal{D}$ be a discrete probability distribution. Then $\mathcal{D}$ has a time oblivious generating algorithm if and only if the following holds:*

1. *$\mathcal{D}$ has finite support.*
2. *$\mathcal{D}$ is rational, i.e. all the probabilities of possible outputs are rational.*

This characterization answers the questions above, we cannot sample from $Geo(\frac{1}{2})$ in a time oblivious way, and we can generate a biased coin if and only if p is rational. We will prove the theorem by showing each direction separately, and begin by showing:

▶ **Lemma 2.4.** *If a distribution $\mathcal{D}$ has a time oblivious algorithm, then it is rational with finite support.*

Proof. Let $\mathcal{A}$ be a time oblivious generating algorithm for $\mathcal{D}$. Since $\mathcal{A}$ has output distribution $\mathcal{D}$, it means in particular that it outputs at some level, i.e. $\mathcal{L}(\mathcal{A})$, is not empty. Let k be the first output level of $\mathcal{A}$. Denote by m_k the number of output nodes of length k, and by e_j the number of nodes which output j in that level. The number of sequences of length k is 2^k and so we get that: $0 < m_k \leq 2^k < \infty$. Observe that the probability to get an output j in level k is exactly $\frac{e_j}{m_k}$. Since $\mathcal{A}$ is time oblivious, conditioning on $\mathcal{T}(\mathcal{A}) = k$ yields the same output distribution $\mathcal{D}$ and so for all j we have $d_j = \frac{e_j}{m_k}$ and therefore $d_j \in \mathbb{Q}$ and $\mathcal{D}$ is rational. Notice that $d_j = \frac{e_j}{m_k}$ means that $d_j > 0$ implies that $d_j \geq \frac{1}{m_k}$ and therefore $|supp(\mathcal{D})| \leq m_k$. We get that $\mathcal{D}$ is rational with finite support. ◀

We will later prove the other direction of Theorem 2.3 by describing a time oblivious algorithm to output $\mathcal{D}$. In the binary tree view described in Section 2.1 being time oblivious means that in each output level the conditional distribution given that the level was reached is $\mathcal{D}$ (i.e. the output nodes of the level are distributed exactly according to $\mathcal{D}$). From now on, in light of Lemma 2.4, if $\mathcal{D}$ is generated by a time oblivious algorithm we may assume $supp(\mathcal{D}) = \{1, \ldots, n\}$ and use the notation $d_j = \frac{p_j}{q_j}$ for the output distribution and let $q := LCM(q_1, .., q_n)$.[3] We now prove some properties of general time oblivious algorithms. We will need the following definition:

▶ **Definition 2.5.** *A node $a_0 \ldots a_n$ is called* reachable *if no prefix $a_0 \ldots a_m$ with $m < n$ is an output node.*

The following lemma applies for any time oblivious algorithm $\mathcal{A}$:

▶ **Lemma 2.6.** *Let $\mathcal{A}$ be a time oblivious algorithm with a finite and rational output distribution $\mathcal{D}$ where the LCM of the probabilities is q. Let t_k be the number of nodes in level k that are either unreachable or output nodes, and m_k the number of output nodes at level k, then:*

1. *The number m_i of output nodes in level i is a multiple of q.*
2. *t_k satisfies: $t_k = \sum_{i=0}^{k} 2^{k-i} m_i$. In particular t_k is a multiple of q.*

Proof. Proof in Appendix B. ◀

A corollary of Lemma 2.6 is:

▶ **Corollary 2.7.** *Let $\mathcal{D}$ be a discrete distribution. $\mathcal{D}$ can be generated in finite worst case complexity if and only if $\mathcal{D}$ is rational with finite support and $q = 2^k$.*

Proof. Proof in Appendix B. ◀

[3] The LCM is the Least Common Multiple of a set of natural numbers, that is the smallest natural number that is divisible by the given set of numbers.

We now describe a time oblivious algorithm that generates $\mathcal{D}$. Following Lemma 2.4, we assume that the support of $\mathcal{D}$ is $[n]$, and $\mathcal{D}$ is given by a list of length n of the probabilities $d_i = \frac{p_i}{q_i}$ for co-prime p_i and q_i. Since the distribution has finite support, $q = LCM(q_1, \dots, q_n)$ can be computed efficiently (and we assume is part of the input). By Lemma 2.6 we know that any time oblivious algorithm should output a multiple of q elements in each level. Observe that if a level has q reachable nodes then we can partition these nodes into sets $\{S_i\}_{i=1}^{n}$, where $|S_i| = d_i \cdot q = \frac{p_i}{q_i} \cdot q \in \mathbb{N}$ and output i at S_i.

With this in mind, the algorithm works as follows: the algorithm iterates level by level and if there are q reachable nodes then it assign them q outputs similarly to the S_i's above. Picking arbitrary reachable nodes may be inefficient (even though it will be optimal in terms of randomness, the formal definition may be found in Definition 2.12). To make the algorithm efficient the algorithm picks consistently the q left most reachable nodes in each level. This is now formally described in Algorithm 1:

Algorithm 1 Gen($\mathcal{D}$,$\mathcal{R}$).

```
$q \leftarrow LCM(q_1, \dots, q_k)$
$n \leftarrow 0$
while True do
    $n \leftarrow n + 1$
    if $(2^n \bmod 2q) \geq q$ then                    ▷ If there are $q$ reachable nodes in level $n$
        if $(\mathcal{R}_n \bmod 2q) \leq q - 1$ then      ▷ Is $\mathcal{R}_n$ one of the $q$ leftmost reachable nodes
            Return GetValue($\mathcal{D}$, $(\mathcal{R}_n \bmod 2q)$)
```

Algorithm 2 GetValue($\mathcal{D}, j$).

```
$q \leftarrow LCM(q_1, \dots, q_k)$
$s_0 \leftarrow 0$
for $i = 1$ to $k$ do
    $s_i \leftarrow s_{i-1} + q \cdot \frac{p_i}{q_i}$
Return $i$ such that $s_{i-1} \leq j < s_i$                ▷ Binary Search the value of $i$
```

The following lemma justifies the comments in Algorithm 1 and specifies properties of its output nodes:

▶ **Lemma 2.8.**

1. *The number of reachable nodes in level n is exactly $2^n \bmod 2q$.*
2. *Let $a_0 \dots a_{n-1}$ be the node in the tree that represents reading the bits $a_0, \dots, a_{n-1}$ from the randomness tape. The number of reachable nodes preceding a reachable node $a_0 \dots a_{n-1}$ of Algorithm 1 is $a_0 \dots a_{n-1} \bmod 2q$.*
3. *Let t_n be the number of nodes in level n which are either unreachable or output nodes. For all n we have: $2^n - t_n < q$.*

Proof. Proof in Appendix B. ◀

We now finish the proof of Theorem 2.3 and show that Algorithm 1 is time oblivious:

▷ Claim 2.9. Algorithm 1 is time-oblivious algorithm and generates $\mathcal{D}$.

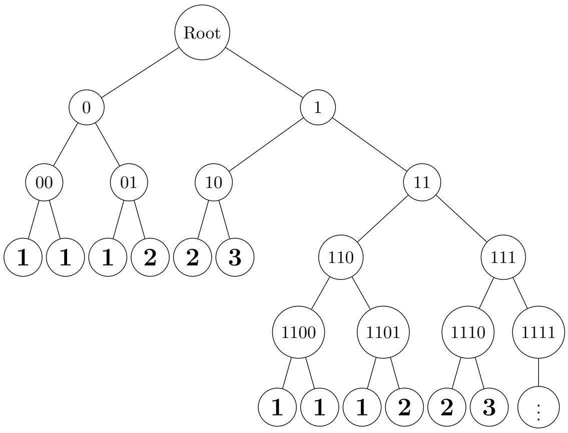

Figure 1 Distribution Generating Tree of $\mathcal{D}_0$ using Algorithm 1.

Proof. By Parts 1,3 of Lemma 2.8 at each level n of Algorithm 1 there are q output nodes that correspond to the binary expansion of $\{\lfloor\frac{2^n}{2q}\rfloor\cdot 2q, \lfloor\frac{2^n}{2q}\rfloor\cdot 2q+1, .., \lfloor\frac{2^n}{2q}\rfloor\cdot 2q+q-1\}$ and Algorithm 2 outputs i on the nodes that correspond to the numbers $\{\lfloor\frac{2^n}{2q}\rfloor\cdot 2q+s_{i-1}, .., \lfloor\frac{2^n}{2q}\rfloor\cdot 2q+s_i-1\}$.

Therefore we get that $\mathbb{P}[\mathcal{O}(\mathcal{A})=i \mid \mathcal{T}(\mathcal{A})=n]=\frac{1}{q}\cdot q\frac{p_i}{q_i}=d_i$ which implies that $\mathcal{O}(\mathcal{A})$ is $\mathcal{D}$ when conditioning on $\mathcal{T}(\mathcal{A})=n$. Thus $\mathcal{O}(\mathcal{A})$ and $\mathcal{T}(\mathcal{A})$ are independent random variables. ◁

To demonstrate how the algorithm works, we show an example of the generating tree for the distribution: $\mathcal{D}_0=\{\frac{1}{2},\frac{1}{3},\frac{1}{6}\}$.

▶ **Example 2.10.** The LCM q of the denominators of $\mathcal{D}_0$ is 6 and so:

$$\mathcal{D}_0=\left\{\frac{3}{6},\frac{2}{6},\frac{1}{6}\right\}$$

This gives the following distribution generating tree:

2.3 Randomness Efficiency

We now show the sampling algorithm is the most efficient one in terms of randomness. To formalize what it means to be most efficient in terms of randomness, for an algorithm $\mathcal{A}$ let $\ell_i(\mathcal{A})$ be the probability that $\mathcal{A}$ produces an output on level i and denote by $S_k=\sum_{i=1}^{k}\ell_i(\mathcal{A})$, the probability that the algorithm will produce an output up until, and including, level k.

▶ **Definition 2.11.** *Let $\mathcal{A}$ and $\mathcal{A}'$ be two algorithms with the same output distribution $\mathcal{D}$. We say $\mathcal{A}$ (weakly) dominates $\mathcal{A}'$ in efficiency if for all k: $S_k(\mathcal{A}')\leq S_k(\mathcal{A})$. Furthermore, we say $\mathcal{A}$ strictly dominates $\mathcal{A}'$ if $\mathcal{A}$ weakly dominates $\mathcal{A}'$ and there exists some k for which $S_k(\mathcal{A}')<S_k(\mathcal{A})$.*

With this definition we can now define what it means for an algorithm to be "optimal":

▶ **Definition 2.12.** *An algorithm $\mathcal{A}$ that generates $\mathcal{D}$ is* **optimal***, if for all algorithms $\mathcal{A}'$ that generate $\mathcal{D}$, $\mathcal{A}$ (weakly) dominates $\mathcal{A}'$.*

▶ Remark 2.13. For non-negative integer valued random variable X: $\mathbb{E}[X]=\sum_{k=1}^{\infty}\mathbb{P}[X\geq k]$ and therefore if $\mathcal{A}$ weakly dominates $\mathcal{A}'$ then $\mathbb{E}[\mathcal{T}(\mathcal{A})]\leq\mathbb{E}[\mathcal{T}(\mathcal{A}')]$ which means that if $\mathcal{A}$ is optimal then the expected number of random bits used by $\mathcal{A}$ is the minimum possible.

We will show that Algorithm 1 is optimal. First we need:

▶ **Observation 2.14.** *Let t_k be the number of unreachable or output nodes level k, then $S_k = \frac{t_k}{2^k}$.*

Proof. Proof in Appendix B. ◀

▶ **Theorem 2.15.** *Let $\mathcal{D}$ be a finite and rational distribution where the LCM of the denominators of the probabilities is q, then Algorithm 1 is an optimal time oblivious algorithm for generating $\mathcal{D}$.*

Proof. Proof in Appendix B. ◀

In addition to its optimality Algorithm 1 is essentially unique in the following sense:

▷ Claim 2.16. All optimal algorithms have the same number of output nodes in each level as Algorithm 1.

Proof. Let $\mathcal{A}$ be Algorithm 1 and let $\mathcal{A}'$ be another optimal time oblivious algorithm for $\mathcal{D}$. Since $\mathcal{A}$ and $\mathcal{A}'$ are both optimal we have: $\forall k : S_k(A) = S_k(A')$. By definition of S_k this implies $\forall k : \ell_k(A) = \ell_k(A')$. Therefore $\mathcal{A}$ and $\mathcal{A}'$ have the same output levels with q output nodes at each output level. ◁

We can now estimate the number of random bits Algorithm 1 reads from the tape:

▶ **Lemma 2.17.** *The expected number of bits read from the tape, $\mathbb{E}[\mathcal{T}(\mathcal{A})]$, by Algorithm 1 is $\log_2 q + \Theta(1)$.*

Proof. Proof in Appendix B. ◀

We now analyze the complexity of Algorithm 1. Recall that $n = |supp(\mathcal{D})|$. We assume the input is a list of numbers (p_i, q_i), the LCM q and the size of the support $n = |supp(\mathcal{D})|$:

Preparation time worst case $O(n)$. The array of s_i described in Algorithm 2 can be computed ahead of time in $O(n)$ running time by iteratively applying $s_i = s_{i-1} + q \cdot \frac{p_i}{q_i}$.

Sampling time expected $O(\log q)$. Remember that from Lemma 2.17 we know that the expected number of bits the algorithm reads is $\log_2 q + \Theta(1)$ bits. The algorithm takes extra $O(\log q)$ time to binary search what to output in Algorithm 2 so the sampling time is $O(\log q)$. One should keep in mind that the first level that is an output level is $\lceil \log_2 q \rceil$, and therefore the loop in Algorithm 1 may start at that level. Similar analysis to Lemma 2.17 implies that the expected number of iterations executed by the loop is $\Theta(1)$.

Space complexity worst case $O(\log q)$. We can deduce $O(\log q)$ space complexity in expectation since $\mathcal{T}(\mathcal{A})$, the expected number of bits read from the tape, is $\log_2 q + O(1)$ and the memory needed for $\mathcal{R}_n, 2^n$ is proportional to $\mathcal{T}(\mathcal{A})$. To get $O(\log q)$ worst case, notice that the algorithm doesn't actually need to know $\mathcal{R}_n$, 2^n but only ($\mathcal{R}_n$ mod $2q$), (2^n mod $2q$) and this can be maintained using $O(\log q)$ bits.

Algorithm 1 is also oblivious in the much stronger sense: all reachable nodes have the same control-flow i.e. they execute each line in the algorithm exactly the same number of times, and in the same order. Therefore, given a fixed time implementation of mod $2q$, and a fixed time implementation that compares two numbers of size up to $2q$ the running time of this implementation of $\mathcal{A}$ will not leak information on the output[4].

[4] Fixed time mod operations are required since in some circumstance even having the same control flow does not guarantee fixed time, as was shown by the Hertzbleed attack [24]

Notice that the parameters above cannot be improved. Preparation time is essential for general $\mathcal{D}$, since any sampling algorithm must read the probabilities of the outputs. The sampling time cannot be improved by Lemma 2.17, it also cannot be worst case by Corollary 2.7. Space complexity $\Omega(\log q)$ is needed to represent output range of size $\Omega(q)$.

From the discussion in this section, we conclude that defending against time attacks may cost unbounded slowdown, i.e.:

▶ **Observation 2.18.** *For every n there exist a distribution $\mathcal{D}$ for which $\mathbb{E}[\mathcal{T}(\mathcal{A})]/\mathbb{E}[\mathcal{T}(\mathcal{A}')] \geq n$ where $\mathcal{A}'$ is the optimal generation algorithm for $\mathcal{D}$ (not necessarily time oblivious) and $\mathcal{A}$ is any time oblivious algorithm for $\mathcal{D}$.*

Proof. Consider the distribution $\mathcal{D}$ of a biased coin with $p = \frac{1}{2^{2n}}$. Since the separating bit algorithm takes 2 random bits in expectation we know that $\mathbb{E}[\mathcal{T}(\mathcal{A}')] \leq 2$ since $\mathcal{A}'$ is optimal.

Let $\mathcal{A}''$ be Algorithm 1. Since it is optimal, we deduce from Remark 2.13 that $\mathbb{E}[\mathcal{T}(\mathcal{A})] \geq \mathbb{E}[\mathcal{T}(\mathcal{A}'')]$. $\mathcal{A}''$ has one output level which is $2n$ and therefore $\mathbb{E}[\mathcal{T}(\mathcal{A})] \geq 2n$. By dividing these two inequalities we conclude that $\mathbb{E}[\mathcal{T}(\mathcal{A})]/\mathbb{E}[\mathcal{T}(\mathcal{A}')] \geq n$. ◀

We refer the intrested reader to Appendix A where we consider time oblivious randomized algorithms that does not leak information on their input, as well as their input from the running time. We show that the algorithm of Karp and Luby [17, 18] to sample a satisfying assignment from a DNF formula, can be transformed to be time oblivious and efficient, but there is no efficient time oblivious algorithm that does not leak x from the running time.

2.4 Approximate Time Oblivious Sampling a Distribution

The "time oblivious" condition may sometimes be too strict, since many distributions used in real life applications do not have a finite support or rational probabilities. Moreover, even if the distribution does have finite support and rational probabilities, in some situations Algorithm 1 can be considerably slower than the Knuth-Yao sampler (that is optimal among all samplers in the random coin flips model). In other situations the time oblivious sampler has similar number of coin flips used to the Knuth-Yao sampler, e.g. when sampling a uniform integer in $[n]$. Therefore it is desired to have a definition of "approximate obliviousness."

We stress that whether using an approximation is an appropriate solution depends on the specific application of the sample in a randomized algorithm as well as on the approximation's guarantee. For example, if $\mathcal{M}$ is a pure DP mechanism, and the sampling time used by $\mathcal{M}$ leaks a small amount of information on the database, then $\mathcal{M}$ may not be pure DP given the running time. On the other hand this sort of approximation can be applied to an approximate DP mechanism with a small cost to δ. We will consider time oblivious DP mechanism in more detail in Section 3.

We suggest the following definition for "approximate time oblivious sampling" that preserves the privacy of the sample even when the running time has leaked. Note that for sampling algorithm $\mathcal{A}$ we let $\mathcal{O}(\mathcal{A})$ be its output. The definition essentially says that given the running time the conditional distribution is close to the original in a point-wise sense.

▶ **Definition 2.19.** *Let $\mathcal{D}$ be a distribution and $X \sim \mathcal{D}$. An algorithm $\mathcal{A}$ is (ε, δ)-approximate time oblivious sampler of $\mathcal{D}$ if for any $T \subseteq \mathbb{N}$ and for any $S \subseteq supp(D)$:*

$$e^{-\varepsilon}\mathbb{P}[X \in S] - \delta \leq \mathbb{P}[\mathcal{O}(\mathcal{A}) \in S \mid \mathcal{T}(\mathcal{A}) \in T] \leq e^{\varepsilon}\mathbb{P}[X \in S] + \delta.$$

The main benefit of considering the approximation above is that, as we shall see, the sampling complexity will *not* depend on the LCM of the distribution. If $\delta = 0$ then we say $\mathcal{A}$ is a pure time oblivious sampler of $\mathcal{D}$ and if $\delta > 0$ then it is an approximate time oblivious sampler. Allowing (pure) type of approximation yield that all distributions of finite support can be sampled approximately:

▷ Claim 2.20. Let $\mathcal{D}$ be a discrete distribution with support size n and let

$$H := H(\mathcal{D}) = \max_{x \in supp(\mathcal{D})} \log \left(\frac{1}{\mathbb{P}[x]} \right).$$

Then there exist an ε-pure time oblivious sampler of $\mathcal{D}$ that uses $H + \log \frac{1}{\varepsilon} + O(1)$ random bits in the worst case.

Proof. Proof in Appendix B ◁

We note that in the pure variant of Definition 2.19, only distributions of finite support can be sampled, as $supp(\mathcal{D})$ must contain all the outputs in the first output level of $\mathcal{A}$. We will use this observation again in Section 3. We also note that the number of coin flips used is essentially tight, since a pure sampler that outputs after k coin tosses must satisfy $He^{\varepsilon} \geq 2^k$, otherwise the maximizer of H would not satisfy the pure sampling condition.

We can replace of the dependence on $H(\mathcal{D})$, by a dependence on $\delta > 0$ if we allow the use of an (ε, δ)-approximate time oblivious sampler:

▷ Claim 2.21. Let $\mathcal{D}$ be a discrete distribution with $supp(\mathcal{D}) = [n]$, then there exist a $(0, \delta)$-approximate time oblivious sampler that uses $\log n + \log \frac{1}{\delta} + O(1)$ bits worst case.

Proof. Proof in Appendix B. ◁

Many distributions used in applications are of infinite support, e.g. the geometric distribution, discrete Laplace, etc. With respect to Definition 2.19 one must consider an approximate time oblivious sampler (that is not pure). Notice that a consequence of Definition 2.19 is that every subset $S \subseteq supp(\mathcal{D})$ of measure $> \delta$ must satisfy that $supp(\mathcal{O}(\mathcal{A})) \cap S \neq \emptyset$. This suggests that the the support of the distribution generated by the sampler should be of measure at least $1 - \delta$.

In general consider a distribution $\mathcal{D}$ and let $S_\delta \subseteq supp(\mathcal{D})$ denote a subset of minimal size that satisfies for $X \sim \mathcal{D}$: $\mathbb{P}[X \in S_\delta] \geq 1 - \delta$.

▷ Claim 2.22. For any distribution $\mathcal{D}$ there exist an $(0, \delta)$ approximate time oblivious sampler that uses $\log(|S_{\delta/2}|) + \log \frac{1}{\delta} + O(1)$ random bits in the worst case.

Proof. To ease the notation we assume that $supp(\mathcal{D}) = \mathbb{N}$, X is a random variable with distribution $\mathcal{D}$, and $p_i = \mathbb{P}[X = i]$. Also assume $S_{\delta/2} = [n]$ for $n = |S_{\delta/2}|$. Let $\mathcal{D}'$ be the distribution supported on $[n]$ in which the probability to get i is q_i where:

$$q_i = \left\{ \begin{array}{ll} p_i + \mathbb{P}\left[X \notin [n]\right], & \text{if } i = 1 \\ p_i, & \text{otherwise} \end{array} \right\}$$

By applying the sampler of Claim 2.21 to $\mathcal{D}'$ we obtain a sampler that uses $\log n + \log \frac{1}{\delta} + O(1)$ random coins. Notice that since the time is fixed and $\varepsilon = 0$ then Definition 2.19 coincides with total variation (TV) distance of distributions. Therefore since by construction $\mathcal{D}$ and $\mathcal{D}'$ are of TV distance $\frac{\delta}{2}$ and $\mathcal{D}'$ and the distribution produced by Claim 2.21 is of distance $\frac{\delta}{2}$ the sampler distribution is of TV distance δ and it outputs on a single time, therefore satisfies Definition 2.19. ◁

We remark that for distributions of infinite support tail bounds imply bounds on S_δ and therefore imply bounds on the efficiency of an approximate time oblivious sampler. For example, let $\mathcal{D}$ be a discrete distribution on $\mathbb{Z}$ satisfying $\mathbb{P}[|X| > t] \leq c_1 exp(c_2 t)$ for some positive constants c_1, c_2 and t large enough (i.e. $\mathcal{D}$ is discrete, sub-exponential distribution).

The tail bound implies that $|S_\delta| = O\left(\log \frac{1}{\delta}\right)$ and thus the approximate sampler is efficient and needs $O\left(\log \frac{1}{\delta}\right)$ random bits by Claim 2.22.

This in particular applies to the sub-exponential distributions such as geometric and Laplace which are commonly used in differential privacy.

3 Differential Privacy Mechanisms and Timing Attacks

We gave a full characterization on the distributions that can be sampled in a time oblivious way. This can be interpreted as a negative result for time oblivious sampling, since many algorithms use distributions that are not of finite support and also assume that elements may have irrational probabilities of being an outcome. A prominent example in the context of differential privacy (DP) is the application of a *Laplace*(ε) R.V. that has an infinite support. It is often sampled inside a differentially private mechanism, but by Theorem 2.3 it cannot be sampled by a time oblivious algorithm. Therefore the execution of the mechanism will actually leak information that will make it non-DP (at least in the *pure* sense).

This may be considered as the conceptual starting point of the work of Balcer and Vadhan [2]. In their work, they addressed the problem of DP mechanisms implemented on “finite computers” to address the issues of timing attacks and infinite output range. They constructed DP algorithms on finite range, with worst case running time to handle those issues and emphasized that each sampling must be of a distribution $\mathcal{D}$ which is finite an rational.

One issue that motivates the formal discussion of the sampler as part of the timing of a mechanism is that by Corollary 2.7 we get that rounding to rationals, as done in [2], isn't enough and the rounding must be to *dyadic* rationals; this was first observed in the work of [8] in relation to DP. This issue is clearly dependent on the model of computation, but shows the delicacy needed in the rounding procedure to ensure worst case running time.

Another issue is that the sampling running time of an algorithm is affected by the running time of the sampler it uses. By Observation 2.18 it may be that a proposed algorithm originally uses samples that take expected $O(1)$ time to generate, but using the time oblivious samples must run in running time $\omega(1)$. This may affect total performance of an algorithm.

The main aim of this section is to define time oblivious DP mechanisms and specifically, time oblivious pure DP mechanisms and investigate their properties. Even though time oblivious sampling is restrictive, we show that time oblivious pure DP mechanism are far more flexible. In particular, we prove that any pureDP mechanism can be transformed to a time oblivious with almost the same guarantees. We begin with the definition of time oblivious DP mechanism.

▶ **Definition 3.1.** *Let $\mathcal{M}\colon \mathcal{C} \to \mathcal{R}$ be a randomized algorithm. We say $\mathcal{M}$ is (ε, δ)-***differentially private** *time oblivious mechanism if for every pair of neighboring datasets D and D', every subset $S \subseteq \mathcal{R} \times \mathbb{N}$*

$$\mathbb{P}[(\mathcal{M}(D), \mathcal{T}(\mathcal{M}(D))) \in S] \leq e^{\varepsilon} \cdot \mathbb{P}[(\mathcal{M}(D'), \mathcal{T}(\mathcal{M}(D'))) \in S] + \delta.$$

In the definition above, if $\delta = 0$ the mechanism is said to be *pure* time oblivious DP, otherwise it is approximate time oblivious DP. $\mathcal{C}$ denotes the set of possible database (rather then a distribution), and we assume that $\mathcal{C}$ is connected as a graph with respect to the neighbouring relation. The time oblivious DP condition does not impose any condition on $\mathcal{M}(D)$ in terms of finiteness or rationality. The time oblivious DP condition means that levels in the distribution generating trees from Section 2.2 satisfy that the conditional distribution on neighbouring datasets are almost the same up to the prescribed parameters.

The following lemma says that the time oblivious condition implies that all databases have the same outputs at each level.

▶ **Lemma 3.2.** *Let $\mathcal{M}$ be a time oblivious ε-pure DP mechanism on a connected set of databases, then for any two databases D and D'':*

$$supp(\mathcal{M}(D)|\mathcal{T}(\mathcal{M}(D)) = t) = supp(\mathcal{M}(D'')|\mathcal{T}(\mathcal{M}(D'')) = t).$$

Proof. By the connectivity of $\mathcal{C}$ it's enough to show this for neighbouring databases D, D'. If $r \in \mathcal{R}$ is in the support of the conditional distribution on level n of $\mathcal{M}$ on D then by the pure DP condition:

$$0 < \mathbb{P}[\mathcal{M}(D) = r \mid \mathcal{T}(\mathcal{M}(D)) = t] \leq e^{\varepsilon} \cdot \mathbb{P}[\mathcal{M}(D') = r \mid \mathcal{T}(\mathcal{M}(D')) = t]$$

Therefore $\mathbb{P}[\mathcal{M}(D') = r \mid \mathcal{T}(\mathcal{M}(D')) = t] > 0$ and r is in the support of the conditional distribution on level t of $\mathcal{M}(D')$. ◀

Pure DP mechanism for approximate counting

One fundamental problem considered in the DP literature is counting. That is, given a database D output an approximation to the number of elements satisfying some property. We now devise a time oblivious pure DP mechanism for approximate counting up to a factor 2. This example captures critical properties of time oblivious DP algorithms.

Let $c(D)$ be the true count of D. For each database we need to construct a tree and by Lemma 3.2 at each level we must have the same outputs for all possible databases D. The mechanism will have at level $i + 2$ only one possible output which is 2^i. If $2^k \leq c(D) < 2^{k+1}$ the mechanism will have one output node in each of the levels $2, .., k + 1, k + 4, k + 5 \dots$, at level $k + 2$ the algorithm outputs on $2^{k+1} - c(D) + 1$ of the nodes the value 2^k, while at level $k + 3$ the algorithm outputs 2^{k+1} on $2c(D) - 2^{k+1} + 1$ nodes. Notice that

$$\sum_{i=2}^{k+1} 2^{-i} + (2^{k+1} - c(D) + 1)2^{-(k+2)} + (2c(D) - 2^{k+1} + 1)2^{-(k+3)} + \sum_{i=k+4}^{\infty} 2^{-i} = \frac{3}{4}.$$

Therefore by adding another output at the 2^{nd} level which will always be 1 we get a probability distribution on output nodes and with probability at least $\frac{1}{4}$ the output is either 2^k or 2^{k+1} so this algorithm approximates $c(D)$ up to factor 2 with constant probability. Moreover, notice that neighboring databases have identical trees up to 1 node in one level $k + 2$ and 2 nodes in level $k + 3$. This algorithm is $\ln 3$ pure DP since at each output level the number of output nodes changes by at most 2 and since there is at least one output node of value 2^i at level $i + 2$ we get that if there are s output nodes at the level then it's enough to have ε satisfy $s + 2 \leq e^{\varepsilon} s$ which holds for $\varepsilon = \ln 3$ since $s \geq 1$.

The mechanism above has several drawbacks, which we will soon show must exist in all time oblivious pure DP mechanisms. One drawback is that with constant probability the mechanism outputs an irrelevant output since with probability $\frac{1}{4}$ the output is 1 for any database. Another drawback is that the amount of randomness used scales with $c(D)$ in contrast to the not time oblivious algorithm of adding a discrete Laplace noise to $c(D)$.

We turn to specifying and proving the claims stated above. We begin with showing that a time oblivious pure DP mechanism on infinitely many databases gives an irrelevant output with some constant probability. To define what this means we assume we have a *utility function* $\mathbf{U}\colon \mathcal{C} \times \mathcal{R} \to \{0, 1\}$ for which $\mathbf{U}(D, r) = 1$ if r is useful information of D. We say that $\mathbf{U}$ is *sofic* if for any $r \in \mathcal{R}$ there are only finitely many databases D such that $\mathbf{U}(D, r) = 1$.

▶ **Definition 3.3.** *For a mechanism $\mathcal{M}$ the utility of the mechanism on a database D is defined by $\mathbf{U}_{\mathcal{M}}(D) = \mathbb{E}\left[\mathbf{U}(\mathcal{M}(D))\right]$, we omit $\mathcal{M}$ when it is clear from context. The guaranteed utility of a mechanism $\mathcal{M}$ is $G_{\mathbf{U}}(\mathcal{M}) := \inf_{D\in\mathcal{C}} \mathbf{U}(D)$.*

We show that with some constant probability the algorithm gives an irrelevant answer for most databases using the definition of guaranteed utility.

▷ Claim 3.4. Let $\mathbf{U}$ be a sofic utility function. Then any time oblivious pure DP mechanism $\mathcal{M}$ with infinitely many possible databases has $G_{\mathbf{U}}(\mathcal{M}) \leq 1 - \frac{1}{2^s}$ where s is the first output level of $\mathcal{M}$.

Proof. Proof in Appendix B. ◁

Similarly to guaranteed utility, we define the guaranteed expected runtime of the algorithm to be $G_{\mathcal{T}}(\mathcal{M}) = \sup_{D\in\mathcal{C}} \mathbb{E}[\mathcal{T}(\mathcal{M}(D))]$. Even though it seems natural to require for a time oblivious ε-pure DP to have a $G_{\mathcal{T}}(\mathcal{M})$ finite (with some parameter that may depend on ε) it cannot be done without serious consequences on the utility of the mechanism.

▶ **Proposition 3.5.** *Let $\mathbf{U}$ be a sofic utility function and $\mathcal{M}$ is a pure DP mechanism on infinitely many possible databases. Then $G_{\mathcal{T}}(\mathcal{M})$ is finite implies that $G_{\mathbf{U}}(\mathcal{M}) = 0$*

Proof. Proof in Appendix B. ◀

We remark that all proofs above can be easily modified to handle a utility function with range $[0, 1]$ such that for every $r \in \mathcal{R}$ and $x > 0$ there are finitely many D with $\mathbf{U}(r, D) > x$.

It is natural to wonder how does the expressive power of pure DP mechanisms compare to the expressive power of time oblivious pure DP mechanisms, i.e. does a ε pure DP algorithm has a time oblivious counterpart?

We now get to the main result of this section. We prove that *any* ε-pure DP mechanism M has a time oblivious pure DP mechanism with almost the same security guarantee and utility. Therefore this shows that pure DP mechanisms *can be defended against timing attacks* with a small cost to their utility and and security guarantees.

▶ **Theorem 3.6.** *Let $\mathcal{M}$ be any ε-pure DP mechanism with a discrete range $\mathcal{R}$. For any $\gamma > 0$, $\varepsilon' > \varepsilon$ there is a time oblivious ε' pure DP mechanism $\mathcal{M}_{obl}$ such that $\|\mathcal{M}_{obl}(D) - \mathcal{M}(D)\|_{TV} < \gamma$.*

Proof. The idea is to use a similar construction to the approximate counting mechanism described above by providing that each level in the tree will correspond to a different element in $\mathcal{R}$. Intuitively we will view the levels of the tree as an "abacus", and the change of the probability distributions of neighbouring datasets will correspond to change in the number of beads that outputs in each layer.

We assume $\mathcal{R} = \{r_i\}_{i=1}^{\infty}$, and we start by picking s, the depth of the first input level which we will decide later, and an integer t which corresponds to the number of output nodes that must be in every level.

The element r_i will be output at level $s+i$. For the mechanism $\mathcal{M}_{obl}$ to have outputs close to $\mathcal{M}(D)$ for any D we need that the probability to output at the level that corresponds to r_i is roughly $p_i = \mathbb{P}[\mathcal{M}(D) = r_i]$; let q_i denote the probability to output r_i in $\mathcal{M}_{obl}(D)$. We pick $q_i \approx \frac{t}{2^{s+i}} + \frac{2^s-t}{2^s}p_i$. Define $N_i^+ = \lceil (2^{s+i} - t2^i)p_i \rceil + t$ and $N_i^- = \lceil (2^{s+i} - t2^i)p_i \rceil + t - 1$ to be the number of output nodes in level $s + i$ with the two options for estimating q_i from below and from above. Notice that by the choice of N^+ and N^- we know that

$$1 + \frac{1}{2^s} \geq \sum 2^{-(s+i)} N_i^+ \geq 1 \geq \sum 2^{-(s+i)} N_i^- \geq 1 - \frac{1}{2^s}.$$

Therefore, there exists a signing of the N_i (this is done by looking at the binary expansion of the sum above) that gives a probability distribution. Notice that the following inequalities hold

$$\frac{N_i^+}{2^{s+i}} \geq \mathbb{P}[\mathcal{M}_{obl}(D) = r_i] \geq \frac{N_i^-}{2^{s+i}}.$$

To bound the privacy guarantee of $\mathcal{M}_{obl}$ we need to bound $N_i^+(D)/N_i^-(D')$ for neighboring database D, D', where we abuse the notation to describe which N_i belongs to which database. But since $\mathcal{M}$ is ε-pure DP we get that $N_i^+(D)/N_i^-(D') \leq \frac{te^\varepsilon+1}{t-1}$. Therefore by picking t large enough such that $\frac{te^\varepsilon+1}{t-1} \leq e^{\varepsilon'}$ we can ensure the DP guarantee and by picking s large enough that implies $\frac{t+1}{2^s} < \gamma$ (The extra masking of t nodes at each output level + 1 for rounding) the proof follows. ◀

▶ Remark 3.7. Notice that since $\|\mathcal{M}_{obl}(D) - \mathcal{M}(D)\|_{TV} < \gamma$ it follows that for any utility function (not necessarily sofic) $\mathbf{U}\colon \mathcal{C} \times \mathcal{R} \to [0,1]$, the utility of $\mathbf{U}_{\mathcal{M}_{obl}}(D) \geq \mathbf{U}_{\mathcal{M}}(D) - \gamma$.

The proof above produces a tree for $\mathcal{M}_{obl}(D)$ each $D \in \mathcal{C}$ by picking the signing for the N_i. We leave it as an open question to provide mechanisms $\mathcal{M}(D)$ for which the time oblivious pure DP mechanism $\mathcal{M}(D)$ is efficiently computable from the input D.

By Theorem 3.6 we get that any pure DP mechanism with a discrete range has a time oblivious pure DP with mechanisms with similar privacy guarantee. We saw in Proposition 3.5 that infinite guaranteed expected running time cannot be avoided for pure DP mechanism. Approximate DP mechanisms do not suffer from this issues.

▷ Claim 3.8. There is a $(0, \delta)$ mechanism for approximate counting up to an additive factor $\frac{1}{\delta}$ with expected number of used bits $O\left(\log \frac{1}{\delta}\right)$.

Proof. Return the true count + a uniform number in $[1, \ldots, \frac{1}{\delta}]$, with the uniform number sampled by Algorithm 1. ◁

4 Future Work and Open Problems

Our investigation focused on the question of time oblivious sampling, and we used a combinatorial lens to give a full characterization of the distributions that can be sampled from in a time oblivious way. These distributions have a finite support and rational weights. The combinatorial perspective also helped us define an algorithm to sample from such distributions and show it is optimal in a strong sense of the randomness used. In certain situations distributions are not given explicitly but are given in a rather succinct form. A notable example is sampling via a Markov Chain process which have found numerous applications in TCS. This lead us to ask:

▶ **Question 1.** *Is it possible to* efficiently *convert a Markov Chain algorithm, which samples from a finite and rational distribution, into a time oblivious algorithm with exactly the same output distribution?*

A natural consumer of our results is the area of differential privacy. Here we had both bad news and good news: the bad news are for mechanisms that we require to produce useful results (where the utility is some arbitrary but non trivial function of a points in the domain and range). It is impossible to guarantee usefulness with high probability ($(1 - g(n)$ where g is a function whose limit is 0). On the other hand we argued that it is possible to take any mechanisms and make it time oblivious without changing the distribution by much (and hence it utility) while preserving its differential privacy.

▶ **Question 2.** *One important question is whether the process of turning a DP mechanism into a timing oblivious one can preserve the computational efficiency of the scheme.*

In this paper we revisited the issues arising from timing attacks and specifically investigated Time oblivious sampling, where the emphasis was an exactly producing a distribution. In a companion paper we explore notions and methods for protecting keyed functions from timing attacks [6].

References

1 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, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 623–639. IEEE Computer Society, 2015. doi:10.1109/SP.2015.44.

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

3 Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In *Advances in Cryptology – ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 494–524. Springer, 2018. doi:10.1007/978-3-030-03326-2_17.

4 David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203. Plenum Press, New York, 1982. doi:10.1007/978-1-4757-0602-4_18.

5 Luc Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986. doi:10.1007/978-1-4613-8643-8.

6 Yoav Ben Dov, Liron David, Moni Naor, and Elad Tzalik. Resistance to timing attacks revisited: Protecting the keys. Technical report, Weizmann Institute of Science, Rehovot, 2023.

7 Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013. doi:10.1007/978-3-642-40041-4_3.

8 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, St. Petersburg, Russia, May 28 – June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006. doi:10.1007/11761679_29.

9 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

10 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

11 Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 – November 03, 2017*, pages 1857–1874. ACM, 2017. doi:10.1145/3133956.3134028.

12 David Feldman, Russell Impagliazzo, Moni Naor, Noam Nisan, Steven Rudich, and Adi Shamir. On dice and coins: Models of computation for random generation. *Inf. Comput.*, 104(2):159–174, 1993. doi:10.1006/inco.1993.1028.

13 Christina Ilvento. Implementing the exponential mechanism with base-2 differential privacy. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, November 9-13, 2020*, pages 717–742. ACM, 2020. doi:10.1145/3372297.3417269.

14 Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003 Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003. doi:10.1007/978-3-540-45146-4_27.

15 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.

16 Yael Tauman Kalai and Leonid Reyzin. A survey of leakage-resilient cryptography. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 727–794. ACM, 2019. doi:10.1145/3335741.3335768.

17 Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 56–64. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.35.

18 Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989. doi:10.1016/0196-6774(89)90038-2.

19 Donald E. Knuth and Andrew C. Yao. The complexity of nonuniform random number generation. *Algorithms and Complexity: New Directions and Recent Results, edited by J.F. Traub*, 1976.

20 Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology – CRYPTO '96, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9.

21 Richard J. Lipton and Jeffrey F. Naughton. Clocked adversaries for hashing. *Algorithmica*, 9(3):239–252, 1993. doi:10.1007/BF01190898.

22 Ilya Mironov. On significance of the least significant bits for differential privacy. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 650–661. ACM, 2012. doi:10.1145/2382196.2382264.

23 Gerald Tenenbaum. *Introduction to analytic and probabilistic number theory*, volume 163 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, Rhode Island, third edition. edition, 2015.

24 Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In *Usenix Security '22: 31st USENIX Security Symposium 2022*, pages 679–697. Usenix, 2022.

A The Good News and the Bad News: DNF Sampling

Recall that a DNF formula on n variables is a formula of the form $C_1 \vee C_2 \vee \ldots \vee C_m$ where each term C_i has $1 \leq \ell_i \leq n$ literals and is of the form $C_i = y_1^i \wedge y_2^i \wedge \ldots \wedge y_{\ell_i}^i$ where each y_j^i is a literal. It is worth noting that the task of finding a satisfying assignment to a DNF formula is easy and straightforward, however counting the number of satisfying assignments is $\#P-complete$. Jerrum, Valiant and Vazirani [15] provide a general discussion on generating a uniform object given an efficient algorithm that approximates to the counting problem.

Karp and Luby [17, 18] tackle the other direction. They describe an efficient algorithm for sampling a uniformly satisfying assignment to a DNF formula and use it for approximate counting.

Let $\phi = C_1 \vee \ldots \vee C_m$ be a DNF formula with n variables, and let S_i be the set of satisfying assignments to term C_i. Let ℓ_i be the number of literals in C_i and note that $|S_i| = 2^{n-\ell_i}$. Further denote by $S = \sum_i |S_i|$ and by $p_i = \frac{|S_i|}{S}$, which is the relative weight of C_i. The algorithm works as follows:

1. Sample a clause with probability proportional to its weight, meaning clause C_i is chosen with probability p_i.
2. Sample a random assignment $\pi \sim S_i$.
3. Let k be the number of clauses that are satisfied by π. With probability $\frac{1}{k}$ return π, else restart the algorithm from the beginning.

In order to convert the algorithm to be time oblivious, for Step 1, notice that the running time may leak information about the chosen clause. The LCM q of the distribution in step 1 is $\sum_j |S_j|$. We can therefore use Algorithm 1 with $q = \sum_j |S_j|$ and conclude that Step 1 is done in a time oblivious way. Note that $\forall i\colon\ |S_i| \leq 2^n$ and there are m clauses, therefore the expected running time of this step is $O(n + \log m)$.

Next, the running time of Step 2 may leak information about the chosen assignment. If the clause has ℓ literals, one can simply choose a uniform assignment to the $n - \ell$ remaining literals. However, this does reveal what ℓ is, which yields information about what clause was chosen in the case where the clauses are of different sizes. To hide ℓ, we toss n coins and ignore the first ℓ, thus making Step 2 of the algorithm run in time independent of ℓ and the chosen assignment, making it time oblivious. This gives us a running time of n always.

Finally, for Step 3, we need to toss a $\frac{1}{k}$-biased coin. To make this step time oblivious we need to hide the value of k, which is the number of satisfied clauses by the assignment π chosen in Step 2. Since there are m clauses, we know that $1 \leq k \leq m$. In order to hide what k is, we need to take a value q which is divisible by all numbers from 1 to m, i.e. $q = LCM(1, \ldots, m)$. With this q we can run Algorithm 1 and output heads on the first $\frac{q}{\ell}$ output nodes, and tails on the rest. This makes the distribution of the running time of Step 3 independent of k, and overall Step 3 is time oblivious. The following is a standard number theoretic estimate (which may be found in [23] p.37):

▶ **Fact A.1.** $\ln(LCM(1, 2, \ldots, n)) = \Theta(n)$.

By Lemma 2.17, we need $O(\log q) = \log\left(LCM(1, \ldots, m)\right)$ bits in expectation, instead of $O(\log m)$. By Fact A.1 we conclude that the expected running time of Step 3 is $O(m)$. Notice that this is an exponential increase relative to the $O(\log m)$ bits that is needed to generate a uniform element of $\{1, \ldots, m\}$.

Finally the expected number of iterations of the algorithm before it produces an output is at most m, since k is bounded above by m. Putting it all together we get:

▶ **Proposition A.1.** *There exists an efficient time oblivious algorithm for uniform DNF sampling running in expected time $O(mn + m^2)$*

While we showed that the running time of the algorithm above does not leak information about what assignment was chosen, it does reveal information about the structure of the formula. For example, the running time leaks information about the number of literals and number of clauses in the formula. When the formula is public information this leakage does not give any new information, but there might be cases where we wish to hide the structure of the formula in addition to what assignment was chosen.

It is therefore natural to ask: is there an efficient time oblivious algorithm for DNF sampling that hides the formula as well as the chosen assignment?

A.1 An Exponential Lower Bound for Hiding the Formula

We formalise what it means to "hide the structure of the formula":

▶ **Definition A.2.** *Let $x \in \{0,1\}^n$ be an input to a randomized algorithm $\mathcal{A}$. We say $A(x)$ is an **input hiding time oblivious generating algorithm** if its running time distribution is independent of both the output distribution and the input x.*

The motivation for the definition above is to defend against timing attacks in situations where a randomized algorithm runs on an input which we would like to be kept secret, as well as the output. We show that DNF sampling cannot be done efficiently in an input hiding time oblivious way, we will show a different problem which cannot be done efficiently, and then get the result on DNF via a reduction.

Consider the problem: given $x \in \{1, \dots, 2^n\}$ sample a random element in $\{1, \dots, x\}$. This task can be done efficiently using $O(\log x)$ random bits. However, in the input hiding setting we wish to hide x in addition to the sample. This problem cannot be solved efficiently and in a time oblivious way.

▶ **Lemma A.3.** *The task of sampling a random element $k \in \{1, \dots, x\}$ given $x \in \{1, \dots, 2^n\}$ in an input hiding time oblivious way takes at least $\Omega(2^n)$ random bits.*

Proof. Let $\ell_m(x = i)$ be the probability that the algorithm outputs on level m given $x = i$. In order for the sample to be time oblivious, we need the running time to be independent both of the choice of x and k. This means that the running time distribution of the algorithm should be the same for all values of x. In particular: for all m and i, $\ell_m(x = i) = \ell_m(x = 1)$

From the observation above we know that the algorithm must have the same output levels, and same number of output nodes for all values of x. Let q be the number of nodes in the first output level. We get that q must be divisible by all values from 1 to 2^n, and therefore $q \geq LCM(1, \dots, 2^n)$. By Fact A.1 we need $\Theta(2^n)$ to represent $LCM(1, 2, \dots, 2^n)$, and the result follows. ◀

Applying the lemma, we get the following theorem on DNF sampling:

▶ **Theorem A.4.** *Sampling a uniform satisfying assignment of a DNF formula in an input hiding time oblivious way cannot be done efficiently and requires $\Omega(2^n)$ bits in expectation.*

Proof. Given $x \in \{1, \dots, 2^n\}$ notice that we can write a formula with the literals $a_0, \dots, a_{n-1}$ consisting of at most n clauses and x satisfying assignments. We denote the formula corresponding to x by ϕ_x. If $x = 2^k$ we set $\phi_x = \overline{a_0} \wedge \dots \wedge \overline{a_{(n-1)-k-1}} \wedge a_{n-1-k}$. It is easy to check that for $x = \sum_{i \in S} 2^i$ for some $S \subseteq \{0, 1, \dots, n-1\}$ one can set $\phi_x = \vee_{i \in S} \phi_{2^i}$ and obtain in such a way for each $x \in \{1, ..., 2^n\}$ a formula with x satisfying assignments and at most n clauses.

We know by Lemma A.3 that to sample a uniform element from $\{1, \dots, x\}$ given x any algorithm must use $\Omega(2^n)$ bits in expectation in case we want to algorithm to be input hiding time oblivious case and by Proposition A.1 there is a time oblivious algorithm that samples from the formulas above using $O(n^2)$ random bits. Therefore we conclude that hiding the formula cannot be done efficiently. ◀

B Missing Proofs

B.1 Missing Proofs in Section 2

Proof of Lemma 2.6. If $\mathcal{A}$ is time oblivious then the output distribution $\mathcal{O}(\mathcal{A})$ is $\mathcal{D}$ when conditioning on $\mathcal{T}(\mathcal{A}) = k$, therefore: $\frac{p_j}{q_j} = d_j = \frac{e_j}{m_k}$ where m_k is the number of output nodes in the k^{th} level and e_j is the sequences of length k on which $\mathcal{A}$ outputs j. We get that $e_j = m_k \cdot \frac{p_j}{q_j}$ and as each e_j is a natural number we get that for all j: $q_j \mid m_k$ and therefore $q \mid m_k$. This concludes Part 1.

For Part 2, notice that in the binary tree each output node in level l has 2^{k-l} descendants in level k that it makes unreachable, and also that an output node can not have another output node as a descendent, therefore $t_k = \sum_{i=0}^{k} 2^{k-i} m_i$. By the first part each m_i is divisible by q and therefore t_k is a multiple of q. ◀

Proof of Corollary 2.7. If $q = 2^k$ then we can clearly output all elements in level k since $\mathcal{D}$ is rational with finite support. For the other direction, we may assume that the algorithm reads exactly t bits from $\mathcal{R}$. This also implies that $\mathcal{D}$ is rational with finite support. By Lemma 2.6 $q\ |2^t$ and therefore $q = 2^k$ for some k. ◀

Proof of Lemma 2.8. For Part 1, observe that the statement is true if $q = 2^m$ for some m. The test $2^n \geq q$ and the test $(2^n \bmod 2q) \geq q$ are exactly the same until the first output level. Since q is a power of 2 then all the nodes in level m will be output nodes and there will not be any more output nodes.

For q that is not a power of 2 we prove the claim by induction on the level. The base case is level 1. Indeed q is not a power of 2 which implies that $q > 2$ and as level 1 has 2 nodes, both nodes are reachable.

Assume the statement is true for all levels up to k. We now show the statement is true for $k+1$. By the induction hypothesis let $r = 2^k \bmod 2q$ be the number of reachable nodes in level k. if $r < q$ we know that k is not an output level. In this case each reachable node $a_0 \ldots a_{k-1}$ becomes two reachable nodes in level $k+1$, since each node has two children: $a_0 \ldots a_{k-1}0$ and $a_0 \ldots a_{k-1}1$. From this we get that the number of reachable nodes in level $k+1$ is $2r$, and the statement holds since:

$$2^{k+1} \equiv 2 \cdot 2^k \equiv 2r \pmod{2q}.$$

If $q < r < 2q$ then k is an output level. Write $r = q + s$ for some $1 \leq s < q$, and observe that q nodes will be output nodes in that level. This means that the number of reachable nodes in level $k+1$ will be $2s$ and indeed:

$$2^{k+1} \equiv 2 \cdot 2^k \equiv 2r \equiv 2(q+s) \equiv 2q + 2s \equiv 2s \pmod{2q}.$$

This concludes part 1 of Lemma.

For Part 2 notice that at each output level the output nodes are the q leftmost reachable nodes, and we know there are exactly $2^n \bmod 2q$ reachable nodes from Part 1 the binary sequences of $\{\lfloor \frac{2^n}{2q} \rfloor \cdot 2q, \lfloor \frac{2^n}{2q} \rfloor \cdot 2q + 1, .., 2^n - 1\}$ correspond to the reachable nodes. Part 2 now follows by taking $\bmod 2q$ on the sequence above.

Finally, by Part 1 the number of reachable nodes in each level is between 0 and $2q - 1$, and in addition, if there are at least q reachable nodes, then q nodes produce an output and therefore $2^n - t_n < q$. ◀

Proof of Observation 2.14. Recall that $S_k = \sum_{i=1}^{k} \ell_i(\mathcal{A})$ is the probability that the algorithm produced an output up until level and including k. By Part 2 of Lemma 2.6 we get:

$$S_k = \sum_{i=0}^{k} \ell_i(\mathcal{A}) = \sum_{i=0}^{k} 2^{-i} m_i = \frac{1}{2^k} \sum_{i=0}^{k} 2^{k-i} m_i = \frac{t_k}{2^k}. \qquad \blacktriangleleft$$

Proof of Lemma 2.17. For the lower bound: the first output level k satisfies $2^k \geq q$ and $\mathcal{T}(\mathcal{A}) \geq k$ since it is the *first* output level. Taking expectation it follows that $\mathbb{E}[\mathcal{T}(\mathcal{A})] \geq \lceil \log_2 q \rceil$.

For the upper bound: Recall that by construction of Algorithm 1 the algorithm outputs on nodes from left to right, and at each level there are less than q nodes which are reachable but not output nodes. From this we get that after the first time the algorithm reads a 0, it will produce an output after at most $\log q$ more steps. Observe that the probability to read the first 0 in the i^{th} level is a $Geo\left(\frac{1}{2}\right)$ R.V. and therefore we get:

$$\mathbb{E}[\mathcal{T}(\mathcal{A})] \leq \sum_{i=1}^{\infty} \frac{1}{2^i}(i + \log q) = \log q \sum_{i=1}^{\infty} \frac{1}{2^i} + \sum_{i=1}^{\infty} \frac{i}{2^i} = \log q + 2. \qquad \blacktriangleleft$$

Proof of Theorem 2.15. Let $\mathcal{D}$ be a distribution and $\mathcal{A}$ be Algorithm 1 for $\mathcal{D}$. Assume towards a contradiction that $\mathcal{A}$ is not optimal. This means that there exists a time oblivious algorithm $\mathcal{A}'$ which generates $\mathcal{D}$ and a k such that $S_k(A') > S_k(A)$. By Observation 2.14 $S_k(A') = \frac{t'_k}{2^k}$ and $S_k(A) = \frac{t_k}{2^k}$. From the assumption about $\mathcal{A}'$ we get that $t'_k > t_k$. From Lemma 2.6 we know that t'_k and t_k are both multiples of q which means $t'_k \geq t_k + q$. From Lemma 2.8 we know that $2^k - t_k < q$ which means that $t_k > 2^k - q$. Taking both inequalities into account we get that $t'_k > 2^k$. Recall that t'_k is the number of unreachable or output nodes of algorithm $\mathcal{A}'$ at level k, and therefore cannot be bigger than 2^k. Thus we get a contradiction and conclude $\mathcal{A}$ is optimal. ◀

Proof of Claim 2.20. We assume that $supp(\mathcal{D})$ is $[n]$ and define $p_i = \mathbb{P}[X = i]$. Let k be a parameter of the number of coins that the sampler uses before returning an answer, and assume $k > H + c$ for large enough c that will be picked later. Define $p_i^+ = \frac{\lceil p_i 2^k \rceil}{2^k}$ and $p_i^- = \frac{\lfloor p_i 2^k \rfloor}{2^k}$.

The sampler $\mathcal{A}$ will have $\mathbb{P}[\mathcal{O}(\mathcal{A}) = i]$ is either p_i^+ or p_i^- and will always output after reading all k bits. Therefore it will satisfy that $\mathbb{P}[\mathcal{T}(\mathcal{A}) \in \{k\}] = 1$ which will make the conditioning on the time in Definition 2.19 redundant. It is possible to always "sign" $p_i^{\pm}$ (i.e. pick between p_i^+ or p_i^-) to obtain a distribution (the sum of the probabilities is 1). We will show that for any such signing algorithm $\mathcal{A}$ is an approximate time oblivious sampler of $\mathcal{D}$. Notice that since $k > H + c$ we have that $2^k \cdot p_i = r_i + t_i$ for some integer $r_i > 2$ and $0 \leq t_i < 1$:

$$\frac{p_i^+}{p_i^-} = \frac{\lceil p_i 2^k \rceil}{\lfloor p_i 2^k \rfloor} \leq \frac{r_i + 1}{r_i - 1} \leq 1 + \frac{2}{r_i - 1}.$$

If $\frac{2}{r_i - 1} < \varepsilon$, then we could conclude with the inequality $1 + x \leq e^x$ that $\frac{p_i^+}{p_i^-} \leq e^{\varepsilon}$, and therefore $\frac{p_i^+}{p_i} < e^{\varepsilon}$ as well as $\frac{p_i}{p_i^-} \leq e^{\varepsilon}$ since we know $p_i^+ \geq p_i \geq p_i^-$.

Therefore it is enough to find k for which $\frac{2}{r_i - 1} < \varepsilon$. Notice that by the definition of H we have for $k = H + c$ that $2^k \cdot p_i \geq 2^c$ Therefore $c = \log \frac{1}{\varepsilon} + 4$ suffices. ◁

Proof of Claim 2.21. We use the notation $p_i := \mathbb{P}[X = i]$ for $X \sim \mathcal{D}$. We also continue with the notation defined in Claim 2.20, p_i^+, p_i^- with $k = \log n + \log\left(\frac{1}{\delta}\right)$. Clearly $p_i^+ - p_i^- = \frac{1}{2^k} \leq \frac{\delta}{n}$. Let $\mathcal{D}'$ be a *distribution* that correspond to some signing $p_i^\pm$ and we will again take $\mathcal{A}$ that outputs according to $\mathcal{D}$ after $k = \log n + \log\frac{1}{\delta}$ coin tosses. For $S \subseteq [n]$:

$$\begin{aligned}\mathbb{P}[X \in S] - \delta \cdot \frac{|S|}{n} = \sum_{i \in S}\left(p_i - \frac{\delta}{n}\right) &\leq \sum_{i \in S} p_i^- \\ &\leq \Pr[\mathcal{O}(\mathcal{A}) \in S] \\ &\leq \sum_{i \in S} p_i^+ \leq \sum_{i \in S}\left(p_i + \frac{\delta}{n}\right) = \mathbb{P}[X \in S] + \delta \cdot \frac{|S|}{n}.\end{aligned}$$

Therefore since $|S| \leq n$

$$\Pr[\mathcal{O}(\mathcal{A}) \in S] - \delta \leq \mathbb{P}[X \in S] \leq \Pr[\mathcal{O}(\mathcal{A}) \in S] + \delta. \qquad \blacktriangleleft$$

B.2 Missing Proofs in Section 3

Proof of Claim 3.4. Let D be some database and let k be the index of the first output level of $\mathcal{M}(D)$. By Lemma 3.2 we know that there are $r_1, \ldots, r_\ell \in \mathcal{R}$ that appear in the output level of all possible databases. Since $\mathbf{U}$ is sofic and there are infinitely many possible databases we can find a database D' such that $\mathbf{U}(D', r_1) = 0$ and therefor $G_{\mathbf{U}}(D) < 1$ and thus the guaranteed utility of $\mathcal{M}$ satisfies $\mathbf{U}(\mathcal{M}) < 1$. ◁

Proof of Proposition 3.5. Assume that $G_{\mathcal{T}}(\mathcal{M}) < m$. Let D be any database and $\tau > 0$ and let $r_1, \ldots, r_\ell$ be the elements of $\mathcal{R}$ that D outputs in levels $1, \ldots, \tau m$. We know that there is a database D' that satisfies $\mathbf{U}(D', r_i) = 0$ but by Markov's inequality $\mathbb{P}[\mathcal{T}(\mathcal{M}(D')) > \tau m] < \frac{1}{\tau}$ and therefore we get that $G_{\mathbf{U}}(\mathcal{M}) \leq \mathbf{U}(D') \leq \frac{1}{\tau}$. Since $\tau > 0$ is arbitrary we conclude that $G_{\mathbf{U}}(\mathcal{M}) = 0$. ◀