Cryptϵ: Crypto-Assisted Differential Privacy On Untrusted Servers

ABSTRACT

In the last decade, differential privacy has steadily become the de-facto standard for achieving strong privacy guarantees in data analysis. It is typically implemented either in the "central" or "local" model. In the former, a trusted centralized server collects the records from the data owners in the clear and outputs differentially private statistics; while in the latter, the data owners do not necessarily trust the server and hence individually randomizes their inputs. Thus the central model assumes the presence of a trusted server which is dispensed with in the local model. This increased privacy guarantee of the local model, however, comes at the cost of strictly lower accuracy and restricted algorithmic expressibility for differentially private programs. In this work, we propose a two-server model for differential privacy, Cryptϵ that
1) achieves the accuracy guarantees of the central model without the need of any trusted server
2) provides strictly greater expressibility than the local model.
Cryptϵ achieves this "best of both worlds" result with the assistance of two cryptographic primitives - linear homomorphic encryption and Yao's garbled circuit. Cryptϵ is equipped with a set of primitives which can be used by an data analyst to obtain differentially private answers for a useful class of programs. Further, we propose four optimization schemes to improve the efficiency of Cryptϵ. We demonstrate the feasibility of Cryptϵ in for practical differentially private analysis on untrusted servers with an extensive empirical evaluation on real datasets.

CCS CONCEPTS

• **Security and privacy** → *Public key encryption*; *Privacy-preserving protocols.*

KEYWORDS

Differential Privacy, Crypto Service Provider, Linear Homomorphic Encryption

1 INTRODUCTION

Differential privacy has steadily emerged as a compelling privacy definition that allows us to glean useful information about data alongside providing privacy at the granularity of individuals. It is essentially a property of a function such that its output should disallow any inference about the presence (or absence equivalently) of any individual record in the function's input dataset [20]. Formally, it requires that for any outcome of a randomized function, that outcome should be nearly equally likely with and without any one record.

Depending on infrastructural constraints, like the viable trust model in a given setting, differential private algorithms in practice can have two different styles of implementation. The more common and historically precedent implementation is the central differential privacy (CDP) model where a trusted data curator collates data from all individuals into a centrally held dataset and processes it in a privacy preserving way. The curator is trusted to store the data in the clear and mediates upon queries posed by a mistrustful analyst, interested in learning some synopsis of this dataset. Privacy is enforced by the curator by adding uncertainty (e.g. random noise drawn from Laplace distribution of scale parameter $\frac{1}{\epsilon}$) to the answers for analyst's queries before releasing them. This differentially private noise has a standard deviation of $O(\frac{1}{\epsilon})$. Herein lies an important observation - since every query requires only a single instance of random noise to be added to the answer (by the curator), central differential privacy can thus achieve only a constant error of $O(\frac{1}{\epsilon})$ where ϵ is our chosen differential privacy parameter.

Thus a major attribute of the central differential privacy setting is its prerequisite of a centralized server (the data curator) that can be fully trusted to store the data in the clear and release only differentially private computations. However, such a strong trust assumption is ill-suited for many practical scenarios and a single point of data breach in the CDP setting can be catastrophic for the entire dataset. Moreover, in the wake of stringent data protection regulations like GDPR, HIPAA etc, a trusted data curator is saddled with legal and ethical obligations to uphold the privacy of its data, making its practical implementation to be cumbersome. These premises have lead to the rise of a competing model called the local differential privacy (LDP). In this model too, each data owner sends in their data to a central aggregator. However, unlike in the central differential privacy setting, the aggregator in the LDP model is untrusted and every individual has to randomize its inputs before communicating it to the central aggregator. Hence the private data is concealed from the untrusted aggregator who attempts to infer statistics about the true population from the perturbed data instead. LDP techniques thus enable private data analysis without reliance on any trusted third party. Most notable use cases for LDP include observing most frequent software features and measuring their performance and failure characteristics; collecting data about users' default browser homepage and search engine et al.

Regrettably, this ease of adopt-ability for LDP comes with an utilitarian price tag. Since each reporting data owner has to perform independent coin flips, the resulting noise induces a binomial distribution. This binomial distribution can be approximated by a normal distribution; the magnitude of this random Gaussian noise however can be very large, its standard deviation grows in proportion to the square root of the report count $O(\frac{\sqrt{n}}{\epsilon})$, and the noise is

Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

in practice higher by an order of magnitude [9, 24, 25, 60]. Thus, if a billion individuals' reports are analyzed, then a common signal from even up to a million reports may be missed. The construct of the LDP model in fact imposes additional penalties in terms of the algorithmic expressibility. Kasiviswanathan et al. in [38] showed that the power of the local model is equivalent to that of the statistical query model [39] from learning theory and there exists an exponential separation between the accuracy and sample complexity of local and central algorithms. As a consequence, the local model requires enormous amounts of data [38] to obtain reliable population statistics. Unsurprisingly, only large corporations like Google [9, 24, 25] and Apple [1], with billions of user base have had successful commercial deployment of LDP.

From the above discussion, thus we see that although the minimal trust assumption of LDP makes it conducive for easy practical adoption, its accuracy guarantees are strictly limited as compared to that of CDP. In order to bridge this gap, we propose a new model for differential privacy, Cryptϵ that
(1) achieves the accuracy guarantees of CDP without the need of a trusted server
(2) provides strictly greater expressibility than that of LDP
In Cryptϵ the differentially private computations are outsourced to two non-colluding but untrusted servers, namely the Analytics Server (AS) and the Cryptographic Service Provider (CSP). The relaxation of not having trusted servers is effectuated by the use of cryptographic primitives, specifically linear homomorphic encryption and Yao's garbled circuits. The AS plays a role akin to that of the data curator in CDP or the data aggregator in LDP while the CSP initializes and manages the cryptographic primtives in Cryptϵ. Cryptϵ enables the computation of differentially private algorithms by executing programs expressed as a sequence of Cryptϵ primitives. At the very outset, the data owners submit their encrypted data to the AS which collates all the data records into a single encrypted database. On receiving a data query from an external analyst expressed in the form of a Cryptϵ program, the AS executes it on the encrypted database via some interactions with the CSP. Cryptϵ is designed to reveal no extra information (beside that released by the differentially private outputs itself) to the two servers under the assumption of non-collusion (this condition, for example, can be enforced by law). The main contributions of this work are
•**New approach-** We propose a new two-server model for implementing differential privacy which integrates the constant error bounds of CDP with the low trust assumption of LDP.
• **Expressive Language** - Cryptϵ supports a set of 9 primitives which can be used to compose differentially private programs to answer a large class useful queries.
• **Optimizations** - We propose four optimizations (two of them are differentially private) for Cryptϵ that provide substantial performance gain over the base case implementation.
•**Generalized multiplication using linear homomorphic encryption**- We propose a very efficient way of performing $n-$ way multiplications using linear homomorphic encryptions.
•**Practical for real-world usage** - We showcase the practicality of Cryptϵ via extensive evaluation on real datasets.

Related Work

Differential Privacy - Introduced by Dwork et al. in [21], differential privacy has enjoyed immense attention from both academia and industry in the last decade. Some of the most recent directions in the CDP model include [3, 10, 14, 17, 18, 22, 31–34, 43, 44, 46, 53–55, 64–66, 68–70]. The most prominent work in LDP include [6, 7, 16, 24, 25, 56, 60–62, 71]. Recently it has been showed that augmenting the local differential privacy setting by an additional layer of anonymity can improve the privacy guarantees (or decrease error bound equivalently) [9, 15, 23]. An important point to be noted here is that the power of this new model (known as shuffler/mixnet model) lies strictly between that of traditional LDP and CDP. The two-server model of Cryptϵ differs from this line of work in three major ways. Firstly, Cryptϵ results in no reduction in expressibility as compared to that of the CDP model (see Appendix E). Secondly, the shuffler/mixnet model results in an approximate DP guarantee $(\epsilon\sqrt{\frac{\log\frac{1}{\delta}}{n}}, \delta)$ which incurs an expected error of $O(\epsilon\sqrt{\log\frac{1}{\delta}})$. In practice, δ has to be at least $\frac{1}{n}$ in order to get some meaningful privacy. In contrast Cryptϵ achieves the the same order of accuracy guarantees as that of the CDP model. Finally, the shuffler/mixnet model and Cryptϵ have certain differences in their respective trust assumptions. A more detailed discussion is presented in Appendix E.
Two-Server Model - The two-server model is a popular choice especially for privacy preserving machine learning approaches where typically one of the servers manages the cryptographic primitives while the other handles computation. Examples of this include [26–28, 41, 48, 51, 52].
Homomorphic Encryption - There has been a surge in practical privacy preserving solutions employing homomorphic encryptions in the recent past enabled by improved constructs. A lot of the aforementioned two-server models employ homomorphic encryption [28, 41, 51, 52]. Additionally it is used in [11, 12, 29, 30, 36]. A more detailed discussion on related work is presented in Appendix D.

2 BACKGROUND

In this section we give a brief introduction to definitions and primitives relevant to Cryptϵ.

2.1 Differential Privacy

Definition 1. *An algorithm $\mathcal{A}$ satisfies ϵ-differential privacy (ϵ-DP), where $\epsilon > 0$ is a privacy parameter, iff for any two datasets D and D′ that differ in a single record, we have*

$$\forall t \in Range(\mathcal{A}), Pr\big[\mathcal{A}(D) = t\big] \leq e^{\epsilon} Pr\big[\mathcal{A}(D') = t\big] \quad (1)$$

where $Range(\mathcal{A})$ denotes the set of all possible outputs $\mathcal{A}$.

When applied multiple times, the differential privacy guarantee degrades gracefully as follows.

Theorem 1. *(Sequential Composition). If $\mathcal{A}_1$ and $\mathcal{A}_2$ are ϵ_1-DP and ϵ_2-DP algorithms that use independent randomness, then releasing the outputs $\mathcal{A}_1(D)$ and $\mathcal{A}_2(D)$ on database D satisfies $\epsilon_1 + \epsilon_2$-DP.*

There exist advanced composition theorems that give tighter privacy loss bounds, but we use Theorem 1 for our paper.

2.2 Cryptographic Primitives

Linearly Homomorphic Encryption (LHE). Let $(\mathcal{M}, +)$ be a finite group. A LHE scheme for messages in $\mathcal{M}$ is defined by three algorithms

• **Key Generation** (*Gen*) -This algorithm takes the security parameter κ as input and outputs a pair of secret and public keys, $(s_k, p_k) \leftarrow Gen(\kappa)$.

• **Encryption** (*Enc*) - This is a randomized algorithm that encrypts a message $m \in \mathcal{M}$ via the public key p_k, to generate ciphertext $\mathbf{c} \leftarrow Enc_{pk}(m)$.

•**Decryption** (*Dec*) - This is a deterministic function that uses the secret key s_k to recover the plaintext m from ciphertext $\mathbf{c}$.

In addition, LHE supports the operator $\oplus$ that allows the summation of ciphers as follows:

Operator $\oplus$ - Let $c_1 \leftarrow Enc_{pk}(m1), \ldots, c_a \leftarrow Enc_{pk}(m_a), a \in \mathcal{Z}_{>0}$. Then we have $Dec_{sk}(c_1 \oplus c_2 \ldots \oplus c_a) = m_1 + \ldots + m_a$.

One can multiply a cipher $c \leftarrow Enc_{sk}(m)$ by a plaintext positive integer a by a repetitions of $\oplus$. We denote this operation by $cMult(a, c)$ such that $Dec_{sk}\big(cMult(a, c)\big) = a \cdot m$.

Labeled Homomorphic Encryption(labHE). Let (Gen, Enc, Dec) be an LHE scheme with security parameter κ and message space $\mathcal{M}$. Assume that a multiplication operation is given in $\mathcal{M}$, i.e., is a finite ring. Let also $\mathcal{F} : \{0,1\}^s \times \mathcal{L} \rightarrow \mathcal{M}$ be a pseudo-random function with seed space $\{0,1\}^s$(s= poly(κ)) and the label space $\mathcal{L}$. Define

•**labGen**(κ) - On input κ, it runs $Gen(\kappa)$ and outputs (sk, pk)

•**localGen**(pk) - For each user i and with the public key as input, it samples a random seed $\sigma_i \in \{0,1\}^s$ and computes $pk_i = Enc_{pk}(\underline{\sigma_i})$ where $\underline{\sigma_i}$ is an encoding of σ_i as an element of $\mathcal{M}$. It outputs (σ_i, pk_i).

•**labEnc**$_{pk}(\sigma_i, m, \tau)$: On input a message $m \in \mathcal{M}$ with label $\tau \in \mathcal{L}$ from user i, it computes $b = \mathcal{F}(\sigma_i, \tau)$ (known as the mask) and outputs the labeled ciphertext $\mathbf{c} = (a, d) \in \mathcal{M} \times C$ with $a = m - b$ (known as the hidden message) in $\mathcal{M}$ and $d = Enc_{pk}(b)$. For brevity we just use notation **labEnc**$_{pk}(m)$ to denote the above functionality, in the rest of paper.

•**labDec**$_{sk}(\mathbf{c})$ - This functions inputs a cipher $\mathbf{c} = (a, d) \in \mathcal{M} \times C$ encrypted under the labHE scheme and decrypts it as $m = a - Dec_{sk}(d)$.

Secure Computation Garbled circuit, also known as Yao's protocol [45, 67], is a generic method for secure computation that allows two-party evaluation of a function $f(x_1, x_2)$ in the presence of semi-honest adversaries. The protocol is run between two data owners with respective private inputs x_1 and x_2 such that at the end, no party learns more than what is revealed from the output value $f(x_1, x_2)$. A more detailed discussion is presented in Appendix A.

3 SYSTEM OVERVIEW

This section provides an overview of Cryptϵ. We begin this section with the desiderata for Cryptϵ that motivates our design followed by a discourse on the different components of Cryptϵ and the associated trust assumptions. Next we describe the functionality of the different modules of Cryptϵ and a discussion on a few additional architectural choices we make for Cryptϵ. This is followed by a brief security sketch of Cryptϵ.

Table 1: Notations

Symbol	Explanations
Boldface	- represents encrypted data
$\tilde{}$	- represents one-hot-coding
$\hat{}$	-represents a differentially private output
A	- an attribute
s_A	- size of domain of attribute A
$dom(A) = \{v_1, \ldots, v_{s_A}\}$	- domain of attribute A
$ct_{A,i}$	- # records with value v_i for attribute A
m	-# number of data onwers
$\tilde{\mathcal{D}}$	- encrypted database with records in per-attribute one-hot-coding
$x \times y$ table **T**	- an encrypted table with x records in one-hot-coding and y columns one for each attribute; serves as one of the inputs to a transformation primitive
B	- A m - lengthed vector such that each entry $\mathbf{B}[i]$ represents whether record $r_i, i \in [m]$ is relevant to the program at hand
V	-represents a vector
c	- represents a scalar
$\mathcal{P}$	- represents a set

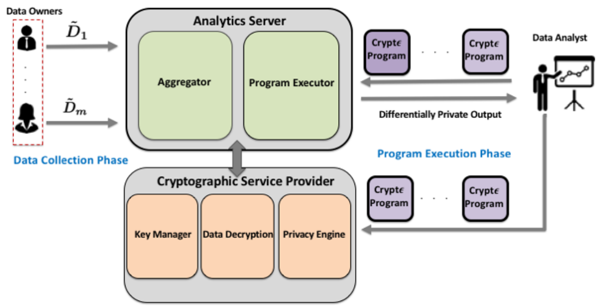


Figure 1: **Cryptϵ System Setting: The AS runs the Cryptϵ programs. The CSP manages the cryptographic primitves.**

3.1 Cryptϵ Design

The desiderata for the functionality of Cryptϵ are as follows

(1) The architecture of Cryptϵ should be able to support any differentially private algorithm that is permitted in the CDP model at the same order of accuracy guarantee as that of CDP, i.e. constant $O(\frac{1}{\epsilon})$ error bound.

(2) Raw data records cannot be stored in the clear in any server, i.e., the server(s) in Cryptϵ are untrusted.

(3) Data owners must be off-line.

Recall from our discussion in section 1, CDP gives constant error bounds, however it relies on a trusted server for this. In contrast LDP requires no trusted server but results in limited accuracy. The first two principles thus capture Cryptϵ's objective of achieving the "best of both worlds", i.e., the accuracy guarantees and expressibility of the CDP model with the trust assumptions similar to that of the LDP model. The third principle stems from the fact that another undesirable trait of LDP is its inherent requirement for the data owners to be on-line during every query processing. It is so because in LDP depending on the query at hand, every time the data owners need to compute a different noisy measurement that best answers it. This is in general a poor practice as maintaining active communication channels for the data owners might be unwieldy especially with a large number of data owners in geographically dispersed locations.

The first desideratum can be achieved if we collate the data into a single dataset such that any measurement can be performed on the entire dataset at once followed by a single instance of noise addition as opposed to aggregation over noisy measurements on individual data. The second desideratum mandates that the data hence collected should be suitably protected via cryptographic primitives (e.g. encryption scheme). Thus the first two requirements combined, necessitates Cryptϵ to be able to perform differentially private computations on the encrypted data itself. This leads us to use linear homomorphic encryption and secure computation (specifically Yao's garbled circuits) as the cryptographic primitives of our choice for Cryptϵ. Hence by virtue of secure computation, we can potentially implement all the functionalities of the CDP model in Cryptϵ. However, bearing the practical efficiency of the cryptographic primitives in mind, we impose certain limitations on the expressibility of Cryptϵ as discussed in section 3.5. From the third desideratum, since the data owners are no longer in the loop to monitor every query answering, there is a need for a third party agent that would watchdog the overall privacy budget expenditure. Guided by the above principles we design a two-server model for Cryptϵ with the following components

(1) **Data-Owners (DO)**- Each data-owner $\mathrm{DO}_i, i \in [m]$ has a private data record D_i and is willing to share it only if encrypted.
(2) **Analytics Server (AS)** - The AS wants to run a set of differentially private programs on the dataset $\mathcal{D} = \bigcup_{i=1}^{m} D_i$ but has access only to the encrypted copies of $D_i, i \in [m]$.
(3) **Cryptographic Service Provider (CSP)** - The CSP manages the cryptographic primitives used in Cryptϵ and interacts with the AS to compute the noisy answers. It is also responsible for monitoring the privacy overall budget expenditure.

3.2 Trust Model

Both the servers, AS and CSP are completely untrusted in Cryptϵ. Thus from the data owners perspective the trust assumption is similar to that of LDP; the data owners need not place their trust in any external entity. However there are two modest differences in the Cryptϵ setting from that of LDP.
(1) We assume that the AS and the CSP do not collude with each other and follow the honest-but-curious (or semi-honest) adversary model. Moreover we assume that each data owner has a private channel with the AS. This is to prevent any third party (including the CSP) from eavesdropping.
(2) The adversary (specifically the two servers AS and CSP) is now reduced to a computationally bounded one as opposed to the information theoretic unbounded adversary in LDP.

3.3 Cryptϵ Modules

Cryptographic Service Provider (CSP)

(1) **Cryptographic Key Manager** - The foremost duty of the CSP is to initialize the encryption scheme of Cryptϵ. This task is handled by the *Cryptographic Key Manager* module which generates the key pair (sk, pk) for the labHE scheme. It stores the secret key, sk with itself and releases the public key, pk. Note that since only the CSP has access to the secret key sk, it is the only entity capable of decryption in Cryptϵ.
(2) **Data Decryption** - The CSP being the only entity capable of decryption, any measurement of the data (even noisy) has to involve the CSP. The *Data Decryption* module is tasked with handling all such interactions with the AS.
(3) **Privacy Engine** - Cryptϵ starts of with a total privacy budget of ϵ^B which is unanimously agreed upon by all the data owners. Note that the mechanism of deciding ϵ^* should be piloted by social prerogatives [37, 42] and is currently outside the scope of Cryptϵ. For executing every program, the AS has to interact with the CSP at least once (for decrypting the noisy answer) thereby giving the CSP the opportunity to monitor the AS's actions in terms of privacy budget expenditure. The *Privacy Engine* module hence maintains a public ledger that records the privacy budget spent in executing every individual program. Once the privacy cost incurred reaches the capped limit of ϵ^B, the CSP refuses to decrypt any further answers thereby ensuring that the privacy budget is not exceeded. The ledger is completely public allowing any data owner to verify it as and when desired.

Data Owners (DO)

(1) **Data Encoder** - Each data owner $\mathrm{DO}_i, i \in [m]$ has a private data record D_i of the form $\langle A_1, ...A_l \rangle$ where A_j is an attribute. At the very outset, every data owner DO_i represents his/her private record D_i in its respective per attribute one-hot-coding format. The one-hot-coding is a way of representation for categorical attributes and is illustrated by the following example. If the database schema in Cryptϵ is given by $\langle Age, Gender \rangle$ then corresponding one-hot-coding representation for a data owner $DO_i, i \in [m]$ with the record $\langle 30, Male \rangle$, is given by $\tilde{D_i} = \langle [\underbrace{0, \ldots, 0}_{29}, 1, \underbrace{0, \ldots, 0}_{70}], [1, 0] \rangle$.
(2) **Data Encryption** - The *Data Encryption* module stores the public key pk of the labHE scheme used in Cryptϵ which is announced by the CSP. This key is used for an element-wise encryption of the data owners encoded record of per attribute one-hot-codings. In our aforementioned example, we get $\tilde{\mathbf{D}} = \langle [\underbrace{labEnc_{pk}(0), \ldots, }_{29} labEnc_{pk}(1), \underbrace{\ldots, labEnc_{pk}(0)}_{70}], [labEnc_{pk}(1), labEnc_{pk}(0)] \rangle$. Finally the data owner sends this encrypted record to the AS via a secure channel.This is the only interaction that a data owner ever participates in, all the program executions are carried out by the AS and the CSP with the data owners being completely off line.

Analytics Server (AS)

(1) **Aggregator** - The *Aggregator* collects the encrypted records $\tilde{\mathbf{D}}_\mathbf{i}$ from each of the data owners DO_i and collates them into a single encrypted database $\tilde{\mathcal{D}}$.
(2) **Program Executor** - The *Program Executor* is the most important module of the AS and is tasked with the execution of Cryptϵ programs. It takes as input a Cryptϵ program from an external analyst, alongside the appropriate privacy parameter ϵ and releases the differentially private output computed with the assistance of the CSP. Cryptϵ supports 9 different primitives on the encrypted data and a Cryptϵ program is an execution plan expressed as a sequence

of these Cryptϵ primitives. The primitives can be broadly classified into two types- transformation primitives and measurement primitives. Transformation primitives allow certain modifications on the encrypted data and are performed almost entirely by the AS. The measurement primitives on the other hand reveal some noisy measurement on the data and requires interaction with the CSP. A typical Cryptϵ program execution consists of a series of transformation on the encrypted data followed by a measurement primitive involving the CSP to decrypt the noisy answer. Both the AS and the CSP partake in the computation of the random noise to be added to the program output. This two-fold noise addition is necessary because had only one of them added the noise, then after the release of the differentially private output, that party can simply subtract out the noise to reveal the true private answer. Since in Cryptϵ the point of noise addition is just at the two servers, unlike at every individual in LDP, we are able to get constant error bounds just like in CDP (see section 7.1).

3.4 Cryptϵ at work

The complete workflow of Cryptϵ is outlined as follows
(1) **Setup Phase** - This is the very first step in Cryptϵ where the key manager of CSP generates the key pair for labHE (sk, pk), publishes pk and stores sk.
(2) **Data Collection Phase** - In the next phase, the *Data Encoder* and *Data Encryption* modules of every data owner, work to produce the encrypted data records which are then submitted to the AS. The data owners are relieved of all other duties in the Cryptϵ setting and can go completely off-line. The *Aggregator* module of the AS then aggregates these encrypted records into a single encrypted database $\tilde{\mathcal{D}}$.
(3) **Program Execution Phase** - In this phase, the AS executes a Cryptϵ program with some interaction with the CSP and generates a differentially private output.
The *Setup* and *Data Collection* phases occur just once at the very beginning, every subsequent program is handled via the corresponding *Program Execution* phase. Figure 1 shows the diagrammatic representation of Cryptϵ. A comparative analysis of Cryptϵ, CDP and LDP is presented in Appendix Table 5.

3.5 Discussion on Cryptϵ architecture

In this section we discuss and justify the architectural choices we make for Cryptϵ. The Cryptϵ setting is based on the premise that there are m individual data owners, each possessing a private data record, and the AS is the primary entity who is interested in learning certain differentially private statistics on the collective data. The primary goal is to be able to achieve the constant error accuracy guarantees of CDP without relying on a trusted server. This leads to the following requirements.
(1) We should have an efficient implementation of the cryptographic primitives that are practical.
(2) The AS should be shouldering the major chunk of the workload for any Cryptϵ program execution. Interactions with the CSP should be minimal and only related to data decryption.
(3) In the context of differential privacy, *sensitivity* is the value by which any individual can affect the output of a function. Cryptϵ programs should allow easy sensitivity analysis for the composition of the Cryptϵ primitives.
The first two requirements, have been instrumental behind our choice of using LHE for Cryptϵ. Firstly, LHEs have very efficient implementations in practice. Moreover the homomorphism allows certain operations (addition and multiplication when extended to labHE) to be performed directly on the encrypted data. This enables the AS to perform most of the data transformations by itself which conforms to our second requirement. Specifically for every Cryptϵ program, the AS processes the whole database and transforms it into concise representations (like an encrypted scalar or a short vector) which is then decrypted with the help of the CSP.

It is interesting to note that we could have had an alternative implementation for Cryptϵ where the private database is equally shared between the two servers and they engage in a secret share based secure computation protocol for computing the differentially private answers. However, this would require both the servers to do almost equal amount of work for every program. Such an arrangement would be justified only if both the servers are equally invested in learning the differentially private statistics and is ill-suited for Cryptϵ owing to our second requirement. Additionally, a secret-share based computation would be much more communication intensive resulting in a performance hit. Yet another alternative implementation could have been to engage the m data owners and the AS in a $m + 1$ secure computation protocol. This approach has two disadvantages. Firstly, as the number of data owners grow, the communication and computation overheads of this $m + 1$ secure computation protocol would get prohibitively high. Secondly, this violates our requirement of off-line data owners. Thus by introducing a separate entity in the form of the CSP, we factor out the onus of participating in a secure multi party computation from the data owners and capture their computation power in a secure protocol between just the CSP and the AS instead irrespective of the number of data owners.

The class of programs supported by Cryptϵ is any program that can be expressed as a composition of Cryptϵ primitives followed by arbitrary post-processing. Cryptϵ supports 7 transformation primitives and 2 measurement primitives. The noisy measurements allowed are the Laplace mechanism and the Noisy max (see section 4.2), two of the most popular and standard differentially private algorithms. The main constraint behind the design of the transformation primitives is that their composition should allow seamless sensitivity analysis. Sensitivity analysis of arbitrary relational algebraic queries is a hard problem, for e.g. sensitivity of conjunctive counting queries with join operator is unbounded [19]. Hence all the transformations that Cryptϵ supports have bounded stability [47] such that the sensitivity analysis of any program written using them is straightforward. In fact, we showcase that the proposed Cryptϵ primitives are indeed very powerful, enabling us to answer practically useful queries, such as linear counting queries etc.
Recall from our discussion in section 3.1, the architecture of Cryptϵ imposes no fundamental restriction on its expressibility and in theory we can enjoy the same algorithmic power as that of CDP. However, from the discussion above, we restrict the expressibility in the current implementation of Cryptϵ to ensure practical efficiency of Cryptϵ programs. Nevertheless, there is a separation between Cryptϵ and LDP as explained in Appendix E2. via Cryptϵ's feasibility for an example set of three problems namely DNF queries,

variable selection problem (Noisy-Max) and computing the number of distinct values for an attribute with a very large domain.

3.6 Cryptϵ Security Sketch

Here we present a brief security sketch of Cryptϵ. There are two security requirements for Cryptϵ-
(1) Firstly, all the programs executed by Cryptϵ should produce differentially private outputs.
(2) Secondly, the views of the two servers AS and CSP should be indistinguishable from that of a true CDP execution (modulo our assumption of a computationally bounded adversary). In other words the transcription of all the messages exchanged between the two servers can be simulated from just the differentialy private program outputs.
For the first security requirement, recall that any measurement of the data in the clear can be performed only by the measurement primitives with at least one interaction with the CSP. As mentioned in the preceding section, the measurement primitives supported by Cryptϵ implement the standard differentially private algorithms namely the Laplace mechanism and the Noisy-Max. Thus all Cryptϵ outputs are noisy.

The formal proof of the second security requirement is presented in the Appendix B; in this section we give the readers an intuitive understanding of why Cryptϵ is secure. From the AS's perspective, it has to interact with CSP for any measurement of the data. Recall that in Cryptϵ both the AS and the CSP are responsible for the noise addition. Hence, the CSP always adds noise of its own before releasing the decrypted data back to the AS. Moreover, in addition to the *Program Executor* module of the AS, the data analyst communicates the Cryptϵ program and the privacy parameter ϵ to the *Privacy Engine* module of the CSP. This enables the CSP to compute the sensitivity of the program for itself (required while adding its own noise) thereby thwarting any privacy budget over expenditure attempts under the semi-honest assumption. As for the CSP, its interactions with the AS can be categorized as
• Yao's garbled circuits with differentially private outputs
• decryption of ciphers for differentially private results
• decryption of ciphers for masked data
The garbled circuits are inherently semantically secure and the random masks in the third type of interactions hides the true plaintext from the CSP. Thus even the CSP is just allowed only a differentially private view of the database.

4 CRYPTϵ PRIMITIVES

Cryptϵ provides a set of Cryptϵ primitives for the data analyst. A sequence of these primitives forms a program which can be run by Cryptϵ. In this section, we first define these primitives and then show how to use these primitives to write useful programs. The analyst is given a database schema $\langle A_1, \ldots, A_k \rangle$ and an encrypted instance of this database, represented by $\tilde{\boldsymbol{\mathcal{D}}}$. There are two types of Cryptϵ primitives: (i) transformations and (ii) measurements, which are summarized in Table 2.

4.1 Transformation Primitives

Transformation primitives take an encrypted data as input and output a transformed encrypted data. These primitives thus work completely on the encrypted data and do not expend any privacy budget. Three types of data are considered in this context: (1) an encrypted table of x rows and y columns/attributes, denoted by $\tilde{\boldsymbol{T}}$, where each attribute value is represented by encrypted one-hot-encoding of the value; (2) an encrypted vector of numbers, denoted by $\boldsymbol{V}$; and (3) an encrypted scalar, denoted by $\boldsymbol{c}$. In addition, every encrypted table $\tilde{\boldsymbol{T}}$ of x rows has a encrypted bit vector $\boldsymbol{B}$ of size x to indicate whether the record is relevant to the program at hand. If the i^{th} bit value of B is 1, then the ith row in $\tilde{\boldsymbol{T}}$ will be used for answering the current program and vice versa. The input to the very first transformation primitive in Cryptϵ program is the encrypted database $\tilde{\boldsymbol{\mathcal{D}}}$ with all bits of $\boldsymbol{B}$ set to be 1. For brevity, we just use $\tilde{\boldsymbol{T}}$ to represent both the encrypted table $\tilde{\boldsymbol{T}}$ and $\boldsymbol{B}$. The transformation primitives are detailed below.
(1) **CrossProduct** $\times_{(A_i,A_j)\to A'}(\tilde{\boldsymbol{T}})$: This primitive transforms the two encrypted one-hot-encodings for attributes A_i and A_j in $\tilde{\boldsymbol{T}}$ into a single encrypted one-hot-encoding of a new attribute A'. The domain of the new attribute A' is the cross product of the domains for A_i and A_j. The resulting table $\tilde{\boldsymbol{T}}'$ has one column less than $\tilde{\boldsymbol{T}}$. Note that the construction of the one-hot-coding of the y-dimensional domain can be computed by repeated application of this transformation.
(2) **Project** $\pi_{\bar{A}}(\tilde{\boldsymbol{T}})$: This primitive projects $\tilde{\boldsymbol{T}}$ on a subset of attributes $\bar{A}$ of the input table. All the attributes that are not in $\bar{A}$ are discarded in the output table $\tilde{\boldsymbol{T}}'$.
(3) **Filter** $\sigma_{\phi}(\tilde{\boldsymbol{T}})$: This primitive specifies a filtering condition, represented by a boolean predicate ϕ defined over a subset of attributes $\bar{A}$ of the input table $\tilde{\boldsymbol{T}}$. The predicate can be expressed as a conjunction of range conditions over $\bar{A}$, i.e. for a row $r \in [x]$ in $\tilde{\boldsymbol{T}}$, $\phi(r) = \bigwedge_{A_i \in \bar{A}} (r.A_i \in V_{A_i})$, where $r.A_i$ is value of attribute A_i in row r and V_A is a subset of values (can be a singleton too) that attribute A_i can take. For example, $Age \in [30, 40] \wedge Gender = M$ can be one such filtering condition. The Filter primitive affects only the associated encrypted bit vector of $\tilde{\boldsymbol{T}}$ and keeps the actual table completely unchanged. In this primitive, if any row $r \in [x]$ in $\tilde{\boldsymbol{T}}$ does not satisfy the filtering condition ϕ, the corresponding r^{th} bit in $\boldsymbol{B}$ will be set to $labEnc_{pk}(0)$; otherwise, the corresponding bit value in $\boldsymbol{B}$ is kept unchanged. Thus the Filter transformation suppresses all the records that are extraneous to answering the program at hand (i.e., does not satisfy ϕ) by explicitly zeroing the corresponding indicator bits and outputs the table $\tilde{\boldsymbol{T}}'$ with the updated indicator vector.
(4) **Count** $count(\tilde{\boldsymbol{T}})$: This primitive simply counts the number of rows in $\tilde{\boldsymbol{T}}$ that are pertinent to the program at hand, i.e. the number of 1s in its associated bit vector $\boldsymbol{B}$. This primitive outputs an encrypted scalar $\boldsymbol{c}$.
(5) **GroupByCount*** $\gamma_A^{*,count}(\tilde{\mathbf{T}})$: The GroupByCount* primitive buckets the input table $\tilde{\mathbf{T}}$ into groups of records having the same value for an attribute A. The output of this transformation is thus, an encrypted vector $\mathbf{V}$ which is the encrypted histogram for A. This primitive serves as a preceding transformation for other Cryptϵ primitives such as NoisyMax, CountDistinct.
(6) **GroupByCount** $\gamma_A^{count}(\tilde{\mathbf{T}})$: This primitive is similar to the

Table 2: Cryptϵ Primitives

Types	Name	Notation	Input	Output	Functionality
Transformation	CrossProduct	$\times_{A_i,A_j \to A'}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\tilde{\boldsymbol{T}}'$	Generates a new attribute A' (in one-hot-coding) to represent the data for both the attributes A_i and A_j
	Project	$\pi_{A^*}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\tilde{\boldsymbol{T}}'$	Discards all attributes but A^*
	Filter	$\sigma_{\phi}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\boldsymbol{B}'$	Zeros out records not satisfying ϕ in $\boldsymbol{B}$
	Count	$count(\cdot)$	$\tilde{\boldsymbol{T}}$	$\boldsymbol{c}$	Counts the number of 1s in $\boldsymbol{B}$
	GroupByCount*	$\gamma_A^{*,count}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\boldsymbol{V}$	Returns encrypted histogram of A
	GroupByCount	$\gamma_A^{*,count}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\tilde{\boldsymbol{V}}$	Returns encrypted histogram of A in one-hot-encoding
	CountDistinct	$count^*(\cdot)$	$\boldsymbol{V}$	$\boldsymbol{c}$	Counts the number of non-zero values in $\boldsymbol{V}$
Measurement	Laplace	$Lap_{\epsilon,\Delta}(\cdot)$	$\boldsymbol{V}$	$\hat{V}$	Adds Laplace noise to $\boldsymbol{V}$
	NoisyMax	$NoisyMax^k_{\epsilon,\Delta}(\cdot)$	$\boldsymbol{V}$	$\hat{\mathcal{P}}$	Returns indices of the top k noisy values

aforementioned GroupByCount* primitive. The only difference between the two is that, GroupByCount outputs the encrypted histogram of attribute A with each count represented in one-hot-coding, $\tilde{V}$. This transformation allows us to answer queries based on the count of a particular value for attribute A.
(7) **CountDistinct** $count^*(\mathbf{V})$: As mentioned above, this primitive is always preceded by a GroupByCount* primitive. Hence the input vector $\mathbf{V}$ is an encrypted histogram for some attribute A and this primitive returns the number of distinct values of A that appear in $\tilde{\mathcal{D}}$ by counting the non-zero entries of V.

4.2 Measurement Primitives

The measurement primitives take encrypted vector of counts $\boldsymbol{V}$ (or a single count $\boldsymbol{c}$) as input and return noisy measurements on it in the clear. These implement two of the most popular differentially private mechanisms: Laplace mechanism and Noisy-Max mechanism. Both mechanisms add noise η drawn from Laplace distribution, denoted by $Lap(b)$, where $\Pr[\eta = x] \propto e^{-\frac{|x|}{b}}$. The scale of the noise depends on the transformations applied on the base database D. Let the sequence of transformations ($\bar{\mathcal{T}} = (\mathcal{T}_l, \ldots, \mathcal{T}_1)$) applied on D to get V be $\bar{\mathcal{T}}(D) = \mathcal{T}_l(\cdots \mathcal{T}_2((\mathcal{T}_1(D))))$. We define the *sensitivity* of a sequence of transformations as the maximum change to the output of this sequence of transformations when changing a row in the input database, i.e., $\Delta_{\bar{\mathcal{T}}} = \max_{D,D'} \|\bar{\mathcal{T}}(D) - \bar{\mathcal{T}}(D')\|_1$ where D and D' differ but a single record. The sensitivity of $\bar{\mathcal{T}}$ can be upper bounded by the product of the stability (definition in Appendix A2) of these transformation primitives, i.e. $\Delta_{\bar{\mathcal{T}}=(\mathcal{T}_l,\ldots,\mathcal{T}_1)} = \prod_{i=1}^{l} \Delta\mathcal{T}_i$. The transformation operators in Table 2 have a stability of 1, except for GroupByCount and GroupByCount* which have a stability of 2.
(1) **Laplace** $Lap_{\epsilon,\Delta}(\mathbf{V}/\boldsymbol{c})$: This primitive implements the classic Laplace mechanism [21]. Given an encrypted vector $\boldsymbol{V}$ or an encrypted scalar $\boldsymbol{c}$, a privacy parameter ϵ and sensitivity Δ of the preceding transformations, the primitive adds a noise vector (scalar) from $Lap(\frac{\Delta}{\epsilon})$. This primitive ensures ϵ-differential privacy when data analyst views the plaintext answer.
(2) **NoisyMax** $NoisyMax^k_{\epsilon,\Delta}(\mathbf{V})$: Noisy-Max is a type of differentially private selection mechanism [21] where the goal is to determine the query with the highest value out of n different noisy query outputs. The algorithm works as follows. First, generate each of the N answers and then add independent Laplace noise from the distribution $Lap(\frac{\Delta}{\epsilon})$ to each of them. The index of the largest noisy value is then reported as the noisy max. Thus the NoisyMax primitive of Cryptϵ takes as input of encrypted vector $\boldsymbol{V}$ where each vector element is a count alongside privacy parameter ϵ and sensitivity Δ. It then adds noise drawn from $Lap(\frac{\Delta}{\epsilon})$ to each vector element and computes the indices of the top k elements $\hat{\mathcal{P}}$.

4.3 Program Examples

Consider a database schema $\langle Age, Gender, NativeCountry, Department, Salary\rangle$. We show several Crypt$\epsilon$ program examples in Table 3 over this database.

5 IMPLEMENTATION

In this section we describe the implementation of Cryptϵ. First we discuss our novel proposed technique for extending the *labMult* operation of labHE to support $n > 2$ multiplicands. Then we describe the implementations for each primitive. Last, we use the example programs from the previous section to illustrate the performance of Cryptϵ.

5.1 General n-way Multiplication for labHE

In addition to the operations supported by a LHE scheme, labHE supports multiplication of two labHE ciphers.
•**labMult**$(\mathbf{c}_1, \mathbf{c}_2)$ - On input two labHE ciphers $\mathbf{c}_1 = (a_1, d_1)$ and $\mathbf{c}_2 = (a_2, d_2)$, it computes a "multiplication" ciphertext $\mathbf{e} = labMult(\mathbf{c}_1, \mathbf{c}_2) = Enc_{pk}(a_1, a_2) \oplus cMult(d_1, a_2) \oplus cMult(d_2, a_1)$. Observe that $Dec_{sk}(\mathbf{e}) = m_1 \cdot m_2 - b_1 \cdot b_2$.
•**labMultDec**$_{sk}(d_1, d_2, \mathbf{e})$ - On input two encrypted masks d_1, d_2 of two labHE ciphers $\mathbf{c}_1, \mathbf{c}_2$ and the output $\mathbf{e}$ of $labMult(\mathbf{c}_1, \mathbf{c}_2)$, it decrypts the product as $m_3 = Dec_{sk}(\mathbf{e}) + Dec_{sk}(d_1) \cdot Dec_{sk}(d_2) = m_1 \cdot m_2$.
In this paper we propose an efficient way of extending the *labMult* operation for a n-way multiplication. Let's consider an example case where we want to multiply the respective ciphers of 4 messages $\{m_1, m_2, m_3, m_4\} \in \mathcal{M}^4$ and obtain $\mathbf{e} = labEnc_{pk}(m_1 \cdot m_2 \cdot m_3 \cdot m_4)$. Note that the reason why we can't directly use *labMult* for a 4− way multiplication is because, the "multiplication" cipher $\mathbf{e} = labMult(\mathbf{c}_1, \mathbf{c}_2)$ does not have a corresponding label, i.e., it is not in the correct labHE cipher representation. Thus for generalizing the *labMult* operation for $n > 2$ multiplicands what we have to do is to generate a label and a seed for every intermediary product of two

Table 3: Examples of Cryptϵ Program

Cryptϵ Program	**Description**
P1: $Lap_{\epsilon,1}(count(\sigma_{Age\in[50,60]}(\pi_{Age}(\tilde{\mathcal{D}}))))$	Counts the number of records satisfying $Age \in [50, 60]$
P2: $NoisyMax^5_{\epsilon,1}(\gamma^{*,count}_{Age}(\tilde{\mathcal{D}}))$	Outputs the 5 most frequent age values.
P3: $Lap_{\epsilon,2}(\gamma^{*,count}_{Age\times Gender}(\pi_{Age\times Gender}(\times_{Age,Gender\rightarrow Age\times Gender}(\tilde{\mathcal{D}}))))$	Outputs the marginal over the attributes *Age* and *Gender*.
P4: $Lap_{\epsilon,2}(\gamma^{*,count}_{Age\times Gender}(\sigma_{NativeCountry=Mexico}(\pi_{Age\times Gender,NativeCountry}(\times_{Age,Gender\rightarrow Age\times Gender}(\tilde{\mathcal{D}})))))$	Outputs the marginal over *Age* and *Gender* for Mexican employees.
P5: $Lap_{\epsilon,1}(count(\sigma_{Age=30\wedge Gender=Male\wedge NativeCountry=Mexico}(\pi_{Age,Gender,NativeCountry}(\tilde{\mathcal{D}}))))$	Counts the number of male employees of Mexico having age 30.
P6: $Lap_{\epsilon,2}(count^*(\gamma^{*,count}_{Age}(\sigma_{Gender=Male}(\pi_{Age,Gender}(\tilde{\mathcal{D}})))))$	Counts the number of distinct age values for the male employees.
P7: $Lap_{\epsilon,2}(count(\sigma_{Count\in[10,m]}(\gamma^{count}_{Age}(\pi_{Age}(\tilde{\mathcal{D}})))))$	Counts the number of age values having at least 10 records.

Algorithm 1 *genLabMult* - generate label for *labMult*

```
    Input: $\mathbf{c_1} = labEnc_{pk}(m_1) = (a_1, d_1)$ where
        $a_1 = m_1 - b_1, d_1 = Enc_{pk}(b_1)$
        $\mathbf{c_2} = labEnc_{pk}(m_2)$
        $a_2 = m_2 - b_2, d_2 = Enc_{pk}(b_2)$
    Output: $\mathbf{e} = labEnc_{pk}(m_1 \cdot m_2)$
    AS:
1:  Computes $\mathbf{e}' = labMult(\mathbf{c_1}, \mathbf{c_2}) \oplus Enc_{pk}(r)$ where $r$ is a random mask
2:  Sends $\mathbf{e}', d_1, d_2$ to CSP
    CSP:
3:  Decrypts $\mathbf{e}'$, to get $Dec_{sk}(\mathbf{e}') = m_1 \cdot m_2 - b_2 \cdot b_1 + r$
4:  Computes $b_1 \cdot b_2$ from $d_1$ and $d_2$.
5:  Removes $b_1 \cdot b_2$ from $e'$ to compute $e'' = m_1 \cdot m_2 + r$
6:  Picks a seed $\sigma'$ and label $\tau'$
7:  Computes $\bar{a} = e'' - b' = m_1 \cdot m_2 + r - b', b' = \mathcal{F}(\sigma', \tau')$ and $d' = Enc_{pk}(b')$
8:  Send $\bar{e} = (\bar{a}, d')$ to AS
    AS:
9:  Computes true cipher $\mathbf{e} = (a', d')$ where $a' = \bar{a} - r = m_1 \cdot m_2 - b'$
```

Algorithm 2 GroupByCount $\gamma^{count}_{A}(\tilde{\mathbf{T}})$

```
    Input: $\tilde{\mathbf{T}}$
    Output: $\tilde{V}$
    AS:
1:  Computes $\mathbf{V} = \gamma^{*,count}_{A}(\tilde{T})$.
2:  Masks the encrypted histogram $\mathbf{V}$ for attribute $A$ as follows
        $\mathcal{V}[i] = \mathbf{V}[i] \oplus labEnc_{pk}(M[i])$
        $M[i] \in_R [m], i \in [|V|]$
3:  Sends $\mathcal{V}$ to CSP.
    CSP:
4:  Decrypts $\mathcal{V}$ as $\mathcal{V}[i] = labDec_{sk}(\mathcal{V})$.
5:  Converts each entry of $\mathcal{V}$ to its corresponding one-hot-coding and encrypts it, $\tilde{\mathcal{V}}[i] = labEnc_{pk}(\tilde{\mathcal{V}[i]})$
6:  Sends $\tilde{\mathcal{V}}$ to AS.
    AS:
7:  Rotates every entry by its corresponding mask value to obtain the desired encrypted one-hot-coding $\tilde{V}[i]$.
        $\tilde{V}[i] = RightRotate(\tilde{\mathcal{V}}, M[i])$
```

multiplicands. This can be done as shown by Algorithm 1. Note that the mask r protects the value of $m_1 \cdot m_2$ from the CSP in step 5. Similarly since b' is not known to the AS, $m_1 \cdot m_2$ remains hidden from the AS in step 9. In the aforementioned example, the AS first generates $\mathbf{e_{12}} = labEnc_{pk}(m_1 \cdot m_2)$ and $\mathbf{e_{34}} = labEnc_{pk}(m_3 \cdot m_4)$ using Algorithm 1. Both of this operations can be done in parallel in just one interaction round between the AS and the CSP. In the next round, now the AS can again use Algorithm 1 with inputs $\mathbf{e_{12}}$ and $\mathbf{e_{34}}$ to obtain the final answer $\mathbf{e}$. Thus for a generic $n - way$ multiplication the order of multiplication can be, in fact, parallelized as shown in Figure 4 (Appendix) to require a total of $\lceil \log n \rceil$ rounds of communication with the CSP.

5.2 Primitive Implementation

Here we explain the implementation details of two of the Cryptϵ primitives, the rest are covered in Appendix B.

(1) **GroupByCount** $\gamma^{*,count}_{A}(\tilde{\mathbf{T}})$- The GroupByCount primitive is implemented by Algorithm 2. In the first step the AS uses the GroupByCount* primitive to generate the encrypted histogram $\boldsymbol{V}$ for attribute A. Note that since each entry of $\mathbf{V}$ is a count of records, its value ranges from $\{0, ..., m\}$. The AS then masks $\boldsymbol{V}$ (step 2) and sends it to the CSP. The purpose of this mask is to hide the true histogram from the CSP. Next the CSP generates the encrypted one-hot-coding representation for this masked histogram $\tilde{\mathcal{V}}$ (steps 4-5) and returns it back to the AS. The AS can simply rotate $\tilde{\mathcal{V}}[i], i \in [|V|]$ by its respective mask value $M[i]$ (step 7) and get back the true encrypted histogram in one-hot-coding $\tilde{\boldsymbol{V}}$. Note that the GroupByCount primitive could have an alternative implementation using a Yao's garbled circuit that takes an input the encrypted vector and outputs the corresponding one-hot-coding representation. However this would require the circuit to decrypt and re-encrypt $O(m)$ data inside it which would be very computationally heavy.

(2) **Laplace** $Lap_{\epsilon,\Delta}(\mathbf{V})$ - Recall that both AS and CSP have to add Laplace noise to the output in Cryptϵ. Hence the Laplace primitive has two phase. In the first phase, the AS adds an instance of encrypted Laplace noise to the encrypted input to generate $\hat{\mathcal{V}}$. This acts as the input to the second phase which is executed by the CSP. Here the CSP decrypts $\hat{\mathcal{V}}$ and adds a second instance of the Laplace noise to generate the final noisy output $\hat{V}$ in the clear. The Laplace primitive with an encrypted scalar $\boldsymbol{c}$ as the input is implemented in a similar way.

Note: Thus in Cryptϵ we add two separate instances of the Laplace noise as opposed to just one instance of noise like in the traditional CDP setting. Thus although the errors of both Cryptϵ and CDP are of the same order $O\big(\frac{1}{\epsilon}\big)$, yet quantitatively the error values of Cryptϵ are twice that of the traditional CDP setting. This can be addressed by a joint computation of a single instance of noise by both AS and CSP as discussed in appendix E.1. Another observation is that this double noise addition does not affect the differential privacy guarantee. After the addition of the first instance of noise by the AS, the second can be seen as a post-processing step and differential privacy is post-processing immune. Hence our results Cryptϵ programs are still differentially private.

5.3 Classification of Cryptϵ Programs

Cryptϵ programs can be grouped into three classes based on the number and type of interaction between the AS and the CSP.

(1) **Class I - Single Decrypt Interaction Programs:**

Recall that the output of all the transformation primitives are encrypted. Since the CSP has exclusive access to the secret key, it is the only entity in the Cryptϵ setting capable of decryption. Thus for releasing any result (albeit noisy) in the clear, we need to interact at least once with the CSP so that it can decrypt the encrypted noisy answer. Cryptϵ supports this type of interactions via the two measurement primitives. Some Cryptϵ programs require only one round of interaction at the very end to release the noisy output. All other transformations can be performed by the AS via homomorphic operations on the encrypted data records. Typically these programs are counting queries on a single attribute or noisy max on a single attribute. Examples of this type of programs are P1 and P2 from Table 3.

(2) **Class II : LabHE Multiplication Interaction Programs-**

Recall that labeled homomorphic encryption allows multiplication of two ciphers. Generalization to a n-multiplicants, $n > 2$ case can be done using the protocol described in section 5.1. However it requires intermediate interactions with the CSP. Thus all Cryptϵ programs that require multiplication of more than two ciphers need interaction with the CSP. All programs with more than three attributes in its Boolean predicate would thus fall under this class. P3, P4 and P5 from table 3 fall in this class of Cryptϵ programs.

(3) **Class III : Other Interaction Programs-**

The GroupBy primitive requires an intermediate interaction with the CSP (for generating the encrypted one-hot-coding). The CountDistinct primitive also uses a Yao's garbled circuit (see section C) and hence requires interaction with the CSP. This is in addition to the interaction required for decrypting the noisy answer (as explained above). Thus any program with the GroupBy or CountDistinct primitive requires two rounds of interaction in the least. P6 and P7 from Table 3 are examples of this class of Cryptϵ programs.

5.4 Optimization

In this section we present several optimization techniques used by Cryptϵ. These optimizations preserve differential privacy.

Differential Privacy Based Optimizations

(1) **DP Indexing**

This optimization is motivated by the fact that the number of records needed by many programs is much smaller than the total dataset size. For example, the program P5 in Table 3 only needs records that have $Age = 30$. Hence, we create an index for the database on some attribute of choice A. There are two heuristics that can be considered for selecting this indexing attribute A. First, A should be very frequently queried. This is intuitive as this would mean a larger fraction of the queries will benefit from this optimization. Second, if $\{v_1, ...v_n\} \subset dom(A)$ is the set of most frequently queried values for attribute A, then $ct_{A,i}, i \in [n] << m$ where $ct_{A,i}$ is the number of records in $\tilde{\mathcal{D}}$ having value v_i for attribute A. This would ensure that the first selection performed alone on A will filter out majority of the records and reduce the intermediate dataset size to be considered for the subsequent predicate.

Given an encrypted database $\tilde{\mathcal{D}}$ and an attribute A of $\tilde{\mathcal{D}}$, we first use a garbled circuit to sort $\tilde{\mathcal{D}}$ over A into $\tilde{\mathcal{D}}_s$ and then build a differentially private indexing on the sorted database $\tilde{\mathcal{D}}_s$. The details are provided in Appendix C.2. Next we build the index on A given a privacy budget ϵ_A as follows. Let $P = (P_1, \ldots, P_k)$ be a uniform partition on the sorted domain of A such that each partition contains $\frac{s_A}{k}$ consecutive domain values. We compute a noisy CDF $\hat{C}$ vector $\hat{V}$ of length k such that $\hat{V}[i] = \sum_j ct_{A,j} + \eta_i$ where $i \in [k], j \in P_i$ and $\eta_i \sim Lap(1/\epsilon_A)$. Next the AS computes a noisy CDF, $\hat{C}$ over the k bins using the noisy counts in $\hat{V}$ using inference based on isotonic regression [35]. For executing a program conditioned as $\phi = A \in [v_s, v_e]$, we compute first compute $i_{start} = \hat{C}[r_s - 1], v_s \in [\frac{s_A}{k} \cdot (r_s - 1) + 1, \frac{s_A}{k} \cdot r_s]$, i.e., v_s belongs to bin r_s and $i_{end} = \hat{C}[r_e], v_e \in [\frac{s_A}{k} \cdot (r_e - 1) + 1, \frac{s_A}{k} \cdot r_e]$, i.e., v_e belongs to bin r_e. We then run the program on this subset of records in $[i_{start}, i_{end}]$. Note that for increased accuracy we can also consider preceding bins of $r_s - 1$ for i_{start} and succeeding bins of r_e for i_{end}.

Optimized feature - This optimization speeds up the execution time by reducing the number of total records to be processed for the program execution.

Trade-off - The trade-off is a possible increase in error as the noisy selection of records from the index might miss some of the records that do satisfy the filter condition.

Privacy Cost - Assuming the index is constructed with a privacy parameter ϵ, a selection of a subset of records using it will result in a ϵ - DP intermediate result. If ϵ' is the parameter used for the subsequent measurement primitives, then by Theorem 1, the total privacy parameter is $\epsilon + \epsilon'$.

(2) **DP Range Tree**

Range queries constitute a very popular category of queries for typical databases and range trees are a popular data structure constructed to speed up range query answering. A 1-dimensional range tree for an attribute A is an ordered data structure such that the leaf nodes correspond to the individual counts $ct_{A,i}, v_i \in dom(A)$ while each parent node stores the sum of the counts of its children. Hence an useful optimization for Cryptϵ can be pre-computing the noisy range tree for some of the most popular attributes. This can be useful for programs P1 and P2 in Table 3. The sensitivity for each such noisy range tree is $\log k$ where k is the domain size of the attribute on which the tree is constructed. For answering any arbitrary range query, we need to access at most $2 \log k$ nodes of the range tree. Thus to answer all possible range queries for the given attribute, the total squared error accumulated is $O(\frac{k^2(\log k)^2}{\epsilon})$. In contrast for the naive case, we would have incurred error $O(\frac{k^3}{\epsilon})$.

Hence this range tree optimization not only gives us a huge performance boost but also results in better answer accuracy.
Optimized Feature - The DP range tree optimization not only reduces the execution time, but also the expected error when executed over multiple range queries.
Trade-off - The trade-off for this optimization is its storage cost ($O(2 \cdot s_A)$).
Privacy Cost - If the range tree is constructed with privacy parameter ϵ_R then any measurement on it is ϵ_R-DP.

Implementation Based Optimizations

(1) **Precomputation**

Recall that the data owners DO$_i$ send per-attribute encrypted one-hot-codings of their data. We can use the CrossProduct primitive to generate the one-hot-coding of data across two attributes. However due to the intermediate interactions required with the CSP, the CrossProduct is a very costly operation. Hence an useful optimization can be pre-computing the one-hot-coding for the data across a set of popular attributes $A^* \subset \mathcal{A}$ so that during subsequent program executions, the AS can get the desired representation via simple look-ups. This can be useful for example P3 in Table 3.
Optimized Feature - This optimization improves the execution time of Cryptϵ programs. Moreover, once the multi-attribute one-hot-codings are created it can be re-used for all subsequent programs.
Trade-off - The trade-off for this pre-computation is the storage cost (O($s_{A^*} = \prod_{A \in A^*} s_A$)) incurred to store the multi-attribute one-hot-codings for A^*.
Privacy Cost - Since this computation is carried completely on the encrypted data, we do not expend any privacy budget.
(2) **Off-line Processing**

Recall that the in the implementation of GroupByCount primitive, the CSP needs to generate the encrypted one-hot-codings for the masked histogram (step 5 in Algorithm 2). But the one-hot-coding representation for any such count would simply be a vector of $m-1$ $labEnc_{pk}(0)$ ciphers and 1 $labEnc_{pk}(1)$ cipher. Thus one useful optimization can be generating these ciphers for 0 and 1 in an off-line phase. (This is similar to the idea of off-line generation of Beaver's multiplication triples [8] used in secure multi-party computation.) In this way the encryption time will not be a part of the program execution time thereby giving us a performance boost.
Optimized Feature - This optimization results in a reduction in the run time of Cryptϵ programs.
Trade-off - The trade-off for this optimization is the storage cost (O($m \cdot s_A$)) incurred to store the ciphers for attribute A.
Privacy Cost - The computation is carried completely on the encrypted data, we do not expend any privacy budget.

6 EXPERIMENTAL EVALUATION

In this section we experimentally evaluate Cryptϵ to assess its practical utility based on two parameters, the accuracy and the performance of the Cryptϵ programs. Specifically
Q1: Does Cryptϵ programs have constant error bounds depending only on the privacy parameter ϵ which is at least $O(\sqrt{m})$ times lower than that for the corresponding state-of-the-art LDP implementation?
Q2: Is the program execution timings for Cryptϵ practical?
Q3: Does the proposed optimizations provide substantial performance improvement over the base case Cryptϵ implementation?

6.1 Methodology

Programs: To answer the aforementioned questions we ran the experiments on 4 of the Cryptϵ programs previously outlined in Table 3, namely P1,P3,P5 and P7. In this section we rename P1,P5,P7 and P3 as Program A, Program B, Program C and Program D respectively. Additional experimental results for programs P2,P4 and P6 from Table 3 are presented in Appendix.
Dataset: We ran our experiments on the Adult dataset from the UCI repository [2] which has been extracted from the 1994 census data. The dataset is of size 32, 651.
Metrics:
Accuracy: For accuracy the following metrics are used
• Programs A, B and C use absolute error = $|c - \hat{c}|$ where c is the true count and $\hat{c}$ is the noisy output.
• Programs D uses the $L1 - Norm$ error metric given by $Error = \sum_i |V[i] - \hat{V}[i]|, i \in [|V|]$.
For each of the programs, we report the mean error values over 10 repetitions.
Performance: For measuring performance we report the mean total execution time in seconds of each program, over 10 repetitions.

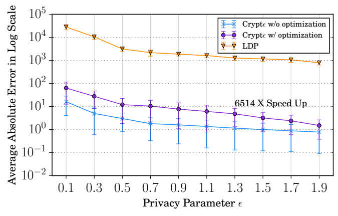
For the rest of the section, we consider the optimized implementation to be the default implementation for Cryptϵ and refer to the unoptimized implementation as the base case.

6.2 Experimental Results

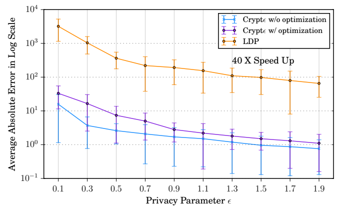
Accuracy

In this section we evaluate **Q1** by performing a comparative analysis between the empirical accuracy of the aforementioned four Cryptϵ programs (both optimized and unoptimized) and that of the corresponding state-of-the-art LDP implementations [60].

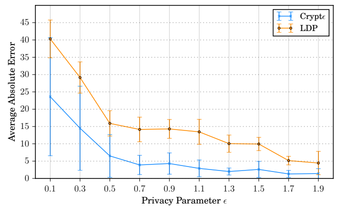
For Program A, Cryptϵ constructs a DP range tree over the attribute Age with the full privacy budget, ϵ. Program B is optimized by constructing a DP index over the attribute $NativeCountry$. For our experiments, we use 20% of the total privacy budget towards constructing the index, the remaining 80% is reserved for the rest of the program execution. The rational behind selecting the value 20% will be discussed in section 6.2.3.
Observations: The main observation with respect to accuracy is that the mean error for a single frequency count for Cryptϵ is of the order $O(\frac{1}{\epsilon})$. In fact quantitatively it is close to $\frac{2}{\epsilon}$. This conforms to our expectation as we add two instances of Laplace noise at the AS and the CSP. In contrast, the mean error of the corresponding LDP implementation is of the order $O(\frac{\sqrt{m}}{\epsilon})$. For e.g., in Figure 2a which shows our error analysis for Program A, we observe that the mean error of base case Cryptϵ is 16.10 for $\epsilon = 0.1$ and 0.78 for $\epsilon = 1.9$. The mean error values with the DP range tree optimization are slightly poorer (at most 6 × worse) as compared to that of the base case. For e.g., for $\epsilon = 0.1$, the mean error for the optimized implementation is 63.5 while that for the base case is 16.1. This is so because the sensitivity of the range tree is $\log k$



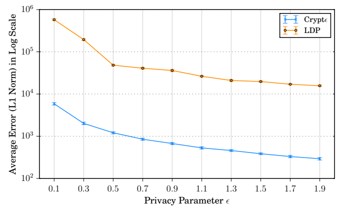
(a) **Accuracy Analysis of Program A**



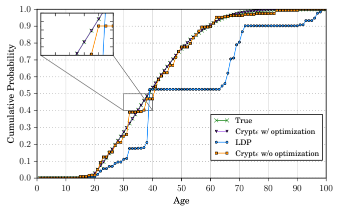
(b) **Accuracy Analysis Program B**



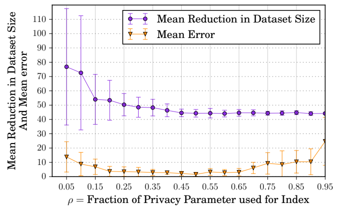
(c) **Accuracy Analysis of Program C**



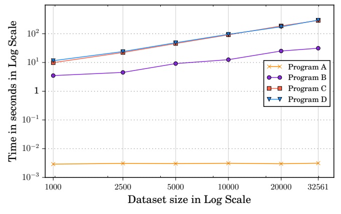
(d) **Accuracy Analysis of Program D**



(e) **DP Range Tree Evaluation**



(f) **Privacy Budget Analysis for DP Index**



(g) **Cryptϵ Program Scalability**

Figure 2: **Empirical Evaluation of Cryptϵ Programs**

where k is the number of leaves (100 in this case) and a range query can take up to $2\log k$ nodes for answering. Thus for a single query, programs executed on the DP range tree has an expected error of $O(\frac{log k}{\epsilon})$. The accuracy gain for the range tree kicks in for answering multiple range queries as showcased in Figure 2e (see section 6.2.3). In contrast, the corresponding LDP implementation has at least $1000 \approx \frac{11}{2} \cdot \sqrt{m} = \frac{11}{2} \cdot \sqrt{323561}$ times worse error values. This observation is intuitive as for answering Program A we need to read 11 noisy counts from the frequency oracle (for the range [50,60]) and the factor of $\frac{1}{2}$ comes from the fact that the errors in base case Cryptϵ are roughly $\frac{2}{\epsilon}$. For e.g., the mean error for LDP is 1747× higher than that of base case Cryptϵ for $\epsilon = 0.1$.

Figure 2b shows the error analysis for Program B. From the figure, we observe that as expected, $\epsilon = 0.1$ results in a mean error of 15.8 for base case Cryptϵ while for $\epsilon = 1.9$ we get a mean error of 0.76. Using the DP index, gives slightly higher error values (around 2×) than that of base case Cryptϵ. It is so because, the noisy index might miss out some the records that do satisfy the filter condition. A very loose upper bound on the error of the program execution with the DP index can be $\frac{1}{0.2\epsilon} + \frac{2}{0.8\epsilon} = \frac{7.5}{\epsilon}$. Thus even with the DP index optimization we get constant error bounds (since the noise added is independent of the number of records and depends only on the privacy parameter and the number of partitions/bins). On the other hand, the corresponding LDP implementation is at least $90 \approx \frac{\sqrt{32561}}{2}$ times worse as compared to base case Cryptϵ. For e.g., for $\epsilon = 0.1$ the mean error for LDP is 200× worse than that of the base case Cryptϵ implementation.

We make similar observations for Program D whose results are showed in Figure 2d. Specifically we observe that the error values of the LDP implementation is around $\frac{\sqrt{m}}{4} = 45$ times higher than that of Cryptϵ. There is an extra $\frac{1}{2}$ factor since the corresponding Cryptϵ program is 2− stable (due to the GroupByCount* primitive). For e.g., $\epsilon = 0.1$ results in almost 98× higher error for the LDP implementation as compared to that of Cryptϵ.

Recall Program C outputs the number of age values with at least 100 records. Thus Program C does not directly report counts of records in the database but some statistic on an attribute based on the counts. Even here we observe that the accuracy of Cryptϵ is significantly higher than that of the corresponding LDP implementations. For e.g., Figure 2c shows that for $\epsilon = 0.1$ the mean error for Cryptϵ is 23.6 while that for the LDP implementation is 40.3.

In this paragraph we list some additional observations about the accuracy of the aforementioned Cryptϵ programs and their corresponding LDP implementations. For Program B, from Figure 2b, we observe that the LDP implementation errors are roughly $\frac{1}{11}$× that of the corresponding LDP errors for Program A. It is so because Program B outputs a single noisy count only as opposed to requiring 11 noisy counts for Program A. Additionally from Figure 2d we observe that the mean error values for Cryptϵ (both base case and optimized) are approximately $200 \cdot 2$× larger than the corresponding values for that for Programs A and B. For e.g., the mean error for Program B is 5862.4 and 292.9 for $\epsilon = 0.1$ and $\epsilon = 1.9$ respectively. It is so because, the attribute *Age* has domain size 100 while attribute *Gender* is of size 2. Hence *Age* × *Gender* is of domain size 200. The additional factor of 2 comes from the 2-stability of Program D. Similarly, the mean errors for the LDP implementation of Program D are around $18 = 200/11$ times that of the corresponding values for Program A and 200× that of the corresponding values for Program B.

Optimizations

In this section we evaluate **Q2** by analysing the speed-up in execution time **DP Range Tree** For Program A we see that the total time taken for execution for the base case Cryptϵ implementation is about 19 seconds while using the range tree optimization we get a 6514× speed up in timings. It is so because the time required by the AS in the optimized implementation becomes almost negligible because it simply needs to do a memory fetch to read off the answer from the pre-computed range tree instead of computing it from the entire encrypted database.

Table 4: Execution Time Analysis for Cryptϵ Programs

Time in (s)		Program			
		A	**B**	**C**	**D**
Base	AS	18.89	650.78	290	101840.8316
	CSP	0.0027	550.34	30407.73	78131.27
	Total	18.8927	1201.12	30697.73	179972
Optimized	Total	0.0031	30.13	290.98	298.81
	Speed Up ×	6514.7	40	105.49	602.29

In addition to a performance boost, the DP range tree optimization also increases the accuracy of Cryptϵ over multiple range queries as shown in Figure 2e. For this experiment, we construct the c.d.f over the attribute *Age* (with domain size 100) by first executing 100 range queries, where the i-th query outputs the noisy count for age values in $[1, i], i \in [100]$. This is followed by a isotonic regression post-processing to get a proper c.d.f. We use a total privacy budget of $\epsilon = 1$ for all the 100 queries. Thus in the base case each query gets privacy parameter $\frac{\epsilon}{100}$ while the DP range tree is constructed with privacy parameter ϵ and sensitivity $\rceil \log 100 \lceil$. From a visual analysis of the plots in Figure 2e, we observe that the plots for the true c.d.f and the one obtained from optimized Cryptϵ are almost indistinguishable. In fact the mean L1-Norm error of the c.d.f for the optimized Cryptϵ is just 0.0045. Although the plot for the unoptimized implementation Cryptϵ captures the overall shape of the c.d.f correctly, it shows slight aberrations in some points and has a mean L1-Norm error of 1.211. On the other hand, the c.d.f obtained from the LDP implementation is way off and results in a mean L1-Norm error of 12.93.

DP Index- For Program B, we observe that the base case implementation takes around 20 minutes to run. However a DP index over the attribute $NativeCountry$ reduces the execution time to about 2s only giving us a 40× speedup. It is so because, only about 2% of the data records satisfy $NativeCountry$=Mexico. Thus the index reduces the number of records to be considered for the program execution drastically thereby resulting in a huge performance boost. Let ρ represent the fraction of the privay parameter used towards constructing the DP index. As mentioned before, $\rho = 0.2$ in this experimental setup. In Figure 2f we study how the expected speed up in execution time and the accuracy of the final result vary as a function of ρ for Program B for a total privacy parameter of $\epsilon = 1$. We measure the expected speed up as $\frac{\text{Total dataset size=32561}}{\text{\# of records to be considered as indicated by the DP index}}$. From Figure 2f we observe that the error reduces till $\rho = 0.2$ and stabilises until $\rho = 0.65$ after which it starts rising again. This rise in error is because, as we keep increasing ρ, the amount of privacy parameter left for the measurement primitive keeps decreasing and after a certain point ($\rho = 0.65$ in our experimental setup), starts dominating the total error. Also we observe that at about $\rho = 0.2$, the index is able to approximately identify the correct set of records and hence increasing ρ further does not affect the speed up or its contribution to the total error by much. Hence we choose $\rho = 0.2$ for our experiments. A formal accuracy vs speed up trade-off analysis would be very helpful in this regard and is part of our future work.

Pre-computation- For Program D the unoptimized execution time on the dataset of 32561 records is around 2 days. This is so because the CrossProduct primitive used in the program needs to perform 200 · 32561 *labMult*() operations which is very time consuming. Hence from Table 4 we see that, pre-computing the 2-dimensional attribute over *Age* and *Gender* is a very useful optimization as now the execution reduces to just about 4 minutes giving us a 717.6× speed up.

Off-line Processing The most costliest primitive for Program C is the specifically since the CSP has to generate for the one-hotcodings. In the base case this takes about . But with the off-line geneates then the execution time cuts down to just which gives us a speed uo of

Performance

Figure 2g showcases how well the Cryptϵ programs scale. All the reported execution times are for optimized implementations. For Program A we see that the the execution time remains unchanged for all the dataset sizes. It is so because once the range tree is constructed, irrespective of the dataset size, the program execution just involves reading the answer directly from the modes of the tree followed by a decryption by the CSP. For Program B, the execution time basically depends on the % of the records in the given dataset that satisfy the condition $NativeCountry = Mexico$ (as this is roughly the number of records that will be retrieved from the noisy index). From the figure, we observe that Program B scales almost linearly with the dataset size. whch is a of For Program C clearly, the cost is dominated by the is . Thus we conclude that the optimized implementations Cryptϵ program running times are practical and only takes about a few minutes to run. The execution time for the optimized Program D is dominated by the cost of $\oplus$ operations for the GroupBy primitive which scales linearly with the number of data records. Another important observation from Table 4 is that the time taken by the AS is significantly larger than that (except Program A due to the DP range tree optimization). This indicates that the AS performs conforms to our second requirement in section.

7 CONCLUSION

In this paper we have proposed a new implementation setting for differential privacy that allow us to achieve the constant error bounds of the central differential privacy setting without the need for any trusted server. This is achieved via the assistance of cryptographic primitives specifically linear homomorphic encryption and Yao's garbled circuits.

One possible extension of the current work can be the development of a Cryptϵ Compiler. Recall that currently the data analyst spells out the explicit Cryptϵ program (i.e., the sequence of Cryptϵ primitives and their arguments) to the AS. Thus a useful future work can be constructing a Cryptϵ compiler that takes as input only a user specified query in a high-level-language and a total privacy budget for the query. The Cryptϵ compiler should then be able to formalize an optimized Cryptϵ program expressed in terms of Cryptϵ primitives with automated sensitivity analysis and subsequent optimal per measurement primitive privacy budget allocation. Another future work direction can be developing a formal performance-accuracy trade-off framework for choosing the optimal privacy budgeting for the proposed optimizations. Yet another goal for future work is to support a larger class of queries like joins et al.

REFERENCES

[1] Apples differential privacy is about collecting your databut not your data.

[2] A.Asuncion and D. Newman. Uci machine learning repository, 2010.

[3] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *2012 IEEE 12th International Conference on Data Mining*, pages 1–10, Dec 2012.

[4] A. Agarwal, M. Herlihy, S. Kamara, and T. Moataz. Encrypted databases for differential privacy, 2018. https://eprint.iacr.org/2018/860.

[5] G. Barthe, M. Gaboardi, B. GrÃľgoire, J. Hsu, and P.-Y. Strub. Proving differential privacy via probabilistic couplings. *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science - LICS âĂŽ16*, 2016.

[6] R. Bassily and A. Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 127–135, New York, NY, USA, 2015. ACM.

[7] R. Bassily and A. Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 127–135, New York, NY, USA, 2015. ACM.

[8] D. Beaver. Precomputing oblivious transfer. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, pages 97–109, Berlin, Heidelberg, 1995. Springer-Verlag.

[9] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 441–459, New York, NY, USA, 2017. ACM.

[10] J. Blocki, A. Blum, A. Datta, and O. Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, Oct 2012.

[11] J. W. Bos, W. Castryck, I. Iliashenko, and F. Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In M. Joye and A. Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, pages 184–201, Cham, 2017. Springer International Publishing.

[12] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[13] T. H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In A. D. Keromytis, editor, *Financial Cryptography and Data Security*, pages 200–214, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[14] T. Chanyaswad, A. Dytso, H. V. Poor, and P. Mittal. Mvg mechanism: Differential privacy under matrix-valued query. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 230–246, New York, NY, USA, 2018. ACM.

[15] A. Cheu, A. D. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via mixnets. *CoRR*, abs/1808.01394, 2018.

[16] G. Cormode, T. Kulkarni, and D. Srivastava. Marginal release under local differential privacy. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 131–146, New York, NY, USA, 2018. ACM.

[17] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. *2012 IEEE 28th International Conference on Data Engineering*, Apr 2012.

[18] W.-Y. Day and N. Li. Differentially private publishing of high-dimensional data using sensitivity control. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 451–462, New York, NY, USA, 2015. ACM.

[19] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, 2006.

[20] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3&#8211;4):211–407, Aug. 2014.

[21] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3&#8211;4):211–407, Aug. 2014.

[22] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 51–60, Washington, DC, USA, 2010. IEEE Computer Society.

[23] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. *CoRR*, abs/1811.12469, 2018.

[24] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1054–1067, New York, NY, USA, 2014. ACM.

[25] G. Fanti, V. Pihur, and ÃŽlfar Erlingsson. Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries, 2015.

[26] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Secure linear regression on vertically partitioned datasets. *IACR Cryptology ePrint Archive*, 2016:892, 2016.

[27] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017:345–364, 2017.

[28] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In B. Preneel and F. Vercauteren, editors, *Applied Cryptography and Network Security*, pages 243–261, Cham, 2018. Springer International Publishing.

[29] I. Giacomelli, S. Jha, R. Kleiman, D. Page, and K. Yoon. Privacy-preserving collaborative prediction using random forests, 2018.

[30] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. R. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.

[31] M. Hardt, K. Ligett, and F. Mcsherry. A simple and practical algorithm for differentially private data release, 2012.

[32] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2339–2347, USA, 2012. Curran Associates Inc.

[33] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70, Oct 2010.

[34] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021âĂŞ1032, Sep 2010.

[35] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1-2):1021–1032, Sept. 2010.

[36] E. Hesamifard, H. Takabi, and M. Ghasemi. Cryptodl: Deep neural networks over encrypted data, 2017.

[37] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth. Differential privacy: An economic method for choosing epsilon. *2014 IEEE 27th Computer Security Foundations Symposium*, pages 398–410, 2014.

[38] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 531–540, Oct 2008.

[39] M. Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6):983–1006, Nov. 1998.

[40] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 341–352, New York, NY, USA, 1992. ACM.

[41] S. Kim, J. Kim, D. Koo, Y. Kim, H. Yoon, and J. Shin. Efficient privacy-preserving matrix factorization via fully homomorphic encryption: Extended abstract. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 617–628, New York, NY, USA, 2016. ACM.

[42] J. Lee and C. Clifton. How much is enough? choosing ϵ for differential privacy. In *Proceedings of the 14th International Conference on Information Security*, ISC'11, pages 325–340, Berlin, Heidelberg, 2011. Springer-Verlag.

[43] C. Li, M. Hay, G. Miklau, and Y. Wang. A data- and workload-aware algorithm for range queries under differential privacy. *Proceedings of the VLDB Endowment*, 7(5):341âĂŞ352, Jan 2014.

[44] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 123–134, New York, NY, USA, 2010. ACM.

[45] Y. Lindell and B. Pinkas. A proof of security of yao&#x2019;s protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, Apr. 2009.

[46] M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy. *Proceedings of the VLDB Endowment*, 10(6):637âĂŞ648, Feb 2017.

[47] F. 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*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[48] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, May 2017.

[49] A. Narayan and A. Haeberlen. Djoin: Differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 149–162, Berkeley, CA, USA, 2012. USENIX Association.

[50] T. T. Nguyên, X. Xiao, Y. Yang, S. C. Hui, H. Shin, and J. Shin. Collecting and analyzing data from smart device users with local differential privacy. *CoRR*, abs/1606.05053, 2016.

[51] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, pages 801–812, New York, NY, USA, 2013. ACM.

[52] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348, May 2013.

[53] A. Nikolov, K. Talwar, and L. Zhang. The geometry of differential privacy. *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC âĂŹ13*, 2013.

[54] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 757–768, April 2013.

[55] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *Proc. VLDB Endow.*, 6(14):1954–1965, Sept. 2013.

[56] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 192–203, New York, NY, USA, 2016. ACM.

[57] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.

[58] E. Shi, T.-H. Hubert Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. volume 2, 01 2011.

[59] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 991–1008, Berkeley, CA, USA, 2018. USENIX Association.

[60] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, pages 729–745, Berkeley, CA, USA, 2017. USENIX Association.

[61] T. Wang, N. Li, and S. Jha. Locally differentially private heavy hitter identification, 2017.

[62] T. Wang, N. Li, and S. Jha. Locally differentially private frequent itemset mining. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 127–143, May 2018.

[63] S. L. Warner. âĂIJrandomized response: A survey technique for eliminating evasive answer bias.âĂİ. *Journal of the American Statistical Association*, 60 60, no. 309:63âĂŞ69, 1965.

[64] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: Differential privacy with reduced relative errors. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 229–240, New York, NY, USA, 2011. ACM.

[65] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010.

[66] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *2012 IEEE 28th International Conference on Data Engineering*, pages 32–43, April 2012.

[67] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, Oct 1986.

[68] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism. *Proceedings of the VLDB Endowment*, 5(11):1352âĂŞ1363, Jul 2012.

[69] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Optimizing batch linear queries under exact and approximate differential privacy. *ACM Trans. Database Syst.*, 40(2):11:1–11:47, June 2015.

[70] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 587–595.

[71] Z. Zhang, T. Wang, N. Li, S. He, and J. Chen. Calm: Consistent adaptive local marginal for marginal release under local differential privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 212–229, New York, NY, USA, 2018. ACM.

A BACKGROUND CNTD

Differential Privacy Cntd

Theorem 2. *(Sequential Composition). If μ_1 and μ_2 are ϵ_1-DP and ϵ_2-DP algorithms that use independent randomness, then releasing the outputs $\mu_1(D)$ and $\mu_2(D)$ on database D satisfies $\epsilon_1 + \epsilon_2$-DP.*

There exist advanced composition theorems that give tighter bounds on privacy loss, but we use sequential composition as defined above for our paper.

Secure Computation cntd In this section we will be describing garbled circuits in more detail. The function f is evaluated via a Boolean circuit. First the garbler associates two random cryptographic keys K^0_{wi} and K^1_{wi} with each wire w_i of the circuit such that they correspond to the bits $b_i = 0$ and $b_i = 1$ respectively.

Stability of a transformation

Definition 2. *A transformation $\mathcal{T}$ is defined to be $t-$ stable if for two datasets D and D', we have*

$$|\mathcal{T}(D) \ominus \mathcal{T}(D')| \leq t \cdot |D \ominus D'| \tag{2}$$

where $\ominus$ denotes the symmetric difference (i.e., $D \ominus D'$ is the set of records in D or D', but not both.

Transformations with bounded stability scale the differential privacy guarantee of their outputs, by their stability constant [47].

Theorem 3. *Let μ provide ϵ-differential privacy, and let $\mathcal{T}$ be an arbitrary t-stable transformation. The composite computation $\mu \circ \mathcal{T}$ provides $(\epsilon \cdot t)$-differential privacy.*

Noisy Max Cntd The Noisy Max algorithm has two fold advantage over the naive implementation of finding the maximum count. Firstly, noisy-max applies "information minimization" as rather than releasing all the noisy counts and allowing the analyst to find the max and its index, only the index corresponding to the maximum is made public. Secondly, the noise added is much smaller than that in the case of the naive implementation (it has sensitivity $m \cdot \Delta$).

B SECURITY ANALYSIS

B.1 Security Model

The involved parties in Cryptϵ are m data owners $DO_1, \ldots, DO_m$ and two servers AS and the CSP. In this paper we consider a semi-honest adversary Adv who can corrupt any subset of the data owners and at most one of the two servers. This captures our assumption that the two servers are non-colluding, i.e. if one is controlled by the adversary, the second one behaves honestly. Thus the requisite security definition should state that Adv only learns the data of the corrupted data owners and the final differentially private output but nothing else about the honest data owners. In other words, the view of the adversary Adv should be computationally indistinguishable from a truly differentially private protocol. Answering a query in Cryptϵ can be considered to be a two-phase protocol. In the first phase the data owners submit there data encrypted under a labeled homomorphic encryption scheme to the AS. Let this phase be represented by protocol Π_1. This is followed by the actual computation of the query answer via a joint computation of an equivalent Cryptϵ program by the As and the CSP. Let protocol Π_2 represent this second phase. Using this the ideal functionality of Cryptϵ can be represented by Figure 3. To formally prove the security, we use the standard simulation based definition [?]. Let us consider a public function $\phi : (\{0,1\}^k)^n \mapsto \{0,1\}^t$ and let $P_1; \cdots; P_n$ be n players modeled as PPT machines. Each player P_i holds the value $x_i \in \{0,1\}^k$ and wants to compute the value

Parties:- *AS, CSP and* DO_i, $\forall i \in \{1, ..., m\}$
Input : $-D_i, \forall i \in \{1, ..., m\}$, *Query Q*
Output:- *Differentially private output for Q,* $\hat{c}$

Phase 1: *AS, CSP and* DO_i *jointly run protocol* Π_1
Phase 2: *AS and CSP run protocol* Π_2 *to generate* $\hat{c}$

Figure 3: **Ideal functionality $\mathcal{F}$ of Cryptϵ**

$\phi(x_1; \cdots ; x_n)$ while keeping his input private. The players can communicate among them using point-to-point secure channels in the synchronous model. If necessary, we also allow the players to use a broadcast channel. To achieve their goal, the players jointly run a n-party MPC protocol . The latter is a protocol for n players that is specified via the next-message functions: there are several rounds of communication and in each round the player P_i sends to other players a message that is computed as a deterministic function of the internal state of P_i (his initial input x_i and his random tape k_i) and the messages that P_i has received in the previous rounds of communications. The view of the player P_j , denoted by $ViewP_j()$ (a1, . . . , an), is defined as the concatenation of the private input aj , the random tape kj and all the messages received by Pj during the execution of Π. Finally, the output of Π for the player P_j can be computed from the view ViewPj . In order to be private, the protocol Π needs to be designed in such a way that a curious player P_i cannot infer information about $x_j, j \neq i$ from his view $ViewP_i$. More precisely, we have the following definition.

Definition 3. *Let S a subset of at most n − 1 players, the protocol* Π *realizes* ϕ *with privacy against S if it is correct and there exists a PPT algorithm Sim such that* $(ViewP_i)_{P_i \in S}$ *and* $Sim(\{x_i, P_i \in S\}, \phi(x_1, ..., x_n))$ *are computationally indistinguishable for all inputs.*

The evaluation of any Cryptϵ program can be seen as an MPC for m + 2 parties: $DO_1, ..., DO_m$, AS and CSP.

Theorem 4. *Let* $S \subset \{1, ..., m\}$ *then* Π *realizes* ψ *with correctness and privacy against the adversaries* $Adv_1 = AS \cup \{DO_i, i \in S\}$ *and* $Adv_2 = CSP \cup \{DO_i, i \in S\}$

Proof. Let simulators S_1 and S_2 simulate the view of Adv_1 and Adv_2 respectively. Let l be the length of the data-records when expressed in the suitable attribute-wise one-hot-coding representation.
$S_1 = Sim_1(\{D_i, i \in S\}, \hat{c}, pk, \eta_1)$

(1) Run $Gen(\kappa) \mapsto (pk, sk)$
(2) Draw $\eta' \in Lap(\frac{1}{\epsilon})$
(3) Compute $c' = \hat{c} - \eta' - \eta_1$
(4) For all $i \in \{1, ..., m\}$ if $i \in S$ then encrypt D_i. Otherwise compute $Enc_pk(D_i)$ as a random l-lengthed vector $r_i \in C^l$.
(5) Output $\left(\{D_i, i \in S\}, pk, D', Enc_{pk}(c'), \hat{c}\right)$ where $D' = \{r_i\}, i \in \{1, ..., m\} i \notin S$ as computed in step 4.

It follows from the semantic security of the encryption scheme that the simulation output has the same distribution of the views of the corrupted parties in adv_1 in the protocol π. $S_2 = Sim_2(D_i, i \in S, \hat{c} = c + \eta_1 + \eta_2, pk, sk,)$

(1) Run $Gen(\kappa) \mapsto (pk, sk)$
(2) $\bar{c} = \hat{c} - \eta_2 = c + \eta_1$
(3) Output $\left(\{D_i, i \in S\}, pk, Enc_{pk}(\bar{c}), \hat{c}\right)$

Thus the simulation output has the same distribution of the views of the corrupted parties in Adv_2 in the protocol Π.

□

Table 5: Comparative analysis of the different DP models

Features	LDP	CDP	Cryptϵ
# Servers	1	1	2
Trust Assumption for Server	Untrusted	Trusted	Untrusted Semi Honest Non-Colluding
Data Storage in Server	Noisy	Clear	Encrypted
Adversary	Unbounded	Unbounded	Bounded
Error	$O\left(\frac{\sqrt{(n)}}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon}\right)$
Impossibility results	Everything beyond SQ model	Algorithms w/ Unbounded Sensitivity	Same as that of CDP[1]

C CRYPTϵ IMPLEMENTATION

C.1 Cryptϵ Primitives Implementation

1) **CrossProduct**($\tilde{\mathbf{T}}, A_i, A_j$): Let D_1 and D_2 be the encrypted one-hot-coding corresponding to two values v_1 and v_2 (integral representation) for attributes A_1 and A_2 respectively. The corresponding encrypted one-hot-encoding for the two-dimensional attribute $A_1 \times A_2$ is given by

$$D_{1\times2}[(i-1)\cdot s_{A_2} + j] = labMult(D_1[i], D_2[j]) \quad (3)$$
$$i \in [s_{A_1}], j \in [s_{A_2}] \quad (4)$$

For this particular case, only $D_{1\times2}[(v_1-1)\cdot s_{A_2} + v_2] = Enc(1)$ while all other indices will equate to $Enc(0)$. Note that when computing the one-hot-encoding for a t-dimensional attribute $t > 2$, for the actual implementation, instead of calling t iterative instances of CrossProduct() we use the *genLabMult*() operator of labeled homomorphic encryption to speed up the computation. 2) **Project**($\tilde{\mathbf{T}}, A^*$)- The implementation of the Project transformation is very straightforward, it simply drops off all but the attributes in A^* from $\tilde{\mathbf{T}}$ and returns the truncated table.

3) **Filter**($\tilde{\mathbf{T}}, \phi$)- As discussed in the preceding section, the predicate ϕ is expressed in a special form of conjunctions of range conditions given by eq **??**. Now for a range condition $A \in \{v_1, ...v_t\}$, assuming $\tilde{\mathbf{R}}_{\mathbf{A}}[i]$ is the corresponding one-hot-coding for the i^{th} record's value for attribute A, consider the following

$$\mathbf{c}^i_A = \bigoplus_{j=1}^{t} \tilde{\mathbf{R}}_A[i][v_1] \quad (5)$$

where $\tilde{\mathbf{R}}_A[i][v]$ is the v^{th} index of corresponding one-hot-coding. Clearly if the i^{th} record satisfies the condition $A \in \{v_1, ...v_t\}$, then exactly one of the values in $\{\tilde{\mathbf{R}}_A[i][v_j]\}, j \in \{1, ..., t\}$ will be a

[1]Theoretically no restriction, however practical constraints maybe present

cipher for 1. Thus $c^i_A = 1$ if record i satisfies the range condition and 0 otherwise. If the condition is instead an equality predicate $A == v$ then $\mathbf{c}^i_A = \tilde{\mathbf{R}}_A[i][v]$. Now considering ϕ is given by eq **??**, let us define

$$\mathbf{c}^i = genLabMult(\mathbf{c}^i_{A_1}, ..., \mathbf{c}^i_{A_r}) \tag{6}$$

$$A^* = \bigcup_{j=1}^{r} A_j \tag{7}$$

It is easy to see that c^i=1 iff record i satisfies ϕ. Let $\mathbf{B}'$ be the indicator vector before the execution of the current instance of the Filter transformation. The final step is to multiply the $\mathbf{c}^i$s with the corresponding indicator bits and obtain the updated indicator vector $\mathbf{B}$ as follows

$$\mathbf{B}[i] = labMult(\mathbf{c}^i, \mathbf{B}'[i]) \tag{8}$$

The above step zeros out some additional records which were found to be extraneous by some preceding filter conditions. Clearly $\mathbf{B}$ is the output of the Filter transformation.

Avoid Indicator Vector Multiplication

When the Filter transformation is applied for the very first time in a Cryptϵ program and the input predicate is conditioned on a single attribute $A \in \{v_1, ..., v_k\}$, then we can do the following optimization. Consider

$$\mathbf{b}[i] = \bigoplus_{j=1}^{k} \tilde{\mathbf{R}}_A[i][v_j], i \in [m] \tag{9}$$

where $\tilde{\mathbf{R}}_A[i]$ is the one-hot-coding for attribute A for the i^{th} record. Since this is the first instance of the Filter primitive, the current indicator vector $\mathbf{B}$ will be all 1-vector. Thus $\mathbf{b}$ is itself the updated indicator vector and we can avoid the unnecessary multiplication $labMult(\mathbf{b}[\mathbf{i}], \mathbf{B}[i])$.

4) **Count**($\hat{\mathbf{T}}$) - The Count primitive takes the associated bit vector $\mathbf{B}$ of its input table $\tilde{T}$ and simply adds up its entries to return

$$\mathbf{c} = \bigoplus_{i=1}^{m} \mathbf{B}[i] \tag{10}$$

5) GroupBy*($\tilde{\mathbf{T}}, A$) - The GroupBy* transformation makes use of three other transformations Project, Filter and Count and is implemented as follows

a) $\tilde{\mathbf{T}}_1$=Project($\tilde{\mathbf{T}}, A$)
b) $\mathbf{B}$ = current indicator bit vector
c) for $i = 1 : s_A$
   Intialize bit vector to $\mathbf{B}$
   $\phi_i = (A == v_{i,A})$
   $\tilde{\mathbf{T}}_2$ = Filter($\tilde{\mathbf{T}}_1, \phi_i$)
   $\mathbf{C}[i]$ = Count($\tilde{\mathbf{T}}_2$)
   end for
d) Output $\mathbf{C}$

6) CountDistinct($\mathbf{V}, \epsilon$) - The CountDistinct primitive is implemented as follows

a) Firstly the AS creates a mask vector drawn uniformly at random from $[m]^{s_A}$, i.e.,

$$M[i] \in_R [m], i \in [|V|]$$

b) AS masks the encrypted true count vector $\mathbf{V}$ as follows

$$\boldsymbol{\mathcal{V}}[i] = \mathbf{V}[i] \oplus labEnc_{pk}(M[i])$$

and sends it to the CSP

c) CSP decrypts the masked encrypted vector as

$$\mathcal{V}[i] = labDec_{sk}(\mathbf{V}[i]), i \in [|V|]$$

d) Next the CSP generates the following garbled circuit that
   i) takes the mask M as an input from the AS
   ii) takes a random number r as an input from the CSP
   iii) takes the decrypted masked vector $\mathcal{V}$ as an input from the CSP
   iv) removes the mask M from $\mathcal{V}$ as

$$V[i] = \mathcal{V}[i] - M[i], i \in [|V|]$$

   v) counts the number of non-zero entries of V as C
   vi) adds the laplace noises

$$C = C + r$$

   and returns C

e) The AS evaluates the above circuit and gets output C
f) The AS gets $labEnc_{pk}(r)$ from the CSP and generates $labEnc_{pk}(C)$ to compute

$$\mathrm{C} = labEnc_{pk}(C) - labEnc_{pk}(r)$$

8) Laplace($\mathbf{V}, \epsilon$)- Recall that both AS and CSP have to add Laplace noise to the output in Cryptϵ. Hence the Laplace primitive has two components. The first component is executed by the AS wherein,

(1) AS generates a noisy vector η such that $\eta \in [Lap(\frac{1}{\epsilon})]^{|V|}$
(2) encrypts η and adds it to the input vector as

$$\boldsymbol{\eta} = labEnc_{pk}(\eta)$$
$$\hat{\mathbf{V}}[i] = \mathbf{V}[i] \oplus \boldsymbol{\eta}[i], i \in [|V|]$$

This encrypted noisy vector $\hat{\mathbf{V}}$ is the input for the second phase of the Laplace primitive which is executed by the CSP as follows

(1) Decrypts $\hat{\mathbf{V}}$

$$\hat{V} = labDec_{sk}(\hat{\mathbf{V}})$$

(2) Generates a noisy vector η' such that $\eta' \in [Lap(\frac{1}{\epsilon})]^{|\hat{V}|}$
(3) Finally adds the noise η' to $\hat{V}$

$$\hat{\mathcal{V}}[i] = \hat{V}[i] + \eta'[i], i \in [|\hat{V}|]$$

(4) Returns $\hat{\mathcal{V}}$ to AS

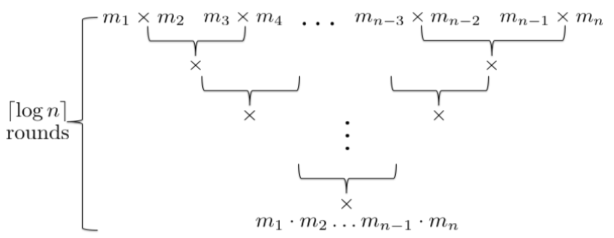
9)NoisyMax($\mathbf{V}, \epsilon, k$)- The input to the NoisyMax primitive is an encrypted vector $\mathbf{V}$ where each entry $V[i]$ is a count. The primitive is implemented via the following steps.

(1) First the AS adds noise to the input encrypted vector as follows

$$\eta \in [Lap(\frac{1}{\epsilon})]^{|V|}$$
$$\boldsymbol{\eta} = labEnc_{pk}(\eta)$$
$$\hat{\mathbf{V}}[i] = \mathbf{V}[i] + \boldsymbol{\eta}[i], i \in [|V|]$$

(2) Next the AS creates a mask vector M drawn uniformly at random from $[m]^{s_A}$, i.e.,

$$M[i] \in_R [m], i \in [|V|]$$



$m_1 \times m_2$ $m_3 \times m_4$ ... $m_{n-3} \times m_{n-2}$ $m_{n-1} \times m_n$
$\lceil \log n \rceil$ rounds
$m_1 \cdot m_2 \ldots m_{n-1} \cdot m_n$

Figure 4: $genLabMult()$ - **Batching of multiplicands for labHE**

(3) AS masks the encrypted noisy vector $\hat{\mathbf{V}}$ as follows

$$\mathcal{V}[i] = \hat{\mathbf{V}}[i] \oplus labEnc_{pk}(M[i]), i \in [|V|]$$

and sends it to the CSP

(4) CSP decrypts the masked encrypted noisy vector as

$$\mathcal{V}[i] = labDec_{sk}(\hat{\mathbf{V}}[i]), i \in [|V|]$$

(5) Next, the following garbled circuit is evaluated which
  i takes noisy masked vector $\mathcal{V}$ as an input from the CSP
  ii takes mask M as the input from AS
  iii removes the mask from $\mathcal{V}$ as

$$\hat{V}[i] = \mathcal{V}[i] - M[i], i \in [|V|]$$

  iv computes the top k element over $\hat{V}$ and returns $arg_{top\ k} \max \hat{V}[i]$)

C.2 DP Index Optimization Implementation

The DP index optimization can be implemented via a garbled circuit that

(1) takes the entire database $\tilde{\mathcal{D}}$ as an input and the attribute A as an input from the AS.
(2) takes the secret key sk as an input from the CSP
(3) Decrypts $\tilde{\mathcal{D}}$
(4) Sorts the decrypted database on A, i.e., the first ct_{A,v_1} rows are the ones with value v_1 for attribute A, the next ct_{A,v_2} are the records with value v_2 for attribute A and so on.
(5) re-encrypts the sorted database
(6) Divide the domain of A into k bins such that each bin contains s_A/k consecutive domain values.
(7) Construct a k lengthed vector $\hat{V}$ such that $\hat{V}[i] = \sum_j ct_{A,j} + \eta_i, i \in [k], j \in [\frac{s_A}{k}(i-1)+1, \frac{s_A}{k}i]$ where η_i is a random laplace noise drawn from the distribution $Lap(\frac{k}{\epsilon})$
(8) Return $\hat{V}$ and sorted $\tilde{\mathcal{D}}_{sort}$

D RELATED WORK

D.1 Differential Privacy

Introduced by Dwork et al. in [21], differential privacy has enjoyed immense attention from both academia and industry in the last decade. In this section, we will discuss some of the most recent directions in differential privacy in both the CDP and LDP models. An interesting line of work in the CDP model has been towards proposing "derived" mechanisms [14] or "revised algorithms" [10] whose privacy guarantee can be deduced from basic mechanisms (like exponential mechanism, Laplace mechanism etc) via the composition theorems and the post-processing immunity property [21]. Such mechanisms have the advantage of better utility as their design leverages on specific properties of the query and the data. One such technique is based on data partition and aggregation [3, 17, 34, 54, 55, 65, 66, 70] and is helpful in answering histogram queries. Another technique involves non-uniform data weighting where each data sample is weighed based on their query contribution. Research in this line of work include [18, 31, 32]. Yet another popular method is to utilize past/auxiliary information to improve the utility of the query answers. Examples are [22, 33, 46, 53, 64, 68, 69]

The local differential privacy model was first introduced by Kasiviswanathan et al. in[38]. Possibly the most simple LDP technique is the randomized response [63] protocol which was proposed by Warner in 1960s. In the recent times, constructing a frequency oracle to estimate the frequencies of any value in the domain, is perhaps the most fundamental LDP problem. The state-of-the-art locally differentially private histogram estimator solutions are [6, 24, 60]. However, when the domain size of the input values is extremely large, it might be computationally infeasible to construct the histogram over the entire domain. To tackle this, specialized algorithms to compute the most frequently occurring values, also known as the heavy hitters, have been proposed [7, 25, 61]. Another practical setting can be when the user's data is a set of items and the aggregator is interested in the k most frequent item sets. This problem is addressed in [56, 62]. In [16, 71] the authors propose efficient constructions of marginal tables in the local differential privacy setting. Due to their attractive trust model, LDP has also enjoyed significant industrial adoption. Google has integrated RAPPOR [24, 25] with Chrome. It is primarily tasked with collecting user statistics like default browser homepage, default search engine et al in order to monitor malicious hijacking of user settings. Apple [1] has also deployed differential privacy to collect of data like most frequent emojis, help with auto-completion of spellings etc. Samsung [50] proposed a similar system which enables the collection of both categorical (like screen resolution) as well as numerical data (like time of usage, battery volume), although it is not clear whether they went ahead with the actual deployment.

Recently it has been showed that augmenting the local differential privacy setting by an additional layer of anonymity can improve the privacy guarantees. The first work to study this was PROCHLO [9] implementation by Google. In [9] the authors propose a Encode, Shuffle, Analyze (ESA) architecture which relies on an explicit intermediate shuffler that processes the randomized LDP reports from users to ensure their anonymity. PROCHLO necessitates this intermediary to be trusted, this is implemented via trusted hardware enclaves (Intel's SGX). However, as showcased by recent attacks [59], it is notoriously difficult to design a truly secure hardware in practice. Motivated by PROCHLO, the authors in [23], present a tight upper-bound on the worst-case privacy loss. Formally, they show that any permutation invariant algorithm satisfying ϵ-local differential privacy will satisfy $O(\epsilon\sqrt{\frac{\log(\frac{1}{\delta})}{n}}, \delta)$ -central differential privacy. Cheu et al in [15] demonstrate privacy amplification by the same factor for 1-bit randomized response by using a mixnet architecture to provide the anonymity. Another important result from this work is that, they prove that the power of the mixnet model lies strictly between those of the central and local models.

A parallel line of work involves efficient use of cryptographic primitives for differentially private functionalities. In [4] Agarwal et al. propose an algorithm for computing histogram over encrypted data. Rastogi et al. [57] and Shi et al. [58] proposed algorithms that allow an untrusted aggregator to periodically estimate the sum of n users' values in a privacy preserving fashion. However both the schemes are irresilient to user failures. Chan et al. tackle this in [13] by constructing binary interval trees over the users.

D.2 Two-Server Model

The two-server model is a popular choice for privacy preserving machine learning techniques. In [52], [28], [26] and [27], the authors propose privacy preserving ridge regression systems with the help of a cryptographic service provider. While [27] uses a hybrid multi-party computation scheme with a secure inner product technique, [52] proposes a hybrid approach combining homomorphic encryptions and Yao's garbled circuits. Gascon et al. [26] extended the results in [52] to include vertically partitioned data and [28] solves the problem using just linear homomorphic encryption. Zhang et al in [48] also propose secure machine learning protocols using a privacy-preserving stochastic gradient descent method. Their main contribution includes developing efficient algorithms for secure arithmetic operations on shared decimal numbers and proposing alternatives to non-linear functions such as sigmoid and softmax tailored for MPC computations. In [51] and [41] the authors solve the problem of privacy-preserving matrix factorization. In both the papers, use a hybrid approach combining homomorphic encryptions and Yao's garbled circuits for their solutions.

D.3 Homomorphic Encryption

With improvements made in implementation efficiency and new constructions developed in the recent past, there has been a surge in practicable privacy preserving solutions employing homomorphic encryptions. A lot of the aforementioned two-server models employ homomorphic encryption [28, 41, 51, 52]. In [12, 30, 36] the authors enable neural networks to be applied to homomorphic-ally encrypted data. Linear homomorphic encryption is used in [29] to enable privacy-preserving machine learning for ensemble methods while uses fully-homomorphic encryption to approximate the coefficients of a logistic-regression model. [11] uses somewhat- homomorphic encryption scheme to compute the forecast prediction of consumer usage for smart grids.

E DISCUSSION

E.1 Joint Laplace Noise Generation

Recall that in Cryptϵ both the servers, AS and CSP has to add two separate instances of laplace noise before releasing the output. Thus the error incurred in Cryptϵ is quantitatively twice that of the traditional CDP model. However there is an alternative way of jointly computing a single instance of the laplace noise via a secure multi party computation protocol [49]. For this, the CSP generates a garbled circuit that

(1) takes a l-bit random string, S_1 as an input from the CSP
(2) takes another l-bit random string S_2 as an input from the AS
(3) performs $S = S_1 xor S_2$ and uses it to generate an instance of random noise, η drawn from the distribution $Lap(\frac{1}{\epsilon})$ following the fundamental law of transformation of probabilities
(4) encrypts η and returns $\boldsymbol{\eta} = labEnc_{pk}(\eta)$

Hence using this approach, we need to add just one instance of the Laplace noise and thus get back the exact same accuracy guarantees of the CDP model. However owing to the garbled circuit, this implementation is computationally heavier and hence we go for the two phase noise addition implementation for Crypt ϵ in this paper.

E.2 Separation from LDP model

As mentioned in section 1, the power of the LDP model is strictly lesser than that of the CDP model [15, 38]. However, by virtue of secure computation, we can potentially implement all the functionalities of the CDP model in our two-server model. Functional efficiency might be a point of contention in certain cases but nothing in Cryptϵ's architecture pose any restriction on its algorithmic expressibility. Recall that the power of the LDP model is equivalent to that of the "Statistical Query Model". In this section we showcase three different queries cases that are computable efficiently in Cryptϵ but infeasible in the standard LDP model. An important point to be noted here is that the power of the shuffler or mixnet model (which is obtained by augmenting LDP with anonymization via shuffling), as proposed in [9, 15, 23], lies strictly between that of traditional LDP and CDP. Thus the two-server model of Cryptϵ differs from this line of work in three major ways. Firstly, as discussed above, Cryptϵ results in no reduction in expressibility as compared to that of the CDP model. Secondly, the mixnet/shuffler model results in an approximate DP guarantee $(\epsilon\sqrt{\frac{\log\frac{1}{\delta}}{n}}, \delta)$ which incurs an expected error of $O(\epsilon\sqrt{\log\frac{1}{\delta}})$. In practice, δ has to be at least $\frac{1}{n}$ in order to get some meaningful privacy. In contrast Cryptϵ achieves the the same order of accuracy guarantees as that of the CDP model. Finally, the shuffler / mixnet model requires some additional trust assumptions as compared to that of the base LDP model. Google's implementation relies on a trusted intermediary shuffler which they implement via trusted hardware enclaves. However truly secure hardware enclaves are notoriously difficult to achieve in practice [59]. The mixnet model on the other hand requires a mix network or mixnet which is a protocol involving several computers that inputs a sequence of encrypted messages, and outputs a uniformly random permutation of those messages' plaintexts. Their trust assumption is that at least one of the servers needs to behave honestly. For Cryptϵ both the servers are completely untrusted under the constraint that they are non-colluding and follow the protocols semi-honestly.

DNF Queries

The class of DNF queries fall outside the scope of statistical query models [40]. Hence it is infeasible to answer counting queries based on a predicate with a disjunction in the LDP model. However, we can answer them in Cryptϵ as follows. Consider a DNF query

$$\phi = (A_{11} \wedge ... \wedge A_{1k}) \vee ... \vee (A_{t1} \wedge ...A_{tl}), t, k, l \in \mathcal{Z}_{\geq 0} \quad (11)$$

Let $Attribute(\phi)$ denote the set of all attributes in $\mathcal{A}$ that appear in the boolean condition ϕ. For e.g., if $\phi = \big((\mathcal{A}_1 == v_1) \wedge \mathcal{A}_2 == v_2) \vee \mathcal{A}_3 == v_3\big)$, then we have $Attribute(\phi) = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$.

(1) Firstly, the AS computes the attribute set $A^* = Attribute(\phi)$.
(2) Next the AS performs a Project transformation on inputs attribute set A^* and the entire encrypted database $\tilde{\mathcal{D}}$.
(3) Let $A^* = \{A^*_1, A^*_2, \ldots, A^*_t\}, t \leq k$. The AS constructs the encrypted one-hot-coding over the entire t-dimension 'attribute' $\mathcal{A}^* = \times_{i=1}^{t} A^*_i$ by $(t-1)$ iterative application of the cross product transformation.
(4) Note that the result of the preceding step is a $m \times 1$ table where the $i^{th}, i \in [m]$ record corresponds to the encrypted one-hot-coding over the entire t-dimension domain space of $\mathcal{A}^*$ of data owner DO$_i$. Now the AS simply applies the Filter transformation on this table with predicate ϕ' such that ϕ' is the equivalent of ϕ when expressed in terms of the new attribute $\mathcal{A}^*$.
(5) This is followed by performing the Count transformation and the Laplace transformation to obtain the final result.

Variable Selection Problem

The variable selection problem is an optimization problem described as follows. Given a set of counting queries, the problem finds the query with nearly largest value, i.e., computes an approximate argmax. In [15] Cheu et al. prove that the sample complexity of this problem in the "one-message" mixnet model (i.e., each user send only a single message into the shuffle) is exponentially larger than that of the CDP model. The variable-selection problem is actually equivalent to the exponential mechanism[21] in the CDP model. Moreover the exponential mechanism is simply a variant of the "Report Noisy-Max" algorithm with a different noise distribution [5]. Thus essentially, the NoisyMax primitive in Cryptϵ is capable of solving the variable-selection problem efficiently.

Number of distinct values

Consider the problem of computing the number of distinct values out of a set of m user data where the domain of the values is S and $m << |S|$. In the LDP model, for small sizes of S, one can construct a frequency oracle and compute the number of values with non-zero count with some careful thresholding [60]. However, if the size of S is huge then it becomes computationally infeasible to deploy the aforementioned mechanism. For example, if the values correspond to different URLs, since the total domain size is 2^{64}, computability limitations make this problem infeasible to be solved in the LDP setting. Although for our discussion in the paper we have considered the one-hot-coding as our preferred data encoding scheme, Cryptϵ architecturally can support any arbitrary encoding scheme. For instance, for URLs the data owners can instead use the domain name based encoding (i.e., subdomain.secondleveldomain.topleveldomain) for encrypting their data. Following this, an appropriate garbled circuit to count the number of distinct values from this encrypted dataset (which can be defined as a new Cryptϵ primitive) can answer the above query in the Cryptϵ setting.