Cryptϵ: Crypto-Assisted Differential Privacy on Untrusted Servers

Amrita Roy Chowdhury
University of Wisconsin-Madison
amrita@cs.wisc.edu

Chenghong Wang
Duke University
cw374@duke.edu

Xi He
University of Waterloo
xihe@uwaterloo.ca

Ashwin Machanavajjhala
Duke University
ashwin@cs.duke.edu

Somesh Jha
University of Wisconsin-Madison
jha@cs.wisc.edu

ABSTRACT

Differential privacy (DP) is currently the de-facto standard for achieving privacy in data analysis, which is typically implemented either in the "central" or "local" model. The local model has been more popular for commercial deployments as it does not require a trusted data collector. This increased privacy, however, comes at the cost of utility and algorithmic expressibility as compared to the central model.

In this work, we propose, Cryptϵ, a system and programming framework that (1) achieves the accuracy guarantees and algorithmic expressibility of the central model (2) without any trusted data collector like in the local model. Cryptϵ achieves the ~~"best of both worlds"~~ by employing two non-colluding untrusted servers that run DP programs on encrypted data from the data owners. In theory, straightforward implementations of DP programs using ~~off-the-shelf~~ secure multi-party computation tools can achieve the above goal. However, in practice, they are beset with many challenges like poor performance and tricky security proofs. To this end, Cryptϵ allows data analysts to author logical DP programs that are automatically translated to secure protocols that work on encrypted data. These protocols ensure that the untrusted servers learn nothing more than the noisy outputs, thereby guaranteeing DP (for computationally bounded adversaries) for all Cryptϵ programs. Cryptϵ supports a rich class of DP programs that can be expressed via a small set of transformation and measurement operators followed by arbitrary post-processing. Further, we propose performance optimizations leveraging the fact that the output is noisy. We demonstrate Cryptϵ's practical feasibility with extensive empirical evaluations on real world datasets.

ACM Reference Format:
Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Cryptϵ: Crypto-Assisted Differential Privacy on Untrusted Servers. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3318464.3380596

1 INTRODUCTION

Differential privacy (DP) is a rigorous privacy definition that is currently the gold standard for data privacy. It is typically implemented in one of two models – *centralized differential privacy* (CDP) and *local differential privacy* (LDP). In CDP, data from individuals are collected and stored *in the clear* in a *trusted* centralized data curator which then executes DP programs on the sensitive data and releases outputs to an untrusted data analyst. In LDP, there is no trusted data curator. Rather, each individual perturbs his/her own data using a (local) DP algorithm. The data analyst uses these noisy data to infer aggregate statistics of the datasets. In practice, CDP's assumption of a trusted server is ill-suited for many applications as it constitutes a single point of failure for data breaches, and saddles the trusted curator with legal and ethical obligations to uphold data privacy. Hence, recent commercial deployments of DP [43, 52] have preferred LDP over CDP. However, LDP's attractive privacy properties comes at a cost. Under the CDP model, the expected additive error for a aggregate count over a dataset of size n is at most $\Theta(1/\epsilon)$ to achieve ϵ-DP. In contrast, under the LDP model, at least $\Omega(\sqrt{n}/\epsilon)$ additive expected error must be incurred by any ϵ-DP program [17, 29, 37], owing to the randomness of each data owner. The LDP model in fact imposes additional penalties on the algorithmic expressibility; the power of LDP is equivalent to that of the statistical query model [67] and there exists an exponential separation between the accuracy and sample complexity of LDP and CDP algorithms [65].

SIGMOD '20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380596

In this paper, we strive to bridge the gap between LDP and CDP. We propose, Cryptϵ, a system and a programming framework for executing DP programs that:

- never stores or computes on sensitive data in the clear
- achieves the accuracy guarantees and algorithmic expressibility of the CDP model

Cryptϵ employs a pair of untrusted but non-colluding servers – Analytics Server (AS) and Cryptographic Service Provider (CSP). The AS executes DP programs (like the data curator in CDP) but on *encrypted* data records. The CSP initializes and manages the cryptographic primitives, and collaborates with the AS to generate the program outputs. Under the assumption that the AS and the CSP are semi-honest and do not collude (a common assumption in cryptographic systems [46, 47, 49, 68, 81, 84, 85]), Cryptϵ ensures ϵ-DP guarantee for its programs via two cryptographic primitives – linear homomorphic encryption (LHE) and garbled circuits. One caveat here is that due to the usage of cryptographic primitives, the DP guarantee obtained in Cryptϵ is that of computational differential privacy or SIM-CDP [80] (details in Section 7).

Cryptϵ provides a data analyst with a programming framework to author logical DP programs just like in CDP. Like in prior work [41, 78, 107], access to the sensitive data is restricted via a set of predefined transformations operators (inspired by relational algebra) and DP measurement operators (Laplace mechanism and Noisy-Max [39]). Thus, any program that can be expressed as a composition of the above operators automatically satisfies ϵ-DP (in the CDP model) giving the analyst a proof of privacy for free. Cryptϵ programs support constructs like looping, conditionals, and can arbitrarily post-process outputs of measurement operators. The main contributions of this work are:

- **New Approach**: We present the design and implementation of Cryptϵ, a novel system and programming framework for executing DP programs over encrypted data on two non-colluding and untrusted servers.
- **Algorithm Expressibility**: Cryptϵ supports a rich class of state-of -the-art DP programs expressed in terms of a small set of transformation and measurement operators. Thus, Cryptϵ achieves the accuracy guarantees of the CDP model without the need for a trusted data curator.
- **Ease Of Use:** Cryptϵ allows the data analyst to express the DP program logic using high-level operators. Cryptϵ automatically translates this to the underlying implementation specific secure protocols that work on encrypted data and provides a DP guarantee (in the CDP model) for free. Thus, the data analyst is relieved of all concerns regarding secure computation protocol implementation.
- **Performance Optimizations**: We propose optimizations that speed up computation on encrypted data by at least an order of magnitude. A novel contribution of this work is a DP indexing optimization that leverages the fact that noisy intermediate statistics about the data can be revealed.
- **Practical for Real World Usage**: For the same tasks, Cryptϵ programs achieve accuracy comparable to CDP and 50× more than LDP for a dataset of size $\approx 30K$. Cryptϵ runs within 3.6 hours for a large class of programs on a dataset with 1 million rows and 4 attributes.
- **Generalized Multiplication Using LHE**: Our implementation uses an efficient way for performing n-way multiplications using LHE which maybe of independent interest.

The full version of the paper is available in [6].

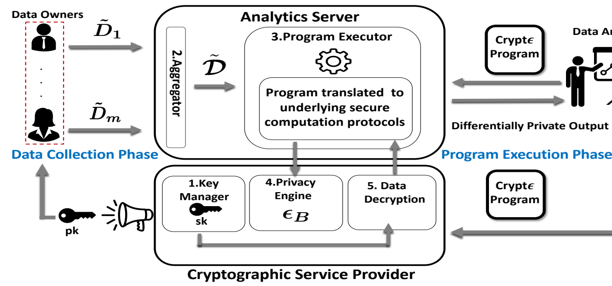


Figure 1: Cryptϵ System

2 CRYPTϵ OVERVIEW

2.1 System Architecture

Figure 1 shows Cryptϵ's system architecture. Cryptϵ has two servers: Analytics server (AS) and Cryptographic Service Provider (CSP). At the very outset, the CSP records the total privacy budget, ϵ^B (provided by the data owners), and generates the key pair, $\langle sk, pk \rangle$ (details in Section 3), for the encryption scheme. The data owners, $\mathrm{DO}_i, i \in [m]$ (m = number of data owners), encrypt their data records, D_i, in the appropriate format with the public key, pk, and send the encrypted records, $\tilde{\boldsymbol{D}}_i$, to the AS which aggregates them into a single encrypted database, $\tilde{\mathcal{D}}$. Next, the AS inputs logical programs from the data analyst and translates them to Cryptϵ's implementation specific secure protocols that work on $\tilde{\mathcal{D}}$. A Cryptϵ program typically consists of a sequence of transformation operators followed by a measurement operator. The AS can execute most of the transformations on its own. However, each measurement operator requires an interaction with the CSP for (a) decrypting the answer, and (b) checking that the total privacy budget, ϵ^B, is not exceeded. In this way, the AS and the CSP compute the output of a Cryptϵ program with the data owners being offline.

2.2 Cryptϵ Design Principles

Minimal Trust Assumptions: As mentioned above, the overarching goal of Cryptϵ is to mimic the CDP model but without a trusted server. A natural solution for dispensing with the trust assumption of the CDP model is using cryptographic primitives [10, 15, 18, 21, 30, 32, 38, 42, 93, 94]. Hence, to accommodate the use of cryptographic primitives, we assume a computationally bounded adversary in Cryptϵ. However, a generic m-party SMC would be computationally expensive. This necessitates a third-party entity that can capture the requisite secure computation functionality in a 2-party protocol instead. This role is fulfilled by the CSP in Cryptϵ. For this two-server model, we assume semi-honest behaviour and non-collusion. This is a very common assumption in the two-server model [46, 47, 49, 68, 81, 84, 85].

Programming Framework: Conceptually, the aforementioned goal of achieving the *best of both worlds* can be obtained by implementing the required DP program using off-the-self secure multi-party computation (SMC) tools like [2–5]. However, when it comes to real world usage, Cryptϵ outperforms such approaches due to the following reasons.

First, without the support of a programming framework like that of Cryptϵ, every DP program must be implemented from scratch. This requires the data analyst to be well versed in both DP and SMC techniques; he/she must know how to implement SMC protocols, estimate sensitivity of transformations and track privacy budget across programs. In contrast, Cryptϵ allows the data analyst to write the DP program using a high-level and expressive programming framework. Cryptϵ abstracts out all the low-level implementation details like the choice of input data format, translation of queries to that format, choice of SMC primitives and privacy budget monitoring from the analyst thereby reducing his/her burden of complex decision making. Thus, every Cryptϵ program is automatically translated to protocols corresponding to the underlying implementation.

Second, SMC protocols can be prohibitively costly in practice unless they are carefully tuned to the application. Cryptϵ supports optimized implementations for a small set of operators, which results in efficiency for all Cryptϵ programs.

Third, a DP program can be typically divided into segments that (1) transform the private data, (2) perform noisy measurements, and (3) post-process the noisy measurements without touching the private data. A naive implementation may implement all the steps using SMC protocols even though post-processing can be performed in the clear. Given a DP program written in a general purpose programming language (like Python), automatically figuring out what can be done in the clear can be subtle. In Cryptϵ programs, however, transformation and measurement are clearly delineated, as the data can be accessed only through a pre-specified set of operators. Thus, SMC protocols are only used for transformations and measurements, which improves performance.

For example, the AHP algorithm for histogram release [108] works as follows: first, a noisy histogram, $\hat{H}$, is released using budget ϵ_1. This is followed by post-processing steps of thresholding, sorting and clustering resulting in $\bar{H}$. Then a final histogram, $\tilde{H}$, is computed with privacy budget $\epsilon - \epsilon_1$. An implementation of the entire algorithm in a single SMC protocol using the EMP toolkit [2] takes 810s for a dataset of size $\approx 30K$ and histogram size 100. In contrast, Cryptϵ uses SMC protocols only for the first and third steps. Cryptϵ automatically detects that the second post-processing step can be performed in the clear. A Cryptϵ program for this runs in 238s (3.4× less time than that of the EMP implementation) for the same dataset and histogram sizes.

Last, the security (privacy) proofs for just stand-alone cryptographic and DP mechanisms can be notoriously tricky [20, 76]. Combining the two thus exacerbates the technical complexity, making the design vulnerable to faulty proofs [60]. For example, given any arbitrary DP program written under the CDP model, the distinction between intermediate results that can be released and the ones which have to be kept private is often ambiguous. An instance of this is observed in the Noisy-Max algorithm, where the array of intermediate noisy counts is private. However, these intermediate noisy counts correspond to valid query responses. Thus, an incautious analyst, in a bid to improve performance, might reuse a previously released noisy count query output for a subsequent execution of the Noisy-Max algorithm leading to privacy leakage. In contrast, Cryptϵ is designed to reveal nothing other than the outputs of the DP programs to the untrusted servers; every Cryptϵ program comes with an automatic proof of security (privacy). Referring back to the aforementioned example, in Cryptϵ, the Noisy-Max algorithm is implemented as a secure measurement operator thereby preventing any accidental privacy leakage. The advantages of a programming framework is further validated by the popularity of systems like PINQ [78], Featherweight PINQ [41], Ektelo [107] - frameworks for the CDP setting.

Data Owners are Offline: Recall, Cryptϵ's goal is to mimic the CDP model with untrusted servers. Hence, it is designed so that the data owners are offline after submitting their encrypted records to the AS.

Low burden on CSP: Cryptϵ views the AS as an extension of the analyst; the AS has a vested interest in obtaining the result of the programs. Thus, we require the AS to perform the majority of the work for any program; interactions with the CSP should be minimal and related to data decryption. Keeping this in mind, the AS performs most of the data transformations by itself (Table 3). Specifically, for every Cryptϵ program, the AS processes the whole database and transforms it into concise representations (an encrypted scalar or a short vector) which is then decrypted by the CSP. An example real world setting can be when Google and Symantec assumes the role of the AS and the CSP respectively.

Separation of logical programming framework and underlying physical implementation: The programming framework is independent of the underlying implementation. This allows certain flexibility in the choice for implementation. For example, we use one-hot-encoding as the input data format (Section 2). However, any other encoding scheme like range based encoding can be used instead. Another example is that in this paper, we use ϵ-DP (pure DP) for our privacy analysis. However, other DP notions like (ϵ, δ)-DP, Rènyi DP [79] can also be used instead. Similarly, it is straightforward to replace LHE with the optimized HE scheme in [22] or garbled circuits with the ABY framework [36].

Yet another alternative implementation for Cryptϵ could be where the private database is equally shared between the two servers and they engage in a secret share-based SMC protocol for executing the DP programs. This would require both the servers to do almost equal amount of work for each program. Such an implementation would be justified only if both the servers are equally invested in learning the DP statistics and is ill-suited for our context. A real world analogy for this can be if Google and Baidu decide to compute some statistics on their combined user bases.

3 BACKGROUND

3.1 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 neighboring datasets D and D' such that $D = D' - t$ or $D' = D - t$, we have*

$$\forall S \subset Range(\mathcal{A}), Pr\big[\mathcal{A}(D) \in S\big] \leq e^{\epsilon} Pr\big[\mathcal{A}(D') \in S\big] \quad (1)$$

The above definition is sometimes called *unbounded DP*. A variant is *bounded-DP* where neighboring datasets D and D' have the same number of rows and differ in one row. Any ϵ-DP algorithm also satisfies 2ϵ-bounded DP [73].

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

Theorem 2. *(Post-Processing) Let $\mathcal{A} : D \mapsto R$ be a randomized algorithm that is ϵ-DP. Let $f : R \mapsto R'$ be an arbitrary randomized mapping. Then $f \circ \mathcal{A} : D \mapsto R'$ is ϵ- DP.*

3.2 Cryptographic Primitives

Linearly Homomorphic Encryption (LHE): If $(\mathcal{M}, +)$ is a finite group, an LHE scheme for messages in $\mathcal{M}$ is:

- **Key Generation** (*Gen*): This algorithm takes the security parameter κ as input and outputs a pair of secret and public keys, $\langle s_k, p_k \rangle \leftarrow Gen(\kappa)$.
- **Encryption** (*Enc*): This is a randomized algorithm that encrypts a message, $m \in \mathcal{M}$, using the public key, p_k, to generate the ciphertext, $\mathbf{c} \leftarrow Enc_{pk}(m)$.
- **Decryption** (*Dec*): This uses the secret key, s_k, to recover the plaintext, m, from the ciphertext, $\mathbf{c}$, deterministically.

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

Operator $\oplus$: Let $c_1 \leftarrow Enc_{pk}(m1), \ldots, c_a \leftarrow Enc_{pk}(m_a)$ and $a \in \mathcal{Z}_{>0}$. 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): Any LHE scheme can be extended to a labHE scheme [13] with the help of a pseudo-random function. In addition to the operations supported by an LHE scheme, labHE supports multiplication of two labHE ciphers (details in full paper [6]).

Garbled Circuit: Garbled circuit [74, 104] is a generic method for secure computation. Two data owners with respective private inputs x_1 and x_2 run the protocol such that, no data owner learns more than $f(x_1, x_2)$ for a function f. One of the data owners, called generator, builds a "garbled" version of a circuit for f and sends it to the other data owner, called evaluator, alongside the garbled input values for x_1. The evaluator, then, obtains the garbled input for x_2 from the generator via oblivious transfer and computes $f(x_1, x_2)$.

4 CRYPTϵ SYSTEM DESCRIPTION

In this section, we describe Cryptϵ's workflow (Section 4.1), modules (Section 4.2), and trust assumptions (Section 4.3).

4.1 Cryptϵ Workflow

Cryptϵ operates in three phases:

(1) **Setup Phase**: At the outset, data owners initialize the CSP with a privacy budget, ϵ^B, which is stored in its *Privacy Engine* module. Next, the CSP's *Key Manager* module generates key pair (sk, pk) for labHE, publishes pk and stores sk.

(2) **Data Collection Phase**: In the next phase, each data owner encodes and encrypts his/her record using the *Data Encoder* and *Data Encryption* modules and sends the encrypted data records to the AS. The data owners are relieved of all other duties and can go completely offline. 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 provided by the data analyst. Cryptϵ programs (details in Sections 5 and 6) access the sensitive data via a restricted set of transformation operators, that filter, count or group the data, and measurement operators, which are DP operations to release noisy answers. Measurement operators need interactions with the CSP as they require (1)

decryption of the answer, and (2) a check that the privacy budget is not exceeded. These functionalities are achieved by CSP's *Data Decryption* and *Privacy Engine* modules.

The *Setup* and *Data Collection* phases occur just once at the very beginning, every subsequent program is handled via the corresponding *Program Execution* phase.

4.2 Cryptϵ Modules

Cryptographic Service Provider (CSP)

(1) **Key Manager**: The *Key Manager* module initializes the labHE scheme for Cryptϵ by generating its key pair, $\langle sk, pk\rangle$. It stores the secret key, sk, with itself and releases the public key, pk. The CSP has exclusive access to the secret key, sk, and is the only entity capable of decryption in Cryptϵ.

(2) **Privacy Engine**: Cryptϵ starts off with a total privacy budget of ϵ^B chosen by the data owners. The choice of value for ϵ^B should be guided by social prerogatives [8, 62, 70] and is currently outside the scope of Cryptϵ. For executing any program, the AS has to interact with the CSP at least once (for decrypting the noisy answer), thereby allowing the CSP to monitor the AS's actions in terms of privacy budget expenditure. The *Privacy Engine* module gets the program, P, and its allocated privacy budget, ϵ, from the data analyst, and maintains a public ledger that records the privacy budget spent in executing each such program. Once the privacy cost incurred reaches ϵ^B, the CSP refuses to decrypt any further answers. This ensures that the total privacy budget is never exceeded. The ledger is completely public allowing any data owner to verify it.

(3) **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.

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-encoding format. The one-hot-encoding is a way of representation for categorical attributes and is illustrated by the following example. If the database schema is given by $\langle Age, Gender\rangle$, then the corresponding one-hot-encoding representation for a data owner, $\mathrm{DO}_i, i \in [m]$, 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 labHE which is announced by the CSP. Each data owner, $\mathrm{DO}_i, i \in [m]$, performs an element-wise encryption of his/her per attribute one-hot-encodings using pk and sends the encrypted record, $\tilde{\mathbf{D}}_\mathbf{i}$, to the AS via a secure channel. This is the only interaction that a data owner ever participates in and goes offline after this.

Analytics Server (AS)

(1) **Aggregator**: The *Aggregator* collects the encrypted records, $\tilde{\mathbf{D}}_i$, from each of the data owners, DO_i, and collates them into a single encrypted database, $\tilde{\mathcal{D}}$.

(2) **Program Executor**: This module inputs a logical Cryptϵ program, P, and privacy parameter, ϵ, from the data analyst, translates P to the implementation specific secure protocol and computes the noisy output with the CSP's help.

4.3 Trust Model

There are three differences in Cryptϵ from the LDP setting:

(1) **Semi-honest Model**: We assume that the AS and the CSP are *semi-honest*, i.e., they follow the protocol honestly, but their contents and computations can be observed by an adversary. Additionally, each data owner has a private channel with the AS. For real world scenarios, the semi-honest behaviour can be imposed via legal bindings. Specifically, both the AS and the CSP can swear to their semi-honest behavior in legal affidavits; there would be loss of face in public and legal implications in case of breach of conduct.

(2) **Non-Collusion**: We assume that the AS and the CSP are *non-colluding*, i.e., they avoid revealing information [64] to each other beyond what is allowed by the protocol definition. This restriction can be imposed via strict legal bindings as well. Additionally, in our setting the CSP is a third-party entity with no vested interested in learning the program outputs. Hence, the CSP has little incentive to collude with the AS. Physical enforcement of the non-collusion condition can be done by implementing the CSP inside a trusted execution environment (TEE) or via techniques which involve using a trusted mediator who monitors the communications between the servers [12].

(3) **Computational Boundedness**: The adversary is *computationally bounded*. Hence, the DP guarantee obtained is that of computational differential privacy or SIM-CDP [80]. There is a separation between the algorithmic power of computational DP and information-theoretic DP in the multi-party setting [80]. Hence, this assumption is inevitable in Cryptϵ.

5 CRYPTϵ OPERATORS

Let us consider an encrypted instance of a database, $\tilde{\mathcal{D}}$, with schema $\langle A_1, \ldots, A_l\rangle$. In this section, we define the Cryptϵ operators (summarized in Table 1) and illustrate how to write logical Cryptϵ programs for DP algorithms on $\tilde{\mathcal{D}}$. The design of Cryptϵ operators are inspired by previous work [78, 107].

5.1 Transformation operators

Transformation operators input encrypted data and output a transformed encrypted data. These operators thus work completely on the encrypted data without expending any privacy budget. Three types of data are considered in this context: (1) an encrypted table, $\tilde{\boldsymbol{T}}$, of x rows and y columns/attributes

Table 1: Cryptϵ Operators

Types	Name	Notation	Input	Output	Functionality
Transformation	CrossProduct	$\times_{A_i,A_j \rightarrow A'}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\tilde{\boldsymbol{T}}'$	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
	GroupByCountEncoded	$\tilde{\gamma}_A^{count}(\cdot)$	$\tilde{\boldsymbol{T}}$	$\tilde{\boldsymbol{V}}$	Returns encrypted histogram of A in one-hot-encoding
	CountDistinct	$countD(\cdot)$	$\boldsymbol{V}$	$\boldsymbol{c}$	Counts the number of non-zero values in $\boldsymbol{V}$
Measurement	Laplace	$Lap_{\epsilon,\Delta}(\cdot)$	$\boldsymbol{V} \backslash \boldsymbol{c}$	$\hat{V}$	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

where each attribute value is represented by its encrypted one-hot-encoding; (2) an encrypted vector, $\boldsymbol{V}$; and (3) an encrypted scalar, $\boldsymbol{c}$. In addition, every encrypted table, $\tilde{\boldsymbol{T}}$, of x rows has an encrypted bit vector, $\boldsymbol{B}$, of size x to indicate whether the rows are relevant to the program at hand. The i-th row in $\tilde{\boldsymbol{T}}$ will be used for answering the current program only if the i-th bit value of $\boldsymbol{B}$ is 1. The input to the first transformation operator in Cryptϵ program is $\tilde{\mathcal{D}}$ with all bits of $\boldsymbol{B}$ set to 1. For brevity, we use just $\tilde{\boldsymbol{T}}$ to represent both the encrypted table, $\tilde{\boldsymbol{T}}$, and $\boldsymbol{B}$. The transformation operators are:

(1) **CrossProduct** $\times_{(A_i,A_j)\rightarrow A'}(\tilde{\boldsymbol{T}})$: This operator 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 $\tilde{\boldsymbol{T}}$. Thus, the construction of the one-hot-encoding of the entire y-dimensional domain can be computed by repeated application of this operator.

(2) **Project** $\pi_{\bar{A}}(\tilde{\boldsymbol{T}})$: This operator 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 from the output table $\tilde{\boldsymbol{T}}'$.

(3) **Filter** $\sigma_{\phi}(\tilde{\boldsymbol{T}})$: This operator specifies a filtering condition that is represented by a Boolean predicate, ϕ, and 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 $r \in \tilde{\boldsymbol{T}}$, $\phi(r) = \bigwedge_{A_i \in \bar{A}} (r.A_i \in V_{A_i})$, 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) that A_i can take. For example, $Age \in [30, 40] \wedge Gender = M$ can be a filtering condition. The Filter operator affects only the associated encrypted bit vector of $\tilde{\boldsymbol{T}}$ and keeps the actual table untouched. If any row, $r \in \tilde{\boldsymbol{T}}$, does not satisfy the filtering condition, ϕ, the corresponding 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 operator 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 operator outputs an encrypted scalar, $\boldsymbol{c}$.

(5) **GroupByCount** $\gamma_A^{count}(\tilde{\mathbf{T}})$: The GroupByCount operator partitions the input table, $\tilde{\mathbf{T}}$, into groups of rows having the same value for an attribute, A. The output of this transformation is an encrypted vector, $\mathbf{V}$, that counts the number of unfiltered rows for each value of A. This operator serves as a preceding transformation for other Cryptϵ operators specifically, NoisyMax, CountDistinct and Laplace.

(6) **GroupByCountEncoded** $\tilde{\gamma}_A^{count}(\tilde{\mathbf{T}})$: This operator is similar to GroupByCount. The only difference between the two is that GroupByCountEncoded outputs a new table that has two columns – the first column corresponds to A and the second column corresponds to the number of rows for every value of A (in one-hot-encoding). This operator is useful for expressing computations of the form "count the number of age values having at least 200 records" (see P7 in Table 2).

(7) **CountDistinct** $countD(\mathbf{V})$: This operator is always preceded by GroupByCount. Hence the input vector, $\mathbf{V}$, is an encrypted histogram for attribute, A, and this operator returns the number of distinct values of A that appear in $\tilde{\mathcal{D}}$ by counting the non-zero entries of V.

5.2 Measurement operators

The measurement operators 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 two operators correspond to two classic DP mechanisms – Laplace mechanism and NoisyMax [39]. Both mechanisms add Laplace noise, η, scaled according to the transformations applied to $\tilde{\mathcal{D}}$.

Let the sequence of transformations applied on $\tilde{\mathcal{D}}$ to get V be $\bar{\mathcal{T}}(D) = \mathcal{T}_l(\cdots \mathcal{T}_2((\mathcal{T}_1(D))))$. The *sensitivity* of a sequence of transformations is defined as the maximum change to the output of this sequence of transformations [78] 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 in a single row. The sensitivity of $\bar{\mathcal{T}}$ can be upper bounded by the product of the stability [78] of these transformation operators, i.e., $\Delta_{\bar{\mathcal{T}}=(\mathcal{T}_l,\ldots,\mathcal{T}_1)} = \prod_{i=1}^{l} \Delta\mathcal{T}_i$. The transformations in Table 1 have a stability of 1, except for GroupByCount and GroupByCountEncoded which are 2-stable. Given ϵ and $\Delta_{\bar{\mathcal{T}}}$, we define the measurement operators:

(1) **Laplace** $Lap_{\epsilon,\Delta}(\mathbf{V}/\boldsymbol{c})$: This operator implements the classic Laplace mechanism [39]. Given an encrypted vector, $\boldsymbol{V}$, or an encrypted scalar, $\boldsymbol{c}$, a privacy parameter ϵ and sensitivity Δ of the preceding transformations, the operator adds noise drawn from $Lap(\frac{2\Delta}{\epsilon})$ to $\boldsymbol{V}$ or $\boldsymbol{c}$ and outputs the noisy answer.

(2) **NoisyMax** $NoisyMax^{k}_{\epsilon,\Delta}(\mathbf{V})$: Noisy-Max is a differentially private selection mechanism [39, 48] to determine the top k highest valued queries. This operator takes in an encrypted vector $\boldsymbol{V}$ and adds independent Laplace noise from $Lap(\frac{2k\Delta}{\epsilon})$ to each count. The indices for the top k noisy values, $\hat{\mathcal{P}}$, are reported as the desired answer.

5.3 Program Examples

A Cryptϵ program is a sequence of transformation operators followed by a measurement operator and arbitrary post-processing. Consider a database schema $\langle Age, Gender, NativeCountry, Race\rangle$. We show 7 Crypt$\epsilon$ program examples in Table 2 over this database.

We will use P1 in Table 2 to illustrate how a Cryptϵ program can be written and analyzed. Program P1 aims to compute the cumulative distribution function (c.d.f.) of attribute *Age* with domain $[1, 100]$. The first step is to compute 100 range queries, where the i-th query computes the the number of records in $\tilde{\mathcal{D}}$ having $Age \in [1, i]$ with privacy parameter ϵ_i. The sequence of transformation operators for each range query is $count(\sigma_{Age\in[1,i]}(\pi_{Age}(\tilde{\mathcal{D}})))$. All these three operators are 1-stable and hence, the sensitivity of the resulting range query is upper bounded by the product of these stability values, 1 [78]. Thus, the subsequent measurement operator Laplace for the i-th range query takes in privacy budget ϵ_i and sensitivity $\Delta = 1$, and outputs a noisy plaintext count, $\hat{c}_i$. At this stage the program is $\sum_{i=1}^{100} \epsilon_i/2$-DP by Theorem 1[39] (recall we add noise from $Lap(2 \cdot \Delta/\epsilon_i)$ in Section 5.2). After looping over the 100 ranges, P1 obtains a noisy plaintext output $\hat{V} = [\hat{c}_1, \ldots, \hat{c}_{100}]$ and applies a post-processing step, denoted by $post_{c.d.f}(\hat{V})$. This operator inputs a noisy histogram, $\hat{V}$, for attribute A and computes its c.d.f $\hat{V}' = [\hat{c}'_1, \ldots, \hat{c}'_{100}]$ via isotonic regression [58] $\min_{\hat{V}'} \|\hat{V}' - \hat{V}\|_2 \; s.t. \; 0 \le \hat{c}'_1 \le \cdots \le \hat{c}'_{100} \le |\tilde{\mathcal{D}}|$. Hence, by Theorem 2, P1 is $\epsilon/2$-DP, where $\epsilon = \sum_{i=1}^{100} \epsilon_i$. However, since Crypt$\epsilon$ also reveals the total dataset size, the total privacy guarantee is ϵ-bounded DP (see Section 7 for details).

6 IMPLEMENTATION

In this section, we describe the implementation of Cryptϵ. First, we discuss our proposed technique for extending the multiplication operation of labHE to support $n > 2$ multiplicands which will be used for the CrossProduct operator. Then, we describe the implementations of Cryptϵ operators.

6.1 General labHE n-way Multiplication

The labHE scheme is an extension of a LHE scheme where every ciphertext is now associated with a "label" [13]. This extension enables labHE to support multiplication of two labHE ciphertexts via the $labMult()$ operator (without involving the CSP). However, it cannot support multiplication of more than two ciphertexts because the"multiplication" ciphertext $\mathbf{e} = labMult(\mathbf{c}_1, \mathbf{c}_2)$, ($\mathbf{c}_1$ and $\mathbf{c}_2$ are labHE ciphertexts) does not have a corresponding label, i.e., it is not in the correct labHE ciphertext format. Hence, we propose an algorithm *genLabMult* to generate a label for every intermediary product of two multiplicands to enable generic n-way multiplication (details are in thethe full paper [6]) .

6.2 Operator Implementation

We now summarize how Cryptϵ operators are translated to protocols that the AS and CSP can run on encrypted data.

Project $\pi_{\bar{A}}(\tilde{\boldsymbol{T}})$: The implementation of this operator simply drops off all but the attributes in $\bar{A}$ from the input table, $\tilde{\boldsymbol{T}}$, and returns the truncated table, $\tilde{\boldsymbol{T}}'$.

Filter $\sigma_\phi(\tilde{\boldsymbol{T}})$: Let ϕ be a predicate of the form $r.A_j \in V_{A_j}$. Row i satisfies the filter if one of the bits corresponding to positions in V_{A_j} is 1. Thus, the bit corresponding to row i is set as: $\boldsymbol{B}[i] = labMult(\boldsymbol{B}[i], \bigoplus_{l\in V_{A_j}} \tilde{\mathbf{v}}_j[l])$. The multi-attribute implementation is detailed in the full paper [6].

CrossProduct $\times_{A_i,A_j\rightarrow A'}(\tilde{\boldsymbol{T}})$: The crossproduct between two attributes are computed using $genLabMult()$ described above.

Count $count(\tilde{\boldsymbol{T}})$: This operator simply adds up the bits in $\boldsymbol{B}$ corresponding to input table $\tilde{\boldsymbol{T}}$, i.e., $\bigoplus_i \boldsymbol{B}[i]$.

GroupByCount $\gamma^{count}_{A}(\tilde{\mathrm{T}})$: The implementations for Project, Filter and Count are reused here. First, Cryptϵ projects the input table $\tilde{\boldsymbol{T}}$ on attribute A, i.e. $\tilde{\boldsymbol{T}}_1 = \pi_A(\tilde{\boldsymbol{T}})$. Then, Crypt$\epsilon$ loops each possible value of A. For each value v, Cryptϵ initializes a temporary $\boldsymbol{B}_v = \boldsymbol{B}$ and filters $\tilde{\boldsymbol{T}}'$ on $A = v$ to get an updated $\boldsymbol{B}'_v$. Finally, Cryptϵ outputs the number of 1s in $\boldsymbol{B}'_v$.

GroupByCountEncoded $\tilde{\gamma}^{count}_{A}(\tilde{\mathrm{T}})$: For this operator, the AS first uses GroupByCount to generate the encrypted histogram, $\boldsymbol{V}$, for attribute A. Since each entry of $\mathbf{V}$ is a count of rows, its value ranges from $\{0, \ldots, |\tilde{\boldsymbol{T}}|\}$. The AS, then, masks $\boldsymbol{V}$ 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

Table 2: Examples of Cryptϵ Program

Cryptϵ Program	**Description**
P1: $\forall i \in [1, 100], \hat{c}_i \leftarrow Lap_{\epsilon_i,1}(count(\sigma_{Age\in(0,i]}(\pi_{Age}(\tilde{\mathcal{D}})))); post_{c.d.f}([\hat{c}_1, \ldots, \hat{c}_{100}])$	Outputs the c.d.f of *Age* with domain [1, 100].
P2: $\hat{P} \leftarrow NoisyMax^5_{\epsilon,1}(\gamma^{count}_{Age}(\tilde{\mathcal{D}}))$	Outputs the 5 most frequent age values.
P3: $\hat{V} \leftarrow Lap_{\epsilon,2}(\gamma^{count}_{Race\times Gender}(\pi_{Race\times Gender}(\times_{Race,Gender\rightarrow Race\times Gender}(\tilde{\mathcal{D}}))))$	Outputs the marginal over the attributes *Race* and *Gender*.
P4: $\hat{V} \leftarrow Lap_{\epsilon,2}(\gamma^{count}_{Age\times Gender}(\sigma_{NativeCountry=Mexico}(\pi_{Age\times Gender,NativeCountry}(\times_{Age,Gender\rightarrow Age\times Gender}(\tilde{\mathcal{D}})))))$	Outputs the marginal over *Age* and *Gender* for Mexican employees.
P5: $\hat{c} \leftarrow Lap_{\epsilon,1}(count(\sigma_{Age=30\wedge Gender=Male\wedge NativeCountry=Mexico}(\pi_{Age,Gender,NativeCountry}(\tilde{\mathcal{D}}))))$	Counts the number of male employees of Mexico having age 30.
P6: $\hat{c} \leftarrow Lap_{\epsilon,2}(countD(\gamma^{count}_{Age}(\sigma_{Gender=Male}(\pi_{Age,Gender}(\tilde{\mathcal{D}})))))$	Counts the number of distinct age values for the male employees.
P7: $\hat{c} \leftarrow Lap_{\epsilon,2}(count(\sigma_{Count\in[200,m]}(\tilde{\gamma}^{count}_{Age}(\pi_{Age}(\tilde{\mathcal{D}})))))$	Counts the number of age values having at least 200 records.

histogram $\tilde{\boldsymbol{\mathcal{V}}}$ and returns it to the AS. The AS can simply rotate $\tilde{\boldsymbol{\mathcal{V}}}[i], i \in [|V|]$ by its respective mask value $M[i]$ and get back the true encrypted histogram in one-hot-coding $\tilde{\boldsymbol{V}}$. The details are presented in the full paper [6].

CountDistinct $countD(\boldsymbol{V})$: This operator is implemented by a garbled circuit (details are in the full paper [6]).

Laplace $Lap_{\epsilon,\Delta}(\boldsymbol{V}\backslash\boldsymbol{c})$: The Laplace operator has two phases (since both the AS and the CSP adds Laplace noise). In the first phase, the AS adds an instance of encrypted Laplace noise, $\eta_1 \sim Lap(\frac{2\Delta}{\epsilon})$, to the encrypted input to generate $\hat{\mathbf{c}}$. In the second phase, the CSP first checks whether $\sum_{i=1}^{t} \epsilon_i + \epsilon \leq \epsilon^B$ where ϵ_i represents the privacy budget used for a previously executed program, P_i (presuming a total of $t \in \mathbb{N}$ programs have been executed hitherto the details of which are logged into the CSP's public ledger). Only in the event the above check is satisfied, the CSP proceeds to decrypt $\hat{\mathbf{c}}$, and records ϵ and the current program details (description, sensitivity) in the public ledger. Next, the CSP adds a second instance of the Laplace noise, $\eta_2 \sim Lap(\frac{2\Delta}{\epsilon})$, to generate the final noisy output, $\hat{c}$, in the clear. The Laplace operator with an encrypted scalar, $\boldsymbol{V}$, as the input is implemented similarly.

NoisyMax $NoisyMax^k_{\epsilon,\Delta}(\boldsymbol{V})$: This operator is implemented via a two-party computation between the AS and the CSP using garbled circuits (details are in the full paper [6]).

Note: Cryptϵ programs are grouped into three classes based on the number and type of interaction between the AS and the CSP. For example, P1, P2 and P3 (Table 2) have just one interaction with the CSP for decrypting the noisy output. P4 and P5, on the other hand, require additional interactions for the n-way multiplication of ciphers in the CrossProduct operator. Finally, P6 and P7 require intermediate intercations due to operators CountDistinct and GroupByCountEncoded respectively. The details are presented in the full paper [6] .

7 CRYPTϵ SECURITY SKETCH

In this section, we provide a sketch of the security proof in the semi-honest model using the well established simulation argument [87]. Cryptϵ takes as input a DP program, P, and a privacy parameter, ϵ, and translates P into a protocol, Π, which in turn is executed by the AS and the CSP. In addition to revealing the output of the program P, Π also reveals the number of records in the dataset, $\mathcal{D}$. Let $P^{CDP}(\mathcal{D}, \epsilon/2)$ denote the random variable corresponding to the output of running P in the CDP model under $\epsilon/2$-DP (Definition 1). We make the following claims:

- The views and outputs of the AS and CSP are computationally indistinguishable from that of simulators with access to only $P^{CDP}(\mathcal{D}, \epsilon/2)$ and the total dataset size $|\mathcal{D}|$.
- For every P that satisfies $\epsilon/2$-DP (Definition 1), revealing its output (distributed identical to $P^{CDP}(\mathcal{D}, \epsilon/2)$) as well as $|\mathcal{D}|$ satisfies ϵ-bounded DP, where neighboring databases have the same size but differ in one row.
- Thus, the overall protocol satisfies computational differential privacy under the SIM-CDP model.

Now, let $P^{CDP}_B(\mathcal{D}, \epsilon)$ denote the random variable corresponding to the output of running P in the CDP model under ϵ-bounded DP such that $P^{CDP}_B(\mathcal{D}, \epsilon) \equiv (P^{CDP}(\mathcal{D}, \epsilon/2), |\mathcal{D}|)$. We state the main theorems here and refer the reader to the the full paper [6]for formal proofs.

Theorem 3. Let protocol Π correspond to the execution of program P in Cryptϵ. The views and outputs of the AS and the CSP are denoted as $View^{\Pi}_1(P, \mathcal{D}, \epsilon), Output^{\Pi}_1(P, \mathcal{D}, \epsilon)$ and $View^{\Pi}_2(P, \mathcal{D}, \epsilon), Output^{\Pi}_2(P, \mathcal{D}, \epsilon)$ respectively. There exists Probabilistic Polynomial Time (PPT) simulators, Sim_1 and Sim_2, such that:

- $Sim_1(P^{CDP}_B(\mathcal{D}, \epsilon))$ is computationally indistinguishable ($\equiv_c$) from $(View^{\Pi}_1(P, \mathcal{D}, \epsilon), Output^{\Pi}(P, \mathcal{D}, \epsilon))$, and
- $Sim_2(P^{CDP}_B(\mathcal{D}, \epsilon))$ is $\equiv_c$ to $(View^{\Pi}_2(P, \mathcal{D}, \epsilon), Output^{\Pi}(P, \mathcal{D}, \epsilon))$.

$Output^{\Pi}(P, \mathcal{D}, \epsilon))$ is the combined output of the two parties[1].

The main ingredient for the proof is the composition theorem [87], which informally states: suppose a protocol, Π^g_f, implements functionality f and uses function g as an oracle (uses only input-output behavior of g). Assume that protocol Π_g implements g and calls to g in Π^g_f are replaced by instances of Π_g (referred to as the composite protocol). If Π_f and Π_g are correct (satisfy the above simulator definition), then the composite protocol is correct. Thus, the proof can be done in a modular fashion as long as the underlying operators are used in a blackbox manner (only the input-output behavior are used and none of the internal state are used).

[1]Note that the simulators are passed a random variable $P^{CDP}_B(\mathcal{D}, \epsilon))$, i.e., the simulator is given the ability to sample from this distribution.

Next, every Cryptϵ program expressed as a sequence of transformation operators followed by a measurement operator, satisfies $\epsilon/2$-DP (as in Definition 1). It is so because recall that the measurement operators add noise from $Lap(\frac{2\Delta}{\epsilon})$ (Section 5.2) where Δ denotes the sensitivity of P (computed w.r.t to Definition 1) [39, 48]. However, Cryptϵ reveals both the output of the program as well as the total size of dataset $\mathcal{D}$. While revealing the size exactly would violate Definition 1, it does satisfy *bounded*-DP albeit with twice the privacy parameter, ϵ – changing a row in $\mathcal{D}$ is equivalent to adding a row and then removing a row.

Finally, since every program P executed on Cryptϵ satisfies ϵ-bounded DP, it follows from Theorem 3 that every execution of Cryptϵ satisfies computational DP.

Corollary 4. *Protocol Π satisfies computational differential privacy under the SIM-CDP notion [80].*

Note that Theorem 3 assumes that AS and the CSP do not collude with the users (data owners). However, if the AS colludes with a subset of the users, U, then Sim_1 (Sim_2) has to be given the data corresponding to users in U as additional parameters. This presents no complications in the proof (see the proof in [49]). If a new user u joins, their data can be encrypted and simply added to the database. We discuss extensions to handle malicious adversaries in Section 10.

8 CRYPTϵ OPTIMIZATIONS

In this section, we present the optimizations used by Cryptϵ.

8.1 DP Index Optimization

This optimization is motivated by the fact that several programs, first, filter out a large number of rows in the dataset. For instance, P5 in Table 2 constructs a histogram over *Age* and *Gender* on the subset of rows for which $NativeCountry$ is Mexico. Cryptϵ's filter implementation retains all the rows as the AS has no way of telling whether the filter condition is satisfied. As a result, the subsequent GroupbyCount is run on the full dataset. If there were an index on $NativeCountry$, Cryptϵ could run the GroupbyCount on only the subset of rows with $NativeCountry$=Mexico. But an exact index would violate DP. Hence, we propose a DP index to bound the information leakage while improving the performance.

At a high-level, the DP index on any ordinal attribute A is constructed as follows: (1) securely sort the input encrypted database, $\tilde{\mathcal{D}}$, on A and (2) learn a mapping, $\mathcal{F}$, from the domain of A to $[1, |\tilde{\mathcal{D}}|]$ such that most of the rows with index less than $\mathcal{F}(v), v \in domain(A)$, have a value less than v. The secure sorting is done via the following garbled circuit that (1) inputs $\tilde{\mathcal{D}}$ (just the records without any identifying features) and indexing attribute A from the AS (2) inputs the secret key sk from the CSP (3) decrypts and sort $\mathcal{D}$ on A (4) re-encrypt the sorted database using pk and outputs $\tilde{\mathcal{D}}_s = labEnc_{pk}(sort(\mathcal{D}))$. The mapping, $\mathcal{F}$, must be learned under DP, and we present a method for that below. Let $P = (P_1, \dots, P_k)$ be an equi-width partition on the sorted domain of A such that each partition (bin) contains $\frac{s_A}{k}$ consecutive domain values where s_A is the domain size of A. The index is constructed using a Cryptϵ program that firstly computes the noisy prefix counts, $\hat{V}[i] = \sum_{v \in \cup_{l=1}^{i} P_l} ct_{A,v} + \eta_i$ for $i \in [k]$, where $\eta_i \sim Lap(2k/\epsilon_A)$ and $ct_{A,v}$ denotes the number of rows with value v for A. Next, the program uses isotonic regression [58] on $\hat{V}$ to generate a noisy cumulative histogram $\tilde{C}$ with non-decreasing counts. Thus, each prefix count in $\tilde{C}$ gives an *approximate index* for the sorted database where the values of attribute A change from being in P_i to a value in P_{i+1}. When a Cryptϵ program starts with a filter $\phi = A \in [v_s, v_e]$, we compute two indices for the sorted database, i_s and i_e, as follows. Let v_s and v_e fall in partitions P_i and P_j respectively. If P_i is the first partition, then we set $i_s = 0$; otherwise set i_s to be 1 more than the $i-1$-th noisy prefix count from $\tilde{C}$. Similarly, if P_j is the last partition, then we set $i_e = |\tilde{\mathcal{D}}|$; otherwise, we set i_e to be the $j+1$-th noisy prefix count from $\tilde{C}$. This gives us the DP mapping $\mathcal{F}$. We then run the program on the subset of rows in $[i_s, i_e]$. For example, in Figure 2, the indexing attribute with domain $\{v_1, \cdots, v_{10}\}$ has been partitioned into $k = 5$ bins and if $\phi \in [v_3, v_6]$, $i_s = \tilde{C}[1] + 1 = 6$ and $i_e = \tilde{C}[3] = 13$.

Lemma 5. *Let P be the program that computes the mapping $\mathcal{F}$. Let Π be the Cryptϵ protocol corresponding to the construction of the DP index. The views and outputs of the AS and the CSP are denoted as $View_1^{\Pi}(P, \mathcal{D}, \epsilon_A)$, $Output_1^{\Pi}(P, \mathcal{D}, \epsilon_A)$ and $View_2^{\Pi}(P, \mathcal{D}, \epsilon_A)$, $Output_2^{\Pi}(P, \mathcal{D}, \epsilon_A)$ respectively. There exists PPT simulators Sim_1 and Sim_2 such that:*

- *$Sim_1(P_B^{CDP}(\mathcal{D}, \epsilon_A)) \equiv_c (View_1^{\Pi}(P, \mathcal{D}, \epsilon_A), Output^{\Pi}(\mathcal{D}, \epsilon_A))$, and*
- *$Sim_2(P_B^{CDP}(\mathcal{D}, \epsilon)) \equiv_c (View_2^{\Pi}(P, \mathcal{D}, \epsilon_A), Output^{\Pi}(\mathcal{D}, \epsilon_A))$.*

$Output^{\Pi}(P, \mathcal{D}, \epsilon_A))$ is the combined output of the two parties

The proof of the above lemma is presented in the full paper [6]. Here we present the intuition behind it. From the secure sorting algorithm (steps 1 and 4), it is evident that the servers cannot associate the records of the encrypted sorted dataset, $\tilde{\mathcal{D}}_s$, with the data owners. The AS can learn nothing from $\tilde{\mathcal{D}}_s$ due to the semantic security of the LHE scheme used. This ensures that the DP index construction of Cryptϵ satisfies the SIM-CDP privacy guarantee.

- **Optimized feature**: This optimization speeds up the program execution by reducing the total number of rows to be processed for the program.
- **Trade-off**: The trade-off is a possible increase in error as some of the rows that satisfy the filter condition may not be selected due to the noisy index.

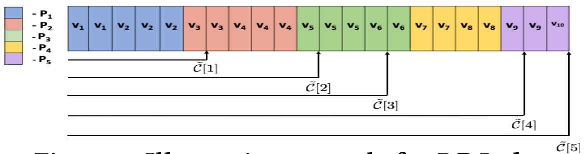


Figure 2: Illustrative example for DP Index

- **Privacy Cost**: Assuming the index is constructed with privacy parameter ϵ_A, the selection of a subset of rows using it will be ϵ_A-bounded DP (Lemma 5). If ϵ_L is the parameter used for the subsequent measurement primitives, then by Theorem 1, the total privacy parameter is $\epsilon_A + \epsilon_L$.

Discussion: Here we discuss the various parameters in the construction of a DP index. The foremost parameter is the indexing attribute A which can be chosen with the help of the following two heuristics. First, A should be frequently queried so that a large number of queries can benefit from this optimization. Second, choose A such that the selectivity of the popularly queried values of A is high. This would ensure that the first selection performed alone on A will filter out the majority of the rows, reducing the intermediate dataset size to be considered for the subsequent operators. The next parameter is the fraction of the program privacy budget, ρ ($\epsilon_A = \rho \cdot \epsilon$ where ϵ is the total program privacy budget) that should be used towards building the index. The higher the value of ρ, the better is the accuracy of the index (hence better speed-up). However, the privacy budget allocated for the rest of the program decreases resulting in increased noise in the final answer. This trade-off is studied in Figures 4a and 4b in Section 9. Another parameter is the number of bins k. Finer binning gives more resolution but leads to more error due to DP noise addition. Coarser binning introduces error in indexing but has lower error due to noise. We explore this trade-off in Figures 4c and 4d. To increase accuracy we can also consider bins preceding i_s and bins succeeding i_e. This is so because, since the index is noisy, it might miss out on some rows that satisfy the filter condition. For example, in Figure 2, both the indices $i_s = \tilde{C}[1] + 1 = 6$ and $i_e = \tilde{C}[3] = 13$ miss a row satisfying the filter condition $\phi = A \in [v_3, v_6]$; hence including an extra neighboring bin would reduce the error.

Thus, in order to gain in performance, the proposed DP index optimization allows some DP leakage of the data. This is in tune with the works in [28, 53, 60, 77]. However, our work differs from earlier work in the fact that we can achieve pure DP (albeit SIM-CDP). In contrast, previous work achieved a weaker version of DP, approximate DP [19], and added one-sided noise (i.e., only positive noise). One-sided noise requires addition of dummy rows in the data, and hence increases the data size. However, in our Cryptϵ programs, all the rows in the noisy set are part of the real dataset.

8.2 Crypto-Engineering Optimizations

(1) **DP Range Tree**: If range queries are common, pre-computed noisy range tree is a useful optimization. For example, building a range tree on *Age* attribute can improve the accuracy for P1 and P2 in Table 2. The sensitivity for such a noisy range tree is $\log s_A$ where s_A is the domain size of the attribute on which the tree is constructed. Any arbitrary range query requires access to at most $2 \log s_A$ nodes on the tree. Thus to answer all possible range queries on A, the total squared error accumulated is $O(\frac{s^2(\log s_A)^2}{\epsilon})$. In contrast for the naive case, we would have incurred error $O(\frac{s_A^3}{\epsilon})$ [58]. Note that, if we already have a DP index on A, then the DP range tree can be considered to be a secondary index on A.

- **Optimized Feature**: The optimization reduces both execution time and expected error when executed over multiple range queries.
- **Trade-off**: The trade-off for this optimization is the storage cost of the range tree ($O(2 \cdot s_A)$).
- **Privacy Cost**: If the range tree is constructed with privacy parameter ϵ_R, then any measurement on it is post-processing. Hence, the privacy cost is ϵ_R-bounded DP.

(2) **Precomputation**: The CrossProduct primitive generates the one-hot-coding of data across two attributes. However, this step is costly due to the intermediate interactions with the CSP. Hence, a useful optimization is to pre-compute the one-hot-codings for the data across a set of frequently used attributes $\bar{A}$ so that for subsequent program executions, the AS can get the desired representation via simple look-ups. For example, this benefits P3 (Table 2).

- **Optimized Feature**: This reduces the execution time of Cryptϵ programs. The multi-attribute one-hot-codings can be re-used for all subsequent programs.
- **Trade-off**: The trade-off is the storage cost (O($m \cdot s_{\bar{A}} = m \cdot \prod_{A \in \bar{A}} s_A$), m = the number of data owners) incurred to store the multi-attribute one-hot-codings for $\bar{A}$.
- **Privacy Cost**: The computation is carried completely on the encrypted data, no privacy budget is expended.

(3) **Offline Processing**: For GroupByCountEncoded, the CSP needs to generate the encrypted one-hot-codings for the masked histogram. Note that the one-hot-encoding representation for any such count would simply be a vector of ($|\tilde{\mathcal{D}}| - 1$) ciphertexts for '0', $labEnc_{pk}(0)$ and 1 ciphertext for '1', $labEnc_{pk}(1)$. Thus one useful optimization is to generate these ciphertexts offline (similar to offline generation of Beaver's multiplication triples [16] used in SMC). Hence, the program execution will not be blocked by encryption.

- **Optimized Feature**: This optimization results in a reduction in the run time of Cryptϵ programs.

- **Trade-off**: A storage cost of $O(m \cdot s_A)$ is incurred to store the ciphers for attribute A.
- **Privacy Cost**: The computation is carried completely on the encrypted data, no privacy budget is expended.

9 EXPERIMENTAL EVALUATION

In this section, we describe our evaluation of Cryptϵ along two dimensions, accuracy and performance of Cryptϵ programs. Specifically, we address the following questions:

- **Q1**: Do Cryptϵ programs have significantly lower errors than that for the corresponding state-of-the-art LDP implementations? Additionally, is the accuracy of Cryptϵ programs comparable to that of the corresponding CDP implementations?
- **Q2**: Do the proposed optimizations provide substantial performance improvement over unoptimized Cryptϵ?
- **Q3**: Are Cryptϵ programs practical in terms of their execution time and do they scale well?

Evaluation Highlights:

- Cryptϵ can achieve up to 50× smaller error than the corresponding LDP implementation on a data of size $\approx 30K$ (Figure 3). Additionally, Cryptϵ errors are at most 2× more than that of the corresponding CDP implementation.
- The optimizations in Cryptϵ can improve the performance of unoptimized Cryptϵ by up to 5667× (Table 3).
- A large class of Cryptϵ programs execute within 3.6 hours for a dataset of size 10^6, and they scale linearly with the dataset size (Figure 5). The AS performs majority of the work for most programs (Table 3).

9.1 Methodology

Programs: To answer the aforementioned questions, we ran the experiments on the Cryptϵ programs previously outlined in Table 2. Due to space limitations, we present the results of only four of them in the main paper namely P1, P3, P5 and P7. The rationale behind choosing these four is that they cover all three classes of programs (Section 6) and showcase the advantages for all of the four proposed optimizations.

Dataset: We ran our experiments on the Adult dataset from the UCI repository [7]. The dataset is of size 32, 651. For the scaling experiments (Figure 5), we create toy datasets of sizes 100K and 1 million by copying over the Adult dataset.

Accuracy Metrics: Programs with scalar outputs (P5, P7) use *absolute error* $|c - \hat{c}|$ where c is the true count and $\hat{c}$ is the noisy output. Programs with vector outputs (P1, P3) use the *L1 error metric* given by $Error = \sum_i |V[i] - \hat{V}[i]|, i \in [|V|]$ where V is the true vector and $\hat{V}$ is the noisy vector. We report the mean and s.t.d of error values over 10 repetitions.

Performance Metrics: We report the mean total execution time in seconds for each program, over 10 repetitions.

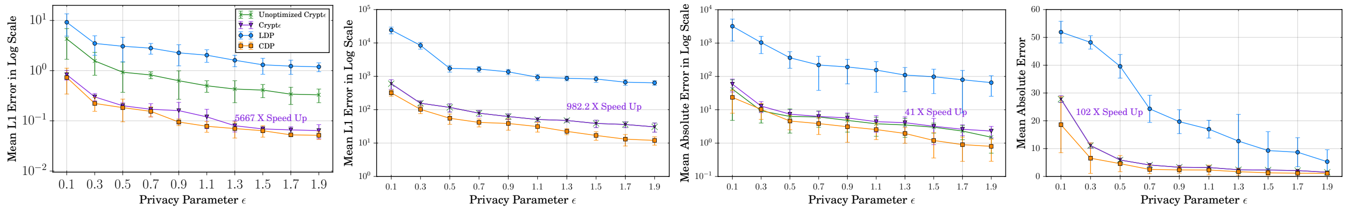
Configuration: We implemented Cryptϵ in Python with the garbled circuit implemented via EMP toolkit [2]. We use Paillier encryption scheme [88]. All the experiments have been performed on the Google Cloud Platform [1] with the configuration c2-standard-8. For Adult dataset, Cryptϵ constructs a DP index optimization over the attribute $NativeCountry$ that benefits programs like P4 and P5. Our experiments assign 20% of the total program privacy parameter towards constructing the index and the rest is used for the remaining program execution. Cryptϵ also constructs a DP range tree over *Age*. This helps programs like P1, P2 and P3. This is our default Cryptϵ implementation.

9.2 End-to-end Accuracy Comparison

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 [96] and CDP (under bounded DP; specifically, using the CDP view Cryptϵ is computationally indistinguishable from as shown in Section 7) [39] implementations.

The first observation with respect to accuracy is that the mean error for a single frequency count for Cryptϵ is at least 50× less than that of the corresponding LDP implementation. For example, Figure 3b shows that for P3, $\epsilon = 0.1$ results in a mean error of 599.7 as compared to an error of 34301.02 for the corresponding LDP implementation. Similarly, P5 (Figure 3c) gives a mean error of only 58.7 for $\epsilon = 0.1$. In contrast, the corresponding LDP implementation has an error of 3199.96. For P1 (c.d.f on *Age*), the mean error for Cryptϵ for $\epsilon = 0.1$ is given by 0.82 while the corresponding LDP implementation has an error of 9.2. The accuracy improvement on P7 (Figure 3d) by Cryptϵ is less significant as compared to the other programs, because P7 outputs the number of age values ($[1 - 100]$) having 200 records. At $\epsilon = 0.1$, at least 52 age values out of 100 are reported incorrectly on whether their counts pass the threshold. Cryptϵ reduces the error almost by half. Note that the additive error for a single frequency count query in the LDP setting is at least $\Omega(\sqrt{n}/\epsilon)$, thus the error increases with dataset size. On the other hand, for Cryptϵ the error is of the order $\Theta(1/\epsilon)$, hence with increasing dataset size the relative the error improvement for Cryptϵ over that of an equivalent implementation in LDP would increase.

For P1 (Figure 3a), we observe that the error of Cryptϵ is around 5× less than that of the unoptimized implementation. The reason is that P1 constructs the c.d.f over the attribute *Age* (with domain size 100) by first executing 100 range queries. Thus, if the total privacy budget for the program is ϵ, then for unoptimized Cryptϵ, each query gets a privacy parameter of just $\frac{\epsilon}{100}$. In contrast, the DP range tree is constructed with the full budget ϵ and sensitivity $\lceil \log 100 \rceil$ thereby resulting in lesser error. For P5 (Figure 3c) however,

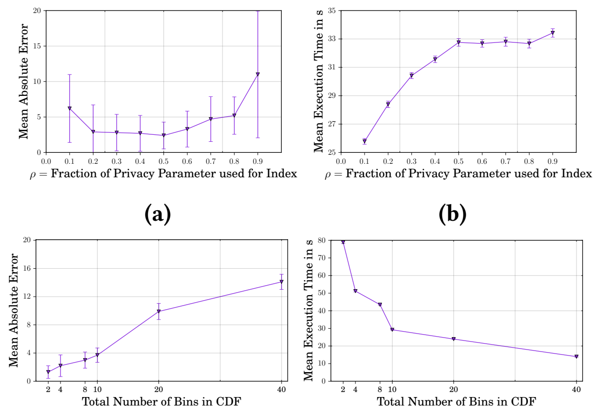


(a) Program 1 **(b) Program 3** **(c) Program 5** **(d) Program 7**

Figure 3: Accuracy Analysis of Cryptϵ Programs

Table 3: Execution Time Analysis for Cryptϵ Programs

Time in (s)		Program			
		1	3	5	7
Unoptimized Cryptϵ	AS	1756.71	6888.23	650.78	290
	CSP	0.26	6764.64	550.34	30407.73
	Total	1756.97	13652.87	1201.12	30697.73
Cryptϵ	Total	0.31	13.9	29.21	299.5
	Speed Up ×	5667.64	982.2	41.1	102.49



(a) **(b)** **(c)** **(d)**

Figure 4: Accuracy and performance of P5 at different settings of the DP index optimization

the unoptimized implementation has slightly better accuracy (around 1.4×) than Cryptϵ. It is because of two reasons; first, the noisy index on *NativeCountry* might miss some of the rows satisfying the filter condition (*NativeCountry*=Mexico). Second, since only 0.8% of the total privacy parameter is budgeted for the Laplace operator in the optimized program execution, this results in a higher error as compared to that of unoptimized Cryptϵ. However, this is a small cost to pay for achieving a performance gain of 41×. The optimizations for P3 (Figure 3b) and P7 (Figure 3d) work completely on the encrypted data and do not expend the privacy budget. Hence they do not hurt the program accuracy in any way.

Another observation is that for frequency counts the error of Cryptϵ is around 2× higher than that of the corresponding CDP implementation. This is intuitive because we add two instances of Laplace noise in Cryptϵ (Section 6.2). For P1, the CDP implementation also uses a range tree.

9.3 Performance Gain From Optimizations

In this section, we evaluate **Q2** (Table 3) by analyzing how much speed-up is brought about by the proposed optimizations in the program execution time.

DP Index: For P5, we observe from Table 3 that the unoptimized implementation takes around 20 minutes to run. However, a DP index over the attribute *NativeCountry* reduces the execution time to about 30s giving us a 41× speed-up. It is so because, only about 2% of the data records satisfy *NativeCountry*=Mexico. Thus the index drastically reduces the number of records to be processed for the program.

Additionally, we study the dependency of the accuracy and execution time of P5 implemented with the DP index on three parameters – (1) fraction of privacy budget ρ used for the index (2) total number of domain partitions (bins) considered (3) number of neighboring bins considered. The default configuration for Cryptϵ presented in this section uses $\epsilon = 2.2$, $\rho = 0.2$, total 10 bins and considers no extra neighboring bin.

In Figure 4a and 4b we study how the mean error and execution time of the final result varies with ρ for P5. From Figure 4a, we observe that the mean error drops sharply from $\rho = 0.1$ to $\rho = 0.2$, stabilises till $\rho = 0.5$, and starts increasing again. This is because, at $\rho = 0.2$, the index correctly identifies almost all the records satisfying the Filter condition. However, as we keep increasing ρ, the privacy budget left for the program after Filter (Laplace operator) keeps decreasing resulting in higher error in the final answer. From Figure 4b, we observe that the execution time increases till $\rho = 0.5$ and then stabilizes; the reason is that the number of rows returned after $\rho = 0.5$ does not differ by much.

We plot the mean error and execution time for P5 by varying the total number of bins from 2 to 40 (domain size of *NativeCountry* is 40) in Figure 4c and 4d respectively. From Figure 4c, we observe that the error of P5 increases as the number of bins increase. It is so because from the computation of the prefix counts (Section 8.1), the amount of noise added increases with k (as noise is drawn from $Lap(\frac{k}{\epsilon})$). Figure 4d shows that the execution time decreases with k. This is intuitive because increase in k results in smaller bins, hence the number of rows included in $[i_s, i_e]$ decreases.

To avoid missing relevant rows, more bins that are adjacent to the chosen range $[i_s, i_e]$ can be considered for the subsequent operators. Thus, as the number of bins considered increases, the resulting error decreases at the cost of higher execution time. The experimental results are presented in Figure 7a and Figure 7b in full paper [6].

DP Range Tree: For P1, we see from Table 3 that the total execution time of the unoptimized Cryptϵ implementation is about half an hour. However, using the range tree optimization reduces the execution time by 5667×. The reason behind this huge speed-up is that 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.

Pre-computation: For P3, the unoptimized execution time on the dataset of 32561 records is around 4 hours (Table 3). This is so because the CrossProduct operator has to perform $10 \cdot 32561$ *labMult* operations which is very time consuming. Hence, pre-computing the one-hot-codings for 2-D attribute over *Race* and *Gender* is very useful; the execution time reduces to less than a minute giving us a 982.2× speed up.

Offline Processing: The most costly operator for P7 is the GroupByCountEncoded operator since the CSP has to generate $\approx 3300K$ ciphertexts of 0 and 1 for the encrypted one-hot-codings. This results in a total execution time of about 8.5 hours in unoptimized Cryptϵ. However, by generating the ciphertexts off-line, the execution time can be reduced to just 5 minutes giving us a speed up of 102.49×.

Another important observation from Table 3 is that the AS performs the major chunk of the work for most program executions. This conforms with our discussion in Section 2.2.

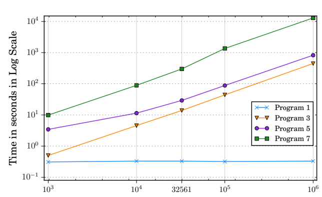


Figure 5: Scalability of Cryptϵ Programs

9.4 Scalability

In this section, we evaluate **Q3** by observing the execution times of the aforementioned four Cryptϵ programs for dataset sizes up to 1 million. As seen from Figure 5, the longest execution time (P7) for a dataset of 1 million records is $\approx$ 3.6 hours; this shows the practical scalability of Cryptϵ. All the reported execution times are for default setting. For P1 we see that the the execution time does not change with the dataset size. This is so because once the range tree is constructed, the program execution just involves reading the answer directly from the tree followed by a decryption by the CSP. The execution time for the P3 and P7 is dominated by the $\oplus$ operation for the GroupByCount operator. The cost of $\oplus$ is linear to the data size. Hence, the execution time for P3 and P7 increases linearly with the data size. For P5, the execution time depends on the % of the records in the dataset that satisfy the condition $NativeCountry = Mexico$ (roughly this many rows are retrieved from the noisy index).

10 EXTENSION OF CRYPTϵ TO THE MALICIOUS MODEL

In this section, we briefly discuss how to extend the current Cryptϵ system to account for malicious adversaries. We present one approach for the extension here and detail another approach in the full paper [6]. The first approach implements the CSP inside a trusted execution environment (TEE) [11, 21, 84]. This ensures non-collusion (as the CSP cannot collude with the AS since its operations are vetted). The measurement operators are implemented as follows (the privacy budget over-expenditure checking remains unchanged from that in Section 6.2 and we skip re-describing it here).

Laplace $Lap_{\epsilon,\Delta}(\boldsymbol{V}\backslash\boldsymbol{c})$: The new implementation requires only a single instance of noise addition by the CSP. The AS sends the ciphertext $\boldsymbol{c}$ to the CSP. The CSP decrypts the ciphertext, adds a copy of noise, $\eta \sim Lap(\frac{2\cdot\Delta}{\epsilon})$, and sends it to the AS.

NoisyMax $NoisyMax^{k}_{\epsilon,\Delta}(\mathbf{V})$: The new implementation works without the garbled circuit as follows. The AS sends the vector of ciphertexts, $\boldsymbol{V}$, to the CSP. The CSP computes $\tilde{V}[i] = labDecrypt_{sk}(\boldsymbol{V}[i]) + \eta[i], i \in [|\boldsymbol{V}|]$, where $\eta[i] \sim Lap(2k\Delta/\epsilon)$ and outputs the indices of the top k values of $\tilde{V}$.

Malicious AS: Recall that a Cryptϵ program, P, consists of a series of transformation operators that transform the encrypted database, $\tilde{\boldsymbol{\mathcal{D}}}$, to a ciphertext, $\boldsymbol{c}$ (or an encrypted vector, $\boldsymbol{V}$). This is followed by applying a measurement operator on $\boldsymbol{c}$ (or $\boldsymbol{V}$). Let P_1 represent the first part of the program P up to the computation of $\boldsymbol{c}$ and let P_2 represent the subsequent measurement operator (performed by the CSP inside a TEE). In the malicious model, the AS is motivated to misbehave. For example, instead of submitting the correct cipher $\boldsymbol{c} = P_1(\tilde{\boldsymbol{\mathcal{D}}})$ the AS could run a different program P' on the record of a single data owner only. Such malicious behaviour can be prevented by having the CSP validate the AS's work via zero knowledge proofs (ZKP) [87] as follows (similar proof structure as prior work [85]). Specifically, the ZKP statement should prove that the AS 1) runs the correct program P_1 2) on the correct dataset $\tilde{\boldsymbol{\mathcal{D}}}$. For this, the CSP shares a random one-time MAC key, $mk_i, i \in [m]$ with each of the data owners, DO_i. Along with the encrypted record $\tilde{\mathbf{D}}_i$, DO_i sends a Pedersen commitment [89] Com_i to the one-time MAC [66] on $\tilde{\mathbf{D}}_i$ and a short ZKP that the opening of this commitment is a valid one-time MAC on $\tilde{\mathbf{D}}_i$. The AS collects all the ciphertexts and proofs from the data owners and computes

$\boldsymbol{c} = P_1(\tilde{\mathbf{D}}_1, \cdots, \tilde{\mathbf{D}}_m)$. Additionally, it constructs a ZKP that $\boldsymbol{c}$ is indeed the output of executing P_1 on $\tilde{\mathcal{D}} = \{\tilde{\mathbf{D}}_1, \cdots, \tilde{\mathbf{D}}_m\}$ [25]. Formally, the proof statement is

$$\boldsymbol{c} = P_1(\tilde{\mathbf{D}}_1, \cdots, \tilde{\mathbf{D}}_m) \wedge \forall i \; \text{Open}(Com_i) = MAC_{mk_i}(\tilde{\mathbf{D}}_i) \quad (2)$$

The AS submits the ciphertext $\boldsymbol{c}$ along with all the commitments and proofs to the CSP. By validating the proofs, the CSP can guarantee $\boldsymbol{c}$ is indeed the desired ciphertext. The one-time MACs ensure that the AS did not modify or drop any of the records received from the data owners.

Efficient proof construction: Our setting suits that of designated verifier non-interactive zero knowledge (DV NIZK) proofs [27]. In a DV NIZK setting, the proofs can be verified by a single designated entity (as opposed to publicly verifiable proofs [54]) who possesses some secret key for the NIZK system. Thus in Cryptϵ, clearly the CSP can assume the role of the designated verifier. The framework for efficient DV NIZKs proposed by Chaidos and Couteau [27] can be applied to prove Eq. (2), as this framework enables proving arbitrary relations between cryptographic primitives, such as Pedersen commitment or Paillier encryption. A detailed construction is given in the full paper [6] which shows that all the steps of the proof involve simple arithmetic operations modulo N^2 where N is an RSA modulus. To get an idea of the execution overhead for the ZKPs, consider constructing a DV NIZK for proving that a Paillier ciphertext encrypts the products of the plaintexts of two other ciphertexts (this could be useful for proving the validity of our Filter operator, for example). In [27], this involves $4logN$ bits of communication and the operations involve addition and multiplication of group elements. Each such operation takes order of 10^{-5} seconds to execute, hence for proving the above statement for 1 million ciphertexts will take only a few tens of seconds.

Malicious CSP: Recall that our extension implements the CSP inside a TEE. Hence, this ensures that the validity of each of CSP's actions in the TEE can be attested to by the data owners. Since the measurement operators (P_2) are changed to be implemented completely inside the CSP, this guarantees the bounded ϵ-DP guarantee of Cryptϵ programs even under the malicious model. Additionally sending the CSP the true ciphers $\boldsymbol{c} = P_1(\tilde{\mathcal{D}})$ also does not cause any privacy violation as it is decrypted inside the TEE.

Validity of the data owner's records: The validity of the one-hot-coding of the data records, $\tilde{\mathbf{D}}_i$, submitted by the data owners DO_i can be checked as follows. Let $\tilde{\mathbf{D}}_{ij}$ represent the encrypted value for attribute A_j in one-hot-coding for DO_i. The AS selects a set of random numbers $R = \{r_k \mid k \in [|domain(A_j)|]\}$ and computes the set $P_{ij} = \{labMult(\tilde{\mathbf{D}}_{ij}[k], labEnc_{pk}(r_k))\}$. Then it sends sets P_{ij} and R to the CSP who validates the record only if $|P_{ij} \cap R| = 1 \; \forall A_j$. Note that since the CSP does not have access to the index information of P_{ij} and R (since they are sets), it cannot learn the value of D_{ij}. Alternatively each data owner can provide a zero knowledge proof for $\forall j, k, \; D_{ij}[k] \in \{0, 1\} \wedge \sum_k D_{ij}[k] = 1$.

11 RELATED WORK

Differential Privacy: Introduced by Dwork et al. in [39], differential privacy has enjoyed immense attention from both academia and industry in the last decade. Interesting work has been done in both the CDP model [9, 23, 31, 34, 35, 40, 55–57, 59, 71, 72, 76, 86, 90, 91, 101–103, 105, 106, 108] and the LDP model [14, 33, 43, 45, 92, 96–98, 109]. Recently, it has been showed that augmenting the LDP setting by a layer of anonymity improves the privacy guarantees [21, 32, 42]. It is important to note that the power of this new model (known as shuffler/mixnet model) lies strictly between that of LDP and CDP. Cryptϵ differs from this line of work in three ways, namely expressibility, precise DP guarantee and trust assumptions (details are in the full paper [6]).

Two-Server Model: The two-server model is popularly used for privacy preserving machine learning approaches where one of the servers manages the cryptographic primitives while the other handles computation [47, 49, 68, 81, 84, 85].

Homomorphic Encryption: Recently, there has been a surge in privacy preserving solutions using homomorphic encryptions due to improved primitives. A lot of the aforementioned two-server models employ homomorphic encryption [49, 68, 84, 85]. Additionally, it is used in [24, 26, 50, 51, 61].

12 CONCLUSIONS

In this paper, we have proposed a system and programming framework, Cryptϵ, for differential privacy that achieves the constant accuracy guarantee and algorithmic expressibility of CDP without any trusted server. This is achieved via two non-colluding servers with the assistance of cryptographic primitives, specifically LHE and garbled circuits. Our proposed system Cryptϵ can execute a rich class of programs that can run efficiently by virtue of four optimizations.

Recall that currently the data analyst spells out the explicit Cryptϵ program to the AS. Thus, an interesting future work is constructing a compiler for Cryptϵ that inputs a user specified query in a high-level-language. The compiler should next formalize a Cryptϵ program expressed in terms of Cryptϵ operators with automated sensitivity analysis. Another direction is to support a larger class of programs in Cryptϵ. For example, inclusion of aggregation operators such as sum, median, average is easily achievable. Support for multi-table queries like joins would require protocols for computing sensitivity [63] and data truncation [69].

Acknowledgement: This work was supported by NSF under grants 1253327, 1408982; and by DARPA and SPAWAR under contract N66001-15-C-4067.

REFERENCES

[1] Google cloud platform. https://cloud.google.com.

[2] https://github.com/emp-toolkit.

[3] https://github.com/encryptogroup/aby.

[4] https://github.com/kuleuven-cosic/scale-mamba.

[5] http://www.multipartycomputation.com/mpc-software.

[6] Full version of the paper, 2019. https://arxiv.org/abs/1902.07756.

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

[8] J. M. Abowd and I. M. Schmutte. An economic analysis of privacy protection and statistical accuracy as social choices. *American Economic Review*, 109(1):171–202, January 2019.

[9] 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.

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

[11] E. Aïmeur, G. Brassard, J. M. Fernandez, and F. S. Mani Onana. Alambic: a privacy-preserving recommender system for electronic commerce. *International Journal of Information Security*, 7(5), Oct 2008.

[12] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 524–540, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[13] M. Barbosa, D. Catalano, and D. Fiore. Labeled homomorphic encryption - scalable and privacy-preserving processing of outsourced data. In *ESORICS*, 2017.

[14] 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.

[15] J. Bater, X. He, S. Y. Tendryakova, A. Machanavajjhala, and J. Duggan. Shrinkwrap: Differentially-private query processing in private data federations. *CoRR*, abs/1810.01816, 2018.

[16] 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.

[17] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 451–468, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: On simultaneously solving how and what. *CoRR*, abs/1103.2626, 2011.

[19] A. Beimel, K. Nissim, and U. Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. *CoRR*, abs/1407.2674, 2014.

[20] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006*, pages 409–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[21] 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.

[22] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan. Optimized homomorphic encryption solution for secure genome-wide association studies. *IACR Cryptology ePrint Archive*, 2019:223, 2019.

[23] 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.

[24] 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.

[25] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 107–122, Berlin, Heidelberg, 1999. Springer-Verlag.

[26] 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.

[27] P. Chaidos and G. Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. *IACR Cryptology ePrint Archive*, 2017:1029, 2017.

[28] T.-H. H. Chan, K.-M. Chung, B. M. Maggs, and E. Shi. Foundations of differentially oblivious algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2448–2467, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics.

[29] T.-H. H. Chan, E. Shi, and D. Song. Optimal lower bound for differentially private multi-party aggregation. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 277–288, Berlin, Heidelberg, 2012. Springer-Verlag.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] D. Demmler, T. Schneider, and M. Zohner. Aby - a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[37] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438, Oct 2013.

[38] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 486–503, Berlin, Heidelberg, 2006. Springer-Verlag.

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

[40] 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.
[41] H. Ebadi and D. Sands. Featherweight pinq, 2015.
[42] 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.
[43] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.
[44] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pages 211–222, New York, NY, USA, 2003. ACM.
[45] G. Fanti, V. Pihur, and ÃŽlfar Erlingsson. Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries, 2015.
[46] 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.
[47] 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.
[48] C. Ge, X. He, I. F. Ilyas, and A. Machanavajjhala. Apex: Accuracy-aware differentially private data exploration. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 177–194, New York, NY, USA, 2019. ACM.
[49] 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.
[50] I. Giacomelli, S. Jha, R. Kleiman, D. Page, and K. Yoon. Privacy-preserving collaborative prediction using random forests, 2018.
[51] 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.
[52] A. Greenberg. Apple's 'differential privacy' is about collecting your data—but not *your* data. *Wired*, Jun 13 2016.
[53] A. Groce, P. Rindal, and M. Rosulek. Cheaper private set intersection via differentially private leakage. Cryptology ePrint Archive, Report 2019/239, 2019. https://eprint.iacr.org/2019/239.
[54] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. Cryptology ePrint Archive, Report 2007/155, 2007. https://eprint.iacr.org/2007/155.
[55] 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.
[56] 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.
[57] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 139–154, New York, NY, USA, 2016. ACM.
[58] 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.
[59] 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.
[60] X. He, A. Machanavajjhala, C. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1389–1406, New York, NY, USA, 2017. ACM.
[61] E. Hesamifard, H. Takabi, and M. Ghasemi. Cryptodl: Deep neural networks over encrypted data, 2017.
[62] 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.
[63] N. M. Johnson, J. P. Near, and D. X. Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.
[64] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. https://eprint.iacr.org/2011/272.
[65] 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.
[66] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.
[67] M. Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6):983–1006, Nov. 1998.
[68] 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.
[69] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: A differentially private sql query engine. *Proc. VLDB Endow.*, 12(11):1371–1384, July 2019.
[70] 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.
[71] 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.
[72] 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.
[73] N. Li, M. Lyu, D. Su, and W. Yang. *Differential Privacy: From Theory to Practice*. Morgan and Claypool, 2016.
[74] Y. Lindell and B. Pinkas. A proof of security of yao’s protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, Apr. 2009.
[75] Y. Lindell and B. Pinkas. A proof of security of yao’s protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, Apr. 2009.
[76] M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy. *PVLDB*, 10:637–648, 2017.
[77] S. Mazloom and S. D. Gordon. Secure computation with differentially private access patterns. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 490–507, New York, NY, USA, 2018. ACM.
[78] 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.
[79] I. Mironov. Renyi differential privacy. *CoRR*, abs/1702.07476, 2017.

[80] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 126–142, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[81] 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.

[82] 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.

[83] 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.

[84] 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 & Communications Security*, CCS '13, pages 801–812, New York, NY, USA, 2013. ACM.

[85] 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.

[86] 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 âĂŽ13*, 2013.

[87] G. Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[88] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[89] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.

[90] 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.

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

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

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

[98] 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.

[99] X. Wang, S. Ranellucci, and J. Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 21–37, New York, NY, USA, 2017. ACM.

[100] 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.

[101] 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.

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

[103] 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.

[104] 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.

[105] 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.

[106] 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.

[107] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. EKTELO: A framework for defining differentially-private computations. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 115–130, 2018.

[108] 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.

[109] 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.