G. G. Dagher, B. C. M. Fung, N. Mohammed, and J. Clark. SecDM: privacy-preserving data outsourcing framework with differential privacy. *Knowledge and Information Systems (KAIS)*, 20 pages, in press. Springer.

SecDM: Privacy-preserving Data Outsourcing Framework with Differential Privacy

Gaby G. Dagher · Benjamin C. M. Fung · Noman Mohammed · and Jeremy Clark

Received: date / Accepted: date

Abstract Data-as-a-service (DaaS) is a cloud computing service that emerged as a viable option to businesses and individuals for outsourcing and sharing their collected data with other parties. Although the cloud computing paradigm provides great flexibility to consumers with respect to computation and storage capabilities, it imposes serious concerns about the confidentiality of the outsourced data as well as the privacy of the individuals referenced in the data. In this paper we formulate and address the problem of querying encrypted data in a cloud environment such that query processing is confidential and the result is differentially private. We propose a framework where the data provider uploads an encrypted index of her anonymized data to a DaaS service provider that is responsible for answering range count queries from authorized data miners for the purpose of data mining. To satisfy the confidentiality requirement, we leverage attribute based encryption to construct a secure kd-tree index over the differentially private data for fast access. We also utilize the exponential variant of the ElGamal cryptosystem to efficiently perform homomorphic operations on encrypted data. Experiments on real-life data demonstrate that our proposed framework preserves data utility, can efficiently answer range queries, and is scalable with increasing data size.

1 Introduction

Cloud computing is a new computing paradigm that enables organizations to have access to a large-scale computation and storage at an affordable price. Data-as-a-service (DaaS) is one of the cloud computing services that allow hosting and managing large-scale databases in the cloud on behalf of the data owner. DaaS is a compelling service for organizations, as they no longer need to invest in hardware, software and operational overheads. However, despite all these benefits, organizations are reluctant to

F. Author
Boise State University
E-mail: gabydagher@boisestate.edu

adopt DaaS model, as it requires outsourcing the data to an untrusted cloud service provider that may cause data breaches.

In recent years, there has been a considerable effort to ensure data confidentiality and integrity of outsourced databases. Several research proposals suggest encrypting the data before moving it to the cloud [1,2]. While encryption can provide data confidentiality, it is less effective in deterring inference attacks. This reality demands new privacy-enhancing technologies that can simultaneously provide data confidentiality and prevent inference attacks due to aggregate query answering.

Privacy-preserving data publishing (PPDP) is the process of anonymizing person-specific information for the purpose of protecting individuals' privacy while maintaining an effective level of data utility for data mining. Different PPDP privacy models provide different types of privacy protection [3]. Differential privacy [4] is a recently proposed privacy model that provides a provable privacy guarantee. Differential privacy is a rigorous privacy model that makes no assumption about an adversary's background knowledge. A differentially-private mechanism ensures that the probability of any output (released data) is equally likely from all nearly identical input data sets and thus guarantees that all outputs are insensitive to any individual's data.

In this paper, we propose a cloud-based query processing framework that simultaneously preserves the confidentiality of the data and the query requests, while providing differential privacy guarantee on the query results to protect against inference attacks. Let us consider the following real-life scenario. Population Data BC (PopData) [1] is a non-profit organization (data bank) responsible (among other things) for storing and managing patient-specific health data received from several hospitals, health organizations and government agencies in the Province of British Colombia, Canada. PopData utilizes explicit identifiers to integrate the data, and then de-identifies the integrated data by separating the explicit identifiers from the rest of the data contents. Data miners who are interested in querying the data initially sign a non-identifiability agreement to prevent them from releasing research data that can be used to re-identify individuals. When PopData receives a data access request, it first authenticates the data miner, verifies that she is working on an approved research project, and then executes the query on the de-identified data and returns the result back to the data miner. Similar organizations can be found in other countries, e.g., the National Statistical Service [2] in Australia.

A major concern in this scenario is *data privacy*. Although the data is de-identified, data miners can still perform (or accidentally release a research results that can leads to) record/attribute linkage attacks and re-identification of individuals, as was shown in the cases of AOL [5] and Netflix [6]. On the other hand, to minimize the workload on PopData, cloud services can be used to store, manage, and answer queries on the integrated data. However, this rises two other concerns. One concern is *data confidentiality*, where the outsourced patient-specific data must be stored in a protected way to prevent the cloud from answering queries from unauthorized data miners, and

[1] PopData: https://www.popdata.bc.ca/

[2] Statistical Data Integration Involving Commonwealth Data: http://statistical-data-integration.govspace.gov.au/

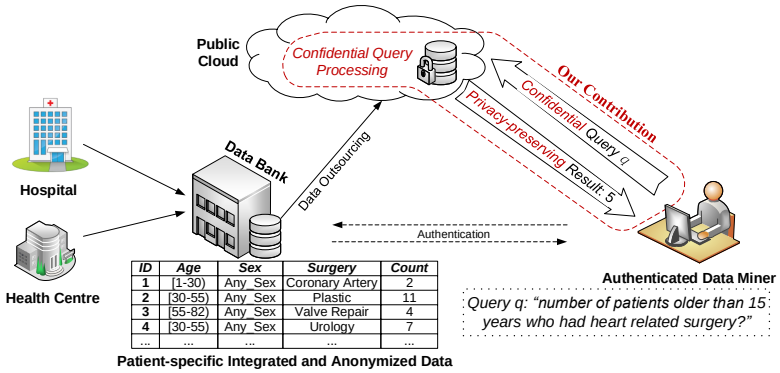


Fig. 1 PRIST: A Privacy-preserving framework for software testing using differential privacy.

to protect against potential multi-tenancy problems due to the sharing of services, resources, and physical infrastructure between multiple independent tenants on the cloud [7]. Another concern is *query confidentiality*, where the cloud should be able to execute query requests from authorized data miners without the ability to know what attributes and attribute values are specified in each query.

As shown by [8], count queries can be quite useful for data mining and statistical analysis applications where miners focus on extracting new trends and patterns from the overall data and are less interested in particular records.

Figure 1 illustrates the overall process of our proposed framework. Each data owner (e.g. hospital, health center) submits its raw data to the data bank (data provider). The data bank first integrates all data together, and then applies a PPDP privacy model on the integrated data such that explicit identifiers of record owners are removed, while other attributes (including sensitive attributes) are anonymized and retained for data analysis. Next, the data bank encrypts the anonymized data and upload it to the service provider (public cloud). Data miners authenticate themselves to the data bank and then submit their encrypted count queries to the cloud. The cloud securely processes each query, homomorphically computes the exact noisy count, and then sends the encrypted result back to the data miner. The proposed framework, named SecDM, achieves data privacy by supporting any privacy algorithm whose output is a contingency table data. Attribute-base Encryption (ABE) and ElGamal schemes are used to achieve data and query confidentiality. We analyze in Section 4.3 the benefit of outsourcing the data to a service provider as compared to having the data bank handle the user queries directly and show that the processing overhead on the data bank is almost 10 times less than the overhead on the service provider. While our framework protects the confidentiality of individual query (data access), we provide a detailed security analysis in Section A.

The intuition of our solution is to generate a kd-tree index for efficient traversal and secure access on the anonymized data, where the index tree is encrypted using *attribute based encryption* and stored on the public cloud. When a data miner desires to query the outsourced data, she sends her proof of identity to the data provider

Table 1 Comparative evaluation of main features in related query processing approaches (properties in columns are positioned as beneficial with fulfilment denoted by ● and partial fulfilment by ○)

	Supported Queries			Security			Client			
Approach	**Exact Query**	**Range Query**	**Similarity Query**	**Data Confidentiality**	**Query Confidentiality**	**Privacy-preserving Result**	**Query Reuse**	**Low Storage Overhead**	**Low Communication Overhead**	**Low Post-processing Overhead**
Private Spatial Decomposition (PSD) [16]	●	●				●		●	●	
SQL over Encrypted Data (SED) [14]	●	●		●	●		●		●	
OPESS [17]	●	●		●	●				●	
Salted IDA [11]	●	●			●					
ASM-PH [15]	●		●		●					
NEST [18]			●					●	●	●
PPNNS [19]	●		●		●	●		●		●
R-tree [20]	●	●		●	○			●	●	
kd-PHR [9]	●	●		●	●			●		○
Our proposed solution SecDM [Section **??**-4.2]	●	●		●	●	●	●	●	●	●
Our proposed solution C-SecDM [Section 4.3]†	●	●		●	●	●		●	●	●

† In C-SecDM, all communications of the data miners go through one party (data provider), and no decryption of query result is needed.

with her query and receives an encrypted version of her query, namely, system query, which she sends to the cloud for processing. The cloud uses the system query to traverse the encrypted kd-tree index and securely compute the total count representing the privacy-preserving answer to the query. The cloud then sends the answer back to the data miner, who in turn decrypts the encrypted results using a decryption key provided originally by the data provider. Our framework protects the confidentiality of each individual query by keeping its predicates hidden from the cloud. However, it does not hide the search pattern of the queries. We provide formal definition of framework properties as well as detailed security analysis in Section A.

The contributions of this paper can be summarized as follows:

Contribution 1. We propose SecDM, a comprehensive privacy-preserving framework for query processing in a cloud computing environment. SecDM maintains the privacy and utility properties of the outsourced data while simultaneously ensuring *data confidentiality*, *query confidentiality*, and *privacy-preserving results*. Previous work [9][10][11][12][13] satisfies only a subset of the aforementioned security features. We refer the reader to Section 2 for a detailed comparison.

Contribution 2. To enable efficient data access on the cloud while maintaining data and query confidentiality, we propose an algorithm for constructing an encrypted kd-tree index while utilizing *attribute based encryption* in order to support range predicates on numerical attributes. We demonstrate the efficiency of our solution by showing that SecDM has linear time complexity w.r.t. the number of attributes, and it is sub-linear w.r.t. the data size on query processing. Extensive experiments on real-life data further confirm these properties.

Contribution 3. Most existing work on the problem of data outsourcing in cloud computing environments either requires the query issuer to have prior knowledge about the data and subsequently requires storage and communication overhead [11], or yields results that require postprocessing on the query issuer's side [14], or both [15]. In contrast, data miners in our proposed framework are considered "lightweight clients" as they are not required to have or store any information about the data, nor are they required to perform post-processing on the results (except for decrypting the results). The communication complexity with the cloud is constant with respect to the size of the dataset and the query type.

2 Related Work

In this section, we review the literature that examines several areas related to our work. Table 1 summarizes the features of the representative approaches, including our proposed solutions.

2.1 Privacy-Preserving Data Publishing

One related area is *privacy-preserving data publishing (PPDP)* [3], where data is published in such a way that useful information can be obtained from the published data while data privacy is preserved. A common PPDP approach is *anonymization*. Several privacy models were proposed in the literature for providing different types of privacy protection. For example, the (α, k)-anonymity model [21] applies generalization and suppression techniques to protect against record and attribute linkages. The ε-differential privacy model [4] aims at protecting against table linkage and probabilistic attacks by ensuring that the probability distribution on the published data is the same regardless of whether or not an individual record exists in the data. Mohammed *et al.* [22] propose a generalization-based anonymization algorithm in a non-interactive setting for releasing differentially private records for data mining. Chen *et al.* [23] propose a method for anonymizing high-dimensional data and releasing synthetic dataset satisfying differential privacy using sampling-based framework to identify attributes' dependencies. Cormode *et al.* [16] propose a framework for using spatial data structures to provide a differentially private description of the data distribution. Xiao *et al.* [24] propose another framework that uses kd-tree based partitioning for differentially private histogram release. These frameworks support range queries while providing privacy guarantee; however, these techniques are not suitable for the outsourcing scenario as they provide no data confidentiality against the cloud service provider. Our work assumes that the outsourced data is a data table that is anonymized to satisfy a privacy requirement. To maximize the data utility for classification analysis, we utilize the anonymization approach in [22].

2.2 Confidentiality in Data Outsourcing

Another area related to our work is *confidentiality in data outsourcing*, where data is stored and managed by one or more untrusted parties that are different from the data owner. Queries are executed on the data while keeping the data confidential and without revealing information about the queries. A commonly used mechanism for ensuring data confidentiality is *encryption*. Some approaches propose to process queries over encrypted data directly. However, such approaches do not provide a good balance between data confidentiality and query execution. For example, methods in [14] [25] attach range labels to the encrypted data, thus revealing the underlying distributions of the data. Other methods depend on order-preserving encryption [26][27]; however, these methods reveal the data order and are subject to inference and statistical attacks. Homomorphic encryption, on the other hand, is a promising public

cryptosystem that allows query execution on encrypted data [1][28]; however, its high computation cost makes it prohibitive in practice. The authors in [29] propose to store encrypted links to the data queries in Blockchain, and use smart contracts to retrieve the data. In this paper, we employ the exponential variation of ElGamal [30] encryption scheme in one area of our solution by taking advantage of its additive homomorphism property. We show that this scheme is efficiently employed because the encrypted message is small enough for the scheme to remain practical.

Instead of processing queries directly over encrypted data, some approaches propose using indexing structures for fast data access and efficient query execution [31][17][32]. Some indexing schemes have constraints on the type of queries they support. For example, hash-based indexing [33] and privacy homomorphism [34] only support equality queries, whereas bucket-based indexing [14] and character-oriented indexing [35][36] support equality queries as well as partially supporting range queries. To support both equality queries and range queries, a category of approaches propose using disk-based indexes such as B-tree [37] and B^+-tree [38] and spatial access indexes such as kd-tree [39] and R-tree [40]. Our work fits in this category because we utilize an encrypted kd-tree index for efficient and secure traversal. Wang *et al.* [11] propose a framework based on B^+-tree index for query processing on relational data in the cloud. However, in order to protect data confidentiality against the cloud, the proposed solution generates a superset of the result and requires the client (querying user) to perform predicates evaluation in order to compute the final result. Hu *et al.* [15] propose a framework based on R-tree index for secure data access and processing of k-nearest-neighbor (kNN) similarity queries. However, the proposed approach partitions the R-tree index constructed over the outsourced data into two indexes, one is hosted by the cloud and the other is hosted by the client. In addition, a high communication bandwidth is required to achieve access confidentiality. Recently, Wang and Ravishankar [20] proposed a framework for performing half-space range queries using an $\widehat{\mathrm{R}}$-tree index that is encrypted using Asymmetric Scalar-product Preserving Encryption (ASPE) scheme [41]. Their method ensures data confidentiality and requires low communication and storage overhead on the client side. However, it does not provide a privacy guarantee, nor does it provide full confidential query processing because it leaks information on the ordering of the minimum bounding box of the leaf nodes and requires result postprocessing because it introduces false positives. Barouti *et al.* [9] proposed a protocol for secure storage of patient health records on the cloud, while allowing health organizations to securely query the data. The proposed protocol, however, does not provide privacy guarantees on the query results, while requiring high communication overhead on the client side.

2.3 Searching on Encrypted Data

Searchable encryption (SE) [42] [43] [44] [45] is a closely related line of work that supports secure searching on encrypted data. SE schemes (except for [42]) enable the data provider to generate a searchable encrypted index over a set of keywords. Most of these indexes, however, leak information about the relation between the keywords and the underlying data, the search pattern, and the access pattern [46]. In contrast,

our proposed framework reveals only the search pattern of the queries to the cloud. Functional encryption (FE) [47] is another related line of work that support searching on encrypted data. It includes identity-based encryption [48], attribute-based encryption [49], and predicate encryption [50]. We choose *cipher-policy attribute-based encryption* (CP-ABE) to construct our searchable encrypted index since CP-ABE supports fine-grained access control that can be utilized to handle not only keywords but also numerical ranges. In [51], the authors propose a PIR solution based on Paillier encryption scheme for privately retrieving a cell from an encrypted data warehouse. That is, it allows users to perform OLAP operations without revealing to the server which operation is performed and which cell is being retrieved.

Unlike the aforementioned approaches, our proposed solution ensures data and query confidentiality and privacy-preserving results while assuming that the client has no prior knowledge about the data being queried and its structure. No further interaction is required between the cloud and the client once the latter has submitted her query to the cloud, and no local refinement is required by the client on the final result. Table 1 summarizes the features of the representative approaches, including our proposed solutions.

3 Problem Formulation

In this section we formally define the research problem. First, we present an overview of the problem of confidential query processing, with privacy guarantee on outsourced data in the cloud in Section 3.1. Next, we define the input components in Section 3.2. We then describe the trust and adversarial model in Section 3.3. Finally, we present the problem statement in Section 3.4.

3.1 Problem Overview

In this paper we examine a cloud computing model consisting of three parties: *data provider*, *data miner*, and *service provider*. The data provider, for example, represents a data bank that owns an integrated patient-specific database. The data miner represents a user who is interested in querying the data for the purpose of performing analytical data mining activities such as classification analysis. The service provider is a public (untrusted) party that facilitates access to IT resources, i.e., storage and computational services.

The data provider desires to make its data available to authorized data miners. Due to its limited resources, the data provider outsources the database to a service provider capable of handling the responsibility of answering count queries from data miners. To prevent the disclosure of patients' sensitive information, the data provider anonymizes its data and generates a set of records that satisfy ε-differential privacy. Even though the outsourced data is anonymized, the data provider wants to protect the data against the service provider so it cannot answer queries on the data from untrusted (unauthorized) data miners. The service provider, however, should be able

PID	Country	Job	Age	Salary	Surgery
1	USA	Engineer	30	45K	Plastic
2	USA	Lawyer	55	45K	Valve Repair
3	Canada	Engineer	30	30K	Coronary Artery
4	USA	Lawyer	45	75K	Plastic
5	Canada	Dancer	18	60K	Coronary Artery
6	USA	Writer	30	45K	Urology
7	Canada	Writer	45	75K	Valve Repair
8	Canada	Dancer	65	45K	Valve Repair

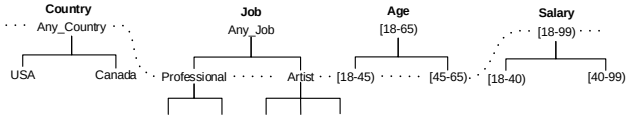


Fig. 2 A raw data table D and its taxonomy trees.

to process count queries from authorized data miners confidentially and return results that provide a certain privacy guarantee.

3.2 System Inputs

In this section we give a formal definition of the input components, namely, *differentially private data* and *user count queries*. Without loss of generality, we assume that the input data is anonymized using an ε-differential privacy model [4], although our approach supports other privacy models that produce contingency-like tables based on generalization and suppression. We choose ε-differential privacy because it provides a strong privacy guarantee while being insensitive to any specific record. We first describe how to generate ε-differentially private records from a relational data, then we explain how to transform the data using taxonomy trees, and finally we define the types of count queries the user can submit.

3.2.1 Differentially Private Data

In this section we review how a data provider can generate ε-differentially private records. We utilize the differentially private anonymization algorithm *(DiffGen)* [22] to maximize the data utility for classification analysis. Suppose a data provider owns an integrated patient-specific data table $D = \{A^I, A^{pr}, A^{cls}\}$, where A^I is an *explicit identifier* attribute such as *SSN* or *Name* for explicitly identifying individuals that will not be used for generating the ε-differentially private data; A^{cls} is a class attribute that contains the class value; and $\mathcal{A}^{pr}$ is a set of k predictor attributes whose values are used to predict the class attribute A^{cls}. We require the class attribute A^{cls} to be categorical, whereas the predictor attributes in A^{pr} are required to be either categorical or numerical. Furthermore, we assume that for each predictor attribute $A_i \in A^{pr}$ a taxonomy tree $\mathbb{T}^{A_i}$ is used in order to specify the hierarchy among the domain values of A_i. Figure 2 shows a raw data table D with four attributes, namely, *Country*, *Job*, *Age*, and *Salary* and the taxonomy tree for each attribute.

The data provider's objective is to generate an anonymized version $\hat{D} = \{\hat{A}^{pr}, NCount\}$ of the data table D, where $\hat{A}^{pr}$ is the set of k generalized predictor attributes, and

Table 2 Differentially-private data table $\hat{D}$

$\hat{Country}$	$\hat{Job}$	$\hat{Age}$	$\hat{Salary}$	*NCount*
Any_Country	Professional	[18-45)	[18-99)	4
Any_Country	Professional	[45-65)	[18-99)	2
Any_Country	Artist	[18-45)	[18-99)	1
Any_Country	Artist	[45-65)	[18-99)	5

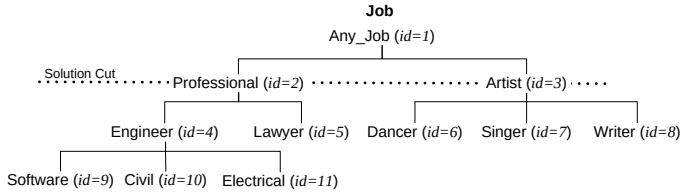


Fig. 3 Taxonomy tree $\mathbb{T}^{Job}$ for attribute *Job*.

$NCount$ is the noisy count of each record in $\hat{D}$. The objective of the data miner is to build a classifier to accurately predict the class attribute A^{cls} by submitting count queries on the generalized predictor attributes $\hat{A}^{pr}$.

Although we focused in this paper on contingency-like data tables, our framework can be extended to handle general type of differentially private data. In this case, a pre-processing step must take place before the secure index tree is generated. The pre-processing step would involve applying a top-down specialization approach (similar to *DiffGen*) and use utility measure functions (e.g. information gain, Max, Gini) to guide the specialization. No privacy budget would be consumed in this step, as the data is already differentially private. According to [52], any processing on a differentially private data does not violate its privacy.

3.2.2 Input Data Transformation

We simplify the representation of the ε-differentially private records $\hat{D} = \{\hat{A}^{pr}, NCount\}$ by mapping the values of each attribute to their integer identifiers from the corresponding attribute's taxonomy tree.

Numerical Attributes. The domain of each numerical attribute $\hat{A}_i \in \hat{A}^{pr}$, consists of a set of ranges that are pair-wise disjoint and can be represented as a continuous and ordered sequence of ranges. We define an order-preserving identification function ID^{op} that assigns an integer identifier to each range $r = [r^{min}, r^{max}]$ such that for any two ranges r_j and r_l, if $r_j^{max} < r_l^{min}$, then $ID^{op}(r_j) < ID^{op}(r_l)$. For example, if the domain of the generalized attribute $\hat{Age}$ is $\Omega(\hat{Age}) = \langle[18, 45), [45, 65)\rangle$, then $ID^{op}([18, 45)) = 1$ and $ID^{op}([45, 65)) = 2$.

Categorical Attributes. The domain of each categorical attribute $\hat{A}_i \in \hat{A}^{pr}$ consists of the set of values $Cut(\mathbb{T}^{A_i})$. We define a taxonomy tree identification function ID^t such that for any two nodes $v_i, v_j : v_j \neq v_i$, if v_i is a parent of v_j, then $ID^t(v_i) < ID^t(v_j)$. If v_i is the root node, then $ID^t(v_i) = 1$. Figure 3 illustrates the taxonomy tree $\mathbb{T}^{Job}$ for attribute *Job*, where each node is assigned an identification value.

Table 3 Transformed Data Table $\hat{D}$

$\hat{Country}$	$\hat{Job}$	$\hat{Age}$	$\hat{Salary}$	*NCount*
1	2	1	1	4
1	2	2	1	2
1	3	1	1	1
1	3	2	1	5

Having defined the mapping functions ID^{op} and ID^{t}, we now transform the ε-differentially private records $\hat{D}$ by mapping the values in the domain of each attribute to their identifiers. That is, for each numerical attribute $\hat{A}_i \in \hat{A}^{pr}$, we map each range $r \in \Omega(\hat{A}_i)$ to its corresponding identification value $ID^{op}(r)$. Similarly, for each categorical attribute $\hat{A}_i \in \hat{A}^{pr}$, we map each value in $v \in \Omega(\hat{A}_i)$ to its identification value from the taxonomy tree $ID^{t}(v)$. Table 3 shows the differentially private data $\hat{D}$ after the transformation.

3.2.3 User Count Queries

The goal of the data miners is to build a classifier based on the noisy count of a query over the generalized attributes $\hat{A}^{pr}$. Therefore, they submit count queries to be processed on the ε-differentially private data $\hat{D}$ and expect to receive a noisy count as a result to each submitted query. We denote by *user count query* any data mining's count query, and it is formally defined as follows:

Definition 1 ***(User Count Query.)*** *A* user count query u *over $\hat{D}$ is a conjunction of predicates $\mathcal{P}_1 \wedge ... \wedge \mathcal{P}_m$ where each predicate $\mathcal{P}_j = (\hat{A}_i \vdash s_i) : 1 \leq j \leq m$ expresses a single criterion such that $\hat{A}_i \in \hat{A}^{pr}$, $\vdash$ is a comparison operator, and s_i is an operand. If $\hat{A}_i$ is a categorical attribute, then $\vdash$ corresponds to the equality operator " $=$ " and s_i is a value from the taxonomy tree $\mathbb{T}^{A_i}$. If $\hat{A}_i$ is a numerical attribute, then s_i is a numerical range $[s_i^{min}, s_i^{max}]$ such that if $s_i^{min} = s_i^{max}$ then $\vdash$ is in $\{>, \geq, <, \leq, =\}$; otherwise, $\vdash$ is the equal operator $(=)$.* ■

In general, a user count query u can be either *exact*, *specific*, or *generic* depending on whether it corresponds to an exact record (equivalence class), or whether it partially intersects with one or more records in the ε-differentially private data $\hat{D}$. Note that both *specific* and *generic* queries correspond to *range queries* in the literature. The following is a formal definition of each type of a user count query.

Definition 2 ***(Exact User Count Query.)*** *A user count query u is* exact *if for each predicate $\mathcal{P} = (\hat{A}_i \vdash s_i) \in u$, $s_i \in \Omega(\hat{A}_i)$.* ■

Definition 3 ***(Specific User Count Query.)*** *A user count query u is* specific *if for each predicate $\mathcal{P} = (\hat{A}_i \vdash s_i) \in u$:*
1. If $\hat{A}_i$ is categorical, then $s_i \in \Omega(\hat{A}_i)$.
2. If $\hat{A}_i$ is numerical, then $s_i \in \Omega(\hat{A}_i)$ or there exists exactly one range $r \in \Omega(\hat{A}_i)$ where $s_i \cap r \neq \phi$ and $s_i \neq r$. ■

Definition 4 ***(Generic User Count Query.)*** *A user count query u is* generic *if for each predicate $\mathcal{P} = (\hat{A}_i \vdash s_i) \in u$:*
1. If $\hat{A}_i$ is categorical, then $s_i \in \mathbb{T}^{A_i}$.

2. If $\hat{A}_i$ is numerical, then $\exists\, r_j, r_l \in \Omega(\hat{A}_i)$ such that $s_i \cap r_j \neq \phi$, $s_i \cap r_l \neq \phi$, and $r_j \neq r_l$. ■

Example 1 The following are examples of user count queries over the ε-differentially private data $\hat{D}$ presented in Table 2:
Exact: $u_1 = (\hat{Job} = \text{“}Artist\text{”}) \wedge (\hat{Age} = [45 - 65))$
Specific: $u_2 = (\hat{Job} = \text{“}Artist\text{”}) \wedge (\hat{Age} = [50 - 57))$
Generic: $u_3 = (\hat{Job} = \text{“}Lawyer\text{”}) \wedge (\hat{Age} = [30 - 70))$

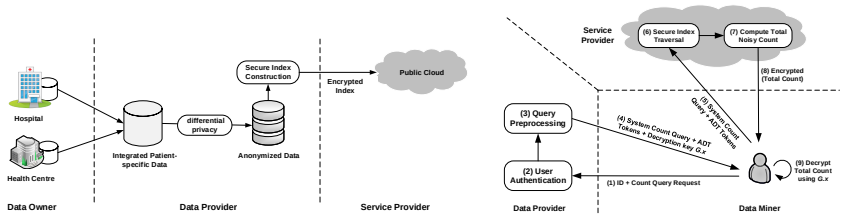
Observe that the queries conform neither to the structure nor to the data in $\hat{D}$. That is, attributes $\hat{Country}$ and $\hat{Salary}$ are missing, the value “Lawyer” is not in the domain $\Omega(\hat{Job})$, and the range $[30, 70]$ spans beyond the values covered by all ranges in $\Omega(\hat{Age})$. All these issues will be addressed in section 4.2.1 when the data miner submits her user count query for preprocessing. ■

3.3 Adversarial Model

SecDM consists of three parties: *data provider* (data bank), *data miner*, and *service provider* (cloud). In our security analysis, the adversary can statically corrupt, in *honest-but-curious (HBC)* [53] fashion, the service provider or the data miner, but not both. The service provider adversary tries to gain access to the contents of the anonymized data, and during query execution tries to infer information about the count queries and their results. On the other hand, the data miner adversary tries to link sensitive information to patients by attempting to gain information about the anonymized records identified by each of her queries, their count values, and the percentage of each query count. We assume the computational power of each adversary is bounded by a polynomial size circuit. We also assume that a protocol is in place to provide secure pair-wise communications between parties in the SecDM framework.

3.4 Problem Statement

Given ε-differentially private data $\hat{D}$, the objective is to design a framework for outsourcing $\hat{D}$ to an untrusted service provider P that can answer exact, specific, and range count queries from authorized data miners on $\hat{D}$. The framework must provide three levels of security: (1) *data confidentiality*, where $\hat{D}$ is stored in an encrypted form such that no useful information can be disclosed from $\hat{D}$ by unauthorized parties; (2) *confidential query processing*, where P is capable of processing the queries on $\hat{D}$ for classification analysis without inferring information about the queries or the underlying anonymized data; and (3) *privacy preservation*, where the result of each query provides a certain privacy guarantee.



(a) Setup phase (b) Query processing phase

Fig. 4 SecDM: A Privacy-preserving framework for confidential count query processing on the cloud.

4 SecDM Framework Solution

4.1 Secure Index Construction

Given the ε-differentially private data $\hat{D}$ with k_c categorical attributes and k_n numerical attributes, the data provider constructs an encrypted index on all attributes in $\hat{D}$ in order to support efficient and secure processing of multi-dimensional range count queries over the k-dimensional data, where $k = k_c + k_n$. That is, it constructs a balanced kd-tree [39] index, where every internal (non-leaf) node is a k-dimensional node that splits the space into two half-spaces, and each leaf node stores a noisy count corresponding to a record in $\hat{D}$.

The kd-tree index is constructed with the procedure *Secure Index Construction* (*buildIndex*) presented in Algorithm 1. *BuildIndex* is a recursive procedure that has four input parameters: $\hat{D}$, depth i, PK, and y. The first input parameter $\hat{D}$ is the set of records for which the kd-tree will be constructed, where each record represents a point in the k-dimensional space. The columns in $\hat{D}$ are *shuffled a priori* to randomize the order of the attributes. The second input parameter i represents the depth of the recursion that determines the split dimension. It ranges between 1 and k, where 1 is the initial value. The third input parameter PK is the public key of the anonymous ciphertext-policy attribute based encryption scheme $\mathbb{A}$, which will be used to secure each internal node in the index tree. To generate this key a security parameter λ is passed to the setup algorithm: $\mathbb{A}.\mathsf{Setup}(1^{\lambda}) \Rightarrow (PK, MSK)$. The last parameter y is the public key of the Exponential ElGamal scheme used to encrypt the noisy counts in the leaf nodes. The function $median$ (Line 5) determines the median value of the domain $\Omega(\hat{A}_i)$, where $\hat{A}_i$ is an attribute from $\hat{D}$. The function $split$ (Line 6) then uses a hyperplane that passes through the median value in order to split $\hat{D}$ into two subsets of records, $\hat{D}_1$ and $\hat{D}_2$. Note that the *median* value is chosen for splitting to ensure a balanced tree where each leaf node is about the same distance from the root of the tree. *BuildIndex* calls itself (Lines 7-8) using $\hat{D}_1$ and $\hat{D}_2$ as inputs in order to determine the left and right children nodes respectively. When the procedure terminates (Line 13), it returns the kd-tree index T. Next, we will discuss how internal nodes (Lines 9-12) and leaf nodes (Line 2) are constructed.

Algorithm 1 *buildIndex*: **Secure Index Construction**

Input: Differentially private data $\hat{D}$, split dimension i, ACP-ABE public key PK, ElGamal public key y
Output: kd-tree index T

1: **if** $|\hat{D}| = 1$ **then**
2: $constLeafNode(\hat{D}, y)$;
3: **return** ;
4: **end if**
5: $cut \leftarrow median(\hat{A}_i, \hat{D})$;
6: $split(\hat{D}, \hat{A}_i, cut, \hat{D}_1, \hat{D}_2)$;
7: $v_{left} \leftarrow buildIndex(\hat{D}_1, (i+1) \bmod k, PK, y)$;
8: $v_{right} \leftarrow buildIndex(\hat{D}_2, (i+1) \bmod k, PK, y)$;
9: create empty node v;
10: $v.split\_dim \leftarrow \hat{A}_i$; $v.split\_value \leftarrow cut$;
11: $v.lc \leftarrow v_{left}$; $v.rc \leftarrow v_{right}$;
12: $v.genCT(PK)$;
13: **return** kd-tree index T;

4.1.1 Internal Nodes Construction

Each internal (non-leaf) node v in the kd-tree index corresponds to one dimension (attribute) $\hat{A}_i \in \hat{D} : 1 \leq i \leq k$ of the k-dimensional space, where the splitting hyperplane is perpendicular to the axis of dimension $\hat{A}_i$, and the splitting value *cut* is determined by the median function (Line 4). Node v has two child nodes, namely, lc and rc, where all records containing values smaller or equal to the *cut* value with regard to $\hat{A}_i$ will appear in the left subtree, whose root is $v.lc$, and all records containing values greater than the *cut* value with regard to $\hat{A}_i$ will appear in the right subtree, whose root is $v.rc$. Furthermore, node v consists of two ciphertexts, $v.CT_{left}$ and $v.CT_{right}$, where the encrypted message in $v.CT_{left}$ is a pointer (Ptr) to the child node $v.lc$, and the encrypted message in $v.CT_{right}$ is a pointer to the child node $v.rc$. The intuition is as follows: The service provider must use the key SK_u provided by the user to be allowed to securely traverse the kd-tree index and compute the answer to the user query u. The structure of SK_u and how it is built is discussed in Section 4.2.1. At any node v in the kd-tree, if SK_u satisfies the access structure of the ciphertext $v.CT_{left}$, the ciphertext is decrypted and a pointer to the child node $v.CT_{left}$ is obtained. Similarly, if SK_u satisfies the access structure of $v.CT_{right}$, the ciphertext is decrypted and a pointer to $v.CT_{right}$ is obtained. If SK_u satisfies both access structures, then two pointers are obtained, indicating that both left and right subtrees must be traversed.

Access Structure. The ciphertexts in each node are generated using the anonymous ciphertext-policy attribute based encryption scheme $\mathbb{A}$, where each ciphertext has an access structure W. Each numerical attribute $\hat{A}_i \in \hat{D}$ is represented in the access structure of a ciphertext by two attributes, $\hat{A}_i^{min}$ and $\hat{A}_i^{max}$, where $\Omega(\hat{A}_i^{min}) = \Omega(\hat{A}_i^{max}) = \Omega(\hat{A}_i)$. On the other hand, each categorical attribute is mapped to one attribute in the access structure of a ciphertext. Below is the formal definition of an access structure of a ACP-ABE ciphertext.

Definition 5 ***(Ciphertext Access Structure*** W***.)*** *Given ε- differentially private data $\hat{D}$ and a node v from the kd-tree index over $\hat{D}$, the access structure of a ciphertext of*

v is the conjunction $W = [W_{\hat{A}_1} \wedge ... \wedge W_{\hat{A}_i} \wedge ... \wedge W_{\hat{A}_k}]$. If $\hat{A}_i$ is a categorical attribute, then $W_{\hat{A}_i}$ corresponds either to the wildcard character " $*$ " or to a disjunction of values from $\Omega(\hat{A}_i)$, where "$W_{\hat{A}_i} = *$" means that attribute $\hat{A}_i$ should be ignored. If $\hat{A}_i$ is a numerical attribute, then $W_{\hat{A}_i} = W_{\hat{A}_i^{min}} \wedge W_{\hat{A}_i^{max}}$, where $W_{\hat{A}_i^{min}}$ and $W_{\hat{A}_i^{max}}$ each corresponds to either the wildcard character " $*$ " or a disjunction of values from $\Omega(\hat{A}_i)$. ■

Note that for a given node v, the access structure of the left and right ciphertexts is mainly concerned with the splitting dimension $v.split\_dim$, and the split value $v.split\_value$ over $\hat{D}_1$ or $\hat{D}_2$, where $\hat{D}_1, \hat{D}_2 \subseteq \hat{D}$. If $v.split\_dim$ is a categorical attribute $\hat{A}_i$, then $W_{\hat{A}_i}$ in the access structure of $v.CT_{left}$ should correspond to the disjunction of all values $val \in \{\Omega(\hat{A}_i) \cup \{1\}\}$ such that $val \leq v.split\_value$ and for $val = 1$, where 1 represents "Any" value. Similarly, $W_{\hat{A}_i}$ in the access structure of $v.CT_{right}$ should correspond to the disjunction of all values $val \in \{\Omega(\hat{A}_i) \cup \{1\}\}$ such that $val > v.split\_value$ or for $val = 1$. On the other hand, if $v.split\_dim$ is a numerical attribute $\hat{A}_i$, then $W_{\hat{A}_i^{min}}$ in the access structure of $v.CT_{left}$ should correspond to the disjunction of all values $val \in \Omega(\hat{A}_i)$ for all $val \leq v.split\_value$, and $W_{\hat{A}_i^{max}}$ in the access structure of $v.CT_{right}$ should correspond to the disjunction of all values $val \in \Omega(\hat{A}_i)$ such that $val > v.split\_value$. Regardless of whether $\hat{A}_i$ is categorical or numerical, all values in $\{W \setminus W_{\hat{A}_i}\}$ should correspond to " $*$ ".

Example 2 Given $\hat{D}$ from Table 3, and given node v from kd-tree index:

a) If $v.split\_dim = Job$ (categorical) and $v.split\_value = 2$, then the access structure of the left and right ciphertexts can be represented as follows:
$W_L = (Country = *) \wedge (Job = 1 \vee Job = 2) \wedge (Age_{min} = *) \wedge (Age_{max} = *) \wedge (Salary_{min} = *) \wedge (Salary_{max} = *)$.
$W_R = (Country = *) \wedge (Job = 1 \vee Job = 3) \wedge (Age_{min} = *) \wedge (Age_{max} = *) \wedge (Salary_{min} = *) \wedge (Salary_{max} = *)$.

b) If $v.split\_dim = Age$ (numerical) and $v.split\_value = 1$, then the access structure of the left and right ciphertexts are:
$W_L = (Country = *) \wedge (Job = *) \wedge (Age_{min} = 1) \wedge (Age_{max} = *) \wedge (Salary_{min} = *) \wedge (Salary_{max} = *)$.
$W_R = (Country = *) \wedge (Job = *) \wedge (Age_{min} = *) \wedge (Age_{max} = 2) \wedge (Salary_{min} = *) \wedge (Salary_{max} = *)$. ■

In procedure *buildIndex* presented in Algorithm 1, each internal node v is created after determining its children nodes V_{left} and v_{right} (Lines 9-12), where function $genCT$ is responsible for creating the left ciphertext CT_{left} and the right ciphertext CT_{right} of the node by calling twice the ACP-ABE algorithm $\mathbb{A}$.Enc() and passing as parameters the public key *PK* of $\mathbb{A}$, a pointer to the child node to be encrypted, and the values in the access structure (without the attribute names):
$v.CT_{left} \leftarrow \mathbb{A}.\mathsf{Enc}(PK, Ptr(v.lc), W_L)$;
$v.CT_{right} \leftarrow \mathbb{A}.\mathsf{Enc}(PK, Ptr(v.rc), W_R)$;

For each attribute $\hat{A}_i$ in W that is assigned a wildcard, e.g. $(Country = *)$, $\mathbb{A}$.Enc() generates a random (*mal-formed*) group elements $[C_{i,j,1}, C_{i,j,2}]$ for each

Algorithm 2 *constLeafNode*: **Leaf Node Construction**

Input: ε-differentially private record R, Exponential ElGamal's public key y
Output: leaf node l

1: create empty node l;
2: $l.NCount \leftarrow \mathbb{G}.\mathsf{Enc}(R.NCount, y, r)$;
3: **for each** numerical attribute $\hat{A}_i \in \hat{D}$ **do**
4: $\quad l \leftarrow genTAG(R.\hat{A}_i)$;
5: **end for**
6: **return** l;

Algorithm 3 *indexUpload*: k**d-Tree Index Upload**

1: The Data Provider submits the kd-tree index T to the Service Provider;
2: The Service Provider receives T;

value in $\Omega(\hat{A}_i)$. On the other hand, for each attribute in W assigned specific values, e.g. $(Job = 1 \vee Job = 2)$, $\mathbb{A}.\mathsf{Enc}()$ generates a *well-formed* group elements for each value specified, i.e. for value 1 and for value 2, and random group elements for each remaining value in $\Omega(Job)$. As a result, all ciphertexts CT generated by $\mathbb{A}.\mathsf{Enc}()$ in the kd-tree index contain the same number of group elements regardless of the access structure.

4.1.2 Leaf Nodes Construction

In procedure *buildIndex*, as the multi-dimensional space is being recursively partitioned a leaf node is created whenever the number of the records being partitioned reaches 1 (Lines 1-2). Procedure *Leaf Node Construction* (*constLeafNode*), presented in Algorithm 2, is responsible for generating the leaf nodes. It takes as input a ε-differentially private record R and Exponential ElGamal's public key y and outputs a leaf node l. After creating an empty node l (Line 1), the noisy count of record R is encrypted using Exponential ElGamal encryption scheme $\mathbb{G}$ and stored in node l (Line 2). We choose the Exponential ElGamal cryptosystem due to its additive homomorphism property, which allows for homomorphically adding encrypted noisy counts together in an efficient way.

For each numerical range value $R.\hat{A}_i$ in R, a hiding commitment function $genTAG()$ is utilized to commit $R.\hat{A}_i$ and randomly generate a unique tag (Line 3-4). Applying $genTAG()$ to the same value using the same randomness always generates the same tag; however, the correspondence between each tag and its value is kept secret. As we will see in Section 4.2, using a deterministic function to generate tags for the numerical range values enables the service provider during query execution to compute the exact percentage of the noisy count of each reported leaf node with respect to the query being processed.

Once the kd-tree index T has been created, it is submitted to the service provider according to Algorithm 3. While Algorithm 3 is trivial, it is required in Section A to prove by simulation the security of the framework.

4.2 Confidential Query Processing

In this section, we illustrate how user count queries are executed in order to determine their exact ε-differentially private answers while preserving the confidentiality of the data as well as the queries. First, we explain how the user query is preprocessed and transformed into a system query. Next, we discuss how the service provider securely traverses the kd-tree index, computes the total count, and then sends the result back to the user.

4.2.1 Query Preprocessing

Upon the receipt of a user's count query u, the data provider first transforms u into a conjunction of subqueries that specify a single-value equality condition over each attribute $\hat{A}_i \in \hat{D}$. Next, it generates a *system count query* SK_u using algorithm $\mathbb{A}.\mathsf{KeyGen}()$ from ACP-ABE scheme. If an attribute $\hat{A}_i$ in $\hat{D}$ is not specified by the user in u, then it will be considered in SK_u as if the user is asking for $(\hat{A}_i = *)$. The following is the formal definition of a system count query:

Definition 6 ***(System Count Query.)*** Given ε-differentially private data $\hat{D}$ with k attributes and a user query $u = \mathcal{P}_1 \wedge ... \wedge \mathcal{P}_m \,|\, \mathcal{P} = (\hat{A}_i \vdash s_i)$, a system count query over $\hat{D}$ is a ACP-ABE user's secret key SK_u representing k subqueries $\{q_{\hat{A}_1}, ..., q_{\hat{A}_k}\}$ such that:

- If $\hat{A}_i$ is a categorical attribute, then $\hat{A}_i$ is represented in SK_u as a tuple of group elements $[D_{i,0}, D_{i,1}, D_{i,2}]$
- If $\hat{A}_i$ is a numerical attribute, however, it is represented in SK_u as two tuples of group elements $[D_{i,0}^{min}, D_{i,1}^{min}, D_{i,2}^{min}]$ and $[D_{i,0}^{max}, D_{i,1}^{max}, D_{i,2}^{max}]$, where each tuple corresponds to the minimum and maximum bound of the range subquery $q_{\hat{A}_i}$, respectively. ■

The total number of group element tuples in a system query SK_u is: $|SK_u| = k_c + 2 \times k_n$, where k_c and k_n are the number of categorical and numerical attributes in $\hat{D}$. $|SK_u|$ is independent of the user query u. We refer the reader to Section **??** for more details on how an ACP-ABE secret key is generated.

Procedure *Query Preprocessing (qPreprocess)* presented in Algorithm 4 illustrates how a system count query SK_u is constructed based on a user's count query u. Once the user has been authenticated successfully using user identification token UIT (Line 2), the next step is to determine the attribute-value pairs in SK_u. For each *categorical* attribute $\hat{A}_i \in \hat{D}$, if predicate $(\hat{A}_i, \vdash, s_i)$ exists in the user count query u and s_i is in the domain of $\hat{A}_i$, then the subquery $(\hat{A}_i, s_i.ID)$ is added to q, where $s_i.ID$ is the identifier of the categorical value s_i in $\hat{A}_i$'s taxonomy tree $\mathbb{T}^{\hat{A}_i}$ (Lines 5-6); otherwise, if s_i is not in the domain of $\hat{A}_i$, then function $findSCS(s_i)$ (Line 8) is utilized to determine the position of s_i in $\hat{A}_i$'s taxonomy tree with regard to the solution cut. If s_i is below the solution cut, then there exists exactly one node n on the path from s_i to the root, such that $n \in \Omega(\hat{A}_i)$. We call such a node the *Solution Cut Subsumer (SCS)* of s_i, and the subquery $(\hat{A}_i, n.ID)$ is then added to q (Line 9). If s_i is above the solution cut or u does not have any predicate that corresponds to a

Algorithm 4 *qPreprocess*: **Query Preprocessing**

Input: ε-differentially private data $\hat{D}$
Output: system count query SK_u, set of attribute distribution tokens $\mathcal{N}$

1: The Data Provider receives user identification token UIT and user count query u from Data Miner
2: **if** user authentication is successful using UIT **then**
3: $q \leftarrow \{\}$; $ADT \leftarrow \{\}$;
4: **for each** categorical attribute $\hat{A}_i \in \hat{D}$ **do**
5: **if** $(\hat{A}_i, \vdash, s_i) \in u$ and $s_i \in \Omega(\hat{A}_i)$ **then**
6: $q \leftarrow q \cup (\hat{A}_i, s_i.ID)$;
7: **else if** $(\hat{A}_i, \vdash, s_i) \in u$ and $s_i \notin \Omega(\hat{A}_i)$ **then**
8: $n \leftarrow findSCS(s_i)$;
9: $q \leftarrow q \cup (\hat{A}_i, n.ID)$;
10: **else if** $(\hat{A}_i, \vdash, s_i) \notin u$ **then**
11: $q \leftarrow q \cup (\hat{A}_i, 1)$;
12: **end if**
13: **end for**
14: **for each** numerical attribute $\hat{A}_i \in \hat{D}$ **do**
15: **if** $(\hat{A}_i, \vdash, s_i) \in u$ **then**
16: $(v_{i,1}, v_{i,2}) \leftarrow compMinMax(\Omega(\hat{A}_i), s_i, \vdash_i)$;
17: $q \leftarrow q \cup (\hat{A}_i^{min}, v_{i,1}) \cup (\hat{A}_i^{max}, v_{i,2})$;
18: $\mathcal{N} \leftarrow \mathcal{N} \cup genADT(\Omega(\hat{A}_i), v_{i1}, v_{i2})$;
19: **else**
20: $q \leftarrow q \cup (\hat{A}_i^{min}, 1) \cup (\hat{A}_i^{max}, range_{max})$;
21: **end if**
22: **end for**
23: $SK_u \leftarrow \mathbb{A}.\mathsf{KeyGen}(MSK, q)$;
24: **return** $SK_u, \mathcal{N}$;
25: **end if**

Algorithm 5 *queryRequest*: **System Count Query Request**

1: The Data Provider sends the following to the Data Miner:
- System count query SK_u corresponding to user count query u
- Set of attribute distribution tokens $\mathcal{N}$
- Exponential ElGamal decryption key x

2: The Data Miner receives SK_u, $\mathcal{N}$, and x from Data Provider;
3: The Data Miner sends SK_u and $\mathcal{N}$ to Service Provider;
4: The Service Provider receives SK_u and $\mathcal{N}$ from Data Miner;

categorical attribute $\hat{A}_i \in \hat{D}$, then the subquery $(\hat{A}_i, 1)$ is added to q (Lines 10-11), where 1 means "ANY" value of $\hat{A}_i$ corresponding to the root node of $\hat{A}_i$'s taxonomy tree $\mathbb{T}^{\hat{A}_i}$.

On the other hand, if $\hat{A}_i$ is a *numerical* attribute and predicate $(\hat{A}_i, \vdash, s_i)$ exists in u, then the values $v_{i,1}$ associated with $\hat{A}_i^{min}$ and $v_{i,2}$ associated with $\hat{A}_i^{max}$ are determined by the function *compMinMax* (Line 15). When $\vdash$ is the equal operator $(=)$, if s_i is a single value, then $v_{i,1} = v_{i,2} = Range(s_i)$, where $Range(s_i)$ is a function that returns the identifier of the range in $\Omega(\hat{A}_i)$ containing s_i; otherwise, if s_i is a range, then $v_{i,1} = Range(Lowerbound(s_i))$ and $v_{i,2} = Range(Upperbound(s_i))$. If $\vdash$ is equal to " $\geq$ ", then $v_{i,1}$ = $Range(s_i)$ and $v_{i,2}$ is the identifier of the highest range in $\Omega(\hat{A}_i)$. Conversely, if $\vdash$ is equal to " $\leq$ ", then $v_{i,1} = 1$ and $v_{i,2}$ = $Range(s_i)$. If predicate $(\hat{A}_i, \vdash, s_i)$ does not exist in u for numerical attribute $\hat{A}_i$

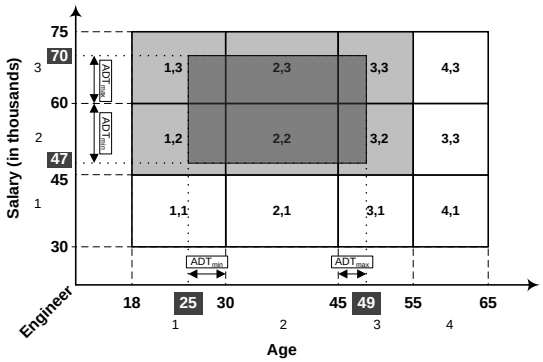


Fig. 5 ADT query overlap.

(Lines 19-20), then $v_{i,1} = 1$ and $v_{i,2}$ is the identifier of the highest range $range_{max} \in \Omega(\hat{A}_i)$.

Example 3 Given Table 2 and Table 3, the following are three different users' queries and their corresponding subqueries in the access structure of the system count query:
a) $u = (Age = 50) \Rightarrow q = (Age^{min}, 2), (Age^{max}, 2)$.
b) $u = (Age = [40 - 70]) \Rightarrow q = (Age^{min}, 1), (Age^{max}, 2)$.
c) $u = (Age \leq 35) \Rightarrow q = (Age^{min}, 1), (Age^{max}, 1)$. ■

Function *genADT* (Line 18) is used to generate *attribute distribution tokens (ADT)* for each numerical attribute $\hat{A}_i$ from $\hat{D}$. Two *ADT* tokens, ADT_{min} and ADT_{max}, are created for each numerical attribute for the purpose of computing the percentages of the noisy counts of the reported leaf nodes upon query execution in order to determine the final answer (total count) of the query. Each *ADT* token consists of two parts: *tag* and *value*. Assuming that *r* is the range for which the *ADT* token is constructed, then $ADT.tag = genTAG(r)$ and $ADT.value$ is the percentage of the partial overlap between query u and range r.

Example 4 Assume that in ε-differentially private data $\hat{D}$, $\Omega(\hat{Age}) = \{[18{-}30), [30{-}45), [45-55), [55-65)\}$, $\Omega(\hat{Salary}) = \{[30-45), [45-60), [60-70)\}$, and user count query u = $(Country =$ "US"$) \wedge (Job =$ "$Engineer$"$) \wedge (Age = [25-49]) \wedge (Salary = [47-70])$. Figure 5 illustrates the equivalence classes of all records (numbered from 1,1 to 4,3), the query u (dark gray rectangle), and the set of leaf nodes identified by u (six light gray rectangles). The range $Age = [25-49]$ spans over three ranges: $[18-30), [30-45)$, and $[45-55)$. Since $[25-49]$ fully spans over $[30-45)$, no *ADT* token is required for $[30-45)$. However, since $[25-49]$ partially overlaps with ranges $[18-30)$ and $[45-55)$, ADT_{min} and ADT_{max} should be created. For range value $[18-30)$, $ADT_{min}.tag = genTAG([18-30))$ and $ADT_{min}.value = \frac{30-25}{30-18} = 42\%$. Similarly, for range value $[45{-}55]$, $ADT_{max}.tag = genTAG([45{-}55))$ and $ADT_{max}.value = \frac{50-45}{55-45} = 50\%$. On the other hand, $Salary = [47-70]$ partially overlaps with ranges $[45-60)$ and $[60-75)$ and ADT_{min} and ADT_{max} must be created. For range value $[45{-}60)$, $ADT_{min}.tag = genTAG([45{-}60))$ and $ADT_{min}.value = \frac{60-47}{60-45} = 87\%$. Similarly, for range value $[60{-}75]$, $ADT_{max}.tag = genTAG([60-75))$ and $ADT_{max}.\ value = \frac{70-60}{75-60} = 67\%$. ■

Algorithm 6 *traverseIndex*: k**d-tree Index Traversal**

Input: kd-tree index root node v, system count query SK_u
Output: set of leaf nodes $\mathcal{R}$

1: **if** v is a leaf node **then**
2: **return** v;
3: **else**
4: **if** $\mathbb{A}.\mathsf{Dec}(v.CT_{left}, SK_u)$ **then**
5: $\mathcal{R} \leftarrow \mathcal{R} \cup traverseIndex(v.lc, SK_u)$;
6: **end if**
7: **if** $\mathbb{A}.\mathsf{Dec}(v.CT_{right}, SK_u)$ **then**
8: $\mathcal{R} \leftarrow \mathcal{R} \cup traverseIndex(v.rc, SK_u)$;
9: **end if**
10: **end if**
11: **return** $\mathcal{R}$;

Once the set of attribute-value pairs q have been determined, the system count query SK_u is then generated by encrypting q with ACP-ABE master secret key ***MSK*** using algorithm $\mathbb{A}.\mathsf{KeyGen}$ (Line 23). Next, the data provider sends the following back to the user: secret key SK_u, the set of ***ADT*** tokens $\mathcal{N}$, and ElGamal decryption key $\mathbb{G}.x$ that will be used eventually to decrypt the final result of the query.

4.2.2 kd-tree Index Traversal

To execute a query u on $\hat{D}$, the data miner sends to the service provider a system count query SK_u and a set of ***ADT*** tokens $\mathcal{N}$. The service provider uses the secret key SK_u to securely traverse the kd-tree index and identify the set of leaf nodes satisfying u, while it uses $\mathcal{N}$ to adjust the noisy count of each identified leaf node in order to compute an accurate final answer to the query.

Procedure *kd-tree Index Traversal (traverseIndex)* presented in Algorithm 6 illustrates how the tree is traversed recursively to answer queries. It takes two input parameters: the root node v of the kd-tree index and a system count query SK_u. If v is an internal node, then the algorithm attempts to decrypt the left ciphertext $v.CT_{left}$ and the right ciphertext $v.CT_{right}$ by separately applying the decryption function Dec from $\mathbb{A}$, with the decryption key SK_u, in order to determine whether it needs to traverse the left subtree, right subtree, or both. If the values of the attributes associated with SK_u satisfy the access structure of $v.CT_{left}$, then the decryption of $v.CT_{left}$ is successful and the procedure $traverseIndex$ calls itself while passing the left child node $v.lc$ as input parameter (Line 4-5). Similarly, if the values of the attributes associated with SK_u satisfy the access structure of $v.CT_{right}$, then the decryption is successful and the procedure $traverseIndex$ calls itself while passing the right child node $v.rc$ as input parameter (Line 7-8). When the algorithm reaches a leaf node v, then v is returned (Lines 1-2). Procedure ***traverseIndex*** eventually returns the set $\mathcal{R}$ containing all leaf nodes satisfying SK_u (Line 11).

Example 5 Given Example 4, assume that v is the root node where $v.split\_dim = Age$ $(\hat{A}_3)$ and $v.split\_value = 2$ $(range[30-45))$. Figure 6.(a) illustrates the access structure of $v.CT_{left}$ and $v.CT_{right}$. Figure 6.(b) shows the system count query (secret key) SK_u that was generated from the user query u such that $Age = [50-60]$ equates to $\hat{A}_3^{min} = 3$ and $\hat{A}_3^{max} = 4$.

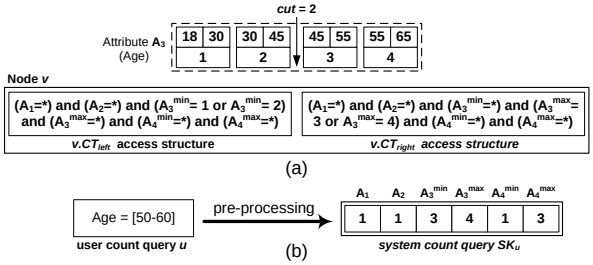


Fig. 6 (a) Access structure of root node v. (b) Generating system count query SK_u from user count query u.

Algorithm 7 *compTCount*: **Total Noisy Count Computation**

Input: set of leaf nodes $\mathcal{R}$, set of attribute distribution tokens $\mathcal{N}$
Output: ciphertext of total count $\langle r, s\rangle$

1: $\langle r, s\rangle \leftarrow \langle 1, 1\rangle$ // initialization
2: **for each** leaf node $l_j \in \mathcal{R}$ **do**
3: $\quad\langle r_j, s_j\rangle \leftarrow l_j.NCount$;
4: $\quad$**for each** token $ADT_i \in \mathcal{N}$ **do**
5: $\quad\quad$**if** $ADT_i.tag \in l_j$ **then**
6: $\quad\quad\quad\langle r_j, s_j\rangle \leftarrow \langle r_j^{ADT_i.value}, s_j^{ADT_i.value}\rangle$; // scalar multiplication
7: $\quad\quad$**end if**
8: $\quad$**end for**
9: $\quad\langle r, s\rangle \leftarrow \langle r.r_j, s.s_j\rangle$; // homomorphic addition
10: **end for**
11: **return** $\langle r, s\rangle$;

Since $\hat{A}_3^{min} = 3$ from SK_u is not in the access structure of $v.CT_{left}$, then the decryption is unsuccessful, and the left subtree will not be traversed. However, $\hat{A}_3^{max} = 4$ from SK_u is in the access structure of $v.CT_{right}$, then the decryption is successful and the procedure $traverseIndex$ traverses the right subtree, whose root node is $v.rc$. ■

4.2.3 Computing Total Noisy Count

Having identified the set of leaf nodes $\mathcal{R}$ satisfying user count query u, the next step is to compute the final answer to the count query.

Procedure *Total Count Computation (compTCount)* presented in Algorithm 7 illustrates how the total noisy count is computed. It takes as input a set of leaf nodes $\mathcal{R}$ and a set of attribute distribution tokens $\mathcal{N}$. For each leaf node l_j, if there is an ADT token ADT_i whose tag matches any of the tags in l_j, then a percentage of the encrypted noisy count $\langle r_j, s_j\rangle$ is computed by raising r_j and s_j to the value associated with ADT_i (Lines 5-6). To homomorphically add two noisy counts together, their first ciphertexts are multiplied together, and the same is done for their second ciphertexts (Line 9). The output of procedure *compTCount* is the encrypted total count $\langle r, s\rangle$ (Line 11).

Algorithm 8 *queryResult*: **User Count Query Result**

Input: Exponential ElGamal decryption key x
Output: Query result ciphertext of total count $\langle r, s \rangle$

1: The Data Miner receives ElGamal encrypted result $\langle r, s \rangle$ from Service Provider;
2: $res_u = \mathbb{G}.\mathsf{Dec}(\langle r, s \rangle, x)$; // Total noisy count decryption
3: **return** res_u;

4.2.4 Computing Query Result

Once ciphertext $\langle r, s \rangle$ has been computed, the service provider returns the ciphertext to the user as the final result. As per Algorithm 8, when the data miner receives the encrypted result $\langle r, s \rangle$, she uses Exponential ElGamal's private key $\mathbb{G}.x$ to decrypt the ciphertext and determine the exact noisy count res_u such that res_u satisfies differential privacy.

4.3 Discussion

4.3.1 Benefits of Outsourcing

To analyze the benefit to the data provider for outsourcing the data to a service provider, we measured the processing overhead of *specific count queries* on the data provider and the service provider when the number of queries ranges from 200 to 1000. We choose *specific count queries* to perform the experiment because they represent the worst-case scenario, where the number of nodes traversed in the kd-tree index is minimized and the number of ADT tokens is maximized. Figure 7 illustrates the results of our experiment, where we observe that the processing overhead on the proxy server is almost 10 times less than the overhead on the service provider side, regardless of the number of the queries. That is, by oursourcing the data, the data provider offloads over 90% of the query processing (computation) to the service provider.

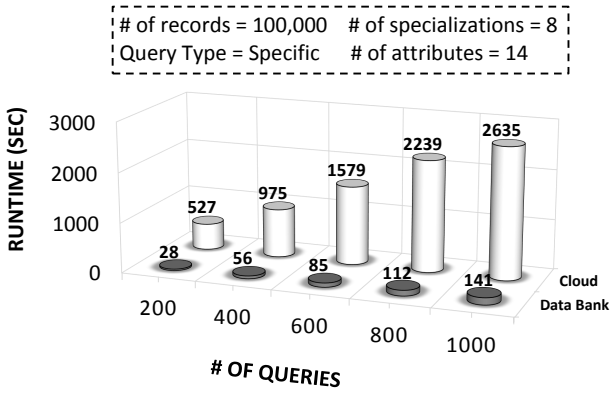


Fig. 7 Performance comparison w.r.t. # of queries.

4.3.2 Centralized SecDM

Our proposed solution SecDM in Section 4 allows data miners to reuse previously generated system queries and eliminates the need to interact with the data provider to generate the same ones again. However, this comes at the expense of requiring the user to interact with two parties (the data provider and the service provider), and to perform public key decryption operations on the results encrypted using Exponential ElGamal. In some scenarios where query reusability is not required, our framework can be easily modified to have all communications go through the data provider, as in the *Centralized SecDM (*C-SecDM*)* framework illustrated in Figure 8. Observe that in C-SecDM, the data miner does not have access to Exponential ElGamal's decryption key $\mathbb{G}.x$, as the decryption is performed by the data provider, and the total count result is then sent in clear text to the data miner via a secure channel.

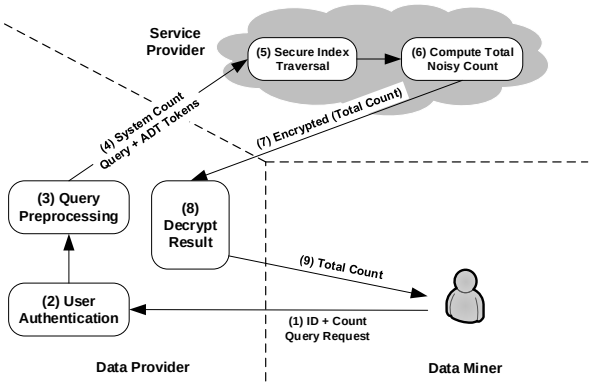


Fig. 8 Centralized SecDM (**C-SecDM**) framework.

4.3.3 Security Tradeoff

To present a practical framework we choose Exponential ElGamal to encrypt the noisy counts because this encryption scheme supports efficient homomorphic addition and integer multiplication operations. These operations are utilized by the cloud to adjust the noisy count of each identified leaf node in the kd-tree index tree using ADT tokens and then to compute the total count. However, in each ADT token, $ADT.value$ must be stored in clear text, which reveals the percentage each noisy count should be multiplied by, without reveling the actual value of the noisy count or its adjusted value. Rather than using Exponential ElGamal, we could have used other encryption schemes that support multiple homomorphic additions and multiplications. However, such schemes are inefficient and will render our solution impractical.

4.3.4 Multiple Data Release

Shabtai *et al.* [54] and Shmueli *et al.* [55] indicate that the anonymization approach should be chosen carefully in a multiple-release outsourcing scenario since it normally differs from the one used in a single-release outsourcing scenario. However, this is out of the scope of this paper.

5 Solution Analysis

5.1 Complexity Analysis

Proposition 1 *The runtime complexity for constructing a kd-tree index from a differentially private data with d equivalent classes and k attributes using Algorithm 1 and Algorithm 2 is bounded by $\mathcal{O}(k \times d \times \log d)$ operations.*

Proof Constructing a kd-tree with d points (equivalent classes) requires $\mathcal{O}(d \times \log d)$ [56]. Each node consists of two ciphertexts, each of which requires $\mathcal{O}(k_c + 2 \times k_n) = \mathcal{O}(k)$, where k_c and k_n are number of categorical attributes and numerical attributes respectively. Therefore, the required number of operations is $\mathcal{O}(k \times d \times \log d)$.

Proposition 2 *The runtime complexity for executing a system query SK_u over a kd-tree index with d leaf nodes using Algorithm 6 and Algorithm 7 is bounded by $\mathcal{O}(\sqrt{d} + r \times k)$ operations, where $r = |\mathcal{R}|$ and $\mathcal{R}$ is the set of reported nodes.*

Proof Since SK_u is an axis-parallel rectangular range query, the time required to traverse a kd-tree and report the points (equivalent classes) stored in its leaves is $\mathcal{O}(\sqrt{d} + r)$ [56]. For each *reported* leaf node, $\mathcal{O}(2 \times k_c) = \mathcal{O}(k)$ time is required to compute the total noisy count. As a result, the number of operations required to traverse the tree and answer SK_u is $\mathcal{O}(\sqrt{d} + r \times k)$.

Proposition 3 *The communication complexity for answering a count query is bounded by $\mathcal{O}(k \times C)$.*

Proof For each query, only 4 messages are generated. The data miner sends two messages: one to the data provider and another to the service provider, and receives two messages: one from the data provider and another from the service provider. Two of the messages are one Elgamal ciphertext each, while each of the other two consists of $2 \times k$ Elgamal ciphertexts (ADT tokens). Hence, the overall communication complexity is $\mathcal{O}(k \times C)$, where $C = \lceil \lg p \rceil$ is the bit length of Elgamal group element, and p is at least 2048 bits.

Proposition 4 *The noisy count answers satisfy ε-differential privacy.*

Proof Generating a differentially private table involves three steps: selecting a candidate for specialization, determining the split value, and publishing the noisy counts. In the following, we will We show that each of these steps preserves differential privacy. We will also use the composition properties of differential privacy to show that the output is differentially private.

To select a candidate for specialization, we first need to compute the utility score (information gain) of each candidate $v \in \cup Cut(\mathbb{T})$:

$$IG(D, v) = H_v(D) - H_{v|c}(D),$$

where $H_v(D)$ is the entropy of candidate v with respect to the class attribute A^{cls}, and $H_{v|c}(D)$ is the conditional entropy given the candidate is specialized. Having computed the score of each candidate, then exponential mechanism is used to select a candidate v_i in a differentially-private manner with the following probability:

$$Select(v_i) = \frac{exp(\frac{\varepsilon}{2\Delta IG} IG(D, v_i))}{\sum_v exp(\frac{\varepsilon}{2\Delta IG} IG(D, v))},$$

where $\Delta IG = \lg |\Omega(A^{cls})|$ is the sensitivity of the *information gain* function.

Determining the split value is also satisfies differential privacy. That is because for *categorical* attributes, taxonomy trees are used. Since the taxonomy tree is fixed, the sensitivity of the split value is 0. Therefore, splitting the records according to the taxonomy tree does not violate differential privacy. As for numerical attributes, the domain is split into s intervals: $I = \{I_1, \ldots, I_s\}$ and then exponential mechanism is used to choose an interval $I_j \in I$ with probability:

$$Select(I_j) = \frac{exp(\frac{\varepsilon}{2\Delta IG} IG(D, v_j)) \times |\Omega(I_j)|}{\sum_{i=1}^{s} (exp(\frac{\varepsilon}{2\Delta IG} IG(D, v_i)) \times |\Omega(I_i)|},$$

where the length of the interval I_j is denoted by $|\Omega(I_j)|$. Choosing a split value from the interval I_j also satisfies ε-differential privacy because the probability of choosing any split value is proportional to $exp(\frac{\varepsilon}{2\Delta IG} IG(D, v_j))$. As for publishing the noisy counts, a $Lap(2/\varepsilon)$ is added to satisfy ε-differential privacy, since the sensitivity of count queries is 1.

Theorem 51 ***Sequential Composition [57].*** *Let each Ag_i provide ε-differential privacy. A sequence of $Ag_i(D)$ over the data set D provides ($\Sigma_i \varepsilon_i$)-dierential privacy.*

Theorem 52 ***Parallel Composition [57].*** *Let each Ag_i provide ε-differential privacy. A sequence of $Ag_i(D_i)$ over a set of disjoint data sets D_i provides ($\Sigma_i \varepsilon_i$)-dierential privacy.*

Based on Theorems 51 and 52, the output data table satisfies ε-differential privacy. Moreover, according to [52], any post-processing on a differentially private data does not violate its privacy. Hence, the computed noisy count answers based on the table also satisfy ε-differential privacy.

6 Performance Evaluation

In this section we evaluate the performance of the SecDM framework. First, we discuss the implementation details, and then we present the experimental results that include data utility, solution construction scalability, the scalability of query processing with respect to the number of records, and the efficiency with respect to the size of the queries.

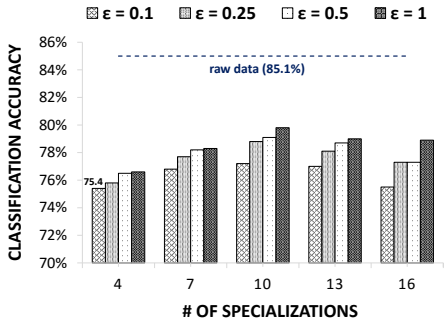


Fig. 9 Classification accuracy of count query results w.r.t. privacy budget ε.

6.1 Implementation and Setup

The SecDM framework is implemented in C++. Experiments were conducted on a machine equipped with an Intel Core i7 3.8GHz CPU and 16GB RAM, running 64-bit Windows 7. The index tree is implemented according to the kd-tree description in [56]. Both of the cryptographic primitives, *ACP-ABE* and *Exponential ElGamal*, were implemented using MIRACL[3], an open source library for big number and elliptic curve cryptography. To implement ACP-ABE, we chose Boneh-Lynn-Shacham (BLS) pairing-friendly curve from [58]: $Y^2 = X^3 + b$, where $b = \sqrt{w + \sqrt{m}}$, $m = \{-1, -2\}$, and $w = \{0, 1, 2\}$. The chosen elliptic curve has a pairing embedding degree of 24 and a AES security level of 256. The pairing $e : G_1 \times G_2 \rightarrow G_T$ is a type 3 pairing where G_1 is a point over the base field, G_2 is a point over an extension field of degree 3, and G_T is a finite field point over the k-th extension, where $k = 24$ is the embedding degree for the BLS curve. To implement *Exponential ElGamal* we randomly choose the message space and calculation modulus p to be a large 2048-bit prime for which $q = (p-1)/\alpha$ is a 256-bit prime. Since *Exponential ElGamal* depends on the multiplicative order of g and having a large collection of ciphertexts, we choose g to be a generator of the multiplicative subgroup $\mathbb{G}_q$ such that $order(g) = q - 1$.

We utilize a real-life *adult* data set [59] in our experiments to illustrate the performance of SecDM framework. The adult data set consists of 45,222 census records containing six numerical attributes, eight categorical attributes, and a *class* attribute with two levels: "$\leq 50K$" and "$> 50K$". A further description of the attributes can be found in [60]. Since the maximum number of attributes is 14, we assume that the number of attributes in a query can range from 2 to 14, and the average number of attributes in a query is 8.

[3] MIRACL: https://certivox.org/display/EXT/MIRACL

6.2 Experimental Results

6.2.1 Data Utility

Blum *et al.* [8] show that count queries are very useful for performing statistical analysis and extracting patterns and trends from the data. We analyze the data utility for count queries by measuring the classification accuracy of the count query results for several values of the privacy budget ε. We generate ε-differentially private records using *DiffGen* algorithm, where the number of specializations is set to 4, 7, 11, 13 and 16, and the utility function $InfoGain(D, v)$ is chosen to determine the score of each candidate v during the specialization process. We utilize C4.5 classifier [61] to measure the classification accuracy of both the raw data and the results of count queries. In each case, we use two-third (2/3) of the records to build (train) the classifier, and one-third (1/3) of the records for testing. Applying C4.5 on the raw data yields a classifier with 85.1% classification accuracy. Figure 9 illustrates the classification accuracy of the count query results for $\varepsilon = 0.1, 0.25, 0.5$ and 1. We observe that our proposed solution maintains high level of data utility as the biggest drop in classification accuracy when compared with the accuracy of raw data is $85.1\% - 75.4\% = 9.7\%$. We also observe that the utility is directly affected by the the privacy budget ε. That is, the more privacy budget is allocated, the higher the classification accuracy is. This trend is due to the fact that a higher privacy budget leads to a more accurate partitioning, and less noise is added to the count of each equivalent class at the leaf level. Our findings are consistent with [22]. Our algorithm has a major advantage over other algorithms, e.g. [62][60], because data miners have better flexibility to perform the required data analysis. For example, the algorithm in [62] allows for interactive queries to build a classifier. That means the data has to be permanently shut down after certain number of queries, which prevents the data miner from building another classifier based on the same data.

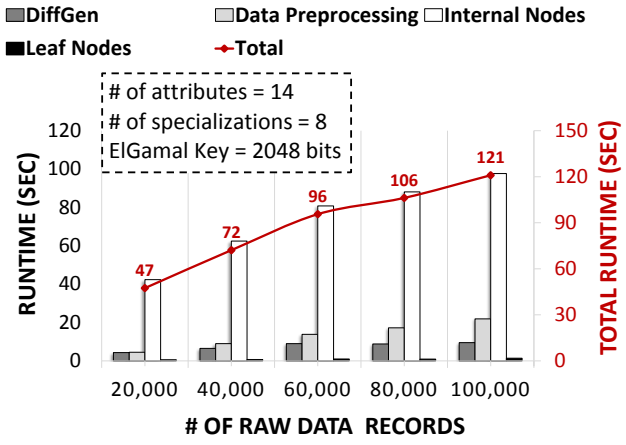
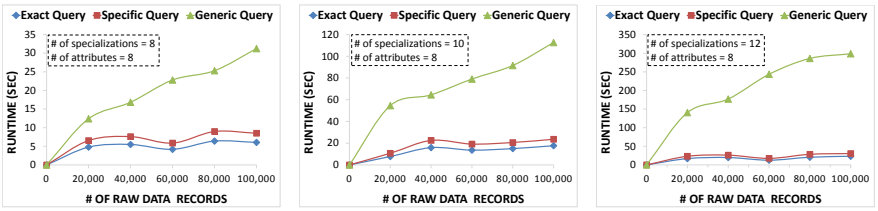


Fig. 10 Scalability of framework construction w.r.t. # of records.



(a) # of specializations = 8 (b) # of specializations = 10 (c) # of specializations = 12

Fig. 11 Scalability of query processing w.r.t. the number of raw data records and the number of specializations.

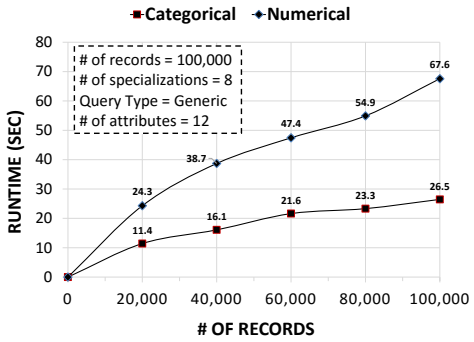


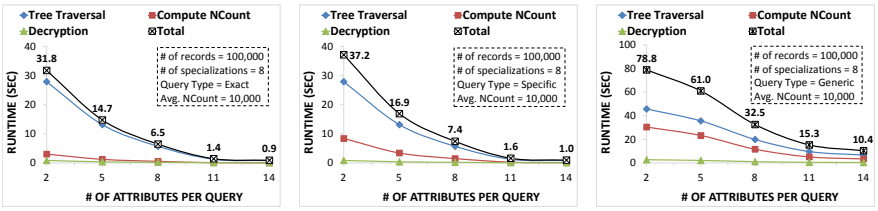
Fig. 12 Performance of categorical and numerical attributes w.r.t. # of records.

6.2.2 Scalability

In this section, we measure the construction scalability of our solution, as well as the query processing scalability.

Solution Construction Scalability There are three major phases involved in constructing the SecDM framework: data anonymization using the *DiffGen* algorithm, data preprocessing, and kd-tree index construction; the latter can be further divided into two subphases: *internal nodes construction* and *leaf nodes construction*. According to procedure $buildIndex$ in Algorithm 1, the complexity for constructing SecDM is dominated by the number of ε-differentially private records, which in turn is impacted by the number of raw data records and the setting of the number of specializations for the *DiffGen* algorithm. The objective is to measure the runtime of each construction phase to ensure its capability to scale up in terms of records size.

Figure 10 depicts the runtime of each of the construction phases, where the number of data records ranges from 20,000 to 100,000 records, and the number of specializations is set to 8. We observe that the runtime of each phase grows linearly as the number of records increases. We also observe that the overall construction runtime scales up linearly as well, as it takes 47 sec to construct the framework for a data set with 20,000 records, 72 sec for 40,000 records, 96 sec for 60,000 records, 106 sec for 80,000 records, and 121 sec for 100,000 records. Since each phase of the algorithm, as well as the overall construction time, grow linearly with respect to the total number



(a) Exact Query (b) Specific Query (c) Generic Query

Fig. 13 Efficiency w.r.t. the number of attributes per a query for *exact*, *specific*, and *generic* queries.

of records, this suggests the construction of SecDM is scalable with regard to the data size.

Query Processing Scalability One major contribution of our work is the development of a scalable framework for query processing on anonymized data in the cloud. Since the number of specializations during the anonymization process impacts the total number of anonymized records, we study the runtime for answering different types of user count queries under a different number of specializations, while the number of raw data records ranges from 20,000 to 100,000. Given the three user count query types, *Exact*, *Specific*, and *Generic*, we randomly create 500 queries of each type, and report the average runtime, where the average number of attributes in each query is 8.

Figures 11(a), 11(b), and 11(c) depict the processing runtime of each type of user count queries when the number of specializations is set to 8, 10, and 12, respectively. In Figure 11(a), we observe that the processing runtime of each query type grows linearly as the number of raw data records continues to increase at the same rate. That is, the processing runtime grows from 4.8 sec for 20,000 records to 6 sec for 100,000 records when the query type is *exact*; from 6.5 sec for 20,000 records to 8.5 sec for 100,000 records when the query type is *specific*; and from 12.4 sec for 20,000 records to 31.2 sec for 100,000 records when the query type is *generic*. Similarly, in Figures 11(b) and 11(c) we observe that the processing runtime of each query type is linear with regard to the number of raw data records for all three types. The increase in the number of specializations leads to a higher number of anonymized records, thus explaining the increase in the average query processing runtime for each query type in Figures 11(a), 11(b), and 11(c).

Figure 12 depicts the performance of categorical and numerical attributes with respect to number of records, when the number of specializations is 8, the query type is generic, and the total number of attributes is 12 (6 categorical and 6 numerical). We observe that the processing runtime of both categorical and numerical attributes grows linearly as the number of raw data records continues to increase at the same rate. Although the number of categorical and numerical attributes in the experiment is the same, processing numerical attributes requires approximately 2.5 the time required for processing categorical attributes. This is mainly due to the need to split each numerical attribute into several ranges and then use exponential mechanism to determine the split points. In contrast, the split points in categorical attributes are fixed and determined based on the taxonomy trees.

6.2.3 Efficiency

To demonstrate the efficiency of our SecDM framework we measure the impact of the number of attributes in a query on the processing time needed by the cloud to process the query and by the user to decrypt the result. We split the query processing phase into two subphases: *tree traversal* and *compute NCount*. We assume the number of specializations is 8, while the number of raw data records is 100,000. We create 500 queries of each query type, and report the average runtime.

Figures 13(a), 13(b), and 13(c) depict the processing runtime of *exact*, *specific*, and *generic* queries, respectively, when the average number of attributes in a query ranges from 2 to 14. We observe that the most dominant phase with regard to the processing runtime is the tree traversal phase, whereas the resulting decryption phase is the least dominant. The total processing runtime of each query type decreases linearly as the number of attributes per query increases. That is, the total runtime decreases from 31.8 sec to 0.9 sec when the number of attributes per query increases from 2 to 14 for *exact* queries, decreases from 37.2 sec to 1 sec when the number of attributes per query increases from 2 to 14 for *specific* queries, and decreases from 78.8 sec to 10.4 sec when the number of attributes per query increases from 2 to 14 for *generic* queries. The total processing runtime improves as the number of attributes increases because adding more attributes to a query makes it more restrictive and, consequently, requires fewer nodes to be traversed in the kd-tree index. Assuming the average noisy count value for each anonymized record is 10,000, we observe that the decryption phase, which involves decrypting Exponential ElGamal ciphertexts, is very small (less than 2 sec) and barely sensitive to the increase in the number of attributes per query.

7 Conclusions and Future Work

In this paper, we propose a privacy-preserving framework for confidential count query processing in a cloud computing environment. Our framework maintains the privacy of the outsourced data while providing data confidentiality, confidential query processing, and privacy-preserving results. Users (data miners) of the system are not required to have prior knowledge about the data, and incur lightweight computation overhead. The framework also allows for query reusability, which reduces the communication and processing time. We perform several experimental evaluations on real-life data, and we show that the framework can efficiently answer different types of queries and is scalable with regard to the number of data records.

As for future work, we plan on investigating how to enable authorized users to self-secure their queries before submitting them to the cloud in order to eliminate the dependency on the data provider. We also plan to investigate a scenario in which data is obtained from multiple data providers and stored in a distributed outsourcing environment. Given that our framework currently supports only user count queries consisting of conjunction of predicates, we will investigate how to support complex queries with various types of Boolean operators, including OR, NOT and XOR.

References

1. T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," in *Proceedings of the 33rd International Conference on Very Large Data Bases (PVLDB)*, 2007, pp. 519–530.
2. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, 2011, pp. 85–100.
3. B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 14:1–14:53, 2010.
4. C. Dwork, "Differential privacy," in *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, 2006, pp. 1–12.
5. M. Barbaro and T. J. Zeller, "A face is exposed for aol searcher no. 4417749," Aug 2006.
6. A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2008, pp. 111–125.
7. T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Proceedings of the 24th IEEE Conference on Advanced Information Networking and Applications (AINA)*, 2010, pp. 27–33.
8. A. Blum, C. Dwork, F. McSherry, and K. Nissim, "Practical privacy: the SuLQ framework," in *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2005, pp. 128–138.
9. S. Barouti, F. Aljumah, D. Alhadidi, and M. Debbabi, "Secure and privacy-preserving querying of personal health records in the cloud," in *Data and Applications Security and Privacy XXVIII (LNCS)*, 2014, vol. 8566, pp. 82–97.
10. F. Giannotti, L. Lakshmanan, A. Monreale, D. Pedreschi, and H. Wang, "Privacy-preserving mining of association rules from outsourced transaction databases," *IEEE Systems Journal (ISJ)*, vol. 7, no. 3, pp. 385–395, 2013.
11. S. Wang, D. Agrawal, and A. El Abbadi, "A comprehensive framework for secure query processing on relational data in the cloud," in *Proceedings of the 8th VLDB International Conference on Secure Data Management (SDM)*, 2011, pp. 52–69.
12. Y. Wang, "Privacy-preserving data storage in cloud using array bp-xor codes," *IEEE Transactions on Cloud Computing (TCC)*, vol. 99, 2014.
13. P. Tysowski and M. Hasan, "Hybrid attribute- and re-encryption-based key management for secure and scalable mobile applications in clouds," *IEEE Transactions on Cloud Computing (TCC)*, vol. 1, no. 2, pp. 172–186, July 2013.
14. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2002, pp. 216–227.
15. H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE)*, 2011, pp. 601–612.
16. G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu, "Differentially private spatial decompositions," in *Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE)*, 2012, pp. 20–31.
17. H. Wang and L. V. S. Lakshmanan, "Efficient secure query evaluation over encrypted xml databases," in *Proceedings of the 32nd International Conference on Very Large Data Bases (PVLDB)*, 2006, pp. 127–138.
18. Y. Hua, B. Xiao, and X. Liu, "Nest: Locality-aware approximate query service for cloud computing," in *Proceedings of the IEEE INFOCOM*, 2013.
19. M. Shaneck, Y. Kim, and V. Kumar, "Privacy preserving nearest neighbor search," in *Machine Learning in Cyber Trust*, 2009, pp. 247–276.
20. P. Wang and C. Ravishankar, "Secure and efficient range queries on outsourced databases using $\widehat{R}$-trees," in *Proceedings of the IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 314–325.
21. R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang, "(α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2006, pp. 754–759.
22. N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu, "Differentially private data release for data mining," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2011, pp. 493–501.

23. R. Chen, Q. Xiao, Y. Zhang, and J. Xu, "Differentially private high-dimensional data publication via sampling-based inference," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, 2015, pp. 129–138.
24. Y. Xiao, L. Xiong, and C. Yuan, "Differentially private data release through multidimensional partitioning," in *Proceedings of the 7th VLDB Conference on Secure Data Management (SDM)*, 2010, pp. 150–168.
25. B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proceedings of the 13th International Conference on Very Large Data Bases (PVLDB)*, 2004, pp. 720–731.
26. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2004, pp. 563–574.
27. F. Emekci, D. Agrawal, A. Abbadi, and A. Gulbeden, "Privacy preserving query processing using third parties," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, 2006, pp. 27–36.
28. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM Symposium on Theory of Computing (STOC)*, 2009, pp. 169–178.
29. G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella, "Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology," *Sustainable Cities and Society (SCS)*, vol. 39, pp. 283–297, 2018.
30. R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Proceedings of the 16th annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, 1997, pp. 103–118.
31. C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proceedings of the IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, 2010, pp. 253–262.
32. T. Ge and S. Zdonik, "Fast, secure encryption for indexing in a column-oriented DBMS," in *Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE)*, 2007, pp. 676–685.
33. E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Balancing confidentiality and efficiency in untrusted relational dbmss," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 2003, pp. 93–102.
34. H. Hacigümüş, B. Iyer, and S. Mehrotra, "Efficient execution of aggregation queries over encrypted relational databases," in *Proceedings of the Database Systems for Advanced Applications (DASFAA)*, 2004, pp. 125–136.
35. Z.-F. Wang, J. Dai, W. Wang, and B.-L. Shi, "Fast query over encrypted character data in database," in *Proceedings of the 1st International Conference on Computational and Information Science (CIS)*, 2004, pp. 1027–1033.
36. Z.-F. Wang, W. Wang, and B.-L. Shi, "Storage and query over encrypted character and numerical data in database," in *Proceedings of the 5th International Conference on Computer and Information Technology (CIT)*, 2005, pp. 77–81.
37. R. Bayer and E. McCreight, "Organization and maintenance of large ordered indices," in *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access and Control*, 1970, pp. 107–141.
38. D. Comer, "Ubiquitous B-Tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, 1979.
39. J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
40. A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1984, pp. 47–57.
41. W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2009, pp. 139–152.
42. D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P)*, 2000.
43. C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," in *Proceeedings of DBSec*, 2008, pp. 127–143.
44. C. Bösch, Q. Tang, P. Hartel, and W. Jonker, "Selective document retrieval from encrypted database," in *Proceedings of ISC*, 2012, pp. 224–241.
45. S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proceedings of the ACM SIGSAC conference on Computer & Communications Security (CCS)*, 2013, pp. 875–888.

46. C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Survey*, vol. 47, no. 2, pp. 18:1–18:51, 2014.
47. D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Proceedings of TCC*, 2011, pp. 253–273.
48. D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," *SIAM Journal of Computing*, vol. 32, no. 3, pp. 586–615, 2003.
49. J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2007, pp. 321–334.
50. D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th Conference on Theory of Cryptography (TCC)*, 2007, pp. 535–554.
51. X. Yi, R. Paulet, E. Bertino, and G. Xu, "Private cell retrieval from data warehouses," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1346–1361, 2016.
52. D. Kifer and B.-R. Lin, "Towards an axiomatization of statistical privacy and utility," in *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2010, pp. 147–158.
53. S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *IACR Cryptology ePrint Archive*, vol. 2011, p. 272.
54. A. Shabtai, Y. Elovici, and L. Rokach, *A Survey of Data Leakage Detection and Prevention Solutions*, ser. SpringerBriefs in Computer Science. Springer, 2012.
55. E. Shmueli, T. Tassa, R. Wasserstein, B. Shapira, and L. Rokach, "Limiting disclosure of sensitive data in sequential releases of databases," *Information Science*, vol. 191, pp. 98–127, 2012.
56. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag TELOS, 2008.
57. 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*, 2009, pp. 19–30.
58. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT)*, 2001, pp. 514–532.
59. K. Bache and M. Lichman, *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2013.
60. B. C. M. Fung, K. Wang, and P. S. Yu, "Anonymizing classification data for privacy preservation," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 19, no. 5, pp. 711–725, 2007.
61. S. Salzberg, "C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993," *Machine Learning*, vol. 16, no. 3, pp. 235–240, 1994.
62. A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD*, ser. KDD '10, 2010, pp. 493–502.
63. O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2.
64. A. Joux, "A one round protocol for tripartite diffie-hellman," in *Proceedings of the 4th International Symposium on Algorithmic Number Theory (ANTS)*, 2000, pp. 385–394.
65. D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology CRYPTO 2004*, ser. Lecture Notes in Computer Science, 2004, vol. 3152, pp. 41–55.
66. P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 2008, pp. 139–148.

A Security Analysis

The proposed framework is sound since all adversaries are non-colluding and semi-honest, according to our adversarial model. In the rest of this section, we focus on proving that the protocol is confidentiality-preserving. We also illustrate the accessibility of the keys in the framework, and show that all keys are properly distributed between the parties.

Privacy by Simulation. Goldreich [63] defines the security of a protocol in the semi-honest adversarial model as follows.

Definition 7 ***(Privacy w.r.t. Semi-honest Behavior) [63].*** *Let $f : (\{0,1\}^*)^m \mapsto (\{0,1\}^*)^m$ be an m-ary deterministic polynomial-time functionality, where $f_i(x_1, \ldots, x_m)$ is the* i*th element of $f(x_1, \ldots, x_m)$. Let Π be an m-party protocol for computing f. The view of the* i*-th party during an execution of Π over $x = (x_1, \ldots, x_n)$ is $\mathsf{view}_i^{\Pi}(x) = (x_i, r_i, m_{i,1}, \ldots, m_{i,t})$, where r_i equals the contents of the* i*th party's internal random tape, and $m_{i,j}$ represents the* j*th message that it received. For $I = \{i_1, \ldots, i_l\} \subseteq \{1, \ldots, m\}$, $\mathsf{view}_I^{\Pi}(x) = (I, \mathsf{view}_{i_1}^{\Pi}(x), \ldots, \mathsf{view}_{i_l}^{\Pi}(x))$. We say that Π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithm (simulator) S such that for every $I \subseteq \{1, \ldots, m\}$:*

$$\{S(I, (x_{i_1}, \ldots, x_{i_l}), f_I(x))\}_{x \in (\{0,1\}^*)^m} \stackrel{c}{\equiv} \{\mathsf{view}_I^{\Pi}(x)\}_{x \in (\{0,1\}^*)^m}$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability. ■

According to Definition 7, it is sufficient to show that we can effectively simulate the view of each party during the execution of the SecDM protocol given the input, output and *acceptable* leaked information of that party, in order to prove that our protocol is secure. We achieve that by simulating each message *received* by a party in each algorithm. If we can simulate the input messages of each party in the protocol based only on its input and output, and the party is not able to recognize that it is dealing with a simulator, that means the protocol does not leak anything to that party since it would have been able to compute its output from its input without the need to be involved in the protocol in the first place.

First, we define the concepts *query distribution* and *query processing threshold*.

Definition 8 ***(Query Distribution.)*** *The distribution of the data mining queries, denoted by U, is the set of all possible queries, where each query consists of $k_c + 2 \times k_n$ integers, each of which maps to a value in the domain of a categorical or numerical attribute.*

Definition 9 ***(Query Processing Threshold.)*** *Query processing threshold, denoted by α, is the maximum number of queries allowed to be processed on a kd-tree before the latter is replaced by a new shuffled and re-encrypted kd-tree submitted by the data provider to the service provider.*

Definition 10 ***(Privacy-preserving Data Outsourcing Framework).*** *Let $\mathcal{F}$ be a framework that enables a service provider (cloud) to answer queries from data miners on hosted (outsourced) data. $\mathcal{F}$ is a privacy-preserving framework if the following properties hold:*

1. ***Correctness.*** *For any user query $u \in U$, the cloud returns res_u to the data miner such res_u is the correct answer to u.*
2. ***Data Confidentiality.*** *A semi-honest adversary $\mathcal{E}$, statically corrupting the service provider, cannot learn anything more about the hosted data from an accepted transcript of $\mathcal{F}$ than she could given only the total number of numerical and categorical attributes, and the size of each attribute's domain.*
3. ***Query Confidentiality.*** *A semi-honest adversary $\mathcal{E}$, statically corrupting the service provider, cannot learn anything about the query.*
4. ***Differentially Private Output.*** *For all $u \in U$, res_u satisfies differential privacy.*

Definition 11 ***(α-Privacy-preserving Data Outsourcing Framework).*** *An outsourcing framework $\mathcal{F}$ is α-privacy-preserving if it satisfies all properties in Definition 10 except that the cloud learns the search pattern of at most α number of queries.*

Theorem A1 SecDM*, as specified in Protocols 1–7, is an α-privacy-preserving data outsourcing framework.*

Proof We proved in Section B Property 1 (correctness) and Property 4 (differentially private output).

To prove Property 2 (data Confidentiality) and Property 3 (query Confidentiality), we build a simulator $\mathcal{S}$ that generates a view that is statistically indistinguishable from the view of $\mathcal{E}$ in real execution. Per Definition 7, the view of the service provider consists mainly of the messages it receives from the other parties. Although we have 8 algorithms, the service provider receives messages from the protocol only in Algorithm 3 - Line 2 (encrypted index from data provider) and Algorithm 5 - Line 4 (encrypted query from data miner). All other steps in all algorithms do not need to be simulated because they either do not involve the service provider at all (e.g. the steps in Algorithm 1, 2 and 4), or involve ciphertext operations (e.g. the steps in Algorithm 6 and 7) which are inherently secure from the security of the cryptosystems used (ABE and Elgamal).

In Algorithm 3 - Line 2, the service provider receives kd-tree index T from the data provider.

Simulation :

1. Supplied with k, the total number of attributes in $\hat{D}$, and the size of each attribute's domain $|\Omega(\hat{A}_i)| : 1 \leq i \leq k$, the simulator $\mathcal{S}$ generates attribute domains $\Omega(\hat{A}'_1), \Omega(\hat{A}'_2), \ldots, \Omega(\hat{A}'_k)$ such that each domain $\Omega(\hat{A}'_i)$ consists of $|\Omega(\hat{A}_i)|$ distinct values, e.g., $1, 2, \ldots, |\Omega(\hat{A}_i)|$.
2. $\mathcal{S}$ constructs a contingency table $\hat{D}'$ with k columns each of which represents one attribute $\hat{A}'_i$, and n records each of which represents one possible combination of attribute values such that $n = \prod_{i=1}^{k} |\Omega(\hat{A}'_i)|$.
3. Supplied with the total number of numerical attributes k_n and categorical attributes k_c in $\hat{D}$ such that $k_n + k_c = k$, the size of each attribute's domain, and the security parameter of ACP-ABE, $\mathcal{S}$ runs $\mathbb{A}.\mathsf{Setup}(1^\lambda)$ to generate public key PK' and master secret key MSK'. Similarly, given the security parameter of ElGamal, $\mathcal{S}$ runs $\mathbb{G}.\mathsf{KeyGen}()$ to generate public key y' and secret key x'.
4. Given $\hat{D}'$, split dimension $i = 1$, PK' and y', $\mathcal{S}$ runs Algorithm 1 and Algorithm 2 to construct a balanced kd-tree T' over $\hat{D}'$:
   (a) In Line 12 of Algorithm 1, n random group elements are generated for each ciphertext CT_{left} or CT_{right} of each internal node v.
   (b) In Line 2 of Algorithm 2, a random ElGamal ciphertext, e.g., encryption of '0', is assigned to the encrypted $NCount$ of each leaf node l.

Indistinguishability Argument : T' is computationally indistinguishable from T.

First, we construct a hybrid tree called T'', and then show the relation between T'' and real the kd-tree T, and between T'' and the simulated kd-tree T'.

1. Let T'' be a kd-tree index over $\hat{D}'' = \hat{D}$ constructed using Algorithm 1 and Algorithm 2, where:
   (a) The ACP-ABE ciphertexts CT_{left} and CT_{right} of each internal node are random group elements, as per Step 4a above.
   (b) The noisy count $NCount$ in each leaf node is a random ElGamal ciphertext, as per Step 4b above.
2. T'' is computationally indistinguishable from T, denoted by $T'' \stackrel{c}{\equiv} T$, because:
   (a) The ACP-ABE ciphertexts in the internal nodes of the kd-tree are IND-CPA-secure under the *decisional bilinear diffie-hellman* (DBDH) assumption [64] and the *decision linear* (D-Linear) assumption [65].
   (b) Since ElGamal is IND-CPA-secure, the distribution of the ciphertext (output) space is independent of the key/message. Therefore, encrypting any message with a random factor is sufficient to generate a computationally indistinguishable $NCount$.
3. T'' is statistically indistinguishable from T', denoted by $T'' \stackrel{s}{\equiv} T'$, because:
   (a) $\hat{D}'' \stackrel{s}{\equiv} \hat{D}'$, where there is one-to-one correspondence between the equivalent classes in $\hat{D}''$ and the records in $\hat{D}'$.
   (b) The random coins used in ACP-ABE encryption in Algorithm 1 are drawn from the same distribution.
   (c) The random coins used in ElGamal encryption in Algorithm 2 are drawn from the same distribution.
4. From Steps (2) and (3), we conclude that $T' \stackrel{c}{\equiv} T$.

In Algorithm 5 - Line 4, the service provider receives system count query SK_u and a set of attribute distribution tokens $\mathcal{N}$.

Simulation :

1. $\mathcal{S}$ obtains α sample queries $\bar{U} = \{u'_1, u'_2, \ldots, u'_\alpha\}$ from U.
2. For each query $u'_i \in \bar{U}$, $\mathcal{S}$ constructs a query pair $(SK_{u'_i}, \mathcal{N}'_i)$ as follows:
   – $\mathcal{S}$ runs $\mathbb{A}.\mathsf{KeyGen}(MSK', u'_i)$ to construct system count query $SK_{u'_i}$.

- $\mathcal{S}$ constructs a set $\mathcal{N}_i'$ containing $2 \times k_n$ ADT tokens, where $ADT.value$ for each token is a randomly generated ElGamal ciphertext, e.g., encryption of '0'.

3. Up to α times, each time a data miner in the real world submits a query, $\mathcal{S}$ submits to the service in the simulation world a different query pair from the set of pairs generated in Step 2.

Indistinguishability Argument :

1. Given any real system query SK_u, $SK_{u_i'} \stackrel{c}{\equiv} SK_u$ because:
   (a) $u_i' \stackrel{s}{\equiv} u$.
   (b) $SK_{u'} \stackrel{c}{\equiv} SK_u$ since $|SK_{u'}| = |SK_u| = k_c + 2 \times k_n$ group element tuples, and the ACP-ABE scheme is IND-CPA-secure.
2. Given any real ADT set $\mathcal{N}$, $\mathcal{N}_i' \stackrel{c}{\equiv} \mathcal{N}$ because:
   - $|\mathcal{N}_i'| = |\mathcal{N}| = 2 \times k_n$.
   - The $ADT.value$ of each token in $\mathcal{N}_i'$ is computationally indistinguishable from the $ADT.value$ of any real token due to the IND-CPA-secure property of ElGamal.

Discussion. The threshold parameter α can range between 1 and ∞. To better understand the impact of revealing α queries to $\mathcal{S}$, we analyze the security when $\alpha = 1$ and $\alpha > 1$.

Case 1 : $\alpha = 1$. This represents the highest security level of our protocol, where one system query is executed per one kd-tree. Since the kd-tree index is constructed by Algorithm 1 as a *balanced* tree and since each path contains all attributes, then no correlation can be established between any two attributes and the attributes are protected when evaluated for splitting the k-dimensional space. As for the data mining query, the service provider cannot determine what attributes are included in the query, nor know what values or ranges the data miner is interested in. Since Algorithm 6 yields how many leaf nodes (equivalent classes) identified, this reveals how general the query is. In general, the more leaf nodes identified by a query, the more general the query is. The revealing of the number of identified leaf nodes, however, won't help the service provider better guess the final result of the query since it cannot access the encrypted noisy counts.

Although setting α to 1 provides the highest security w.r.t. query search pattern, it is impractical due to the cost of reconstructing the kd-tree. We refer the reader to *solution construction scalability* in Section 6.2.2 for more details about the cost of reconstructing the kd-tree.

CASE 2 : $\alpha > 1$. While our proposed framework supports *confidential access* to the data, executing multiple queries on the same kd-tree index reveals the search pattern of the queries, where the service provider is able to determine the number of leaf nodes that overlap between the queries. Let u and u' be two user queries that satisfy the same set of leaf nodes $l = \{l_1, \ldots, l_r\}$, and let *collision set* denote the set of all unique queries that could satisfy l. The size of the collision set can be determined as follows:

$$|\mathsf{collision\ set}(l)| = \prod_{i=1}^{r} \prod_{j=1}^{k} |l_i.Range(\hat{A}_j)| \ : \ \hat{A}_j \text{ is numerical},$$

where $|l_i.Range(\hat{A}_j)|$ denotes the size of the range of attribute $\hat{A}_j$ in the equivalent class represented by leaf node l_i. Note that since the noisy counts are encrypted using ElGamal, the position of the attributes in the tree is hidden and is shuffled every time the kd-tree is constructed, disclosing the search pattern on the differentially private data reveals nothing about the final (noisy) result of each query, nor about the attributes/values in each query. The smaller the value of α is, the less overlap between queries is revealed. Several techniques have been proposed in the literature to address the problem of private search pattern, such as [66]; however, it is out of the scope of this paper.

Note that each time the data provider generates a shuffled and re-encrypted kd-tree, a different ACP-ABE master secret key MSK should be used to prevent the service provider from processing new queries on the old tree.

In our model, we assume the data miner can have access to the entire differentially-private dataset. The data privacy is guaranteed by differential privacy. Therefore, there is no need to simulate the view of the data miner.

Moreover, since our framework returns differentially private results for each count query in a deterministic way, any repetition of queries will leak no extra information about the dat. Also, since count query results are differentially private, our framework is also protected against background knowledge attacks.

The proposed protocol in this paper involves the composition of secure subprotocols in which all intermediate outputs from one subprotocol are inputs to the next subprotocol. These intermediate outputs

Table 4 Key accessibility w.r.t. all parties in SecDM framework

Scheme	Key	Data Bank	Service Provider	Data Miner
$\mathbb{G}$	private key x	generator, full control	no access	read access
$\mathbb{G}$	public key y	generator, full control	read access	read access
$\mathbb{A}$	master secret key MSK	generator, full control	no access	no access
$\mathbb{A}$	public key PK	generator, full control	read access	read access
$\mathbb{A}$	user secret key SK_u	generator, full control	read access	read access

are either simulated given the final output and the local input for each party or computed as random shares. Using the composition theorem (Goldreich [63]), it can be shown that if each subprotocol is secure, then the resulting composition is also secure.

Key Accessibility. Protecting the data distributed between different parties from unauthorized access is an essential part of securing the SecDM framework. We must ensure that all keys are properly distributed such that no party can decrypt any data it is not supposed to have access to in plaintext. Table 4 illustrates the accessibility of each key by each party in SecDM.

Observe that the data provider is the generator of all encryption keys in the system and maintains full control over them. The service provider, on the other hand, has no access to Exponential ElGamal's private key, $\mathbb{G}.x$, that would have allowed her to fully decrypt the contents of each leaf node in the kd-tree index. Moreover, not having access to the ACP-ABE master secret key $\mathbb{A}.MSK$ prevents the service provider from being able to determine the access structures of the ciphertexts in each internal node of the kd-tree index. As for the user (data miner), not having access to $\mathbb{A}.MSK$ prevents her from bypassing authentication and creating her own system count queries.

B Correctness Analysis

The correctness proof is twofold. First, we prove that Algorithm 6 identifies all the leaf nodes satisfying the user count query u. Second, we prove that Algorithm 7 produces the exact total count answer to u, and the answer is differentially private.

Proposition 5 *Given a user count query $u = \mathcal{P}_1 \wedge ... \wedge \mathcal{P}_m$, Algorithm 6 produces a set $\mathcal{R}$ containing all leaf nodes satisfying u.*

Proof To prove the correctness of Algorithm 6 we prove *partial correctness* and *termination*.

1. Partial Correctness. We provide a proof by induction.

Basis. When u includes no predicate for any of the attributes in $\hat{D}$, then each categorical attribute in SK_u is assigned the value 1 (the identifier of the root node of the corresponding taxonomy tree), whereas for each numerical attribute $\hat{A}_i \in \hat{D}$, $\hat{A}_i^{min} = 1$ (the lowest range identifier) and $\hat{A}_i^{max}$ is assigned the highest range identifier in $\Omega(\hat{A}_i)$. When SK_u is used to traverse the kd-tree index, all internal nodes will be traversed until the leaf nodes are reached. That is, if the current node v is internal, $\mathbb{A}.\mathsf{Dec}(v.CT_{left}, SK_u)$ and $\mathbb{A}.\mathsf{Dec}(v.CT_{right}, SK_u)$ will always be true because the attributes in SK_u will always satisfy the access structure in $v.CT_{left}$ and $v.CT_{right}$, and pointers to the left child node and right child node will always be obtained.

Induction Step. Assume that traversing the kd-tree index using SK_u produces the correct set of leaf nodes $\mathcal{R}$ satisfying u. We show that if a new predicate $\mathcal{P} = (\hat{A}_i \vdash s_i)$ is added to u such that $\acute{u} = u + \mathcal{P}$, then traversing the kd-tree index using $SK_{\acute{u}}$ produces the correct set of leaf nodes $\acute{\mathcal{R}}$ satisfying $\acute{u}$. We

observe that $\acute{\mathcal{R}} \subseteq \mathcal{R}$. To complete the proof in this step, we assume that $\mathcal{P}$ corresponds to a categorical attribute; however, the same analogy can be applied to a numerical attribute's predicate. When v is an internal node and $v.split\_dim = \hat{A}_i$, if $s_i.ID \leq v.split\_value$ then $\mathbb{A}.\mathsf{Dec}(v.CT_{right}, SK_u)$ will evaluate to *false*, and no recursive call of procedure *traverseIndex* over node $v.rc$ will be executed. This behaviour is correct because in this case the subtree whose root is $v.rc$ includes the leaf nodes that do not satisfy $\mathcal{P}$, and hence there is no need to search the subtree rooted at $v.rc$. The same logic can be used to reason about the case when $s_i.ID > v.split\_value$.

2. Termination. Each recursive call on a child node partitions the space of the parent node in half. This shows that the algorithm strictly moves from one level to a lower level in the *k*d-tree index while reducing the search space by half until all leaf nodes satisfying u are reached.

Proposition 6 *Given a set of leaf nodes $\mathcal{R}$ generated by a system count query SK_u and a set of attribute distribution tokens $\mathcal{N}$, the output of Algorithm 7 is the exact noisy count answer corresponding to SK_u.*

Proof To prove the correctness of Algorithm 7, we prove *partial correctness* and *termination*.

1. Partial Correctness. We provide a proof by induction.

Basis. When $\mathcal{N} = \phi$, the inner loop will never be executed. In this case, procedure *compTCount* will go through all the leaf nodes in $\mathcal{R}$ and add together all corresponding noisy counts by utilizing the homomorphic addition property of Exponential ElGamal. This is correct because if no *ADT* token was originally generated, then the user query is an *exact* query, and 100% of the noisy count of each leaf node in $\mathcal{R}$ must be used.

Induction Step. Assume that for $\mathcal{N} = \{ADT_1, ..., ADT_l\}$, procedure *compTCount* computes the exact noisy count answer to the user count query u. We show that if a new token ADT_{l+1} for numerical attribute $\hat{A}_i$ is added such that $\acute{\mathcal{N}} = \mathcal{N} \cup ADT_{l+1} = \{ADT_1, ..., ADT_{l+1}\}$, where $\acute{\mathcal{N}}$ corresponds to the system count query $SK_{\acute{u}}$, then procedure *compTCount* computes the exact noisy count answer to the user count query $\acute{u}$. Without loss of generality, we assume that the set of leaf nodes $\mathcal{R}$ remains the same. Since ADT_{l+1} is for numerical attribute $\hat{A}_i$, then $ADT_{l+1}.value$ represents the percentage of the partial intersection between query $\acute{u}$ and attribute $\hat{A}_i$ by definition. If $\acute{u}$ is a *generic* query, then not all leaf nodes in $\mathcal{R}$ will contain a tag that corresponds to $ADT_{l+1}.tag$. However, the noisy count of each leaf node l containing a tag that matches $ADT_{l+1}.tag$ must be adjusted by multiplying $l.NCount$ with $ADT_{l+1}.value$.

2. Termination. We denote by n the initial number of leaf nodes in $\mathcal{R}$. If $n > 0$ then we enter the outer loop. We also denote by m the initial number of *ADT* tokens in $\mathcal{N}$. If $m > 0$ then we enter the inner loop such that after each iteration, the variable *m* is decreased by one, and it keeps strictly decreasing until $m = 0$ where the inner loop terminates. Similarly, the outer loop will terminate as n keeps strictly decreasing until it reaches 0; at that stage the algorithm terminates.