An Efficient and Secure Protocol for Querying High-Dimensional Data in the Cloud

Gaby G. Dagher, Benjamin C. M. Fung, Senior Member, IEEE, Noman Mohammed, and Jeremy Clark

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 $k$-d-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 can efficiently answer range queries and is scalable with increasing data size.

Index Terms—Cloud Computing, Attribute-based Encryption, Differential Privacy.

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 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 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 mining query, 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 [4] 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.

1. PopData: https://www.popdata.bc.ca/
manage, and answer queries on the integrated data. However, this raises 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 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. While our framework protects the confidentiality of individual query (data access), we provide a detailed security analysis in Section 6.3. We analyze in Section 5.4 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.

The intuition of our solution is to generate a \(kd\)-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 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 6.3.

The contributions of this paper can be summarized as follows:

**Contribution 1.** To the best of our knowledge, this is the first work that proposes a comprehensive privacy-preserving framework for query processing in a cloud computing environment. The proposed framework 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.

**Contribution 2.** To ensure efficient data access while maintaining data confidentiality, we present an algorithm for constructing an encrypted \(kd\)-tree index that hides the data from the cloud while allowing for confidential query processing. We utilize attribute based encryption in a unique way to handle range predicates on numerical attributes.

**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.

**Contribution 4.** SecDM has two major steps, namely index construction and query processing. It has linear time complexity on both steps 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.

The rest of the paper is organized as follows: Section 2 reviews the related literature. Section 3 introduces the encryption schemes to be used in the proposed framework. Section 4 provides the formal definition of the data outsourcing problem. Section 5 describes the SecDM framework. In Section 6 we analyze the correctness and security of our proposed framework. Comprehensive experimental results are presented in Section 7. Finally, we conclude the paper in Section 8.

## 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), 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 $(\alpha, k)$-anonymity model \cite{22} applies generalization and suppression techniques to protect against record and attribute linkages. The $\varepsilon$-differential privacy model \cite{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. \cite{24} propose a generalization-based anonymization algorithm in a non-interactive setting for releasing differentially private records for data mining. Cormode et al. \cite{16} propose a framework for using spatial data structures to provide a differentially private description of the data distribution. Xiao et al. \cite{23} 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.

Another PPDP approach for privacy protection is anatomization. Xiao and Tao \cite{24} propose the anatomy method that partitions the data vertically in order to disassociate the relationship between the quasi-identifier (QID) attributes and the sensitive attributes (ST), while satisfying the $\ell$-diversity privacy requirement. This approach generates two tables: a QID table that contains all quasi-identifier attributes, and an ST table that contains all sensitive attributes. Both tables contain an anatomy group attribute (GID) such that all records belonging to the same anatomy group will have the same GID value in both tables. If an anatomy group has an $\ell$ distinct set of sensitive values where each value exists exactly once in the group, then the probability of linking a record to a sensitive value using GID value is $1/\ell$. Since the anatomy approach only considers the single association between QID attributes and ST attributes, Jiang et al. \cite{25} propose a new approach based on anatomy that considers the functional dependencies among the data attributes in a single table. This approach splits the table into several sub-tables according to the functional dependencies such that the decomposed sub-tables satisfy the privacy requirement of $\ell$-diversity for each sensitive association. Nergiz and Clifton \cite{26} propose another decomposition approach using anatomy for query processing on outsourced data that consists of multi-relational tables.

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 \cite{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 \cite{14, 27} attach range labels to the encrypted data, thus revealing the underlying distributions of the data. Other methods depend on order-preserving encryption \cite{28, 29}; 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 \cite{1, 30}; however, its high computation cost makes it prohibitive in practice. In this paper, we employ the exponential variation of ElGamal \cite{31} 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 \cite{32, 17, 53}. Some indexing schemes have constraints on the type of queries they support. For example, hash-based indexing \cite{34} and privacy homomorphism \cite{35} only support equality queries, whereas bucket-based indexing \cite{14} and character-oriented indexing \cite{36, 37} 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 \cite{38} and B+–tree \cite{39} and spatial access indexes such as $kd$-tree \cite{40} and R-tree \cite{31}. Our work fits in this category because we utilize an encrypted $kd$-tree index for efficient and secure traversal. Wang et al. \cite{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 R-tree index that is encrypted using Asymmetric Scalar-product Preserving Encryption (ASPE) scheme [42]. 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 Secure Multiparty Computation

In a distributed data outsourcing environment, the data is partitioned and outsourced to a set of independent and non-colluding servers. To ensure data and query confidentiality, a distributed protocol is needed to securely process the query without disclosing the data in each of the outsourcing servers. Such protocols typically use secure multiparty computation (SMC) [43] [44], a cryptographic technique that computes a secure function from multiple participants in a distributed network. For example, Shaneck et al. [19] propose an approach for computing kNN queries on horizontally partitioned data, Vaidya and Clifton [45] propose an approach for secure answering of top k queries on vertically partitioned data while satisfying k-anonymity, whereas Rastogi and Nath [46] and Shi et al. [47] address the problem of private aggregation of time-series data such that the outcome statistic is differentially private. SMC-based methods are not suitable to our work as they are highly interactive and require heavy computation and storage overhead.

2.4 Searching on Encrypted Data

Searchable encryption (SE) [48] [49] [50] [51] is a closely related line of work that supports secure searching on encrypted data. SE schemes (except for [48]) 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 [52]. In contrast, our proposed framework reveals only the search pattern of the queries to the cloud. Functional encryption (FE) [53] is another related line of work that support searching on encrypted data. It includes identity-based encryption [54], attribute-based encryption [55], and predicate encryption [56]. 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.

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 Preliminaries

The framework presented in this paper utilizes two different types of encryption schemes: an anonymous ciphertext-policy attribute-based encryption scheme [57] and an Exponential ElGamal scheme [51]. In this section, we introduce the two schemes and define for each one the properties and main functions to be used for data and query security in the framework.

3.1 Anonymous Ciphertext-Policy Attribute Based Encryption (ACP-ABE)

Sahai and Waters [58] introduced Attribute-Based Encryption (ABE) as a new concept of public encryption schemes that allow data owners to encrypt their data while setting a policy indicating who can decrypt this data. There are two types of attribute-based encryption schemes: Key-Policy Attribute-Based Encryption (KP-ABE), and Cipher-Policy Attribute-Based Encryption (CP-ABE).

In the key-policy attribute based encryption (KP-ABE) schemes [59] [60] [61], the message is encrypted and a ciphertext with a set of attributes is generated. The decryption of the message is achieved using a secret key with an access structure if the access structure is satisfied by the set of ciphertext attributes. On the other hand, with regard to cipher-policy attribute-based encryption (CP-ABE) schemes [53] [62] [63] [64], a set of attributes is associated with a secret key, while an access structure (ciphertext policy) is associated with a ciphertext. The decryption of the message is achieved using a secret key with a set of attributes if the secret key’s attribute set satisfies the access structure associated with the ciphertext.

In this paper, we utilize CP-ABE to preserve the confidentiality of the data mining queries and the outsourced data (i.e. the data index hosted on the cloud). Our proposed framework requires the CP-ABE scheme to support attributes with multiple values, including the wildcard functionality (to indicate that certain attributes are not relevant to the ciphertext policy), while hiding the details of the access structures associated with ciphertexts. A good candidate satisfying the aforementioned properties is the CP-ABE scheme proposed in [57]. This scheme is secure under the Decisional Bilinear Diffie-Hellman (DBDH) assumption [65] and the Decision Linear (D-Linear) assumption [66], constructed in the multi-valued attribute setting, supports wildcards, and allows the access structure to be expressed in conjunctive normal form (i.e. conjunction - AND of disjunctions - OR). It also prevents the decryptor from obtaining information about the access structure by hiding what values for each attribute is specified in the conjunction of all the attributes.

Given a set of attributes \(\{A_1, \ldots, A_i, \ldots, A_n\}\), where each attribute \(A_i\) has a domain \(\Omega(A_i) =\)
\{v_{i,1}, \ldots, v_{i,j}, \ldots, v_{i,n}\}\), the scheme can be constructed according to the following four algorithms:

**Setup**(1). A trusted authority first runs Gen(1^k) to generate a tuple \(\{p, G, G_T, g, e\}\), where \(G, G_T\) are cyclic groups, \(e : G_1 \times G_2 \rightarrow G_T\) is a bilinear map, and \(\omega \in \mathbb{F}_p\). For each attribute \(A_i : 1 \leq i \leq n\), generate a tuple \((\mathcal{A}_{i,j}^{+}, \mathcal{A}_{i,j}^{-}, \mathcal{A}_{i,j}^0)\) of \(\mathbb{Z}_p^n\) and point \(\mathcal{A}_{i,j} \in G\) for each attribute value \(v_{i,j} \in \Omega(A_i) : 1 \leq j \leq \#(\Omega(A_i))\).

The algorithm outputs public key \(PK = \langle Y, p, G, G_T, g, e, \{(\mathcal{A}_{i,j}^{+}, \mathcal{A}_{i,j}^{-}, \mathcal{A}_{i,j}^0)\}_{1 \leq j \leq \#(\Omega(A_i))}\rangle\), where \(Y = e(g, g)^{\omega}\). It also outputs master secret key \(MSK = \langle \omega, \{(\mathcal{A}_{i,j}^{+}, \mathcal{A}_{i,j}^{-}, \mathcal{A}_{i,j}^0)\}_{1 \leq j \leq \#(\Omega(A_i))}\rangle\).

**KeyGen**(MSK, L). This algorithm takes MSK and a set of attribute values \(L = \{v_{i,1}, \ldots, v_{i,t}, \ldots, v_{i,n}\}\), where \(v_{i,t} \in \Omega(A_i)\), and outputs a user’s secret key \(SK_L = \langle D_0, \{D_{i,0}, D_{i,1}, D_{i,2}\}_{1 \leq i \leq n}\rangle\), where \(D_0 = g^{\omega - s}\), \(D_{i,0} = g^{a_{i,0} + \sum_{j=1}^{t} A_{i,j}^{+} - \sum_{j=1}^{t} A_{i,j}^{-}}\), \(D_{i,1} = g^{a_{i,1} - \lambda_i} \cdot g^{\sum_{j=1}^{t} A_{i,j}^{+} \cdot \lambda_i} + s_i \cdot \lambda_i \in \mathbb{Z}_p^n\) for \(1 \leq i \leq n\) such that \(s = \sum_{i=1}^{n} s_i\). This algorithm will be used to encrypt user queries and generate corresponding system queries that will be executed against the encrypted data on the cloud.

Enc\((PK, M, W)\). This algorithm takes public key \(PK\) and access structure \(W = \{W_1 \land \ldots \land W_i \land \ldots \land W_n\}\), and encrypts a message \(M \in \mathcal{T}\). The result is a ciphertext \(CT = \langle \tilde{C}, C_0, \{C_{i,1,1}, C_{i,1,2}\}_{1 \leq i \leq n}\rangle\), where \(C_0 = g^{Y^r}\), and \(C_0 = g^{Y^r}\). If \(v_{i,j} \in W_j\), then \(C_{i,1,1} = \prod_{1 \leq i \leq n} \mathsf{Enc}(g^{a_{i,1} - \lambda_i}, (\mathcal{A}_{i,j}^{+}, \mathcal{A}_{i,j}^{-}, \mathcal{A}_{i,j}^0))\) (well-formed group elements), where \(r_{i,j} \in \mathbb{Z}_p^n\). If \(v_{i,j} \notin W_j\), then \(C_{i,1,1}, C_{i,1,2} \in \mathbb{Z}_p^n\) (mal-formed group elements).

Dec(\(CT, SK_L\)). This algorithm decrypts ciphertext \(CT\) using user’s secret key \(SK_L\) as follows:

\[
M = \tilde{C} \cdot \prod_{1 \leq i \leq n} \mathsf{Dec}(C_{i,1,1}, C_{i,1,2}) \cdot \mathsf{Dec}(C_0, D_{i,0})\]

3.2 Exponential ElGamal

ElGamal [67] is a public key encryption scheme based on the hardness of computing discrete logarithms (and a related assumption called decisional Diffie-Hellman). We use the exponential variant [31]. It consists of the following algorithms:

KeyGen(). A 2048-bit safe prime \(p\) is chosen randomly such that \(p = \alpha \cdot q + 1\) for a 256-bit prime \(q\) and some integer \(\alpha\) (parameter sizes comply with current NIST recommendations [68]). Let \(g\) be a generator of the multiplicative subgroup \(\mathbb{G}_q\). The private key \(x\) is chosen randomly from \(\mathbb{Z}_q^n\) and the public key \(y = g^x \mod p\) (henceforth, assume all operations are done \(\mod p\)).

Enc\((m, y, r)\). To encrypt a short message \(m\) with public key \(y\), choose random integer \(r\) from \(\mathbb{Z}_q^n\) and compute the ciphertext as \(c = (c_1, c_2) = (g^r, y^m)\).

Dec\((c, x)\). To decrypt ciphertext \(c\) with private key \(x\), first compute \(y^m = c_2 \cdot c_1^{-x}\) and solve for \(m\) (recall it is short) using a lookup table of pre-computed values or appropriate algorithm (such as Pollard’s rho).

Exponential ElGamal is additively homomorphic, meaning a given encrypted message \(\mathsf{Enc}(m, y, r) = (c_1, c_2)\) can be added to a second encrypted message \(\mathsf{Enc}(m', y, r') = (c_1', c_2')\) without decryption: \(\mathsf{Enc}(m + m', y, r + r') = (c_1 \cdot c_1', c_2 \cdot c_2')\).

\(\mathsf{Enc}(m, y, r)\) can also be multiplied by a constant \(\alpha\) homomorphically: \(\mathsf{Enc}(\alpha m, y, r') = (\alpha c_1, c_2')\).

In the rest of the paper we will refer to the ACP-ABE scheme as \(A\), and to the Exponential ElGamal scheme as \(G\).

4 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 4.1. Next, we define the input components in Section 4.2. We then describe the trust and adversarial model in Section 4.3. Finally, we present the problem statement in Section 4.4.

4.1 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 \(\varepsilon\)-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 to process count queries from authorized data miners confidentially and return results that provide a certain privacy guarantee.

4.2 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 \(\varepsilon\)-differential privacy model [4], although our approach supports other privacy models that produce contingency-like tables based on generalization and suppression. We choose \(\varepsilon\)-differential privacy because it provides a strong privacy guarantee while being insensitive to any specific record. We first describe how to generate \(\varepsilon\)-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.
In this section we review how a data provider can generate $\varepsilon$-differentially private records. We utilize the differentially private anonymization algorithm (DiffGen) \cite{dwork2014} 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 $\varepsilon$-differentially private data; $A^{cls}$ is a class attribute that contains the class value; and $A^{pr}$ 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 $T^A_i$ is used to order the hierarchy among the domain values of $A_i$. Figure \ref{fig:datatransformation} 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 $\hat{D}$ of the data table $D$, where $A^{pr}$ is the set of $k$ generalized predictor attributes, and $NCount$ 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 $A^{pr}$.

**DiffGen Algorithm.** The general idea is to anonymize the raw data $D$ by a sequence of specializations, starting from the topmost general state. A specialization, written $v \rightarrow \text{child}(v)$, where child$(v)$ denotes the set of child values of $v$, replaces the parent value $v$ with child values. The specialization process can be viewed as pushing the cut of each taxonomy tree downwards. A cut of the taxonomy tree for an attribute $A_i \in A^{pr}$, denoted by $\text{Cut}(T^A_i)$, contains exactly one value on each root-to-leaf path. In Figure \ref{fig:datatransformation}, the dashed curve represents the solution cut. The specialization starts from the topmost cut and pushes down the cut iteratively by specializing a value in the current cut.

**Example 1.** Consider the raw data set in Figure \ref{fig:datatransformation}. Initially, the algorithm creates one root partition containing all the records that are generalized to \{\text{Country, Job, } [18-45], [18-99]\}. $\text{Cut}(T^A_i)$ is updated to $\{\text{Country, Professional, Artist, } [18-65], [18-99]\}$. Suppose that the next specialization is $[18-65) \rightarrow \{18-40, [40-65]\}$, which creates further specialized partitions. Finally, the algorithm outputs the equivalence groups of each leaf partition with their noisy counts as shown in Table \ref{tab:table2}.

![Fig. 2. A raw data table $D$ and its taxonomy trees.](image2)

![Fig. 3. Algorithm DiffGen for generating $\varepsilon$-differentially private data with noisy counts.](image3)

**Fig. 4. Taxonomy tree $T^{Job}$ for attribute Job.**

![TABLE 2 Differentially-private data table $\hat{D}$](image4)

![Solution Cut](image5)

**4.2.2 Input Data Transformation**

We simplify the representation of the $\varepsilon$-differentially private records $\hat{D} = \{A^{pr}, NCount\}$ 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 $A_i \in A^{pr}$ consists of a set of ranges that are pairwise 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 $r_i$, if $r_{max} < r_{min}$ then $ID^{op}(r_j) < ID^{op}(r_i)$. For example, if the domain of the generalized attribute Age is $\Omega(\text{Age}) = ([18, 45], [45, 65])$, then $ID^{op}([18, 45]) = 1$ and $ID^{op}([45, 65]) = 2$.

**Categorical Attributes.** The domain of each categorical attribute $A_i \in A^{pr}$ consists of the set of values $\text{Cut}(T^{A_i})$. 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 \ref{fig:datatransformation} illustrates the taxonomy tree $T^{Job}$ for attribute Job, where each node is assigned an identification value.

Having defined the mapping functions $ID^{op}$ and $ID^t$, we now transform the $\varepsilon$-differentially private records $\hat{D}$ by mapping the values in the domain of each attribute to their identifiers. That is, for each numerical attribute $A_i \in A^{pr}$, we map each range $r \in \Omega(\hat{A}_i)$ to its corresponding identification value $ID^{op}(r)$. Similarly, for each categorical attribute $A_i \in A^{pr}$, we map each value in $v \in \Omega(\hat{A}_i)$ to its identification value from the taxonomy tree $ID^t(v)$. Table \ref{tab:table3} shows the differentially private data $\hat{D}$ after the transformation.
Definition 4. \( \varepsilon \) the noisy count of a query over the generalized attributes \( A^p \).

The goal of the data miners is to build a classifier based on any data mining’s count query, and it is formally defined as follows:

\[
\text{Definition 2. (User Count Query.)} \quad \text{A user count query } u \text{ over } \hat{D} \text{ is a conjunction of predicates } P_1 \land \ldots \land P_m \text{ where each predicate } P = (A_i \text{ Op } s_i) \text{ } \not\equiv \text{ a single criterion such that } A_i \in A^p, \text{ Op is a comparison operator, and } s_i \text{ is an operand. If } A_i \text{ is a categorical attribute, then } Op \text{ expresses a single criterion such that } = \text{ and } s_i \text{ is a value from the taxonomy tree } T^{A_i}. \text{ If } A_i \text{ is a numerical attribute, then } s_i \text{ is a numerical range } [s_{i\text{ min}}, s_{i\text{ max}}] \text{ such that if } s_{i\text{ min}} = s_{i\text{ max}} \text{ then } Op \in \{>, \geq, <, \leq, =\}; \text{ otherwise, } Op \text{ is the equal operator } (=). 
\]

In general, a user count query 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 \( \varepsilon \)-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 \( P = (A_i \text{ Op } s_i) \in u, s_i \in \Omega(A_i). \)

Definition 3. (Specific User Count Query.) A user count query \( u \) is specific if for each predicate \( P = (A_i \text{ Op } s_i) \in u: \)
1. If \( A_i \) is categorical, then \( s_i \in \Omega(A_i). \)
2. If \( A_i \) is numerical, then \( s_i \in \Omega(A_i) \) or there exists exactly one range \( r \in \Omega(A_i) \) where \( s_i \cap r \neq \emptyset \) and \( s_i \not\in r. \)

Definition 4. (Generic User Count Query.) A user count query \( u \) is generic if for each predicate \( P = (A_i \text{ Op } s_i) \in u: \)
1. If \( A_i \) is categorical, then \( s_i \in T^{A_i}. \)
2. If \( A_i \) is numerical, then \( \exists r_j, r_l \in \Omega(A_i) \) such that \( s_i \cap r_j \neq \emptyset, s_i \cap r_l \neq \emptyset, \) and \( r_j \neq r_l. \)

Example 2. The following are examples of user count queries over the \( \varepsilon \)-differentially private data \( \hat{D} \) presented in Table 2

<table>
<thead>
<tr>
<th>Country</th>
<th>Job</th>
<th>Age</th>
<th>Salary</th>
<th>NCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

4.2.3 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 \( A^p \). Therefore, they submit count queries to be processed on the \( \varepsilon \)-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:

4.4 Problem Statement

Given \( \varepsilon \)-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 \( D \). The framework must provide three levels of security: (1) data confidentiality, where \( D \) is stored in an encrypted form such that no useful information can be disclosed from \( D \) 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.

5 SecDM Framework Solution

In this section we first present an overview of our proposed privacy-preserving framework for confidential query processing on \( \varepsilon \)-differentially private data in the cloud, and then we elaborate on the key steps of the algorithm, including constructing a secure index, securing the data for outsourcing, and executing count queries while preserving their privacy.

5.1 Solution Overview

The objective of our solution is to provide a secure framework that enables data providers to outsource their \( \varepsilon \)-differentially private data \( \hat{D} \) to a service provider (public cloud) such that the confidentiality of the outsourced data is protected, while the service provider is capable of securely answering count queries from authorized data miners without being able to infer any information about the queries and their results. Our framework consists of five algorithms:

Algorithm 1 - Secure Index Construction (buildIndex): For an efficient data retrieval, a secure \( kd \)-tree index is constructed over all categorical and numerical attributes in \( \hat{D} \), where each non-leaf node is encrypted using the ACP-ABE scheme A.

Algorithm 2 - Leaf Node Construction (constLeafNode): Utilized by the buildIndex procedure to construct the leaf nodes in the \( kd \)-tree index. Each leaf node contains a noisy count encrypted using Exponential ElGamal and a set of tags to be utilized during query execution for determining the exact percentage that should be used of the noisy count of each reported leaf node.

Algorithm 3 - Query Preprocessing (qPreprocess): A user (data miner) desiring to query the outsourced data first submits a
query \( u \) to the data provider that preprocess the query and sends back three components: an encrypted version of the query in the form of a secret key \( SK_u \) using anonymous ciphertext-policy attribute based encryption scheme \( \mathbb{A} \), a set of \( ADT \) tokens for producing accurate count results, and a decryption key \( G.x \). The user then submits the encrypted query \( SK_u \) and the \( ADT \) tokens to the service provider.

Algorithm 5 - Index Traversal (traverseIndex): The service provider utilizes the user’s secret key \( SK_u \) in order to securely traverse the index tree and determine the leaf nodes satisfying \( SK_u \).

Algorithm 7 - Total Count Computation (compTCount): Using the set of \( ADT \) tokens, the service provider computes the percentage of the noisy count of each reported leaf node, homomorphically add all counts together, and then sends the encrypted result to the user.

5.2 Secure Index Construction

Given the \( \varepsilon \)-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 [40] 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.

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

**Input:** Shuffled \( \varepsilon \)-differentially private data \( \hat{D} \)
**Input:** split dimension \( i \)
**Input:** ACP-ABE public key \( PK \)
**Input:** Exponential ElGamal public key \( y \)

**Output:** \( kd \)-tree index \( T \)

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

5.2.1 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 $\text{cut}$ is determined by the median function (Line 4). Node $v$ has two child nodes, namely, $l_c$ and $r_c$, where all records containing values smaller or equal to the $\text{cut}$ value with regard to $\hat{A}_i$ will appear in the left subtree, whose root is $v.l_c$, and all records containing values greater than the $\text{cut}$ value with regard to $\hat{A}_i$ will appear in the right subtree, whose root is $v.r_c$. 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 ($\text{Ptr}$) to the child node $v.l_c$, and the encrypted message in $v.CT_{right}$ is a pointer to the child node $v.r_c$. 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 3.1. 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 $A_\ast$, where each ciphertext has an access structure $W$. Each numerical attribute $\hat{A}_i \in D$ is represented in the access structure of a ciphertext by two attributes, $\hat{A}_i^{\text{min}}$ and $\hat{A}_i^{\text{max}}$, where $\Omega(\hat{A}_i^{\text{min}}) = \Omega(\hat{A}_i^{\text{max}}) = \Omega(\hat{A}_i)$. 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 CACP-ABE ciphertext.

**Definition 5.** (Ciphertext Access Structure $W$.) Given $\varepsilon$-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 $W = \{W_{\hat{A}_1} \wedge \ldots \wedge W_{\hat{A}_i} \wedge \ldots \wedge W_{\hat{A}_d}\}$. 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 "$W_{\hat{A}_i} = \ast"$ means that attribute $\hat{A}_i$ should be ignored. If $\hat{A}_i$ is a numerical attribute, then $W_{\hat{A}_i} = W_{\hat{A}_i}^{\text{min}} \wedge W_{\hat{A}_i}^{\text{max}}$, where $W_{\hat{A}_i}^{\text{min}}$ and $W_{\hat{A}_i}^{\text{max}}$ 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 $\text{val} \in \{\Omega(\hat{A}_i) \cup \{1\}\}$ such that $\text{val} \leq v.split\_value$ and for $\text{val} = 1$, 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 $\text{val} \in \{\Omega(\hat{A}_i) \cup \{1\}\}$ such that $\text{val} > v.split\_value$ or for $\text{val} = 1$. On the other hand, if $v.split\_dim$ is a numerical attribute $\hat{A}_i$, then $W_{\hat{A}_i}^{\text{min}}$ in the access structure of $v.CT_{left}$ should correspond to the disjunction of all values $\text{val} \in \Omega(\hat{A}_i)$ for all $\text{val} \leq v.split\_value$, and $W_{\hat{A}_i}^{\text{max}}$ in the access structure of $v.CT_{right}$ should correspond to the disjunction of all values $\text{val} \in \Omega(\hat{A}_i)$ such that $\text{val} > v.split\_value$. Regardless of whether $\hat{A}_i$ is categorical or numerical, all values in $\{\Omega(\hat{A}_i) \setminus \hat{A}_i\}$ should correspond to the disjunction of all values $\text{val} \in \Omega(\hat{A}_i)$ should correspond to the disjunction of all values $\text{val} \in \Omega(\hat{A}_i)$ such that $\text{val} > v.split\_value$. Regardless of

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

a) If $v.split\_dim = \text{Job}$ (categorical) and $v.split\_value = 2$, then the access structure of the left and right ciphertexts can be represented as follows:

$W_L = (\text{Country} = \ast) \land (\text{Job} = 1 \lor \text{Job} = 2) \land (\text{Age}_{\text{min}} = \ast) \land (\text{Age}_{\text{max}} = \ast) \land (\text{Salary}_{\text{min}} = \ast) \land (\text{Salary}_{\text{max}} = \ast)$.

$W_R = (\text{Country} = \ast) \land (\text{Job} = 1 \lor \text{Job} = 3) \land (\text{Age}_{\text{max}} = \ast) \land (\text{Salary}_{\text{min}} = \ast) \land (\text{Salary}_{\text{max}} = \ast)$.

b) If $v.split\_dim = \text{Age}$ (numerical) and $v.split\_value = 1$, then the access structure of the left and right ciphertexts are:

$W_L = (\text{Country} = \ast) \land (\text{Job} = \ast) \land (\text{Age}_{\text{min}} = 1) \land (\text{Age}_{\text{max}} = \ast) \land (\text{Salary}_{\text{min}} = \ast) \land (\text{Salary}_{\text{max}} = \ast)$.

$W_R = (\text{Country} = \ast) \land (\text{Job} = \ast) \land (\text{Age}_{\text{max}} = 2) \land (\text{Salary}_{\text{min}} = \ast) \land (\text{Salary}_{\text{max}} = \ast)$.

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 $\text{genCT}$ is responsible for creating the left ciphertext $v.CT_{left}$ and the right ciphertext $v.CT_{right}$ of the node by calling twice the ACP-ABE algorithm $A_\ast.\text{Enc}()$ and passing as parameters the public key $PK$ of $A_\ast$, a pointer to the child node to be encrypted, and the values in the access structure (without the attribute names):

$v.CT_{left} \leftarrow A_\ast.\text{Enc}(PK, \text{Ptr}(v.l_c), W_L)$;

$v.CT_{right} \leftarrow A_\ast.\text{Enc}(PK, \text{Ptr}(v.r_c), W_R)$;

For each attribute $\hat{A}_i$ in $W$ that is assigned a wildcard, e.g. $(\text{Country} = \ast)$, $A_\ast.\text{Enc}()$ generates a random (mal-formed) group elements $\{C_{ij,1}, C_{ij,2}\}$ for each value in $\Omega(\hat{A}_i)$. On the other hand, for each attribute in $W$ assigned specific values, e.g. $(\text{Job} = 1 \lor \text{Job} = 2)$, $A_\ast.\text{Enc}()$ 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 $\Omega(\text{Job})$. As a result, all ciphertexts $CT$ generated by $A_\ast.\text{Enc}()$ in the $kd$-tree index contain the same number of group elements regardless of the access structure.

**5.2.2 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 $\varepsilon$-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 $G$ 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 deterministic and hiding commitment function $\text{genTAG}()$ is utilized to commit $R.\hat{A}_i$ and randomly generate a unique tag (Line 3-4). Applying $\text{genTAG}()$ to the same value will always generate the same tag (deterministic). Moreover, the correspondence between each tag and its value is kept secret (hiding). As we will see in
Definition 6. (System Count Query). Given \( \varepsilon \)-differentially private data \( \hat{D} \) with \( k \) attributes and a user query \( u = P_1 \land \ldots \land P_m \mid P_i = (\hat{A}_i, Op, s_i) \), a system count query over \( \hat{D} \) is a ACP-ABE user’s secret key \( SK_u \) representing \( k \) subqueries \( \{q_{\hat{A}_1}, \ldots, q_{\hat{A}_k}\} \) 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 3.1 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 \( (\hat{A}_i, Op, s_i) \) 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 \( T^{\hat{A}_i} \) (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 categorical attribute \( \hat{A}_i \in \hat{D} \), then the subquery \( (\hat{A}_i, 1) \) is
Fig. 6. ADT query overlap.

added to $q$ (Lines 10-11), where 1 means “ANY” value of $A_i$ corresponding to the root node of $A_i$’s taxonomy tree $T^{A_i}$.

On the other hand, if $A_i$ is a numerical attribute and predicate $(A_i, Op, s_i)$ exists in $u$, then the values $v_{i,1}$ associated with $A_i^{\text{min}}$ and $v_{i,2}$ associated with $A_i^{\text{max}}$ are determined by the function $\text{compMinMax}$ (Line 15). When $Op$ is the equal operator (=), if $s_i$ is a single value, then $v_{i,1} = v_{i,2} = \text{Range}(s_i)$, where $\text{Range}(s_i)$ is a function that returns the identifier of the range in $\Omega(A_i)$ containing $s_i$; otherwise, if $s_i$ is a range, then $v_{i,1} = \text{Range}(\text{Lowerbound}(s_i))$ and $v_{i,2} = \text{Range}(\text{Upperbound}(s_i))$. If $Op$ is “$\geq$”, then $v_{i,1} = \text{Range}(s_i)$ and $v_{i,2}$ is the identifier of the highest range in $\Omega(A_i)$. Conversely, if $Op$ is “$\leq$”, then $v_{i,1} = 1$ and $v_{i,2} = \text{Range}(s_i)$. If predicate $(A_i, Op, s_i)$ does not exist in $u$ for numerical attribute $A_i$ (Lines 19-20), then $v_{i,1} = 1$ and $v_{i,2}$ is the identifier of the highest range $\text{range}_{\text{max}} \in \Omega(A_i)$.

**Example 4.** 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) $u = (\text{Age} = 50) \Rightarrow q = (\text{Age}^{\text{min}}, 2), (\text{Age}^{\text{max}}, 2)$.

b) $u = (\text{Age} = [40 - 70]) \Rightarrow q = (\text{Age}^{\text{min}}, 1), (\text{Age}^{\text{max}}, 2)$.

c) $u = (\text{Age} \leq 35) \Rightarrow q = (\text{Age}^{\text{min}}, 1), (\text{Age}^{\text{max}}, 1)$.

Function $\text{genADT}$ (Line 18) is used to generate attribute distribution tokens (ADT) for each numerical attribute $A_i$ from $D$. Two ADT tokens, $A^{\text{min}}$ and $A^{\text{max}}$, 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 $ADT^{\text{tag}} = \text{genTAG}(r)$ and $ADT^{\text{value}}$ is the percentage of the partial overlap between query $u$ and range $r$.

**Example 5.** Assume that in $\varepsilon$-differentially private data $D$, $\Omega(\text{Age}) = \{(18 - 30), (30 - 45), (45 - 55), (55 - 65)\}$, $\Omega(\text{Salary}) = \{(30 - 45), (45 - 60), (60 - 70)\}$, and user count query $u = (\text{Country} = \text{"US"}) \wedge (\text{Job} = \text{"Engineer"}) \wedge (\text{Age} = [25 - 49]) \wedge (\text{Salary} = [47 - 70])$. Figure 3 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 $\text{Age} = [25 - 49]$ 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 $[45 - 55]$, $A^{\text{min}}$ and $A^{\text{max}}$ should be created. For range $\text{Age} = [18 - 30]$, $ADT^{\text{min}}.tag = \text{genTAG}(18 - 30)$ and $ADT^{\text{min}}.value = \frac{30 - 25}{50 - 18} = 42\%$. Similarly, for range $\text{Age} = [45 - 55]$, $A^{\text{max}}.value = \frac{45 - 55}{65 - 45} = 50\%$. On the other hand, $\text{Salary} = [47 - 70]$ partially overlaps with ranges $[45 - 60]$ and $[60 - 75]$ and $A^{\text{min}}$ and $A^{\text{max}}$ must be created. For range $\text{Salary} = [45 - 60]$, $A^{\text{min}}.tag = \text{genTAG}(45 - 60)$ and $A^{\text{min}}.value = \frac{45 - 45}{60 - 45} = 87\%$. Similarly, for range $\text{Salary} = [60 - 75]$, $A^{\text{max}}.tag = \text{genTAG}(60 - 75)$ and $A^{\text{max}}.value = \frac{75 - 60}{75 - 60} = 67\%$.

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 $\text{A.KeyGen}$ (Line 23). Next, the data provider sends the following back to the user: secret key $SK_u$, the set of ADT tokens $N$, and ElGamal decryption key $G.x$ that will be used eventually to decrypt the final result of the query.

5.3.2 $kd$-tree Index Traversal

To execute a query $u$ on $D$, the data miner sends to the service provider a system count query $SK_u$ and a set of ADT tokens $N$. 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 $N$ 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 ($\text{traverseIndex}$) 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 $v.CT_{\text{left}}$ and the right ciphertext $v.CT_{\text{right}}$ by separately applying the decryption function Dec from $\text{A}$, 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 $v.CT_{\text{left}}$, then the decryption of $v.CT_{\text{left}}$ is successful and the procedure $\text{traverseIndex}$ 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 $v.CT_{\text{right}}$, then the decryption is successful and the procedure $\text{traverseIndex}$ calls itself while passing the right child node $v.rc$ as input parameter (Line 7-8). When the algorithm reaches a leaf node, then $v$ is returned (Lines 1-2). Procedure $\text{traverseIndex}$ eventually returns the set $R$ containing all leaf nodes satisfying $SK_u$ (Line 11).

**Example 6.** Given Example 5 assume that $v$ is the root node where $v.split\_dim = \text{Age} (A_3)$ and $v.split\_value = 2$
ALGORITHM 7: compCTCount: Total Noisy Count Computation

Input: set of leaf nodes \( R \)
Output: ciphertext of total count \( \langle r, s \rangle \)

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

\((\text{range}[30 - 45])\). Figure 7(a) illustrates the access structure of \( v.CT_{\text{left}} \) and \( v.CT_{\text{right}} \). Figure 7(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 \( A_3^{\text{min}} = 3 \) and \( A_3^{\text{max}} = 4 \).

Since \( A_3^{\text{min}} = 3 \) from \( SK_u \) is not in the access structure of \( v.CT_{\text{left}} \), then the decryption is unsuccessful, and the left subtree will not be traversed. However, \( A_3^{\text{max}} = 4 \) from \( SK_u \) is in the access structure of \( v.CT_{\text{right}} \), then the decryption is successful and the procedure \( \text{traverseIndex} \) traverses the right subtree, whose root node is \( v.r_c \).

5.3.3 Computing Total Noisy Count
Having identified the set of leaf nodes \( R \) satisfying user count query \( u \), the next step is to compute the final answer to the count query.

Procedure Total Count Computation (compCTCount) presented in Algorithm 7 illustrates how the total noisy count is computed. It takes as input a set of leaf nodes \( R \) and a set of attribute distribution tokens \( N \). 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 \( \text{compCTCount} \) is the encrypted total count \( \langle r, s \rangle \) (Line 11).

5.3.4 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 \( G.x \) to decrypt the ciphertext and determine the exact noisy count \( res_u \) such that \( res_u \) satisfies differential privacy.

5.4 Discussion
SecDM 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 \( G.x \), 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.
6.1 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 $O(k \times d \times \log d)$ operations.

**Proof.** Constructing a $kd$-tree with $d$ points (equivalent classes) requires $O(d \times \log d)$ [72]. Each node consists of two ciphertexts, each of which requires $O(k_c + 2 \times k_n) = O(k)$, where $k_c$ and $k_n$ are number of categorical attributes and numerical attributes respectively. Therefore, the required number of operations is $O(k \times d \times \log d)$.

**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 $O(\sqrt{d} + r \times k)$ operations, where $r = |R|$ and $R$ 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 $O(\sqrt{d} + r)$ [72]. For each reported leaf node, $O(2 \times k_c) = O(k)$ 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 $O(\sqrt{d} + r \times k)$.

6.2 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 3.** Given a user count query $u = P_1 \land ... \land P_m$, Algorithm 6 produces a set $R$ 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 $D$, 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 $A_i \in D$, $A_i^{min} = 1$ (the lowest range identifier) and $A_i^{max} = k$ is the highest range identifier in $\Omega(A_i)$. 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, $A.\text{Dec}(v.CT_{left}, SK_u)$ and $A.\text{Dec}(v.CT_{right}, SK_u)$ 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 $R$ satisfying $u$. We show that if a new predicate $P = (A_i \cdot Op \ s_i)$ is added to $u$ such that $\hat{u} = u \cup P$, then traversing the $kd$-tree index using $SK_{\hat{u}}$ produces the correct set of leaf nodes $\hat{R}$ satisfying $\hat{u}$. We observe that $R \subseteq \hat{R}$. To complete the proof in this step, we assume that $P$ 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 $v.split_{dim} = A_i$, if $s_i.ID \leq v.split_{value}$ then $A.\text{Dec}(v.CT_{right}, SK_{\hat{u}})$ will evaluate to false, and no recursive call of procedure $\text{traverseIndex}$ 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 $P$, 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 $kd$-tree index while reducing the search space by half until all leaf nodes satisfying $u$ are reached.

**Proposition 4.** Given a set of leaf nodes $R$ 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 $\text{compTCount}$ will go through all the leaf nodes in $R$ 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 $R$ must be used.

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

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.

Proposition 5. The noisy count answers satisfy \( \varepsilon \)-differential privacy.

Proof. The proposed query processing algorithms operate on a differentially private data table and do not have access to the raw data. Because the input table is differentially private, the computed noisy count answers based on the input data table is also differentially private. Note that any post-processing does not violate the \( \varepsilon \)-differential privacy [73].

6.3 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 [44] defines the security of a protocol in the semi-honest adversarial model as follows.

Definition 7. (Privacy w.r.t. Semi-honest Behavior) [44]. Let \( f : (\{0,1\})^m \rightarrow (\{0,1\})^m \) be an \( m \)-ary deterministic polynomial-time functionality, where \( f_1(x_1, \ldots, x_m) \) is the \( i \)-th element of \( f(x_1, \ldots, x_m) \). Let \( \Pi \) be an \( m \)-party protocol for computing \( f \). The view of the \( i \)-th party during an execution of \( \Pi \) over \( x = (x_1, \ldots, x_n) \) is \( \text{view}^i_{\Pi}(x) = (x_i, r_i, m_{i,1}, \ldots, m_{i,t}) \), 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 \( I = \{i_1, \ldots, i_u\} \subseteq \{1, \ldots, m\} \), \( \text{view}^I_{\Pi}(x) = (I, \text{view}^{i_1}_{\Pi}(x), \ldots, \text{view}^{i_u}_{\Pi}(x)) \). We say that \( \Pi \) securely computes \( f \) in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithm (simulator) \( S \) such that for every \( I \subseteq \{1, \ldots, m\} \):

\[
\{S(I, (x_{i_1}, \ldots, x_{i_u}), f_I(x))\} \neq \{\text{view}^I_{\Pi}(x)\} \quad x \in \{0,1\}^m
\]

where \( \equiv \) 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. The algorithm can then be utilized to simulate the rest of the view.

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 \( \alpha \), 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 \( F \) be a framework that enables a service provider (cloud) to answer queries from data miners on hosted (outsourced) data. \( F \) 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 \( E \), statistically corrupting the service provider, cannot learn anything more about the hosted data from an accepted transcript of \( F \) 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 \( E \), statistically 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. ((\( \alpha \))-Privacy-preserving Data Outsourcing Framework.) An outsourcing framework \( F \) is \( \alpha \)-privacy-preserving if it satisfies all properties in Definition 10 except that the cloud learns the search pattern of at most \( \alpha \) number of queries.

Theorem 6.1. SecDM, as specified in Protocols 1 and 7, is an \( \alpha \)-privacy-preserving data outsourcing framework.

Proof. We proved in Section 6.2 Property 1 (correctness) and Property 4 (differentially private output).

To prove Property 2 (data Confidentiality) and Property 3 (query Confidentiality), we build a simulator \( S \) that generates a view that is statistically indistinguishable from the view of \( E \) in real execution.

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 \( D \), and the size of each attribute’s domain \( |\Omega(A_i)| \) : \( 1 \leq i \leq k \), the simulator \( S \) generates attribute domains \( \Omega(A_1), \Omega(A_2), \ldots, \Omega(A_k) \) such that each domain \( \Omega(A_i) \) consists of \( |\Omega(A_i)| \) distinct values, e.g., \( 1, 2, \ldots, |\Omega(A_i)| \).

2) \( S \) constructs a contingency table \( D' \) with \( k \) columns each of which represents one attribute \( A_i \), and \( n \) records each of which represents one possible combination of attribute values such that \( n = \prod_{i=1}^{k} |\Omega(A_i)| \).

3) Supplied with the total number of numerical attributes \( k_n \) and categorical attributes \( k_c \) in \( D \) such that \( k_n + k_c = k \), the size of each attribute’s domain, and the security parameter of ACP-ABE, \( S \) runs A.Setup(1^{\lambda}) to generate public key \( PK' \).
4) Given $T''$, split dimension $i = 1$, $PK'$ and $y'$, $S$ runs Algorithm 1 and Algorithm 2 to construct a balanced $kd$-tree $T''$ over $D''$

\begin{enumerate}
\item In Line 12 of Algorithm 1, $n$ random group elements are generated for each ciphertext $CT_{left}$ or $CT_{right}$ of each internal node $v$.
\item 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 $l$.
\end{enumerate}

**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 $D'' = \tilde{D}$ constructed using Algorithm 1 and Algorithm 2 where:

\begin{enumerate}
\item The ACP-ABE ciphertexts $CT_{left}$ and $CT_{right}$ of each internal node are random group elements, as per Step 2 above.
\item The noisy count $NCount$ in each leaf node is a random ElGamal ciphertext, as per Step 1 above.
\end{enumerate}

2) $T''$ is computationally indistinguishable from $T$, denoted by $T'' \equiv T$ because:

\begin{enumerate}
\item The ACP-ABE ciphertexts in the internal nodes of the $kd$-tree are IND-CPA-secure under the decisional bilinear Diffie-Hellman (DBDH) assumption [65] and the decisional linear (D-Linear) assumption [66].
\item 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 $NCount$.
\end{enumerate}

3) $T''$ is statistically indistinguishable from $T''$, denoted by $T'' \equiv T$ because:

\begin{enumerate}
\item $D'' \equiv D'$, where there is one-to-one correspondence between the equivalent classes in $D''$ and the records in $D'$.
\item The random coins used in ACP-ABE encryption in Algorithm 1 are drawn from the same distribution.
\item The random coins used in ElGamal encryption in Algorithm 2 are drawn from the same distribution.
\end{enumerate}

4) From Steps (2) and (3), we conclude that $T'' \equiv T$.

In Algorithm 4, Line 4, the service provider receives system count query $SK_u$ and a set of attribute distribution tokens $N'$.

**Simulation**:

1) $S$ obtains $\alpha$ sample queries $\tilde{U} = \{u'_1, u'_2, \ldots, u'_\alpha\}$ from $U$.
2) For each query $u'_j \in \tilde{U}$, $S$ constructs a query pair $(SK_{u'_j}, N'_j)$ as follows:

\begin{itemize}
\item $S$ runs $A, \text{KeyGen}(MSK', u'_j)$ to construct system count query $SK_{u'_j}$.
\item $S$ constructs a set $N'_j$ containing $2 \times k_0$, ADT tokens, where $ADT.value$ for each token is a randomly generated ElGamal ciphertext, e.g., encryption of ’0’.
\end{itemize}

3) Up to $\alpha$ times, each time a data miner in the real world submits a query, $S$ 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 $SK_u$, $SK_{u'_j} \equiv SK_u$ because:

\begin{enumerate}
\item $u'_j \equiv u$.
\item $SK_{u'_j} \equiv SK_u$ since $|SK_{u'_j}| = |SK_u| = k_0 + 2 \times k_0$ group element tuples, and the ACP-ABE scheme is IND-CPA-secure.
\end{enumerate}

**Discussion**. The threshold parameter $\alpha$ can range between 1 and $\infty$. To better understand the impact of revealing $\alpha$ queries to $S$, 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 $\alpha$ 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 7.2.1.1 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:

\[ |\text{collision set}(l)| = \prod_{i=1}^{r} \prod_{j=1}^{k_i} |l_i.Range(\tilde{A}_j)| \] where $|l_i.Range(\tilde{A}_j)|$ denotes the size of the range of attribute $\tilde{A}_j$ 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 $\alpha$ 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 [74]; 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.

**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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Key</th>
<th>Data Bank</th>
<th>Service Provider</th>
<th>Data Miner</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>private key ( x )</td>
<td>generator, full control</td>
<td>no access</td>
<td>read access</td>
</tr>
<tr>
<td>G</td>
<td>public key ( y )</td>
<td>generator, full control</td>
<td>read access</td>
<td>read access</td>
</tr>
<tr>
<td>A</td>
<td>master secret key ( MSK )</td>
<td>generator, full control</td>
<td>no access</td>
<td>no access</td>
</tr>
<tr>
<td>A</td>
<td>public key ( PK )</td>
<td>generator, full control</td>
<td>read access</td>
<td>read access</td>
</tr>
<tr>
<td>A</td>
<td>user secret key ( SK_x )</td>
<td>generator, full control</td>
<td>read access</td>
<td>read access</td>
</tr>
</tbody>
</table>

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, \( G \cdot x \), 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 \( A \cdot MSK \) 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 \( A \cdot MSK \) prevents her from bypassing authentication and creating her own system count queries.

## 7 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 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.

### 7.1 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 [72]. 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 [75]: \( Y^2 = X^3 + b \), 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 \( \epsilon : G_1 \times G_2 \rightarrow G_T \) 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 \( G_q \) such that \( \text{order}(g) = q - 1 \).

We utilize a real-life adult data set [76] 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 [77]. 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. We generate \( \varepsilon \)-differentially private records using the DiffGen algorithm, where the privacy budget \( \varepsilon = 1 \), the number of specializations is set to 8, 10, or 12, and choose the utility function \( \text{Max}(D, v) \) to determine the score of each candidate \( v \) during the specialization process.

### 7.2 Experimental Results

#### 7.2.1 Scalability

##### 7.2.1.1 Solution Construction 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.

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

<table>
<thead>
<tr>
<th>TABLE 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key accessibility w.r.t. all parties in SecDM framework</td>
</tr>
<tr>
<td>Scheme</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

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

Fig. 12. Efficiency w.r.t. the number of attributes per a query for exact, specific, and generic queries.

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

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).

7.2.2 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 12(a), 12(b), and 12(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.

8 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.

References


