Privacy preserving top-k query processing over outsourced data

Sakina Mahboubi

Collège Doctoral Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S***
Et de l'unité de recherche **UMR 5506**

Spécialité: **Informatique**

Présentée par **Sakina Mahboubi**
sakina.mahboubi@inria.fr

Préservation de la Confidentialité des Données Externalisées dans le Traitement des Requêtes Top-k

Soutenue le 21/11/2018 devant le jury composé de :

M. Nicolas ANCIAUX	Directeur de Recherche	Inria	Rapporteur
M. Omar BOUCELMA	Professeur	Université d'Aix-Marseille	Rapporteur
Mme. Hinde BOUZIANE	Maitre de Conférence	Université de Montpellier	Examinatrice
Mme. Karine ZEITOUNI	Professeur	Université de Versailles Saint-Quentin-en-Yvelines	Présidente
M. Reza AKBARINIA	Chargé de Recherche	Inria	Encadrant
M. Patrick VALDURIEZ	Directeur de Recherche	Inria	Directeur de thèse

* **I2S:** ÉCOLE DOCTORALE INFORMATION STRUCTURES SYSTÈMES

Dedication

To my father and mother
To my fiance
To my brother
To my sisters
To all my friends

Acknowledgments

Firstly, I would like to express my special appreciation and thanks to the **Algerian government** for financing this thesis and giving me the opportunity to continue my graduate studies in France. It was really an occasion for me to improve my skills and research spirit. During the three years of my thesis, and for the first time in my life, I lived by myself far from my family in another country. It was a wonderful experience that I really enjoyed.

I would like to express my gratitude to the members of my PhD committee, **Dr. Nicolas Anciaux**, **Dr. Omar Boucelma**, **Dr. Hinde Bouziane**, and **Dr. Karine Zeitouni**. I highly appreciate their patience as well as the time they took out of their busy schedules to read and evaluate my work.

I am extremely thankful for all those who have helped me with my thesis. Starting by extending my sincere thanks to my advisor, **Dr. Reza Akbarinia**, for encouraging my research and for allowing me to grow as a researcher. His advice on both research as well as on my career have been invaluable and it greatly helped me to keep going, even when it was really tough. It is absolutely difficult to find and develop an idea without the help of a specialist in the domain. I found Reza all the time next to me to realize this. Also, I am deeply grateful to my director **Dr. Patrick Valduriez** for his encouragement, support and help specially to finish writing my thesis, I was honored to work with a great researcher like him.

A special thanks to **my family**. Words can not express how grateful I am to **my father** and **mother**, for all the sacrifices that they have made on my behalf. Their prayers sustained me this far. I am also so thankful to **my fiance** for his encouragement and support during these three years.

I would like to thank **my friends** in France and in Algeria for the wonderful time and quality moments that I spent with them. I wish for each and every one of them the best of luck and success in their future personal and professional projects and look forward to meeting them again.

Last, but not least, I would like to thank every person that I met and helped me to be here today and realize this success. Thanks for you all.

Résumé

L'externalisation de données d'entreprise ou individuelles chez un fournisseur de cloud, par exemple avec l'approche Database-as-a-Service, est pratique et rentable. Mais elle introduit un problème majeur : comment préserver la confidentialité des données externalisées, tout en prenant en charge les requêtes expressives des utilisateurs. Dans cette thèse, nous considérons un type important de requêtes, les requêtes top-k, et attaquons le problème du traitement de ces requêtes sur des données cryptées dans le cloud, tout en préservant la vie privée. Une requête top-k permet à l'utilisateur de spécifier un nombre k de tuples les plus pertinents pour répondre à la requête. Le degré de pertinence des tuples par rapport à la requête est déterminé par une fonction de notation.

Nous considérons deux cas : 1) centralisé : les données chiffrées de l'utilisateur sont stockées au niveau d'un noeud unique d'un data center du cloud. 2) distribué : les données chiffrées de l'utilisateur sont partitionnées verticalement (pour des raisons de performance), et les partitions sont chiffrées et réparties sur plusieurs noeuds.

Nous proposons d'abord un système complet, appelé BuckTop, pour le cas centralisé. BuckTop est capable d'évaluer efficacement les requêtes top-k sur des données cryptées, sans avoir besoin de les décrypter dans le cloud. Il inclut un algorithme de traitement des requêtes top-k qui fonctionne sur les données cryptées, stockées dans un noeud du cloud, et retourne un ensemble qui contient les données cryptées correspondant aux résultats top-k. Il est aidé par un algorithme de filtrage efficace qui est exécuté dans le cloud sur les données chiffrées et supprime la plupart des faux positifs inclus dans l'ensemble renvoyé.

Pour le cas distribué, nous proposons deux nouveaux systèmes, appelés SDB-TOPK et SD-TOPK, qui permettent d'évaluer les requêtes top-k sur des données distribuées cryptées sans avoir à les décrypter sur les noeuds où elles sont stockées. De plus, SDB-TOPK et SD-TOPK ont un puissant algorithme de filtrage qui filtre les faux positifs autant que possible dans les noeuds et renvoie un petit ensemble de données cryptées qui seront décryptées du côté utilisateur. Nous analysons la sécurité de notre système et proposons des stratégies efficaces pour la mettre en oeuvre.

Nous avons validé nos solutions par l'implémentation de BuckTop, SDB-TOPK et SD-TOPK, et les avons comparé à des approches de base par rapport à des données synthétiques et réelles. Les résultats montrent un excellent temps de réponse par rapport aux approches de base. Ils montrent également l'efficacité de notre algorithme de filtrage qui élimine presque tous les faux positifs. De plus, nos systèmes permettent d'obtenir une réduction significative des coûts de communication entre les noeuds lors du calcul du résultat de la requête.

Titre en français

Préservation de la confidentialité des données externalisées dans le traitement des requêtes top-k

Mots-clés

- Confidentialité
- Requêtes top-k
- Données sensibles
- Sécurité du cloud
- Données distribuées

Abstract

Outsourcing corporate or individual data at a cloud provider, *e.g.*, using Database-as-a-Service, is practical and cost-effective. But it introduces a major problem: how to preserve the privacy of the outsourced data, while supporting powerful user queries. In this thesis, we consider an important kind of queries, top-k queries, and address the problem of privacy-preserving top-k query processing over encrypted data in the cloud. A top-k query allows the user to specify a number k, and the system returns the k tuples which are most relevant to the query. The relevance degree of tuples to the query is determined by a scoring function.

We consider two cases: 1) centralized: the user encrypted data are stored at a single node of a cloud data center (i.e. computer cluster) 2) distributed: the user encrypted data are vertically partitioned (for scalability and performance reasons), and the partitions are encrypted and distributed across multiple nodes of a cloud data center.

We first propose a complete system, called BuckTop, for the centralized case. BuckTop is able to efficiently evaluate top-k queries over encrypted data outsourced to a singe node, without having to decrypt it in that node. It includes a top-k query processing algorithm that works on the encrypted data stored in the cloud node, and returns a set that is proved to contain the encrypted data corresponding to the top-k results. It also comes with an efficient filtering algorithm that is executed in the cloud node on encrypted data and removes most of the false positives included in the set returned.

For the distributed case, we propose two new systems, called SDB-TOPK and SD-TOPK, that can evaluate top-k queries over encrypted distributed data without having to decrypt at the nodes where they are stored. In addition, SDB-TOPK and SD-TOPK have a powerful filtering algorithm that filters the false positives as much as possible in the nodes, and returns a small set of encrypted data that will be decrypted in the user side. We analyze the security of our system, and propose efficient strategies to enforce it.

We validated our solutions through implementation of BuckTop, SDB-TOPK and SD-TOPK, and compared them to baseline approaches over synthetic and real databases. The results show excellent response time compared to baseline approaches. They also show the efficiency of our filtering algorithm that eliminates almost all false positives. Furthermore, our systems yield significant reduction in communication cost between the distributed system nodes when computing the query result.

Title in English

Privacy Preserving Top-k Query Processing over Outsourced Data

Keywords

- Privacy
- Top-k query processing
- Sensitive data
- Cloud security
- Distributed data

Équipe de Recherche

Zenith Team, Inria & LIRMM

Laboratoire

LIRMM - Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier

Adresse

Université Montpellier
Bâtiment 5
CC 05 018
Campus St Priest - 860 rue St Priest
34095 Montpellier cedex 5

Résumé étendu

Contexte

L'externalisation de données sensibles dans le cloud [33], avec l'approche "Database-as-a-Service" (DaS), pose une question majeure : comment préserver la confidentialité des données externalisées, qui peuvent être manipulées par les employés du cloud par exemple, lors du traitement des requêtes de l'utilisateur ?

La solution classique consiste à crypter les données avant de les externaliser. Ensuite, le défi consiste à répondre aux requêtes des utilisateurs sur des données cryptées. Une solution naïve est de récupérer toutes les données cryptées depuis le cloud vers la machine cliente, les décrypter, puis évaluer la requête sur des données en clair (non cryptées). Cette solution n'est pas pratique, car elle ne permet pas d'exploiter la puissance de calcul considérable qu'offre le cloud pour évaluer les requêtes.

Dans cette thèse, nous considérons un type important de requêtes : les requêtes top-k. Ainsi, nous adressons le problème du traitement des requêtes top-k sur des données cryptées dans le cloud, tout en préservant la confidentialité.

Selon le rapport "Cloud Security Alliance" [19], la sécurité dans le cloud est l'une des principales préoccupations des utilisateurs. La sécurité dans le cloud désigne un vaste ensemble de politiques et de technologies déployées pour protéger les données, les applications et les infrastructures associées au cloud contre différentes menaces telles que la perte de données, l'accès non autorisé aux données, la violation des données, le déni de service, etc. En général, ces menaces peuvent être regroupées en trois modèles principaux [5], selon le type d'attaquant ou d'adversaire dont le but est d'accéder aux données de l'utilisateur et/ou de les altérer : 1) Le modèle *honnête-mais-curieux (honest-but-curious)*, aussi appelé *semi-honnête* où l'attaquant suit les étapes prescrites dans un protocole. Cependant, l'attaquant peut essayer d'apprendre des informations supplémentaires sur les utilisateurs en se basant sur leurs données, ainsi que sur les entrées, les sorties et les messages reçus pendant l'exécution du protocole sécurisé. 2) Le modèle *malveillant (malicious)* où un adversaire peut s'écarter arbitrairement de l'exécution normale d'un protocole et modifier les données de l'utilisateur. 3) Le modèle *déguisé (covert)* qui se situe entre le modèle honnête-mais-curieux et le modèle malveillant. Plus précisément, un adversaire déguisé peut s'écarter arbitrairement des règles d'un protocole, sans modifier les données de l'utilisateur.

Objectifs

Dans cette thèse, nous supposons le modèle honnête-mais-curieux. Ce modèle est bien adapté au problème du traitement des requêtes dans le cloud [46, 24] comme les fournisseurs du cloud ne sont pas des attaquants malveillants qui peuvent modifier les protocoles ou les données des utilisateurs, mais ils peuvent être intéressés à en savoir plus sur les utilisateurs en accédant à leurs données. Toutefois, cela peut porter atteinte à la vie privée des utilisateurs. La *préservation de la vie privée* ou la *confidentialité des données* sont les principales exigences pour assurer la sécurité des données dans le cloud [80]. De nombreux utilisateurs accordent plus d'attention à la protection de leur vie privée lorsqu'ils accèdent à des données stockées dans le cloud ou utilisent les services du cloud. En particulier, ils s'attendent à cacher leur identité en utilisant les services du cloud. Certains utilisateurs souhaitent également que leurs opérations sur les données et les informations extraites d'un cloud soient correctement protégées. La préservation de la vie privée devrait garantir que toutes les données critiques et sensibles doivent être masquées ou cryptées et que seuls les utilisateurs autorisés ont accès aux données dans leur intégralité.

Nous concentrons notre travail de recherche sur les requêtes top-k. Ce type de requête a un grand intérêt pour divers domaines des technologies de l'information tels que les réseaux de capteurs [91], les systèmes de gestion de flux de données [85], crowdsourcing [18, 95], analyse des données spatiales [75], bases de données temporelles [64], bases de données graphes [31], etc. Une requête top-k permet à l'utilisateur de spécifier un nombre k qui spécifie les k tuples les plus pertinents pour la requête. Le degré de pertinence des tuples à la requête est déterminé par une *fonction de notation (scoring)*. Dans une base de données composée de listes d'éléments de données, chaque élément de données a un score local dans chaque liste. La fonction de notation f est une fonction qui calcule une note globale pour chaque élément de données.

Comme exemple simple de requête top-k, prenons l'exemple d'une université qui externalise sa base de données des étudiants vers le cloud, avec des noeuds non fiables. La base de données est cryptée pour des raisons de confidentialité. Dans ce cas, une requête top-k intéressante sur les données cryptées externalisées est la suivante : retourner les k étudiants qui ont les pires moyennes dans certains cours donnés.

Il y a plusieurs approches pour traiter les requêtes top-k sur des données en clair (non cryptées). L'une des approches les plus importantes est Threshold Algorithm (TA) [27] qui fonctionne sur des listes triées selon les valeurs d'attribut. TA peut trouver efficacement les meilleurs résultats grâce à une stratégie intelligente pour décider quand arrêter de lire la base de données. Toutefois, TA et toutes les autres approches efficaces élaborées pour le traitement des requêtes top-k jusqu'à présent supposent l'existence des valeurs d'attributs des éléments de données en clair *c.-à-d.,* les éléments de données sont représentés sous forme lisible pour l'utilisateur. La confidentialité des données de l'utilisateur n'est donc pas préservée.

L'une des principales solutions pour préserver la confidentialité des données dans le cloud est de crypter les bases de données externalisées. Le cryptage des données est le processus d'encodage d'un message ou d'une information de manière à ce que seules les

parties autorisées puissent y accéder. Cette solution assure une grande confidentialité des données. Cependant, la capacité d'effectuer un traitement pratique des requêtes sur des données cryptées reste un défi majeur [8]. Dans cette thèse, nous sommes intéressés par le traitement des requêtes top-k sur des données cryptées.

Quand on réfléchit au traitement des requêtes top-k sur des données cryptées, la première idée qui vient à l'esprit est d'utiliser un schéma de cryptage entièrement homomorphe, qui permet d'effectuer des opérations arithmétiques sur des données cryptées [28]. L'utilisation de ce type de cryptage permet de calculer le score global des éléments de données par rapport aux données cryptées. Cependant, les méthodes actuelles de cryptage entièrement homomorphique sont très coûteuses en termes de temps de cryptage et de décryptage. En outre, elles ne permettent pas de comparer les données cryptées, et de trouver les résultats top-k.

Le cloud est généralement constitué de plusieurs sites (ou centres de données), chacun situé à un endroit géographique différent. Un centre de données (data center) dispose de ses propres ressources informatiques et de stockage, généralement sous la forme d'un cluster d'ordinateurs, qui est composé d'un grand nombre de noeuds de calcul/stockage. Ainsi, pour des raisons de scalabilité et de performance, les données peuvent être réparties sur plusieurs noeuds, soit horizontalement (différents éléments de données sur différents noeuds), soit verticalement (différents champs de données sur différents noeuds). Ces deux formes de partitionnement peuvent être combinées [59]. Dans cette thèse, nous considérons deux cas pour le traitement de requêtes top-k sur des données cryptées : 1) centralisé : les données chiffrées de l'utilisateur sont stockées sur un seul noeud d'un centre de données, ce qui est utile si la base de données de l'utilisateur peut facilement tenir sur un noeud ; 2) distribué : les données chiffrées de l'utilisateur sont partitionnées et les partitions sont chiffrées et réparties sur plusieurs noeuds, ce qui est utile si la base de données est très grande. Nous nous concentrons sur le partitionnement vertical pour deux raisons. Tout d'abord, il est très facile de répondre aux requêtes top-k sur des partitions horizontales : il suffit que chaque neud du centre de données cloud calcule ses données top-k locales, puis le résultat final est obtenu en calculant les données top-k de l'ensemble des données top-k locales. Ensuite, le partitionnement vertical est plus difficile et le problème du traitement des requêtes top-k avec partitionnement vertical reste un grand problème de recherche [14, 93].

Contributions

Cette thèse apporte trois contributions principales :

1. Etat de l'art du traitement de requêtes en préservant la confidentialité des données

Nous présentons en détail les solutions existantes pour le traitement de requêtes sur des données externalisées en préservant la confidentialité. Nous présentons l'externalisation des données en général et certains défis associés, tels que la confidentialité des données.

Nous décrivons ensuite les techniques utilisées pour préserver la confidentialité des données de l'utilisateur, par exemple, l'anonymisation, la confidentialité différentielle et le chiffrement des données. Nous détaillons les différents types de schémas de cryptage qui sont proposés pour assurer le traitement des requêtes sur des données cryptées, par exemple, le cryptage qui préserve l'ordre, le cryptage homomorphe, le cryptage interrogeable, etc. Enfin, nous présentons les principales techniques de traitement des requêtes d'intervalle, des requêtes de recherche des k plus proches voisins (kNN) et des requêtes top-k sur des données cryptées.

Il y a essentiellement trois catégories de techniques qui ont été développées pour les requêtes d'intervalle : 1) les techniques basées sur les structures de données spécialisées : elles utilisent des structures de données spéciales telles que les arbres, les graphes et les index qui sont utilisées pour exécuter des requêtes d'intervalle sur des données chiffrées ; 2) les techniques basées sur le chiffrement qui préserve l'ordre : elles permettent aux bases de données et autres applications de traiter efficacement les requêtes impliquant un ordre des données chiffrées ; et 3) les techniques basées sur la *bucketization* : elles sont basées sur un partage du domaine relatif à un attribut de la base de données en un ensemble de paquets. Chacun d'eux est identifié par une étiquette. L'ensemble des étiquettes permet de construire un index utilisé plus tard par le cloud pour traiter les requêtes.

Les techniques proposées pour préserver la confidentialité des données pour les requêtes kNN sont classées en deux catégories : 1) des techniques centralisées [37, 86] ; et 2) des techniques distribuées où les données sont réparties verticalement ou horizontalement entre un ensemble de noeuds non complices [73, 73]. Presque tous les protocoles proposés pour préserver la confidentialité pendant le traitement des requêtes top-k [81, 82, 13] sont basés sur l'utilisation des techniques de calcul multi-participant sécurisé (Secure Multiparty Computing - SMC) sur des données en clair. Dans le SMC, il y a n participants $(P_1, \ldots, P_n)$ qui détiennent des entrées privées $(a_1, \ldots, a_n)$. Un protocole SMC permet à $(P_1, \ldots, P_n)$ de calculer en collaboration une fonction f sur les entrées $(a_1, \ldots, a_n)$ sans divulguer a_i à P_j, où $1 \leq i, j \leq n$ et $i \neq j$. Pour y parvenir, les participants doivent échanger des messages et effectuer des calculs locaux jusqu'à ce que chacun obtient le résultat souhaité. A notre connaissance, un seul travail introduit une solution pour le traitement des requêtes top-k sur données cryptées [87].

2. BuckTopk

Nous proposons un système complet, appelé *BuckTop*, pour le cas centralisé. BuckTop est capable d'évaluer efficacement les requêtes top-k sur des données cryptées externalisées vers un noeud unique *c.-à-d.* la base de données entière est stockée dans un noeud du cloud. BuckTop utilise une technique de bucketisation après le cryptage des données où il partitionne chaque liste de la base de données dans un ensemble de paquets et affecte à chaque paquet une borne inférieure et supérieure. Ces bornes sont utilisées plus tard pour le traitement des requêtes top-k. BuckTop inclut un algorithme de traitement des requêtes top-k qui fonctionne sur les données cryptées stockées dans le noeud du cloud sans avoir à les décrypter dans ce noeud. L'algorithme de traitement des requêtes top-k retourne un

ensemble de données appelé *candidats top-k* qui contient les données cryptées correspondant aux résultats top-k. BuckTop utilise un algorithme de filtrage efficace qui est exécuté dans le noeud du cloud sur les données cryptées et supprime la plupart des faux positifs inclus dans l'ensemble des candidats top-k renvoyés par l'algorithme de traitement des requêtes top-k. Ce filtrage se fait sans avoir besoin de décrypter les données dans le cloud. Nous avons validé notre système par des expérimentations sur des ensembles de données synthétiques et réelles. Nous avons comparé son temps de réponse avec l'approche basée sur le cryptage qui préserve l'ordre sur des données cryptées, et avec l'algorithme TA sur des données en clair. Les résultats expérimentaux montrent un excellent gain de performance pour BuckTop. Ils montrent aussi que la surcharge d'utiliser BuckTop pour le traitement des requêtes top-k sur des données cryptées est très faible, à cause de l'efficacité de l'algorithme du traitement des requêtes top-k et de l'algorithme de filtrage qui élimine jusqu'à 99% des faux positifs dans le noeud du cloud.

3. SDB-TOPK et SD-TOPK

Pour le cas distribué, nous proposons deux nouveaux systèmes, appelés SDB-TOPK (Secure Distributed Bucket based TOP-K) et SD-TOPK (Secure Distributed TOPK). Ces systèmes sont capables d'exécuter un traitement de requêtes top-k sur des données cryptées réparties sur les noeuds n d'un centre de données. Ils utilisent les mêmes techniques de cryptage et de bucketisation que celles proposées dans le système BuckTop pour crypter la base de données avant de l'externaliser. Ensuite, la base de données externalisée est répartie sur les noeuds de sorte que chaque noeud stocke au moins une liste de la base de données. SDB-TOPK et SD-TOPK utilisent des algorithmes de traitement de requêtes top-k qui sont exécutés dans les noeuds et trouvent un ensemble comprenant les données top-k chiffrées. Ils sont exécutés sans avoir à décrypter les données dans les noeuds où ils sont stockés. De plus, SDB-TOPK et SD-TOPK ont un puissant algorithme de filtrage qui filtre les faux positifs autant que possible dans les noeuds, et renvoie un petit ensemble de données cryptées qui seront décryptées du côté utilisateur pour obtenir le résultat final. Nous avons évalué les performances de nos solutions sur des bases de données synthétiques et réelles. Les résultats montrent un excellent temps de réponse et un excellent coût de communication pour SDB-TOPK et SD-TOPK par rapport aux approches basées sur l'algorithme TA . Ils montrent que leur temps de réponse peut être de plusieurs ordres de grandeur supérieur à celui des algorithmes basés sur TA. Ceci est principalement dû à leurs algorithmes optimisés de traitement des requêtes top-k et de filtrage. Les résultats montrent également des gains significatifs dans les coûts de communication des SDB-TOPK et SD-TOPK par rapport aux autres algorithmes. Ils montrent également l'efficacité de l'algorithme de filtrage qui élimine presque tous les faux positifs, sans décrypter les données.

Directions de recherche

Cette thèse identifie deux directions importantes pour la poursuite de la recherche sur le traitement des requêtes sur des données cryptées. La première est de considérer l'exécution des requêtes *skyline* sur des données cryptées. Comme la requête top-k, la requête skyline [9, 62] est une requête importante qui est utilisée pour réduire le nombre de résultats retournés aux utilisateurs, en éliminant ceux qui ne sont pas intéressants. Ainsi, les requêtes skyline peuvent être très utiles pour réduire le coût de communication entre le cloud et ses utilisateurs. Formellement, une requête skyline est définie comme suit. Avec un ensemble de données D et un ensemble d'attributs m, une requête skyline Q sur D retourne un ensemble de tuples qui ne sont dominés par aucun autre tuple dans les attributs donnés. Un tuple X domine un autre tuple Y, si, sur tous les attributs, X est aussi bon ou meilleur que Y et, sur au moins un attribut, X est meilleur que Y. La dominance entre deux tuples est notée $X > Y$. Les dominances des attributs sont spécifiées dans la requête skyline. La principale différence entre requête skyline et requête top-k est que les résultats des requêtes top-k changent avec le changement de la fonction de notation, tandis que les résultats des requêtes skyline sont fixés pour un ensemble de données. De plus, la taille du résultat de la requête skyline ne peut pas être contrôlée par l'utilisateur et, dans le pire cas, peut être aussi grande que la taille des données. Une solution pour exécuter les requêtes skyline sur des données cryptées est d'utiliser un cryptage qui préserve l'ordre qui nous permet de déterminer la dominance des tuples dans chaque attribut. Cependant, le défi consiste à trouver des solutions plus sûres qui ne divulguent pas l'ordre des données cryptées ou ne les divulguent que partiellement.

La seconde direction de recherche est d'exploiter la technologie des *software guard extensions (SGX)* pour le traitement des requêtes top-k sur des données cryptées dans le cloud. SGX, proposé par Intel, est une nouvelle technologie matérielle qui protège les secrets d'une application des logiciels malveillants en créant des zones de mémoire, appelées enclaves, isolées de code et de données. Ces pages de mémoire non adressables sont réservées à partir de la RAM physique du système, puis cryptées, permettant à l'application d'accéder à ses secrets sans crainte d'être exposée. Les applications SGX sont construites en deux parties : 1) partie de confiance qui se compose des enclaves. Elles résident dans une mémoire cryptée et sont protégées par Intel SGX. Les enclaves sont considérées comme fiables parce qu'elles ne peuvent pas être modifiées après leur construction. Si un utilisateur ou un logiciel malveillant tente de modifier une enclave, cet essai sera détecté et évité par le CPU ; 2) la partie non fiable qui est le reste de l'application. Toute application ou région de mémoire non protégée par Intel SGX est considérée comme non fiable. Nous pourrions utiliser cette technologie, qui est disponible dans certains clouds, par exemple, Azure, pour améliorer le temps de réponse et la sécurité du traitement des requêtes sur des données cryptées. L'idée est d'utiliser les enclaves de confiance pour déchiffrer certaines données dans le cloud, *c-à-d.*, quand cela est nécessaire. Cependant, le défi est d'optimiser les opérations à exécuter dans les enclaves car elles peuvent être rapidement surchargées car leur capacité est limitée.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing encompasses on demand reliable services provided over the Internet (typically represented as a cloud) with easy access to virtually infinite computing, storage and networking resources. Through very simple Web interfaces and at small incremental cost, users can outsource complex tasks, such as data storage, system administration, or application deployment, to very large data centers operated by cloud providers. Thus, the complexity of managing the software/hardware infrastructure gets shifted from the users' organization to the cloud provider. However, outsourcing sensitive data to a cloud provider [33], using Database-as-a-Service (DaS), has a main problem: how to preserve the privacy of the outsourced data, which may be violated by cloud employees for instance, while processing the user's queries.

The popular solution is to encrypt the data before outsourcing it to the cloud. Then, the challenge is to answer user queries over encrypted data. A naive solution is to retrieve the encrypted database from the cloud to the client machine, decrypt it, and then evaluate the query over *plaintext* (non encrypted) data. This solution is not practical, as it does not take advantage of the power provided by the cloud computing resources for evaluating queries. In this thesis, we consider an important kind of queries, top-k queries, and address the problem of privacy-preserving top-k query processing over encrypted data in the cloud.

In the rest of this introduction, we first define the thesis context and objectives, and then introduce our contributions. Finally, we give the thesis organization.

1.1 Thesis Context and Objectives

According to the Cloud Security Alliance [19], cloud security is one of the main concerns for cloud users. Cloud security refers to a broad set of policies and technologies deployed to protect data, applications, and the associated infrastructure of cloud computing. There are a number of security threats, such as data loss, unauthorized access to the data, data breaches, denial of service, etc.

In general, the security threats can be grouped into different models [5], according to the type of attacker or adversary whose aim is to get access to the user data and/or tamper it:

- **Honest-but-Curious Model:** In this model (also called semi-honest), an attacker follows the prescribed steps of a protocol. However, the attacker can try to learn additional information about the users based on the users data, and the input, output, and messages received during the execution of the secure protocol.

- **Malicious Model:** An adversary in the malicious model can arbitrarily diverge from the normal execution of a protocol and change the user data.

- **Covert Model:** This model lies between honest-but-curious and malicious models. More specifically, an adversary under the covert model may deviate arbitrarily from the rules of a protocol, without changing the user data.

In this thesis, we assume the honest-but-curious threat model. This model is well adapted to the problem of query processing in the cloud [46, 24] as the cloud providers are not malicious attackers who can change the protocols or user data, but they may be interested to learn more information about the users by accessing their data. However, this may violate the privacy of users. *Privacy preservation* is one of the main requirements to ensure data security in the cloud [80]. Many users pay more attention to their privacy protection when they access cloud data or use cloud services. In particular, they expect to hide their identity while using the cloud. Some users also want their operations on the data and the information retrieved from a cloud to be properly protected. Privacy preservation should ensure that all critical and sensitive data must be masked or encrypted and that only authorized users have access to data in its entirety.

We focus on top-k queries, which have attracted much attention in several areas of information technology such as sensor networks [91], data stream management systems [85], crowdsourcing [18, 95], spatial data analysis [75], temporal databases [64], graph databases [31], etc. A top-k query allows the user to specify a number k, and the system returns the k tuples which are most relevant to the query. The relevance degree of tuples to the query is determined by a *scoring function*. Given a database composed of lists of data items, each data item has a local score in each list. The scoring function f is a function that calculates an overall score for each data item.

As a simple example of top-k query, consider a university that outsources the students database to the cloud, with non-trusted nodes. The database is encrypted for privacy reasons. Then, an interesting top-k query over the outsourced encrypted data is the following: return the k students that have the worst averages in some given courses.

There are several approaches for processing top-k queries over plaintext (non encrypted) data. One of the most popular approaches is the Threshold Algorithm (TA) [27] that works on sorted lists of attribute values. TA can find efficiently the top-k results because of a smart strategy for deciding when to stop reading the database. However, TA and all other efficient top-k approaches developed so far assume the existence of attribute values of the data items in plaintext *i.e.,* data items represented in the original human-readable form. Thus the privacy of the user data is not preserved.

One of the main solutions for preserving privacy in the cloud is to encrypt the outsourced databases. Data encryption is the process of encoding a message or information

in such a way that only authorized parties can access it. This solution provides strong data privacy. However, the ability to perform practical query processing on encrypted data remains a major challenge [8]. In this thesis, we are interested in processing top-k queries over encrypted data.

When we think about top-k query processing over encrypted data, the first idea that comes to mind is to use a fully homomorphic encryption scheme, which allows performing arithmetic operations over encrypted data [28]. Using this type of encryption allows computing the overall score of data items over encrypted data. However, existing fully homomorphic encryption methods are very expensive in terms of encryption and decryption time. In addition, they do not allow to compare the encrypted data, and find the top-k results.

A cloud is typically made of several sites (or data centers), each at a different geographical location. A data center has its own computing and storage resources, typically as a computer cluster, which is made of multiple computer nodes. Thus, for scalability and performance reasons, data can be partitioned across multiple nodes, either horizontally (different data items on different nodes) or vertically (different data fields on different nodes). These two forms of partitioning can be combined [59]. In this thesis, we consider two cases for top-k query processing over encrypted data: 1) centralized: the user encrypted data are stored at a single node of a data center, which is useful if the database can fit at one node; 2) distributed: the user encrypted data are partitioned and the partitions are encrypted and distributed across multiple nodes, which is useful if the database is very big. We focus on vertical partitioning for two reasons. First, answering top-k queries over horizontal partitions is very easy: it is sufficient that each node of the cloud data center calculates its local top-k data items, then the final top-k result is obtained by calculating the top-k data items of all the local top-k data items. Second, vertical partitioning is more difficult and the problem of top-k query processing with vertical partitioning has been addressed by the research community [14, 93].

1.2 Contributions

The main contributions of this thesis are as follows.

- **A survey of privacy preserving query processing.** We first introduce a general definition of data outsourcing in the cloud. Then, we present the main techniques used to preserve the privacy of the user data stored in the cloud.

- **BuckTopk.** We propose a complete system, called *BuckTop*, for the centralized case. BuckTop is able to efficiently evaluate top-k queries over encrypted data outsourced to a singe node, without having to decrypt it in that node. It includes a top-k query processing algorithm that works on the encrypted data stored in the cloud node, and returns a set that is proved to contain the encrypted data corresponding to the top-k results. It also comes with an efficient filtering algorithm that is executed in the cloud node on encrypted data and removes most of the false

positives included in the set returned by the top-k query processing algorithm. This filtering is done without needing to decrypt the data in the cloud.

- **SDB-TOPK and SD-TOPK.** For the distributed case, we propose two new systems, called SDB-TOPK (Secure Distributed Bucket based TOP-K) and SD-TOPK (Secure Distributed TOPK). SDB-TOPK and SD-TOPK come with top-k query processing algorithms that are executed in the nodes, and find a set including the encrypted top-k data items. They are executed without needing to decrypt the data in the nodes where they are stored. In addition, SDB-TOPK and SD-TOPK have a powerful filtering algorithm that filters the false positives as much as possible in the nodes, and returns a small set of encrypted data that will be decrypted in the user side. We analyze the security of our systems, and propose efficient strategies for enforcing them. We evaluated the performance of our systems over synthetic and real databases. The results show excellent performance compared to baseline approaches. They also show the efficiency of our filtering algorithm that eliminates almost all false positives in the cloud, and reduces the communication cost significantly.

The results of this thesis have been presented in the following papers:

- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Privacy-Preserving Top-k Query Processing in Distributed Systems. Submitted for journal publication, July 2018.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Privacy-Preserving Top-k Query Processing in Distributed Systems, *In European Conference on Parallel Processing (Euro-Par)*, 281-292, 2018.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Answering Top-k Queries over Outsourced Sensitive Data in the Cloud. *In International Conference on Database and Expert Systems Applications (DEXA)*, 218-231, 2018.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Top-k Query Processing over Distributed Sensitive Data, *In International Database Engineering & Applications Symposium (IDEAS)*, 208-216, 2018.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Distributed Privacy-Preserving Top-k Query Processing, *In BDA: Gestion de données - principes, technologies et applications*, 2018.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Top-k Query Processing Over Outsourced Encrypted Data. *Research Report RR-9053*, 2017.
- Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Privacy Preserving Query Processing in the Cloud. *In BDA: Gestion de données - principes, technologies et applications*, 2016, 61-62.

1.3 Organization of the Thesis

This thesis is organized as follows.

Chapter 2: State of the art

In this chapter, we review the state of the art in privacy preserving query processing over outsourced data in the cloud. First, we present the general concept of data outsourcing. Then, we introduce the main techniques proposed to preserve the privacy of the outsourced data and review the top-k query processing algorithms over plaintext data. Finally, we review the main privacy preserving techniques proposed for processing range queries, k-nearest neighbor (kNN) queries and top-k queries.

Chapter 3: Top-k query processing over centralized encrypted data

In this chapter, we present our solutions to process top-k queries over encrypted data in the case of a singe node of a cloud data center. First, we present a basic solution called OPE-based approach which is based on order preserving encryption. Then, we introduce our main contribution: BuckTop system. We describe its architecture, and then its top-k query processing and false positive filtering algorithms. Finally, we report the experiment results that we performed to validate the efficiency of our system.

Chapter 4: Top-k query processing over distributed encrypted data

In this chapter, we present our solutions for privacy preserving top-k query processing in the case where the data are distributed across multiple nodes of a cloud data center. We first propose basic approaches based on TA algorithm called Remote-TA and Block-TA for top-k query processing over distributed encrypted data. Next, we present our two main contributions called SDB-TOPK and SD-TOPK. We describe data encryption and outsourcing steps, top-k query processing and false positive filtering algorithms involved in these systems. Furthermore, we analyze the security of SDB-TOPK and SD-TOPK. Finally, we report the experimental results and show that these systems are very efficient for privacy preserving top-k query processing over distributed encrypted data.

Chapter 5: Conclusion

In this chapter, we summarize our contributions and point out the future research directions.

Chapter 2

Overview of Privacy-Preserving Query Processing over Outsourced Data

2.1 Introduction

Database outsourcing provides users and companies with powerful capabilities to store and process their data in third-party machines managed by a cloud service provider. However, the privacy of the outsourced data is not guaranteed as users typically loose physical access control to their data. The most popular solution to protect outsourced data is to encrypt the data before outsourcing to the cloud. This solution introduces the problem of how to evaluate user queries over the encrypted data.

In this chapter, we present the general concept of database outsourcing [33] and introduce the problem of outsourced data privacy and the existing solutions. Then, we review the top-k query processing algorithms over plaintext data. Finally, we review the state of the art privacy preserving query processing techniques, focusing mainly on range, kNN, and top-k queries.

2.2 Database Outsourcing

The idea of outsourcing data to a third party provider is introduced for the first time by Haĉigümus, et al. in [33] and referred to as "Database as a Service".

Database as a Service (DAS) is a database management concept in which the data owner stores her data in a cloud, and delegates the responsibility of administering and managing the data to the cloud. This paradigm alleviates the need of installing data management software and hardware, hiring administrative and data management crew (personnel) at the company's site. Thus, the data owner can concentrate on her core business logic rather than on the tedious job of data management. Examples of DAS providers are: Amazon, IBM and Google.

The architecture using DAS (see Figure 2.1) consists of 3 main entities: *Data Owner*, *Cloud Service Provider* and *Client*. Generally, data owner and clients are considered

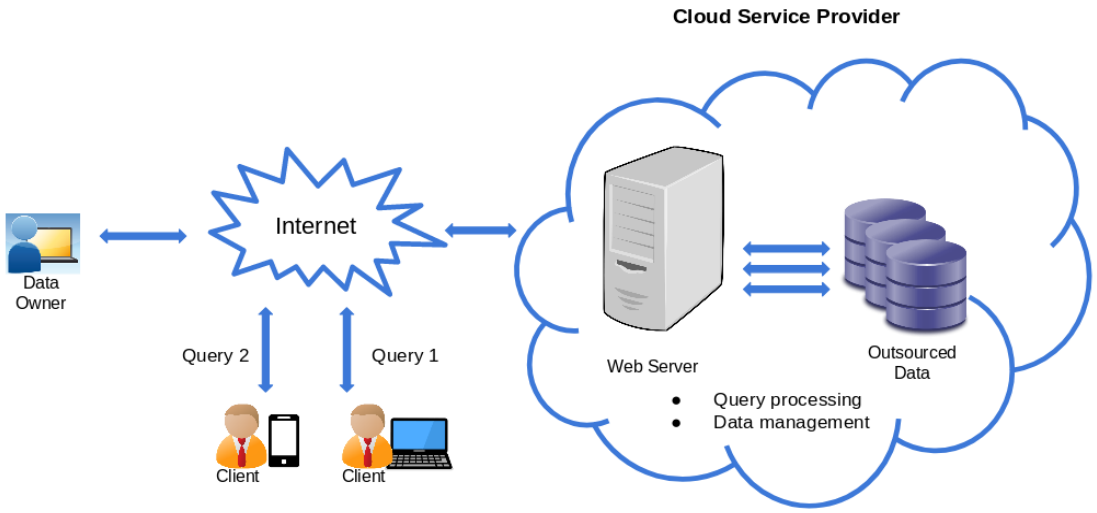

Figure 2.1 – Database Outsourcing Model Architecture

as trustful entities while cloud provider is trustless for the purpose of accessing data in an unauthorized manner. A data owner uploads her data at to the cloud using a high speed communication link. A data owner can insert new data, modify the existing data and delete data. In case of multiple clients, the data owner can set access level permissions for using the data. Data management hardware and software tools are deployed and maintained at the cloud. To ensure the availability of data in case of data crash or version change, standby machines are also maintained so that seamless and uninterrupted database service is provided. When a client submits a query to the cloud, the query is processed in the cloud nodes, and the results are sent back to the client. The clients are given permission to access the data according to their privilege level.

Database outsourcing introduces several challenges in terms of performance, scalability and security and data privacy.

- **Performance:** Since the interaction between the users and the cloud takes place in a different manner than in traditional databases, there are potential overheads introduced by this architecture. Therefore the sources of performance degradation and their significance should be determined.

- **Scalability:** A good DAS must support database and workloads of different sizes. The challenge arises when a database workload exceeds the capacity of a single machine. A DAS must therefore support scale-out, where the responsibility of data management is partitioned amongst multiple nodes to achieve higher throughput.

- **Security and data privacy:** As the data is stored at the cloud, it may be the case that the cloud provider is trustless in terms of disclosing and misusing the data. In this case, security of the database can be hampered in a dramatic way.

2.3 Data Privacy Of Outsourced Databases

Users lose the physical control over their data when they store it in an untrusted cloud. Therefore, the privacy of the outsourced data is not guaranteed. Several techniques are proposed to ensure the privacy of this data. In this section, we present the anonymization, differential privacy and data encryption techniques.

2.3.1 Anonymization

Most of the outsourced database contains sensitive information about individual entities, such as a person, a household, or an organization. In order to protect this sensitive data, the users may anonymize the database tables such that the resulting tables do not disclose any sensitive information. There are different ways to anonymize a table such as suppression and generalization. Suppression is the operation of changing a specific value of an attribute in a tuple by *ANY*, denoted by *. Generalization is the operation of changing a value of an attribute to a more general one. If the data value is numeric, it may be changed to a range of values. Several anonymization techniques have been proposed and the most popular ones are k-anonymity [71] and ℓ-diversity [50]. In both approaches, attributes are partitioned into three categories: (1) some attributes are identifiers that can uniquely identify an individual, such as Name or Social Security Number; (2) some attributes are Quasi-Identifiers (QI), which the adversary may already know (possibly from other publicly-available databases) and which, when taken together, can potentially identify an individual,*e.g.*, Gender, Nationality and Age; (3) some attributes are Sensitive Attributes (SAs), which are unknown to the adversary and are considered sensitive, such as Disease and Salary.

***k*-Anonymity.** Suppose we have a table consisting of *n* tuples each having m quasi-identifying attributes (Gender, Nationality and Age in Table 2.1), and let k *>1* be an integer. The idea is to suppress/generalize some of the entries in the table so as to ensure that for each tuple in the modified table, there are at least k*-1* other tuples in the modified table that are identical to it along the quasi-identifying attributes. The objective is to minimize the extent of suppression and generalization. Note that entries in the column corresponding to the sensitive attribute ("Diseases" in Table 2.1) are not altered. Table 2.2 is a k-anonymized table for $k = 2$ of Table 2.1.

We suppose that an adversary knows the QI attributes of Peter from another database, *e.g.* voter registration, using Table 2.1, she cannot distinguish the first tuple from the second tuple. Thus, she is not sure whether Peter suffers from HIV because the first tuple is linked to HIV, but the second tuple is not.

A k-anonymized table protects individual privacy in the sense that, even if an adversary has access to all the QI attributes of all the individuals represented in the table, she would not be able to track down an individual's tuple further than a set of at least k tuples, in the worst case. Thus, releasing a table after k-anonymization prevents definitive tuple linkages with publicly available databases, and keeps each individual hidden in a crowd of

Name	Gender	Nationality	Age	Disease
Peter	male	Japanese	26	HIV
John	male	Malaysian	30	flu
Mary	female	American	36	HIV
Sally	female	Canadian	40	HIV
Eason	male	American	40	flu
Louis	male	Chinese	36	flu

Table 2.1 – A medical table

Gender	Nationality	Age	Disease
male	Asian	26 - 30	HIV
male	Asian	26 - 30	flu
female	North American	36 - 40	HIV
female	North American	36 - 40	HIV
male	Person	36 - 40	flu
male	Person	36 - 40	flu

Table 2.2 – A 2-anonymous medical table

$k-1$ other people. The privacy parameter k must be chosen according to the application in order to ensure the required level of privacy.

ℓ-Diversity. The concept of ℓ-diversity [50] was introduced to address the limitations of k-anonymity. The latter may disclose sensitive information when there are many identical sensitive attribute (SA) values within an equivalence class,*e.g.*, all persons suffer from the same disease. An equivalence class of a table is a collection of all tuples in the table containing identical QI values, it is also called a *QI-group*.

$\ell-$diversity prevents homogeneity attacks. In the homogeneity attack, while an equivalence class may contain at least k tuples, it is possible that all tuples have the same sensitive attribute value, enabling the adversary to infer the sensitive attribute value of individuals in the equivalence class. Even if not all tuples have the same value, probabilistic inference is still possible. $\ell-$diversity ensures that at least ℓ SA values are well-represented in QI-group i.e. the probability that any tuple in this group is linked to a sensitive value like is at most $1/\ell$. Table 2.3 is a 2-diverse table of Table 2.1. It contains two *QI-groups*. The first one contains the two first tuples while the second contains the last four tuples. For each *QI-group*, the probability that a tuple is linked to HIV is at most 1/2.

Machanavajjhala et al. suggest in [50] that any k-anonymity algorithm can be adapted to achieve ℓ-diversity in different ways,*e.g.*, take a k-anonymous table and suppress groups that are not ℓ-diverse or suppress tuples in groups until all groups are ℓ-diverse.

Gender	Nationality	Age	Disease
male	Asian	26 - 30	HIV
male	Asian	26 - 30	flu
*	Person	36 - 40	HIV
*	Person	36 - 40	HIV
*	Person	36 - 40	flu
*	Person	36 - 40	flu

Table 2.3 – A 2-diverse medical table

2.3.2 Differential Privacy

Differential privacy has received growing attention in the research community. It was originally developed by Dwork, Nissim, McSherry and Smith in [22]. Differential privacy provides rigorous and statistical guarantees against what an adversary can infer from learning the results of some randomized algorithm. In the simplest setting, consider an algorithm that analyzes a dataset and computes statistics about it (such as variance, median, mode, etc.). Such an algorithm is said to be differentially private if by looking at the output, one cannot tell whether any individual's data was included in the original dataset or not. In other words, the guarantee of a differentially private algorithm is that its behavior hardly changes when a single individual joins or leaves the dataset. Formally, we say that a randomized computation M provides ϵ-differential privacy if for any datasets A and B that differ in at most one entry or tuple (called databases neighbors) and any set of possible outcomes $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times e^{\epsilon}$$

The parameter ϵ allows to control the level of privacy. Lower values of ϵ mean stronger privacy. The values typically considered for ϵ are smaller than 1, *e.g.,* 0.01 or 0.1 (for small values we have $e^{\epsilon} \approx 1 + \epsilon$).

To achieve differential privacy, Dwork et. al. propose the Laplace mechanism [22], which perturbs the output of the computation function by explicitly adding scaled random noise from the Laplace distribution. Laplace distribution represents the distribution of differences between two independent variables with identical exponential distributions. It is also called the double exponential distribution. The median mechanism proposed in [69] is an interactive differentially private mechanism that answers arbitrary predicate queries $f_1, \ldots, f_n$ that arrive on the fly without the future knowledge queries, where k could be large or even super-polynomial. It comes to answering more queries exponentially and gives fixed constraints. Theoretically, the mechanism is suitable for defining and identifying the equivalence of queries in the interactive setting. In this setting, to answer the queries from the users, an interactive differential privacy interface is being inserted between the users and the database for the sake of privacy. The Gaussian mechanism [23] and the K-norm mechanism [34] are differentially private mechanisms that are also based on the idea of output perturbation with noise from different distributions. Lei [42] pro-

poses differentially private M-estimators, which perturb the histogram of input data using a scaled noise and further uses the noisy histogram to train the models. Zhang et. al. [94] propose a differentially private functional mechanism that adds a properly scaled Laplace noise to the coefficients of loss function in the polynomial basis.

2.3.3 Data Encryption

Outsourced database services allow users to store sensitive data on a remote, untrusted node and retrieve desired parts of it on request. Many works have addressed the security of outsourced databases by encrypting the data at rest and pushing part of the processing to the cloud. This solution provides strong data privacy, however, the ability to perform practical query processing on encrypted data remains a major challenge. Both the database and the cryptography research communities have shown great interest in querying encrypted data including keyword search [78, 16], k nearest neighbors search (kNN) [37, 86], range queries [32, 36], etc. To process different queries over encrypted data, several techniques have been developed, *e.g.*, Bucketization, order preserving encryption, Searchable encryption, etc.

Bucketization-based technique [32, 35] uses distributional properties of the dataset to partition data and design indexing techniques that allow approximate queries over encrypted data. Unlike cryptographic schemes that aim for exact predicate evaluation, bucketization admits false positives while ensuring all matching data is retrieved. A post-processing step is required at the client side to remove the false positives. These techniques often support limited types of queries and lack of a precise analysis of the performance/security trade-off introduced by the indexes.

Order preserving encryption (OPE) [1, 7, 49, 67] is a deterministic encryption scheme where the order of the ciphertext, *i.e.,* the encoded information obtained as the result of encryption, is the same as that of the original plaintext. OPE allows databases and other applications to process queries involving order *e.g., range queries*, over encrypted data efficiently. OPE relies on the strong assumption that all plaintexts in the database are known in advance.

Searchable symmetric encryption (SSE) allows one to store data at an untrusted node and later search the data for records (or documents) matching a given keyword while maintaining privacy. Many recent works [20, 38, 84] have studied SSE and provided solutions with varying trade-offs between security, efficiency, and the ability to securely update the data after it has been encrypted and uploaded.

All the above mentioned techniques sacrifice some degree of data privacy for more effective querying on encrypted data and provide different levels of security guarantees. Other proposals sacrifice query efficiency for stronger data privacy such as homomorphic encryption scheme.

A homomorphic encryption scheme Hom [79, 61] is a form of encryption that allows computation on encrypted data, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. More formally, it is a public key encryption consists of four algorithms: $(Key_{gen}, E, D, Eval)$.

Given a security parameter λ, the Key_{gen} algorithm returns a secret key sk and a public key pk. $Key_{gen}(\lambda) \rightarrow (sk, pk)$. The secret key is known only to the data owner and it is used to decrypt the data encrypted by the appropriate public key which may be disseminated widely. The encryption algorithm E takes as input a message m, a public key pk and outputs a ciphertext C, $E(pk, m) \rightarrow C$, the decryption algorithm D takes as input the cyphertext C and a secret key sk and outputs a message m, $D(C, sk) \rightarrow m$. the $Eval$ algorithm takes as input a public key pk, a $t - input$ circuit Cr (consisting of addition and multiplication gates modulo 2), and a tuple of ciphertexts $C_1, \ldots, C_t$ (corresponding to the t input bits of Cr, and returns a ciphertext C (corresponding to the output bit of Cr)

Most of the known homomorphic encryption schemes support only a limited set of functions f, which restrict their applicability. The theoretical problem of constructing a fully homomorphic encryption scheme supporting arbitrary functions f, was solved by the breakthrough work of Gentry [28] followed by other works and improvements [83, 77]. Fully homomorphic encryption allows the cloud to compute arbitrary functions over encrypted data, while only clients see decrypted data. However, it is computationally expensive to be used in practice.

2.4 Query Processing

Query processing is an essential technology for database management systems. It allows to efficiently retrieve from the database the data specified in a user query. In this section, we describe the query processing steps, then we present the processing algorithms of some queries like range queries, kNN search queries and top-k queries.

Query processing aims to transform a query described in a high-level declarative language (e.g. SQL) into a correct and efficient execution strategy [4]. The query processor receives a query as input, translates and optimizes this query in several phases into an executable query plan, and executes the plan in order to obtain the results of the query. The query transformation and execution is done by five components [41] (see Figure 2.2): parse, query rewriter, query optimizer, plan refinement/code generation and query execution engine

1. ***Parser.*** In the first phase, the query is parsed and translated into an internal representation (e.g., a query graph [65]) that can be easily processed by the later phases;

2. ***Query rewriter.*** Query rewriter transforms a query in order to carry out optimizations that are useful regardless of the physical state of the system (*e.g.*, the size of tables, presence of indices, locations of copies of tables, speed of machines, etc.). Typical transformations are the elimination of redundant predicates and simplification of expressions;

3. ***Query Optimizer.*** This component carries out optimizations that depend on the physical state of the system. It decides which indices to use to execute a query,

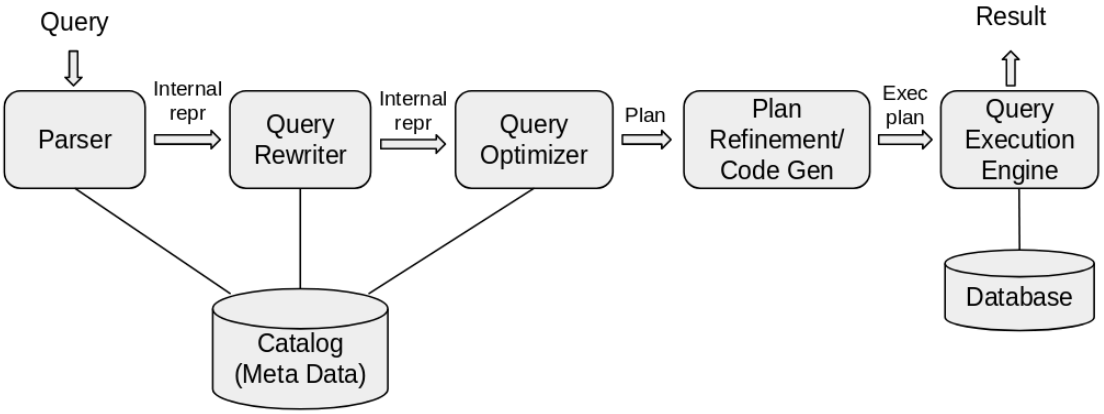

Figure 2.2 – Query Processing Components

which methods (e.g., hashing or sorting) to use to execute the operations of a query (e.g., joins and group-bys), and in which order to execute the operations of a query. The query optimizer also decides how much main memory to allocate for the execution of each operation. As a result of this optimization, a plan is generated. This plan specifies precisely how the query is to be executed. Probably every database system represents plans in the same way: as operator trees. The nodes of a plan are operators, and every operator executes one particular operation (e.g., join, group-by, sort, scan, etc.). The edges of a plan represent consumer-producer relationships of operators;

4. ***Plan Refinement/Code Generation.*** It transforms the plan produced by the optimizer into an executable plan. In some systems, plan refinement also involves carrying out simple optimizations which are not executed by the query optimizer because of a simplification in the implementation of the query optimizer.

5. ***Query Execution Engine.*** This component provides generic implementations for every operator. Most of the query execution engines are based on an iterator model [30]. In such a model, operators are implemented as iterators and all iterators have the same interface. As a result, any two iterators can be plugged together (as specified by the consumer–producer relationship of a plan), and thus, any plan can be executed. Another advantage of the iterator model is that it supports the pipelining of results from one operator to another in order to achieve good performance.

The catalog used in the three first components stores all the information needed in order to parse, rewrite and optimize a query. It maintains the database schema (*i.e.*, definitions of tables, views, user-defined types and functions, etc.), the partitioning schema (*i.e.*, information about what global tables have been partitioned and how they can be reconstructed), and physical information such as the location of copies of partitions of tables, information about indices, and statistics that are used to estimate the cost of a plan. In most relational database systems, the catalog information is stored in a specific relational (meta-) database.

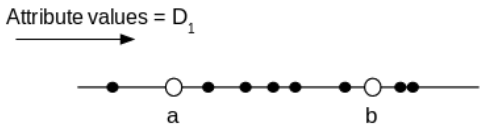

Figure 2.3 – Range query

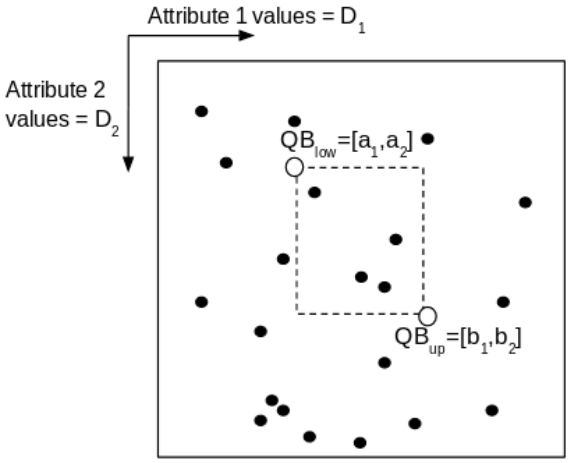

Figure 2.4 – Multi-dimensional range query

2.4.1 Range Queries

Range queries have been used in different domains such as sensor networks [47] spatial databases [60, 63], grid information service [3], peer-to-peer systems [44, 66], etc. A range query is a query that retrieves all data items where some of their attribute values are between an lower bound a and a upper bound b (see Figure 2.3). The result of the range query can be an empty-set or a too-large-set of data items.Unlike in an exact-match query (a query with an equality predicate), it is not generally known in advance how many data items it will return.

Range queries are important to query multidimensional databases [76]. In this case, they are called multidimensional range queries (also called window or rectangular queries). Using a multidimensional range query, the user specifies an interval of values (for each attribute), which the retrieved data tuples have to match. It is the result of the cartesian product of the intervals for all dimensions. The range query can be represented by a hyper-box QB in the multidimensional space . The ranges of query box QB are defined by two boundary points, the lower bound $QB_{low} = [a_1, a_2, \ldots, a_n]$ and the upper bound $QB_{up} = [b_1, b_2, \ldots, b_n]$; where $a_1 \leq b_1, a_2 \leq b_2, \ldots, a_n \leq b_n$ (see Figure 2.4). The purpose of the range query is to select all data tuples inside the query box QB, *i.e.*, to select tuples t satisfying $a_i \leq t \leq b_i$, for $1 \leq i \leq n$.

2.4.2 KNN Search Queries

The K-Nearest Neighbor query (kNN) is a classical problem that has been extensively studied, due to its many important applications, such as spatial databases [40, 48], pattern recognition, DNA sequencing and many others. Given a dataset *P*, a query point *p* and a value *k*, the kNN query returns *k* points $\in P$ which are closest to the query point *p* based on a distance function (*e.g., euclidean distance*)(see Figure 2.5). There is another type

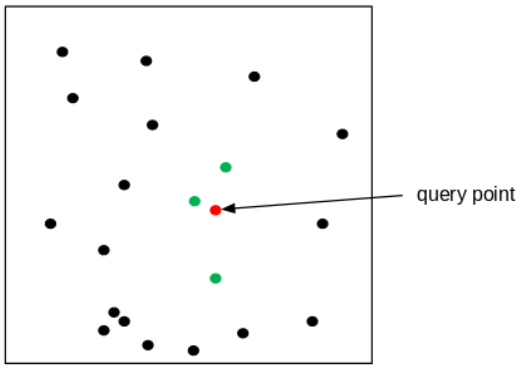

Figure 2.5 – 3 Nearest Neighbors Query

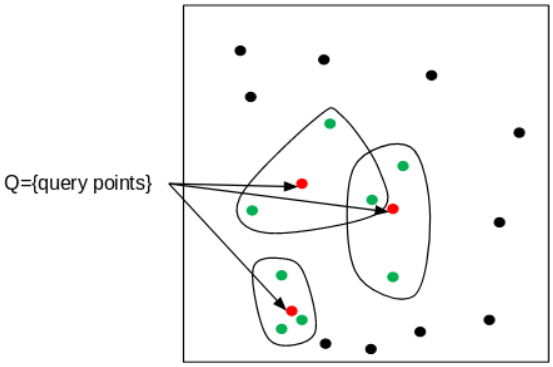

Figure 2.6 – 3 Nearest Neighbors Join Query

Attribute 1	Attribute 2	Attribute 3	Overall score
O_1, 55	O_2, 34	O_2, 43	O_2, 125
O_2, 48	O_1, 30	O_3, 30	O_1, 118
O_3, 42	O_3, 27	O_1, 33	O_3, 109

Table 2.4 – Top-2 query with SUM scoring function

of kNN query which is kNN join query. It can be defined as follows: given a dataset P and a query set Q, for each point $q \in Q$, the kNN query allows to retrieve its k nearest neighbors from points in P (see Figure 2.6).

To search for the K nearest neighbors of a query point q, the distance of the k^{th} nearest neighbor to q defines the minimum radius required for retrieving the complete answer set. Unfortunately, such distance cannot be predetermined with 100% accuracy. Hence, an iterative approach that examines increasingly larger spheres in each iteration can be employed. The algorithm works as follows. Given a query point q, finding nearest neighbors begins with a query sphere defined by a relatively small radius about q, *querydist(q)*. All data spaces that intersect the query sphere have to be searched. Gradually, the search region is expanded until all the K nearest points are found and all the data subspaces that intersect with the current query space are checked. The k data points are the nearest neighbors when further enlargement of query sphere does not introduce new answer points. Starting the search query with a small initial radius keeps the search space as tight as possible, and hence minimizes unnecessary search (has a larger radius that contains all the K nearest points been used).

2.4.3 Top-k Query

Top-k queries have attracted much attention in several areas of information technology such as network monitoring systems, information retrieval, data mining, multimedia databases, spatial data analysis, etc.

By a top-k query, the user specifies a number *k*, and the system should return the *k* most relevant answers to the user. The relevance degree of the answers to the query is determined by a scoring function (see Table 2.4). A common method for efficient top-k query processing is to run the algorithms over *sorted lists* (also called inverted lists) [27]. Let us define them formally.

Let *D* be a database of *n* data items, and $L_1, L_2, ..., L_m$ be *m* lists such that each list L_i contains *n* pairs of the form $(d, s_i(d))$ where $d \in D$ and $s_i(d)$ is a non-negative number that denotes the *local score* of *d* in L_i. Each list L_i is sorted in descending order of its local scores.

The set of *m* lists is called a *database*. The *overall score* of each data item d is calculated as $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$ where *f* is a given *scoring function*. In other words, the overall score is the output of *f* where the input is the local scores of *d* in all lists.

The result of a top-k query is the set of *k* elements that have the highest overall scores among all elements of the database. Formally, the top-k query result is a set of elements $D' \subseteq D$ such that $|D'| = k$, and $\forall d' \in D' \wedge \forall d \in (D - D') \Rightarrow ov(d') \geq ov(d)$.

Almost all of the algorithms for top-k query processing work on data organized in sorted lists. The sorted lists model is simple and general. For example, suppose we want to find the top-k tuples in a relational table according to some scoring function over its attributes. To answer such query, it is sufficient to have a sorted (indexed) list of the values of each attribute involved in the scoring function, and return the k tuples whose overall scores in the lists are the highest.

For processing top-k queries over sorted lists, two modes of access are usually used [27]. The first is *sorted (sequential) access* that allows us to sequentially access the next data item in the sorted list. This access begins with the first item in the list. The second is *random access* by which we look up a given data item in the list.

2.4.4 Top-k Query Processing Algorithms

Most algorithms proposed for top-k query processing over plaintext data [25, 27, 2, 14] use the sorted list model of the database. A naive solution to find the top-k answers is to scan the sorted lists from beginning to end, calculate for each seen data item its overall score using the scoring function, then return the k data items that have the highest overall score. This solution is not efficient, especially for very large databases. Fagin [25] proposes the first efficient algorithm Fagin's Algorithm (FA) which is based on a simple idea: do sorted access in parallel in the sorted lists and stop after finding k data items which are seen in all the lists. Based on FA, Fagin et al. [27] introduce the threshold algorithm (TA). This algorithm comes with an intelligent stop mechanism better than FA; after each sorted access, TA calculates a threshold and stops when it finds k data items that have an overall score greater than or equal to the threshold. FA and TA algorithms use random access to access a given data in the sorted lists, but in some applications, performing random access is impossible or very expensive. Fagin et al. propose in [26] an algorithm called *no random access* to resolve this problem. Akbarinia et al. [2] propose

another algorithm Best Position Algorithm based on the TA algorithm. It uses a different threshold value which is based on the position of the data items accessed in the lists. In the rest of this section, we present in more detail these algorithms, which some of them will be used in the rest of the thesis.

2.4.4.1 Fagin's Algorithm

The basic idea of Fagin's Algorithm (FA) is to scan sequentially the sorted lists until finding k data items that have been seen in all the lists then it stops. After that, FA calculates for each seen data its overall score and return the k data items that have the highest one. FA execution steps are the following:

1. Starting with the first data item in each list, do sorted access in parallel to the sorted lists. Maintain in a set S all the data items that have been seen under the sorted access. If there are k data items in S that have been seen in all the lists, then stop sorted access and pass to the second step, else continue doing sorted access.

2. for each data item $d \in S$, do random access to the lists to get its local scores. By applying the scoring function on the returned local scores, FA calculates the overall score of d and maintains it in a set Y if its overall score is among the k highest scores calculated so far.

3. Return Y.

Fagin shows in [25] that his algorithm is optimal with high probability in the worst case only, i.e, when we have to go far in the lists in order to find the *top-k* data items. This is not the case while using some scoring functions like MAX where FA does additional and useless accesses.

2.4.4.2 Threshold Algorithm

The threshold algorithm (TA) comes with a different stop condition more efficient than that of FA. It uses a threshold to decide if it stops doing sorted access or not. this threshold is calculated by applying the scoring function on the last sees local scores in the lists. TA proceeds as follows:

1. In parallel, do sorted access in the sorted lists. For the data item d seen in a list L_i, do random access on the other lists to get its local score. Calculate the overall score of d using the scoring function. Maintain d in a set Y if its overall score is among the k highest scores of the data items seen so far.

2. After each sorted access, calculate a threshold TH by applying the scoring function on the last seen local score in each list. If there are k data items in Y that have an overall score greater than or equal to TH, then stop doing sorted access and go to step 3. Otherwise, go to step 1.

3. Return Y.

TA assumes that the costs of sorted and random access methods are the same. In addition, TA does not have a restriction on the number of random accesses to be performed. Every sorted access in TA results in up to $m-1$ random accesses, where m is the number of lists.

2.4.4.3 No-Random-Access Algorithm (NRA)

In some applications, performing random access can be impossible, as in the example discussed in [12]. To address this problem, Fagin et al. propose the No Random Access algorithm (NRA) [26], which finds the top-k answers by exploiting only sorted accesses to the lists. NRA proceeds as follows:

1. Let $p_1^{min}, \ldots, p_m^{min}$ be the smallest possible values in lists $L_1, \ldots, L_m$
2. Do sorted access in parallel to lists $L_1, \ldots, L_m$ and at each step do the following:
   (a) Maintain the last seen local scores $ls_1, \ldots, ls_m$ in the m lists
   (b) For each seen data item d, compute a lower bound $Lb = f(v_1, \ldots, v_m)$ where $v_i = s_i(d)$ if d is seen under the sorted access in the list i, otherwise $v_i = p_i^{min}$. Also compute an upper bound $Ub = f(u_1, \ldots, u_m)$ where $u_i = s_i(d)$ if d is seen in the list i, otherwise $u_i = ls_i$. For a data item d' that is not seen in any list, its lower bound $Lb = f(p_1^{min}, \ldots, p_m^{min})$ and its upper bound $Ub = f(ls_1, \ldots, ls_m)$.
   (c) Let A_k be the set of k data items with the largest lower bound values Lb seen so far. If two data items have the same lower bound, then ties are broken using their upper bounds such that the data item with the highest upper bound value wins (and arbitrarily if there is a tie for the lower bound value)
   (d) Let M_k be the k^{th} largest Lb value in A_k.
3. A data item d is said viable if $Ub(d) > M_k$. Stop sorted access when:
   (a) At least k distinct data items have been seen
   (b) There are no viable data items outside A_k. That is, if $Ub(d) \leq M_k$ for all $d \notin A_k$
4. Return A_k.

Mamoulis et al. study the NRA algorithm in [56] under various application requirements. They observe that at some stage during NRA processing, it is not useful to update the upper bounds. Instead, the updates to these upper bounds can be deferred to a later step, or can be reduced to a much more compact set of necessary updates for more efficient computation.

2.4.4.4 Best Position Algorithm

The Best Position Algorithm (BPA) uses a threshold value which is different than that of TA, based on the position of the data items accessed in the lists. This allows BPA to stop processing much earlier than TA. BPA proceeds as follows:

1. Do sorted access in each of the sorted lists. For each seen data d in a list L_i, do random access in the other lists to get its local score and its position in each list. Maintain the obtained data to be used in step 2 and compute the overall score of d. In a set Y, maintain the k data items that have the highest overall scores calculated so far.

2. Let P_i be the set of positions in L_i which are seen under sorted or random access in step 1. Let bp_i, called best position in L_i, be the greatest position in P_i such that any position in L_i between 1 and bp_i is also in P_i. Let $s_i(bp_i)$ be the local score of the data item which is at the position bp_i in the list L_i.

3. Calculate a *best position overall score* $\lambda = f(s_1(bp_1), s_2(bp_2), \ldots, s_m(bp_m))$. If Y contains k whose overall scores are greater than or equal to λ, then stop doing sorted access to the lists. Otherwise, go to step 1.

4. Return Y.

In FA and TA, many read operations are redundant, and BPA tries to minimize those operations. Depending on the number of lists m, BPA can be *(m - 1)* times faster than TA. Furthermore, Akbarinia et al. [2] introduce the Best Position Algorithm 2 (BPA2) which is again (m-1) faster than the first version of their best position algorithm.

2.4.4.5 Best Position Algorithm 2

The Best Position Algorithm 2 (BPA2) algorithm is based on BPA. It is much more efficient than BPA where the number of accesses to the lists done by BPA2 can be about (m-1) times lower than that of BPA. In this algorithm, a new mode of access is defined: direct access. Direct access allows to read the data item which is at a given position in a list. BPA2 works as follows:

1. For each list L_i, let bp_i the best position in L_i. Initially, set $bp_i = 0$.

2. For each list L_i and in parallel, do direct access to position $(bpi + 1)$ in list L_i. As a data item d is seen under direct access in some list, do random access to the other lists to find $d's$ local score in every list. Compute the overall score of d. Maintain in a set Y the k seen data items whose overall scores are the highest among all data items seen so far.

3. If a direct access or random access to a list L_i changes the best position of L_i, then along with the local score of the accessed data item, return also the local score of the data item which is at the best position. Let $s_i(bp_i)$ be the local score of the data item which is at the best position in list L_i.

4. Let best positions overall score be $\lambda = f(s_1(bp_1), s_2(bp_2), \ldots, s_m(bp_m))$. If Y involves k data items whose overall scores are higher than or equal to λ, then stop doing sorted access to the lists. Otherwise, go to step 1.

5. Return Y.

BPA and BPA2 have the same stopping mechanism. Thus, they both stop at the same (best) position. In addition, they see the same set of data items. However, there are two main differences between them. The first difference is that BPA2 does not return the seen positions to the user, it returns only the set Y (which contains at most k data items) and the local scores of the m best positions. The second difference is that with BPA some seen positions of a list may be accessed several times, but with BPA2 each seen position of a list is accessed only once because BPA2 does direct access to the position which is just after the best position and this position is always an unseen position in the list.

2.4.5 Top-k Query Processing over Distributed Data

In distributed applications and systems such as sensor networks, data streams, and peer-to-peer (P2P) systems, data generation and storage is also distributed. Thus an emerging challenge is to support top- k query processing in distributed systems. It is possible to use the existing algorithms to process Top-k queries in distributed systems *e.g.,* TA, but this increases latency and consumes an excessive amount of bandwidth when the number nodes of the distributed system increases. Several algorithms have been proposed to resolve this problem while reducing latency and bandwidth consumption. Cao and al.[14] proposed Three-Phase Uniform Threshold algorithm (TPUT). It is executed in three rounds of communication between nodes. Based on this algorithm, Yu and al.[93] introduce three algorithms: Three-Phase Adaptive Threshold algorithm (TPAT), Three-Phase Object Ranking based algorithm (TPOR) and Hybrid-threshold algorithm (HT).

2.4.5.1 Three-Phase Uniform-Threshold Algorithm (TPUT)

In TPUT, the authors make the assumption that the distributed system is composed of m nodes connected to a central manager. These nodes maintain the sorted lists. TPUT proceeds in three phases, each taking one communication round between the central manager and the nodes to be completed. First, it determines a lower-bound estimation of the k^{th} value. Second, it prunes away ineligible data items as much as possible using this estimation. Third, it looks up the resulting set of data items in all nodes to identify the top-k items. The detailed execution steps are the following:

Lower bound estimation of the k^{th} value: When the central manager receives a top-k query, it informs all the nodes. Each node responds by sending the top-k items in its sorted list. After that and for each returned data item o, the central manager calculates its *partial sum* P: $P(o) = s'_1(o) + s'_2(o) + \cdots + s'_m(o)$ where $s'_i(o) = s_i(o)$ if o is returned by the node i, and $s'_i(o) = 0$ otherwise. At the end of this phase, the central manager set the lower bound τ_1 called "phase-1 bottom" at the k highest partial sum.

Pruning away ineligible data items: Using "phase-1 bottom", the manager fixes a threshold $T = (\tau_1/m)$ and sends it to all the nodes. Each node looks in its list for the data items whose score value is greater than or equals to T and returns them to the central manager. Now, the latter is sure that it has all the data items that can be in the top-k answer set. The reason is that if a data item is not returned by any node, its score value is less than T *i.e.*, its aggregate value is less than τ_1. Therefore, it cannot be in the top-k set. The central manager now starts the pruning task by refining the lower bound estimated in the previous phase. It calculates the new partial sum for all the returned data items and sets a "phase-2 bottom" value, denoted by τ_2, to the k^{th} partial sum. It is clear that $\tau_1 \leq \tau_2 \leq T$. Then, the manager calculates the upper bound of each returned data items and eliminates those who have an upper bound less than τ_2. The upper bound of a data item o is calculated as follows: $U(o) = u'_1(o) + u'_2(o) + \cdots + u'_m(o)$ where $u'_i(o) = s_i(o)$ if o has been returned by the node i, and $u'_i(o) = T$ otherwise. As a result of this phase, the central manager gets the set of top-k candidates denoted by S.

Top-k data item identification. In this phase, the central manager sends the set S to the nodes which respond by sending the local scores of the data items in S. Now, the manager can calculate the exact global score of each data item in S and selects the top-k results.

During the execution of the previous steps, it is remarkable that TPUT does not take into consideration data distribution. It supposes that it is uniform among all nodes which is not realistic due to the heterogeneous nature of distributed systems.

2.4.5.2 Three-Phase Adaptive Threshold Algorithm (TPAT)

TPAT is an extension of TPUT. It divides the "phase-1 bottom" τ_1 among the nodes differently. TPUT assumes that data item scores are uniformly distributed among nodes in the distributed system, *i.e.*, each node contributes approximately the same to the result set of top-k data items. However, this assumption does not consider the case where some nodes may have data items with larger score distributions (hot nodes) and other nodes may have data items with smaller score distributions (cold nodes). To resolve this problem, TPAT proposes to divide τ_1 among nodes according to their data distributions. It is executed in three phases where the first and third one are the same TPUT. The only difference is in the second phases where the uniform threshold $T = \tau_1/m$ is replaced by a set of thresholds $T_1, T_2, \ldots, T_m$ determined by the central manager. These thresholds are calculated according to some statistics sent from the nodes. Then, the thresholds $T_1, T_2, \ldots, T_m$ are sent to $node_1, node_2, \ldots, node_m$, respectively. The rest of this phase is the same as in TPUT except the upper bound calculation, which is calculated as follows: $U(o) = u'_1(o) + u'_2(o) + \cdots + u'_m(o)$ where $u'_i(o) = s_i(o)$ if o has been returned by the node i, and $u'_i(o) = T_i$ otherwise.

2.4.5.3 Three-Phase Object Ranking Based Algorithm

The Three-Phase Object-Ranking based algorithm (TPOR) takes into consideration the heterogeneous nature of distributed systems without using any summary statistics (unlike

TPAT). It uses data item rankings instead of their scores to eliminate ineligible data items which cannot be among the top-k result. TPOR has the same execution phases of TPUT, except the second one, where the central manager broadcasts the list L of the k data items that have the highest partial sum to all nodes. Then, each node looks for the local score of each data item in L and determines the lowest local score T_i among all the k data items in L. Then, it sends the list of data items that have a score value $\geq Ti$ to the central manager. This central manager calculates the partial sums of all the data items seen so far, and identifies the data items with the k highest partial sums. Let τ_2 be the k^{th} highest partial sum called "phase-2 bottom". Now, the central manager tries to prune away more data items, It calculates the upper bounds of the data items seen so far as follows $U_{sum}(O) = u'_1(o) + u'_2(o) + \cdots + u'_m(o)$ where $u'_i(o) = s_i(o)$ if o has been returned by the node i, and $u'_i(o) = T_i$ otherwise. Then the central manager removes data items which have an upper bound less than τ_2 from the candidate set.

2.4.5.4 Hybrid-Threshold Algorithm

In certain cases, the uniform threshold calculated by TPUT for each node to prune data items may be very small. This results in many more data items returned from all the nodes. Alternatively, if a data item in the list L calculated by TPOR in phase 1 ranks very low in some nodes or even does not appear, then those nodes will send almost their entire list to the central manager. To resolve this problem, the Hybrid-Threshold algorithm (HT) is proposed by [57]). HT tries to combine the advantages of both TPOR and TPUT. It works in 3 phases, with an additional patch phase executed if necessary). The first one is the same as in TPUT. In the second phase, the central manager asks each node to send data items whose scores are greater than or equal to a hybrid threshold, which is calculated as the maximum of the uniform threshold $T = \tau_1/m$ from TPUT and the threshold obtained by TPOR. However, this cannot guarantee the correctness of the algorithm. It is possible that some data items in a node whose scores are between the uniform threshold by TPUT and the threshold by TPOR, are top-k data items. Thus, a patch phase is added in order to make the algorithm correctly return the top-k data items. Then, the central manager calculates the new partial sums for all the data items seen so far and identifies the data items with the k highest partial sums. Let the k^{th} partial sum denote τ_2. The central manager calculates $T_{patch} = \tau_2/m$. After the second phase, the central manager knows the lower bounds of the data item scores of nodes, denoted as $T_1, \ldots, T_m$. If $T_{patch} \leq T_i$, the central manager sends T_{patch} to node i and asks node i to send the data items whose scores are greater than or equal to T_{patch}. Since T_{patch} is greater than T, calculated in the beginning of phase 2, the total number of data items sent by HT is no greater than that of TPUT. However, if $T_{patch} > T_i$ for every i, there is no need for this patch phase, *i.e.*, all top-k data item candidates have been considered.

2.5 Privacy-Preserving Query Processing

To provide privacy guarantees for outsourced data in clouds, the main solution is that the user encrypts the data before outsourcing. Several works have addressed the problem of query processing over encrypted data. In this section, we present the main approaches proposed for executing range queries, kNN queries, and top-k queries over encrypted data.

2.5.1 Privacy Preserving Range Query Processing

In the literature, there are essentially three categories of techniques that have been developed for range queries: specialized data structure-based techniques, order-preserving encryption-based techniques and Bucketization-based techniques. Popa et al. have proposed an exhaustive system called CryptDB [68] that ensures SQL query processing over encrypted data, which takes into consideration range query processing.

2.5.1.1 Specialized Data Structure-Based Techniques

To process range queries over encrypted data, several techniques [8, 45, 74, 46] have been proposed that use special data structures such as trees, graphs, and indexes.

In [45], Li and al. propose a prefix-preserving encryption based schema (PPES) that focuses on *interval-matching* or *exact-matching* as query conditions. Interval-matching is defined as a boolean function $f_{[a,b]}(x)$, which returns true if and only if $x \in [a, b]$ where x, a and b are all non-negative integers. Exact-matching is a special case of interval-matching in which a is equal to b. PPES encrypts the whole tuple of a data item in the database as one block: all the attribute values of a data item O are the input of the encryption schema. Then, it associates each encrypted tuple with a number of indexing attributes. The result of this operation (see the example illustrated in Tables 2.5 and 2.6) is a database with one attribute representing the encrypted tuple (the attribute Enc_tuple in Table 2.6) and additional attributes representing the indexes (I_{SSN}, I_{SALARY} and I_{DNO} in 2.6). Each plaintext tuple $t(A_1, \ldots, A_n)$ is mapped onto a tuple $t'(E(t), I_1, \ldots, I_m)$ where $m \leq n$. The attribute E(t) stores an encrypted string that corresponds to the entire plaintext tuple, and each I_i corresponds to the index over some A_j.

To construct the indexes, first, PPES transforms the *interval matching* into *prefix-matching*. The transformation is based on the fact that an arbitrary interval can be converted into a union of prefix ranges. For example, the interval [32, 111], the 8-bit binary representation of which is [00100000, 01101111], can be represented by a set of prefixes {001$*$, 010$*$, 0110$*$} where $*$ is used to denote an arbitrary suffix. Checking the membership of a number in the interval is equivalent of checking that the number matches any of those prefixes in the set. For example, 37 (00100101 in binary) is in the interval as it matches prefix 001$*$, while 128 (10000000 in binary) is not in the interval since it matches none of those three prefixes. PPES integrate the *prefix-preserving encryption* proposed in [90]. A prefix-preserving encryption is a function Ep that for given two numbers a and

FNAME	LNAME	SSN	ADDRESS	SALARY	DNO
John	Smith	123456789	731 Fordren, Storrs, CT	30000	5
Franklin	Wong	333445555	638 Voss, Storrs, CT	40000	5
Alicia	Zelaya	999887777	3321 Castle, Storrs, CT	25000	4
Ahmad	Jabbar	987987987	980 Dallas, Storrs, CT	25000	5
James	Borg	888665555	450 Stone, Storrs, CT	55000	1

Table 2.5 – Employee Database

Enc_tuple	I_{SSN}	I_{SALARY}	I_{DNO}
fjftejcCcWsGqfChXcHuRzoriODCRxvD	068764019	6488	250
tprJMmfjXJNs74fZZfL1TridemjZnWvY	277737042	45639	250
edVI8JvVSjmzXsrmDIiosZabdFnnorwy	080581877	53798	224
z4tzGJUdsyy7Eb0puESatLCXOXckVTWA	203690710	53798	250
zzdqGlqngQgwJurSqsyFrejiia6KCNMk	929644962	20577	59

Table 2.6 – Encrypted Employee Database

b that share a k-bit prefix, $Ep(a)$ and $Ep(b)$ also share a k-bit prefix [45]. we say that two n-bit numbers $a = a_1a_2 \dots a_n$ and $b = b_1b_2 \dots b_n$ share a k-bit prefix $(0 \leq k \leq n)$,if $a = a_1a_2 \dots a_k = b_1b_2 \dots b_k$, and $a_{k+}1 \neq b_{k+1}$ when $k < n$.

After that, the prefix-preserving encryption generates the index as follows. If a plaintext can take any value of a n-bit number, the entire set of plaintexts can be represented by a complete binary tree of height n. This is called the plaintext tree (illustrated in Figure 2.7 using 4-bit plaintexts). Each node in the plaintext tree (excluding the root node) corresponds to a bit position, indicated by the height of the node, and a bit value, indicated by the direction of the branch from its parent node. A prefix-preserving encryption function (see Figure 2.8) is defined by specifying a binary variable for each non-leaf node (including the root node) of the plaintext tree. This variable specifies whether the encryption function "flips" this bit or not. Applying the encryption function results in the rearrangement of the plaintext tree into a ciphertext tree (see Figure 2.9).

To execute range queries over data encrypted using PPES, the query conditions in operations (such as selects and joins) must be translated to corresponding conditions over the cloud representation. This translation function is called Map_{cond}. In [45], the authors studied the SELECT query which is transformed as follows:

attribute = value: since the prefix-preserving encryption is a one-to-one mapping, the mapping is simply defined by $Map_{cond}(A_i = v) \Rightarrow E_p(A_i) = E_p(v)$.

attribute $\leq$ value (attribute $\geq$ value): a query condition $A_i \leq v$ ($A_i \leq v$) is equivalent to an interval-matching of $f[v_{min}, v](A_i)$ ($f[v, v_{max}](A_i)$), where v_{min} (v_{max}) is the lower (upper) bound of the attribute domain. The interval $[v_{min}, v]$($[v, v_{max}]$) can be converted into a union of prefix ranges, $P_1, P_2, \dots, P_l$, using interval-matching prefix-matching transformation. Therefore, $f[v_{min}, v](A_i)$ can be transformed $\{M_{P_1}(A_i)$, $M_{P_2}(A_i)$, $\dots$, $M_{P_l}(A_i)\}$ ($M_{P_k}(A_i)$ denotes the boolean function, which returns true if

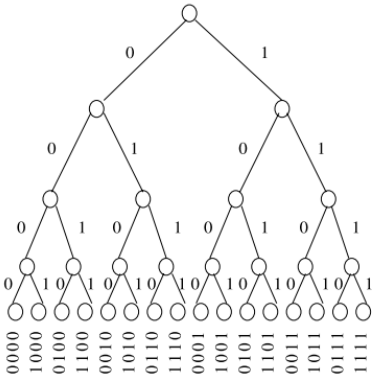

Figure 2.7 – PPES plaintext tree

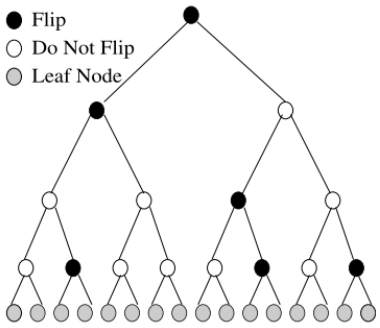

Figure 2.8 – PPES encryption function

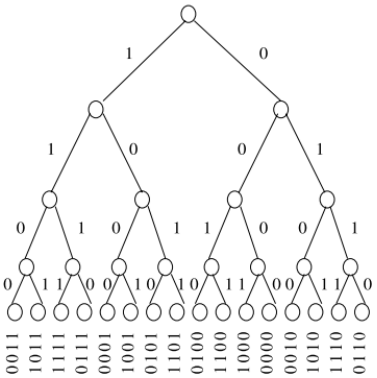

Figure 2.9 – PPES ciphertext tree

and only if the value of A_i matches prefix P_k). Then, the prefix-preserving encryption can be applied on the prefixes. Therefore, the mapping is defined as follows:

$Map_{cond}(A_i \leq v) \Rightarrow \{M_{E_p(P_1)}(E_p(A_i))$ OR $M_{E_p(P_2)}\ (E_p(A_i))$ OR ... OR $M_{E_p(P_l)}(E_p(A_i))\}$.

When the cloud receives this query, it executes it over the encrypted data. The results will be transmitted to the client. The client then can get the query results by applying the decryption function.

There are some other works that create indexes to perform range query processing over encrypted data, *e.g.*, [46, 70]. For example, in [46], the authors propose a query processing schema that achieves index indistinguishability under the indistinguishability against chosen keyword attack (IND-CKA) *i.e.* this schema ensures that no information regarding to the keywords is leaked from the index. They propose to organize all indexing elements in a complete binary tree where each node is represented using a Bloom filter. The tree is called a PBtree (where "P" stands for privacy and "B" stands for Bloom filter). PBtrees allows to achieve index indistinguishability with two important properties. The first one is structure indistinguishability, that is, two sets of data items have the same PBtree structure if and only if the two sets have the same number of data items. Thus, the structure of PBtree of a set of data items is determined solely by the set cardinality, not the value of data items. The second property is node indistinguishability, that is, for any two PBtrees constructed from data sets of the same cardinality, which have the same structure, and for any two corresponding nodes of the two PBtrees, the values of the two nodes are not distinguishable. Thus, the proposed schema prevents an adversary from performing statistical analysis on the index even with domain knowledge.

2.5.1.2 Order-Preserving Encryption-Based Techniques

Order-preserving encryption (OPE) allows databases and other applications to process queries involving order (*e.g.*, range queries), over encrypted data efficiently. OPE not only allows efficient range queries, but also allows indexing and query processing to be done

exactly and as efficiently as for plaintext data. To process a range query, it is sufficient to send the encryption of *a* and *b* (the bounds of the range query) to the cloud who locates the two ciphertexts in logarithmic-time via standard tree-based data structures, and returns all encrypted data between the two ciphertexts.

The ideal security goal for an order-preserving scheme is to reveal no additional information about the plaintext values besides their order (which is the minimum needed for the order-preserving property).

A number of order-preserving encryption schemes have been proposed in the literature [1, 7, 49, 67]. Nevertheless, the security of these schemes is still under discussion [88] because the order of the data is disclosed.

Agrawal et al. [1] introduce the concept of order-preserving encryption. Their encryption scheme allows any comparison operation to be directly applied on encrypted data. It supports equality and range queries such as MIN, MAX, COUNT, GROUP BY and ORDER BY over encrypted data. SUM or AVG operations to a group is not supported by this scheme where the values need to be decrypted in order to get the result. The authors of [1], first, transform the plaintext database into a flat database such that the values are uniformly distributed. This flat database is then transformed into the cipher database such that the data values are distributed according to the targeted distribution. The transformation of the database is performed by splitting it in several buckets and by using linear interpolation inside every bucket. The drawback of this method is that it must take as input all the plaintexts in the database in advance, which is not always practical in real-life applications.

Boldyreva et al. [7] have shown that indistinguishability against chosen-plaintext attack (IND-OCPA) is unachievable by any OPE scheme with stateless encryption and immutable ciphertexts. A chosen-plaintext attack is an attack model which presumes that the attacker can obtain the ciphertexts for arbitrary plaintexts. The goal of the attack is to gain information that reduces the security of the encryption scheme. The authors propose an efficient OPE scheme on the basis of a sampling algorithm for the hypergeometric probability distribution. As IND-OCPA is unachievable for this scheme, they propose a security notion of a random order-preserving function (ROPF) and related primitives asking that an OPE scheme will look "as-random-as-possible" subject to the order preserving constraint. The encryption algorithm of the scheme in [7] behaves similarly to an algorithm that samples a ROPF from a specified domain on-the-fly.

Popa et al. propose the first order-preserving scheme that achieves ideal security. Their basic idea is mutable ciphertexts, which means that over time, the ciphertexts for a small number of plaintext values change. They prove that mutable ciphertexts are needed for ideal security.

Liu and al. in [49] introduce a programmable order-preserving scheme which is secure and easy to use. The architecture in which this scheme is used is composed of: (1) trusted side represented by the query proxy and the user application and (2) an untrusted side represented by the cloud. The scheme is built over the simple linear expressions of the form $a \times x + b = v$ where x is the input value and v its index. The form of the expressions is public, however the coefficients a and b are kept secret (known only by the query proxy)

$$
\begin{array}{lll}
I & ::= & rindex^{sens}_{[a,b]}(v) \mid S; rindex^{sens}_{[a,b]}(v) \\
S & ::= & skindex^{sens}_{[a,b]}(v) \mid \text{if } C \text{ then } S_1 \text{ else } S_2 \mid S_1; S_2 \\
C & ::= & gt(c) \mid ge(c)
\end{array}
$$

Figure 2.10 – Indexing Program Syntaxe

and a must be greater than 0. This basic indexing scheme is secure against ciphertext only attacks while the attacker do not know *a, b* and any input values. But if the attacker gets some input values of some indexes for example x_1 and x_2 with indexes v_1 and v_2, she could recover a and b by solving two linear equations $a \times x_1 + b = v1$ and $a \times x_2 + b = v_2$. To resolve this vulnerability, the authors propose to add some random $noise$ to each index as follows: $a \times x_i + b + noise_i$. Based on this idea, they develop an indexing program that allows different input values to be indexed by different linear indexing expressions and allows indexes to be indexed again (like the Triple Data Encryption Algorithm (3DES) algorithm, which applies the Data Encryption Algorithm (DES) one of the most widely used cryptographic algorithms three times to each ciphertext).

To describe the indexing program, the following terms must be defined: sensitivity, randomized index and sensitivity-keeping index.

Definition 1. *Let V be the set of all input values. The sensitivity of V denoted by sens is the minimum element in the set* $\{|v_1 - v_2||v_1 \in V, v_2 \in V, v_1 \neq v_2\}$. .

Definition 2. *Given the sensitivity sens of input values V, the randomized index of value* $v \in V$ *denoted* $rindex^{sens}_{[a,b]}(v)$ *is* $a \times v + b + noise$*, where* $a > 0$ *and noise is randomly sampled from the range* $[0, a \times sens)$*. denoted*

Definition 3. *Given the sensitivity sens of input values V ,if* $a > 1$*, then the sensitivity-keeping index of value* $v \in V$ *denoted* $skindex^{sens}_{[a,b]}(v)$ *is* $a \times v + b + noise$*, where noise is randomly sampled from the range* $[0, a \times sens - sens]$

The indexing program I has the syntax shown in Figure 2.10. It can be either $rindex^{sens}_{[a,b]}$ or has the form $S; rindex^{sens}_{[a,b]}$, where S is the composition of sensitivity-keeping indexing expressions. S can be a basic sensitivity-keeping indexing expression $skindex^{sens}_{[a,b]}$, a conditional indexing expression, or a sequential composition of expressions. In the conditional indexing expression, C means a condition, which can be $gt(c)$ or $ge(c)$, where c is a constant. The semantics of indexing programs is defined as follows. Suppose v is an input value. Then, $I(v)$ means the application of I to v, generating v's index. If I is $rindex^{sens}_{[a,b]}$, then $I(v) = rindex^{sens}_{[a,b]}(v)$. If I is $S; rindex^{sens}_{[a,b]}$, then $I(v) = rindex^{sens}_{[a,b]}(i)$, where $i = S(v)$. The semantics of indexing steps S is defined inductively. If S is $skindex^{sens}_{[a,b]}$ then $S(v) = skindex^{sens}_{[a,b]}(v)$. If S is the conditional indexing step, then $S(v) = S1(v)$ if v makes the condition C true; otherwise, $S(v) = S_2(v)$. The condition C is $gt(c)$ or $ge(c)$. The condition $gt(c)$ is true if $v > c$, and $ge(c)$ is true if $v \geq c$.If S is a sequential composition of steps, then $S(v) = S_2(i)$, where $i = S_1(v)$.

An indexing program is said well-formed if it is order-preserving. Since in an indexing program the basic indexing expressions $skindex$ and $rindex$ are already order-preserving, it is order-preserving if all conditional indexing expressions are also order-preserving. For any conditional indexing expression if C then S_1 else S_2, where C is $gt(c)$ or $ge(c)$, it is order-preserving if $S_1(c) \geq S_2(c)$. This condition also makes sure there is no overlap among indexes generated by S_1 and S_2.

The programmability of indexes increases the robustness of our index scheme in two aspects. First, input values can be indexed by multiple linear expressions, making brute-force attacks harder. Second, the distribution of indexes can be decoupled from the distribution of input values, making it harder to estimate the range of input values according to the positions of indexes.

2.5.1.3 Bucketization-Based Techniques

Bucketization is useful technique for range query processing over encrypted data. It is based on partitioning the attribute domain of an attribute in a relational database into a set of buckets. Each of them is identified by a tag. The set of tags construct an index used later by the cloud to process the queries. There are several methods for partitioning the values of an attribute, for instance, by dividing the attribute domain to almost equal intervals or creating partitions with equal sizes [32, 36, 35, 43].

Hacigumus et al. [32] proposed the bucketization-based data representation for query processing in Database as a Service model. They develop a technique to split an original query over plaintext relations into 1) a corresponding query over encrypted relations to be run in the cloud; 2) a client query for post-processing results returned by the cloud. To achieve this transformation, they develop an algebraic framework for query rewriting over encrypted representation. For the encryption and storage data, the authors propose that, for each relation $R(A_1, A_2, \ldots, A_n)$, where A_i represents attribute i in the the relation R, they store in the cloud an encrypted relation $R^S(etuple, A_1^S, A_2^S, \ldots, A_n^S)$, where $etuple$ is an encrypted string that corresponds to a tuple in the relation R and A_i^S correspond to the index of the attribute A_i. For data partitioning, they split the domain of an attribute A_i into partitions such that these partitions taken together cover the whole domain, and any two partitions do not overlap. This partitioning step is similar to those used for histogram construction, equi-depth, equi-width partitioning, etc. Then, they assign a random (index) tag to each bucket effectively making every element within a bucket indistinguishable from another. When a query is issued by the client, first, it is determined which buckets intersect the query using the index tag stored on the client (this is typically a small amount of information) and all contents of the intersecting buckets are retrieved from the cloud. The disadvantage is that the query result almost consists of false positives, which have to be eliminated by the client in the post-processing phase. Unfortunately, this proposed bucketization method [32] is not optimal for minimizing false positives.

Hore et al. [35] study the problem of supporting an important class of multidimensional range queries over relational data in a privacy-preserving manner. They propose an approach that computes a secure indexing tag of the data by applying a bucketization

technique, which prevents the cloud from learning exact values of the database data items. In their approach, they assume that the data is encrypted at the row level, *i.e.* each row of the table is encrypted as a single unit, which we will refer to as an e-tuple (the same encryption technique used in the the previous cited work of Hacigumus et al. [32]). The bucketization algorithm consists of two phases. First, bucket creation, it starts with a single bucket and greedily generates a new one in each iteration until the specified number of buckets is formed. The resulting buckets are called "optimal" buckets. The second phase is called controlled diffusion. It re-distributes the data contained in the optimal buckets into a new set of buckets called "composite" buckets such that the average entropy and variance of the sensitive attribute's value distribution is substantially increased. The resulting set of composite buckets are approximately equi-depth buckets and form the final bucketized representation of the client data on the cloud. To retrieve the data elements in response to a range query q, first, the client computes the query overlap with the optimal buckets (denoted as $q(B)$) and then determines the composite buckets that contain at least one element from any bucket in $q(B)$, which are then requested from the cloud.

2.5.1.4 CryptDB

CryptDB [68] is a relational database system that provides privacy. It executes SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. CryptDB is developed to address two threats. The first threat is a curious administrator who tries to learn private data (*i.e.*, honest-but-curious model). The second threat is an adversary that gains complete control of application and DBMS servers.

CryptDB system is composed of two main components:

- A proxy that stores the database schema, the current encryption layers of all columns and a secret master Key. It is responsible of all encryption and decryption operations, and rewriting queries by changing some operators while preserving query semantics,
- A database management system (DBMS) server which stores the encrypted user data, some auxiliary tables (used during the query processing) and the anonymized database schema.

CryptDB equips the DBMS server with CryptDB-specific user-defined functions (UDFs) that enable it to compute certain operations applied over ciphertexts. The DBMS server never sees sensitive data because it never receives decryption keys in plaintext. This prevents a curious database administrator (DBA) from access to the sensitive data.

CryptDB query processing is done in four steps:

1. When the proxy receives a query issued by the application user, it anonymizes each table and column name contained in the query, and encrypts each constant in the query using the master key MK and the best-suited encryption scheme for the desired operation. CryptDB uses some operations of existing crypto systems: Random(RND), Deterministic (DET), Order Preserving Encryption (OPE) and Homo-

morphic Encryption (HOM), in addition to a new cryptographic primitive for joins [68].

2. The proxy verifies if it must give keys to the DBMS server in order to adjust encryption layers before executing the query. If so, it issues an UPDATE query at the DBMS server that invokes a UDF to adjust the encryption layer of the appropriate columns. The proxy sends the encrypted query to the DBMS server.

3. When the DBMS server receives the query, it executes it using standard SQL (occasionally invoking UDFs for aggregation or keyword search).

4. The proxy intercepts the query result from the DBMS server ,decrypts it and returns the plaintext result to the application user.

CryptDB aims to preserve user data privacy not integrity. If an adversary (or malicious DBA) compromises the DBMS server or the proxy, she can then modify or delete the encrypted data. Also, this system is not completely secure since it uses some weak encryption schemes (e.g., order-preserving encryption).

2.5.2 Privacy Preserving Knn Query Processing

In the literature, many techniques have been proposed for secure kNN query processing. We classify these techniques into two categories: centralized and distributed.

Centralized techniques. Several methods are proposed to secure kNN query processing in centralized systems such as [37, 86, 92, 96]. Wong et al. [86] proposed a new encryption scheme called Asymmetric Scalar-Product-preserving Encryption (ASPE) that preserves the scalar product between the query vector (q) and any tuple vector (t_i) from the database (D) for distance comparison, which is sufficient to find kNN. Both the data and query are encrypted using slightly different encryption schemes before outsourcing to the cloud, and all the users know the decryption key. As an improvement, Zhu et al. [96] propose a novel secure kNN method in which the encryption key of the data owner is not disclosed to the users so the data owner participates in query encryption. Each user would like to find out the kNN of her private query point q without disclosing any privacy to data owner and cloud. Before being submitted to the cloud for kNN computation, the query point will be encrypted in a collaborative manner between the user and the data owner such that only the user obtains the encrypted query q'. The data owner cannot reveal her key to the user because Zhu et al. suppose that the user is not trustworthy enough and she may reveal her knowledge about the key to the adversary (the cloud in this work) or it will seriously violate the business secret and privacy. When the cloud recieves the encrypted query q', executes it without learning anything about the original query point.

As an alternative, Hu et al. [37] propose a holistic and efficient solution that is based on Privacy Homomorphism (PH) [21]. PH is an encryption transformation that maps a set of operations on plaintext to another set of operations on ciphertext. In essence, PH supports modular addition, subtraction, and multiplication over encrypted data and enables

complex computations (such as distances) based solely on ciphertext, without decryption. The authors of [37] integrated a provably secure PH seamlessly with a generic index structure to develop a framework that evaluates various types of complex queries, such as kNN queries. Recently, Yao et al. [92] propose a new secure kNN method based on partition-based Secure Voronoi Diagram (SVD). Instead of asking the cloud to retrieve the exact kNN, it is asked to return a relevant encrypted partition ($E(G)$ for $E(D)$) such that G is guaranteed to contain the k-nearest neighbors of q. This solves the SkNN problem accurately by letting the cloud retrieve the k-Nearest Neighbors of q (in encrypted form).

Most of the computations during the query processing step in [37, 92, 96] are performed locally by the end-user, which conflicts with the purpose of database outsourcing model.

Distributed techniques. In these methods, the data is partitioned either vertically or horizontally and distributed among a set of independent, non-colluding parties. Many attempts have been made to secure kNN query processing in a distributed environment. Shaneck et al. [73] propose a privacy-preserving algorithm to perform k-Nearest Neighbor search. The algorithm in privacy-preserving nearest neighbor search [73] is based on Secure Multiparty Computation (SMC) for privately computing kNN points in a horizontally partitioned dataset (*i.e.* each party has a collection of data for the same set of attributes, but for different entities). In SMC, there are *n* parties noted by $(P_1, \ldots, P_n)$ who hold private inputs $(a_1, \ldots, a_n)$. An SMC protocol allows $(P_1, \ldots, P_n)$ to collaboratively compute a function f on inputs $(a_1, \ldots, a_n)$ without disclosing a_i to P_j, where $1 \leq i, j \leq n$ and $i \neq j$. To achieve that, the participating parties have to exchange messages and perform some local computations until all the parties get the desired output. The algorithm proposed in [73] is composed of two main parts. The first computes a superset of the nearest neighborhood and the second one reduces this set to the exact nearest neighbor set.

Ghinita et al. [29] propose a private information retrieval (PIR) based framework for answering kNN queries in location-based services (LBS). LBS are services offered through a mobile phone and take into account the device's geographical location. They typically provide users with useful information about restaurants, coffee shops, stores, concerts, and other places or events which are in a specific geographical area. PIR allows users to retrieve a data item X_i from a set $X = X_i, X_i, \ldots, X_n$ stored in the cloud without revealing i to the cloud. This solution, however, protects only the query privacy (*i.e.* it does not address data privacy and access pattern issues). Note that, in private queries in location-based services [29], the data residing in the cloud are in plaintext format. However, if the data is encrypted to ensure data privacy, it is not clear how a user can obliviously retrieve the output tuples because she does not know the indices that match his input query.

Note that the above data distribution methods are applicable to perform kNN queries over partitioned data in plaintext format among different parties and not over encrypted data. Yousef et al. [24] propose a protocol for computing k-NN on encrypted data. They use Paillier encryption [61] as the underlying cryptographic tool. Paillier encryption is additive homomorphic. Using this property, they developed protocols which compute k-

NN securely in the cloud. Their solution provides very strong security guarantees and is able to hide the data access pattern as well. However, such security comes at the cost of performance.

Manish et al. [39] introduced a new protocol for computing k-NN on encrypted data in the two-party honest-but-curious cloud model. In this model, two non-colluding public cloud clouds voluntarily collaborate with each other to exchange resource and share computing. The proposed protocol is asymptotically faster than the protocol proposed by Youcef et al. [24], without compromising on strong security guarantees. They use LFHE (leveled fully homomorphic encryption) [11], a variant of homomorphic encryption that supports arbitrary functions, and compute squared Euclidean distances directly on encrypted data. In order to compute the ranking among distances, they transform the distances suitably to preserve their order and offload the comparison to a public cloud, which has the secret keys. Since this cloud has access only to transformed results of computations performed on plaintext data, they show that in spite of this knowledge, this honest-but-curious cloud does not learn anything useful about the original database, the results, or the query.

Here, we highlight that the KNN problem should not be confused with the top-k problem in which the given scoring function plays an important role, such that on the same database and with the same k, if the user changes the scoring function, then the output may change. Thus, the proposed solutions proposed for kNN cannot be used for processing the top-k queries over encrypted data.

2.5.3 Privacy Preserving Top-k Query Processing

Almost all the protocols proposed to preserve the privacy during top-k query processing [81, 82, 13] are based on using the secure multiparty computation techniques (SMC) over plaintext data. To the best of our knowledge, [87] is the only work that introduces a solution for top-k query processing over encrypted data.

Vaidya and Clifton [81, 82] study the problem of finding the top-k matching items over vertically-distributed private data. They assume k parties,$P_1, \ldots, P_k$, each party has access to a separate database that gives information about different features. Data is collected for the same set of entities. Also, they make the assumption that none of the parties trust each other completely, *privacy concerns prevent them from disclosing scores or local ordering of the entities*. Their proposed protocol is based on Fagin's algorithm [27] described in Section 2.4.4.1. During the sorted access phase, a mechanism is used to check if there are at least k data items in common to all the parties, as well as giving the union of the sets without revealing the effective order of the data items at any party or even revealing which data item came from which party. Then, each party performs the random access phase independently. In the computation phase, the parties first partially compute scores such that two parties are left with (random) shares of the score for each data item. These two parties then identify a cutoff score that separates the k^{th} item from those below it. This performs a binary search for the appropriate threshold using secure comparisons to guide the search without revealing the score of individual items. As a final

step, the parties can (securely) compare each item with the score to determine if it is in the top-k set or not.

Xiong et al. [89] introduce an algorithm for finding the k largest values among parties holding private sets of values. The protocol preserves privacy in a probabilistic way by randomizing values before distributing them among parties and avoids using cryptographic primitives.

Burkhart et al. [13] proposed two SMC protocols tailored for the problem of preserving the privacy of top-k queries over distributed data: 1) Privacy-Preserving Top-K protocol (PPTK) and 2) Privacy-Preserving Top-K protocol by using Sketches protocol (PPTKS). In the two protocols, the authors assume the existence of two types of parties: n input parties (IN) and m computation parties (CN). The INs are the parties that want to compute the top-k items over their datasets. The CNs help the INs by performing private computations. Each IN locally holds a set of items. An item is defined by an identifying *key* and a corresponding *value*. The goal of the protocols is to compute the k items with the biggest aggregate values over all input sets, without disclosing information about non-top-k items. The aggregate value of an item is simply the SUM of its local values. Also, the protocols should not reveal which INs contribute to a top-k item. With the PPKT protocol, parties put keys into a local hash table to produce a compact representation of a possibly large and sparse space of keys, such as the space of IP addresses. Then, they use SMC primitives (see Table 2.7) to aggregate the local hash tables, resolve collisions, and estimate the top-k key-value pairs. The main idea is that SMC operations are applied to fixed-length hash tables, and this reduces the computational overhead by avoiding expensive key comparison operations. PPTKS extends PPTK using multiple hash tables, *i.e.*, sketches, to reduce the number of collisions experienced by top-k keys and to further improve the estimation accuracy of PPTK.

Xianrui et al. [87] propose a solution for processing top-k queries over encrypted data. In their solution, the data owner encrypts the database using some probabilistic encryption scheme before outsourcing it to the cloud. The authors assume the existence of two different non-colluding semi-honest clouds, S_1 and S_2, where S_1 stores the encrypted database and S_2 holds the secret keys and provides the crypto services. They refer to the cloud S_2 as $CryptoCloud$ and assume that S_2 resides in the cloud environment and is isolated from $S1$. The two nodes S_1 and S_2 do not trust each other, and thus, have to execute secure computations over encrypted data. The Crypto Cloud S_2 is equipped with a cryptographic processor, which stores the decryption key. For top-k query processing, they proposed an algorithm called $SecQuery$ which is based on NRA algorithm [26] (described in Section 2.4.4.3). $SecQuery$ runs two protocols, $EncSort$ [6] and $EncCompare$ [10], position by position in the sorted lists then S_1 computes the worst/best scores based on the items at each position with the collaboration of S_2. Then, S_1 has to update the complete list of encrypted items seen so far with their global worst/best scores. At the end, node $S1$ reports k encrypted data items without learning any data item or its scores. At the end of the protocol, the data item ids can be reported to the client. There are two options to return the data items to the client: either the encrypted records are retrieved and returned to the client, or the client retrieves the records using a secure scheme.

Name	Syntax	Output	Description
sharing	$share(s)$	$[s]$	A secret value s held by an IN is split up in m shares s_i. Each s_i is then distributed to $CN\ i$. The ensemble of all distributed shares $\{s_1, \ldots, s_m\}$ is called a $sharing$ of s and denoted by $[s]$.
reconstruction	$recon([s])$	s	The individual shares of a sharing $[s]$ are combined to reconstruct the secret value s.
addition	$[a] + [b], [a] + b$	$[a + b]$	Adds two sharings (or a sharing and a public value) to get a sharing of the SUM. This also includes subtraction ('-').
multiplication	$[a] \times [b], [a] \times b$	$[a \times b]$	Multiplies two sharings (or a sharing and a public value) to get a sharing of the product.
equal	$aqual([a], [b])$, $equal([a], b)$	if $(a == b)$ then [1] else [0]	Two sharings (or a sharing and a public value) are compared for equality. The output remains secret.
less-than	$lessThan([a], [b])$, $lessThan([a], b)$	if $(a < b)$ then [1] else [0]	Two sharings (or a sharing and a public value) are compared for size. The output remains secret.

Table 2.7 – SMC primitives required for PPKT and PPKTS protocols

2.6 Conclusion

In this chapter, we first introduced database outsourcing and query processing over outsourced plaintext data. Then, we discussed the different techniques used to ensure the privacy of the data while querying them. We presented in more detail the existing solutions for range query processing over encrypted data and the proposed techniques to secure kNN search query and top-k query processing.

In this thesis, we address the problem of top-k query processing over encrypted data. In the best of our knowledge, and as we saw in Section 2.5.3, there is no efficient solution to process top-k query processing over encrypted data except the one of Meng et al. [87]. As we will see in the next chapter, the assumptions that they make in the architecture of their system is different from ours. They assumes the existence of two different non-colluding semi-honest clouds, S_1 and S_2, where S_1 stores the encrypted database and S_2 holds the secret keys and provides the crypto services. The two clouds belong to two different cloud providers and do not trust each other; therefore, they have to execute secure computations on encrypted data. Our assumptions are different, We use one cloud to only store the encrypted database and to process the top-k query processing algorithm over the encrypted data because we the cloud in our contributions is not trusted.

Chapter 3

Top-k Query Processing over Centralized Encrypted Data

3.1 Introduction

Data encryption is the most efficient technique to protect data and ensure their privacy in the cloud. However, this technique a big challenge which is: *How to answer the user queries over encrypted data*. A naive solution is to retrieve the encrypted database from the cloud to the client, decrypt it, and then evaluate the queries over *plaintext (non encrypted)* data. This solution is inefficient, because it does not take advantage of the cloud computing power for evaluating queries.

In this chapter, we consider the case where the encrypted data items are stored in *one node of the cloud*. Then, we propose a novel system, called BuckTop [52, 51], to answer top-k queries over encrypted data in the cloud. BuckTop is designed to encrypt and outsource user sensitive data to the cloud. It comes with a top-k query processing algorithm that is able to process efficiently top-k queries over the encrypted data, without decrypting them in the cloud data centers.

This chapter is organized as follows. Section 3.4 presents a basic solution called OPE-based approach. It is based on Order Preserving Encryption. In Section 3.5, we introduce our main contribution in this chapter: BuckTop. We describe the its top-k query processing and false positive filtering algorithms. Section 3.6 describes our performance evaluation. Finally, Section 3.7 concludes.

3.2 Motivation

Cloud data outsourcing provides users and companies with powerful capabilities to store and process their data in third-party data centers. However, when a user stores her data in a public cloud, she loses the physical access control to the data. Thus, potentially sensitive data gets at risk of security attacks, e.g., from the employees of the cloud provider. According to a recent report published by the Cloud Security Alliance [19], security attacks

are one of the main concerns for the cloud users.

Top-k query processing over encrypted data is critical for many applications that outsource sensitive data. For example, consider a university that outsources the students database in a public cloud, with non-trusted nodes. The database is encrypted for privacy reasons. Then, an interesting top-k query over the outsourced encrypted data is the following: return the k students that have the worst averages in some given courses.

There are many different approaches for processing top-k queries over plaintext data. One of the best known approaches is TA (threshold algorithm) [27] that works on sorted lists of attribute values. TA can find efficiently the top-k results because of a smart strategy for deciding when to stop reading the database. However, TA and its extensions assume that the attribute values are available as plaintext.

To the best of our knowledge, Meng et al [87] propose the only solution for processing top-k queries over encrypted data. They assume the existence of two non-colluding nodes in the cloud, one of which can decrypt the data (using the decryption key) and execute a TA-based algorithm. Our assumptions about the cloud are different, as we do not trust any node of the cloud.

In this chapter, we propose a basic approach called OPE-based that preserve the data privacy while preserving top-k queries in clouds. It uses a combination of the order preserving encryption (OPE) and the FA algorithm for privacy preserving top-k query processing. Then, we propose a complete system, called *BuckTop*, the first efficient approach for processing top-k queries over encrypted data. We call it BuckTop, as it uses the *bucketization* technique to manage the encrypted data in the server. This chapter includes the following contributions:

- A new top-k query processing algorithm that, given a database of encrypted data stored in buckets, returns a set, which is proved to contain the encrypted data corresponding to the top-k results.
- A novel powerful filtering algorithm that significantly filters in the server the false positives included in the set of encrypted data returned by the top-k query processing algorithm. We prove theoretically the correctness of the filtering algorithm.

3.3 Problem Definition

In this section, we first describe the adversary model which we consider, then we state the problem of processing top-k queries over encrypted data in the cloud.

3.3.1 Adversary Model

An adversary model generally specifies what an adversary or attacker is allowed to do during an execution of a secure protocol. In our work, we consider the honest-but-curious adversary model for the cloud. This model is widely used in many solutions proposed for secure processing of the different queries [46].

3.3.2 Problem Statement

Let us now formally state the problem which we address. Let D be a database, and $E(D)$ be its encrypted version such that each data $c \in E(D)$ is the ciphertext of a data $d \in D$, *i.e.*, $c = Enc(d)$ where $Enc()$ is an encryption function. *We assume that the database $E(D)$ is stored in one node of the cloud.*

Given a number k and a scoring function f. Like many previous works on top-k query processing (*e.g.*, [27]), we assume that the scoring function is monotonic. our goal is to develop an algorithm A, such that when A is executed over the database $E(D)$, its output contains the ciphertexts of the top-k results.

A naive approach for top-k query processing over the encrypted database $E(D)$ is to retrieve it from the cloud, decrypt all of its data, run an existing top-k algorithm over the plaintext data, and return the top-k results to the user. However, this approach is impractical, particularly for very large databases.

3.4 OPE-Based Approach

In this section, we propose a basic approach, called OPE-based, that uses a combination of the order preserving encryption (OPE) [1] and the FA algorithm [25] for privacy preserving top-k query processing.

3.4.1 Data Encryption

Let us first explain how the local scores are encrypted. With the OPE-based approach, the local scores (attribute values) in the sorted lists are encrypted using the order preserving encryption technique. We also use a *deterministic* encryption method for encrypting the ID of data items. The *deterministic* encryption generates the same ciphertexts for two equal inputs. This allows us to do random access to the encrypted sorted lists by using the ID of data items.

After encrypting the data IDs and local scores in each sorted list, the lists are sent to the cloud. Notice that all we need is that the lists in the cloud be sorted in the same order they were before encryption.

3.4.2 Top-k Query Processing

Let us now describe how top-k queries can be answered in the cloud over the encrypted data. Given a top-k query Q with a scoring function f, the query is sent to the cloud. Then, the cloud uses the FA algorithm for processing Q as follows. It continuously performs sorted access in parallel to each sorted list, and maintains the encrypted data IDs and their encrypted local scores in a set Y. When there are at least k encrypted data IDs in Y such that each of them has been seen in each of the lists, then the cloud stops doing sorted access to the lists. Then, for each data item d involved in Y, and each list L_i, the cloud performs random access to L_i to find the encrypted local scores of d in L_i (if it

has not been seen yet). The cloud sends Y to the user machine which decrypts the local scores of each item $d \in Y$, computes their overall scores, and find the final k items with the highest overall scores.

Theorem 1. *Given a top-k query with a monotonic scoring function, the OPE-based approach returns a set that includes the encrypted top-k elements.*

Proof. Let Y be the set of data items, which have been seen by top-k query processing algorithm in some lists before it stops. Let $Y' \subseteq Y$ be set of data items that have been seen in all lists. Let $d' \in Y'$ be the data item whose overall score among the data items in Y' is the minimum. In each list L_i, let s'_i be the real (plaintext) local score of d' in L_i.

We show that any data item d, which has not been seen by the algorithm under sorted access, has an overall score that is less than or equal to that of d'. In each list L_i, let s_i be the plaintext local score of d in L_i. Since d has not been seen by the top-k query processing algorithm, and the encrypted data items in the lists are sorted according to their initial order, we have $s_i \leq s'_i$, for $1 \leq i \leq m$. Since, the scoring function f is monotonic, then we have $f(s_1, ..., s_m) \leq f(s'_1, ..., s'_m)$. Thus, the overall score of d is less than or equal to that of d'. Therefore, the set Y contains at least k data items whose overall scores are greater than or equal to that of the unseen data d. □

3.5 BuckTop System

The OPE approach, presented in the previous section, evaluates correctly the top-k queries over encrypted data. But, as shown by our experiments reported in Section 3.6, there may be a high number of false positives which are sent from the cloud to the client, and this renders OPE inefficient in practice. This is why, we propose the BuckTop approach which is much more efficient than OPE.

In this section, we present our BuckTop system. We first describe the architecture of BuckTop, and introduce our method for encrypting the data items and storing them in the cloud. Afterwards, we propose an algorithm for processing top-k queries over encrypted data, and an algorithm for filtering the false positives in the cloud.

3.5.1 System Architecture

The architecture of BuckTop system is shown in Figure 3.1, it has two main components:

- **Trusted client.** It is responsible for encrypting the user data, decrypting the results and controlling the user accesses. The security keys used for data encryption/decryption are managed by this part of the system. When a query is issued by a user, the trusted client checks the access rights of the user. If the user does not have the required rights to see the query results, then her demand is rejected. Otherwise, the query is transformed to a query that can be executed over the encrypted data.

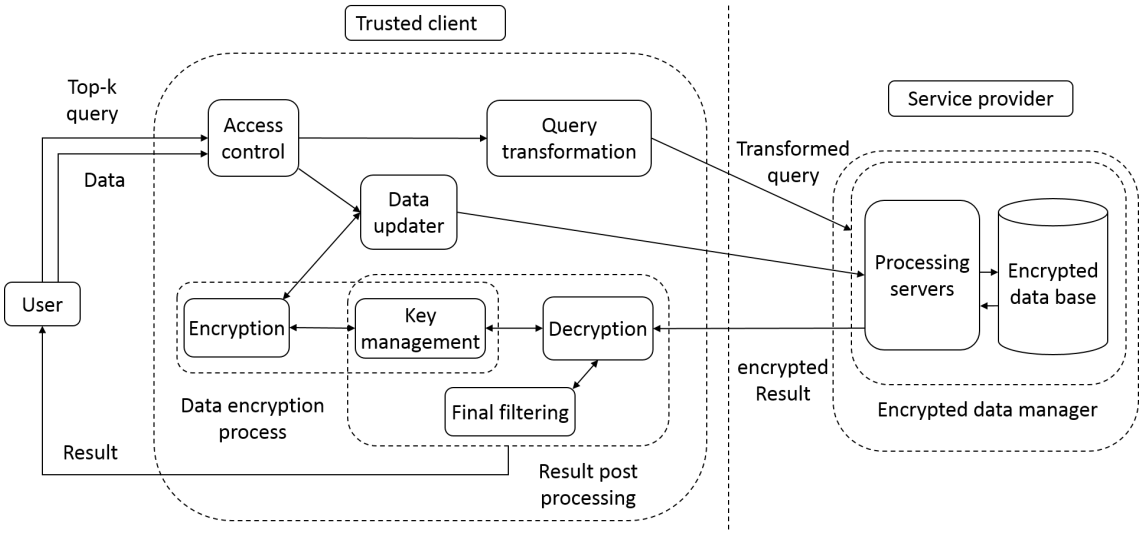

Figure 3.1 – BuckTop system architecture

For example, suppose we have a relation R with attributes att_1, att_2,..., att_m, and the user issues the following query:

*SELECT * FROM R ORDERED BY* $f(att_1, \ldots, att_m)$ *LIMIT* k;

This query is transformed to:

*SELECT * FROM E(R) ORDERED BY* $F(E(att_1),\ldots, E(att_m))$ *LIMIT* k;

where $E(R)$ and $E(att_i)$ are the encrypted name of the relation R and the attribute att_i respectively.

Note that the trusted client component should be installed in a trusted location, *e.g.*, the machine(s) of the person/organization that outsources the data.

- **Service provider.** It is installed in the cloud, and is responsible for storing the encrypted data, executing the queries provided by the trusted client, and returning the results. Note that this component does not keep any security key, and it cannot decrypt the encrypted data in the cloud.

3.5.2 Data Encryption

Let us now present our approach for encrypting and outsourcing the data to the cloud. As mentioned before, the trusted client component of BuckTop is responsible for encrypting the user databases. Before encrypting a database, the trusted client creates sorted lists for all important attributes, *i.e.*, those that may be used in the top-k queries. Then, each sorted list is partitioned into *buckets*. There are several methods for partitioning a sorted list, for example dividing the attribute domain of the list to almost equal intervals or creating buckets with equal sizes [35]. In the current implementation of our system, we use the

latter method, *i.e.*, we create buckets with almost the same size where the bucket size is configurable by the system administrator.

Let b_1, b_2, ..., b_t be the created buckets for a sorted list L_j. Each bucket b_i has a lower bound, denoted by $min(b_i)$, and an upper bound, denoted by $max(b_i)$. A data item d is in the bucket b_i, if and only if its local score (attribute value) in the list L_j is between the lower and upper bounds of the bucket, *i.e.*, $min(b_i) \leq s_j(d) < max(b_i)$.

We use two types of encryption schemes (methods): *deterministic* encryption to encrypt the data item ids and *probabilistic* encryption to encrypt the local scores in the sorted lists. The deterministic encryption allows us to have the same encrypted ID for each data item in all sorted lists.

With the *probabilistic* encryption, for the same plaintexts different ciphertexts are generated, but the decryption function returns the same plaintext for them. Thus, for example if two data items have the same local scores in a sorted list, their encrypted scores may be different. The probabilistic encryption is the strongest type of encryption.

After encrypting the data IDs and local scores of each list L_i, the trusted client puts them in their bucket (chosen based on the local score). Then, the trusted client sends the buckets of each sorted list to the cloud. The buckets are stored in the cloud according to their lower bound order. However, there is no order for the data items inside each bucket, *i.e.*, *the place of the data items inside each bucket is chosen randomly*. This prevents the cloud to know the order of data items inside the buckets.

3.5.3 Top-k Query Processing

The main idea behind top-k query processing in BuckTop system is to use the bucket boundaries to decide when to stop reading the encrypted data from the lists.

Given a top-k query Q including a number k and a scoring function f. To answer Q, the following top-k processing algorithm is executed by the cloud service provider:

1. Let Y be an empty set;
2. Perform sorted access to the lists:
   - 2.1 Read the next bucket, say b_i, from each list L_i (starting from the head of the list);
   - 2.2 For each encrypted data d contained in the bucket b_i:
     - 2.2.1. Perform random access in parallel to the other lists to find the encrypted score and the bucket of d in all lists;
     - 2.2.2. Compute a minimum overall score for d, denoted by $min\_ovl(d)$, by applying the scoring function on the lower bound of the buckets that contain d in different lists. Formally, $min\_ovl(d) = f(min(b_1), min(b_2), ..., min(b_m))$, where b_i is the bucket involving d in the list L_i.
     - 2.2.3. Store the encrypted ID of d, its encrypted local scores, and its *min_ovl* score in the set *Y*.

2.3 Compute a threshold TH as follows: $TH = f(min(b'_1), min(b'_2), ..., min(b'_m))$, where b'_i is the last bucket seen under sorted access in the L_i, for $1 < i < m$. In other words, *TH* is computed by applying the scoring function on the lower bounds of the last seen buckets in the lists.

2.4 If the set Y contains at least k encrypted data items having minimum overall scores higher than *TH*, then stop. Otherwise, go to Step 2.1.

When the top-k query processing algorithm stops, the set Y includes the encrypted top-k data items (see the proof below). This set is sent to the trusted client that decrypts its contained data items, computes the overall scores of the items, removes the false positives (*i.e.*, the items that are in Y but not among the top-k results), and returns the top-k items to the user.

Let us prove that BuckTop's top-k query processing works correctly, but first we have to prove that the minimum overall score of any data item d, *i.e.*, $min\_ovl(d)$, which is computed based on the lower bound of its buckets, is less than or equal to its overall score. We also show that the maximum overall score of d, *i.e.*, $max\_ovl(d)$, is higher than or equal to its overall score.

Lemma 2. *Given a monotonic scoring function f, the minimum overall score of any data item d is less than or equal to its overall score.*

Proof. The minimum overall score of a data item d is calculated by applying the scoring function on the lower bound of the buckets in which d is involved. Let b_i be the bucket that contains d in the list L_i. Let s_i be the local score of d in L_i. Since $d \in b_i$, its local score is higher than or equal to the lower bound of b_i, *i.e.*, $min(b_i) \leq s_i$. Since f is monotonic, we have $f(min(b_1), ..., min(b_m)) \leq f(s_1, ..., s_m)$. Therefore, the minimum overall score of d is less than or equal to its overall score. □

Lemma 3. *Given a monotonic scoring function f, the maximum overall score of any data item d is greater than or equal to its overall score.*

Proof. The proof can be done in a similar way as Lemma 2. □

Now, we show in the following theorem that the output of BuckTop top-k query processing algorithm contains the encrypted top-k data items.

Theorem 4. *Given a top-k query with a monotonic scoring function f, the output of BuckTop top-k query processing algorithm contains the encrypted top-k results.*

Proof. Let Y be the output of the BuckTop top-k query processing algorithm, *i.e.*, the set that contains all the encrypted data items seen under sorted access when the algorithm ends. We show that each data item d that is not in Y ($d \notin Y$), has an overall score that is less than or equal to the overall score of at least k data items in Y. Let s_i be the local score of d in the list L_i. Let b'_i be the last bucket seen under sorted access in the list L_i, *i.e.*, when the algorithm ends. Since d is not in Y, it has not been seen under

sorted access in the lists. Thus, its involving buckets are after the buckets seen under sorted access by the algorithm. Therefore, we have $s_i < min(b'_i)$ for $1 \leq i \leq m$, *i.e.*, the local score of d in each list L_i is less than the lower bound of the last bucket read under sorted access in L_i. Since the scoring function is monotonic, we have $f(s_1, ..., s_m) < f(min(b'_1), min(b'_2), ..., min(b'_m)) = TH$. Thus, the overall score of d is less than *TH*. When the algorithm stops, there are at least k data items in Y whose minimum overall scores are greater than or equal to *TH*. Thus, their overall scores are at least *TH*. Therefore, their overall scores are greater than or equal to that of the data item d. □

3.5.4 False Positive Filtering

In the set Y returned by the top-k query processing algorithm of BuckTop, in addition to the top-k results there may be false positives. Below, we propose a filtering algorithm to eliminate most of them in the cloud, without decrypting the data items. As shown by our experimental results, our filtering algorithm eliminates most of the false positives (more than 99% in the different tested datasets). This improves significantly the response time of top-k queries, because the eliminated false positives do not need to be communicated to the trusted client and should not be decrypted by it.

In the filtering algorithm, we use the *maximum overall score*, denoted by max_ovl of each data item. This score is computed by applying the scoring function on the upper bound of the buckets involving the data item in the lists. The algorithm proceeds as follows:

1. Let $Y' \subseteq Y$ be the k data items in Y that have the highest minimum overall scores (*min_ovl*) among the items contained in Y.

2. Let d_{min} be the data item that has the lowest *min_ovl* score in Y'.

3. For each item $d \in Y$

   3.1 Compute the maximum overall score of d, *i.e.*, $max\_ovl(d)$, by applying the scoring function on the upper bound of the buckets involving d in the lists. Formally, let $max(b_i)$ be the upper bound of the bucket involving d in the list L_i. Then, $max\_ovl(d) = f(max(b_1), max(b_2), ..., max(b_m))$.

   3.2 If the maximum overall score of d is less than or equal to the minimum overall score of d_{min}, then remove d from Y. In other words, if $max\_ovl(d) \leq min\_ovl(d_{min}) \Rightarrow Y = Y - \{d\}$

Let us prove that the filtering algorithm works correctly. The following theorem shows that the filtering algorithm works correctly, i.e., the removed data are only false positives.

Theorem 5. *Any data item removed by the filtering algorithm cannot belong to the top-k results.*

Proof. The proof can be done by considering the fact that any removed data item d has a maximum overall score that is lower than the minimum overall score of at least k data items. Thus, by using Lemmas 2 and 3, the overall score of d is less than or equal to that of at least k data items. Therefore, we can eliminate d. □

Example Consider the encrypted database shown in Figure 3.2, and a top-k query with $k = 3$ and a scoring function that computes the sum of the local scores in the sorted lists.

List 1			List 2			List 3		
bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score
B_{11}	$E(d1)$	$E(27)$	B_{21}	$E(d6)$	$E(28)$	B_{31}	$E(d2)$	$E(22)$
B_{11}	$E(d3)$	$E(30)$	B_{21}	$E(d3)$	$E(29)$	B_{31}	$E(d3)$	$E(25)$
B_{11}	$E(d6)$	$E(26)$	B_{21}	$E(d2)$	$E(26)$	B_{31}	$E(d6)$	$E(27)$
B_{12}	$E(d2)$	$E(15)$	B_{22}	$E(d1)$	$E(24)$	B_{32}	$E(d5)$	$E(21)$
B_{12}	$E(d8)$	$E(20)$	B_{22}	$E(d7)$	$E(21)$	B_{32}	$E(d1)$	$E(20)$
B_{12}	$E(d5)$	$E(24)$	B_{22}	$E(d4)$	$E(19)$	B_{32}	$E(d9)$	$E(18)$
B_{13}	$E(d4)$	$E(14)$	B_{23}	$E(d5)$	$E(16)$	B_{33}	$E(d8)$	$E(17)$
B_{13}	$E(d7)$	$E(12)$	B_{23}	$E(d9)$	$E(13)$	B_{33}	$E(d7)$	$E(14)$
B_{13}	$E(d9)$	$E(11)$	B_{23}	$E(d8)$	$E(10)$	B_{33}	$E(d4)$	$E(11)$
...	...	...	...	...	...	...	...	...

(a) Encrypted database

List 1			List 2			List 3		
bucket ID	min	max	bucket ID	min	max	bucket ID	min	max
B_{11}	24.6	32	B_{21}	25.5	31	B_{31}	21.9	28
B_{12}	14.8	24.1	B_{22}	18	24.1	B_{32}	17.7	21.5
B_{13}	10.7	14.2	B_{23}	9	16.5	B_{33}	10	17.3

(b) Bucket boundaries

Figure 3.2 – Example of an encrypted database with the information about the created buckets

BuckTop algorithm starts by scanning the data items in the fist bucket of each list. For each seen data item in the list i, in our case $d1, d2, d3$ and $d6$, BuckTop does a random access in the other lists to get the min of the bucket where the data item is found and calculates its minimum overall score: $min\_ovl(d_1) = 60.3$, $min\_ovl(d_2) = 62.2$, $min\_ovl(d_3) = 72$, $min\_ovl(d_6) = 72$. Also a threshold is calculated $TH = 72$. Here, BuckTop finds that only two data items that have an overall score greater than or equal to the TH, so it continues the sorted scan. In the 2nd buckets, BuckTop calculates $min\_ovl(d_4) =$

38.7, $min\_ovl(d_5) = 41.5$, $min\_ovl(d_7) = 38.7$, $min\_ovl(d_8) = 33.8$, $min\_(d9) = 37.4$ and $TH = 50.5$. we observe that the minimum overall score of d_3, d_1, d_6 and d_2 is greater than *TH*, so BuckTop stops data scanning and stores all the seen data items in a set Y, *i.e.*, $Y = \{E(d3), E(d1), E(d6), E(d2), E(d5), E(d8), E(d7), E(d4), E(d9)\}$. after that, The filtering algorithm is applied on the data items of *Y*. The k data items that have the highest *min_ovl* scores in *Y* are $Y' = \{d_3, d_6, d_2\}$. Among the data items in Y', the minimum *min_ovl* belongs to d_2, which is $min\_ovl(d_2) = 62.2$. For filtering, we need to compare *max_ovl* score of each data item in *Y - Y'* with $min\_ovl(d_2)$. We find that $max\_ovl(d1) = 77.6$, $max\_ovl(d5) = 62.1$, $max\_ovl(d8) = 57.9$, $max\_ovl(d7) = 55.6$, $max\_ovl(d4) = 55.6$, $max\_ovl(d9) = 52.2$. We observe that we can eliminate d_5, d_8, d_7, d_4 and d_9 from Y because they have a *max_ovl* score less than *min_ovl* score of d_2. As a result, after the filtering, Y will contain the data items $Y = \{E(d_1), E(d_2), E(d_3), E(d_6)\}$. This set includes only one false positive, *i.e.*, d_2, which will be eliminated in the client side.

3.6 Performance Evaluation

In this section, we evaluate the performance of BuckTop using synthetic and real datasets. We first describe the experimental setup, and then report the results of our experiments.

3.6.1 Setup

We implemented our top-k query processing system and performed our tests on real and synthetic datasets. As in some previous work on encrypted data (*e.g.*, [46]), we use the Gowalla database, which is a location-based social networking dataset collected from users locations. The database contains 6 million tuples where each tuple represents user number, time, user geographic position, etc. In our experiments, we are interested in the attribute time, which is the second value in each tuple. As in [46], we decompose this attribute into 6 attributes (year, month, day, hour, minute, second), and then create a database with the following schema R(ID, year, month, date, hour, minute, second), where ID is the tuple identifier. In addition to the real dataset, we have also generated random datasets using uniform and Gaussian distributions.

We compare our solution with the two following approaches:

- *OPE*: this is the OPE-based solution (presented in Section 3.4) that uses the order preserving encryption for encrypting the data scores.
- *TA over plaintext data*: the objective is to show the overhead of top-k query processing by BuckTop over encrypted data compared to an efficient top-k algorithm over plaintext data.

In our experiments, we have two versions of each database: 1) the plaintext database used for running TA; 2) the encrypted database used for running BuckTop and OPE.

In our performance evaluation, we study the effect of several parameters: 1) n: the number of data items in the database; 2) m: the number of lists; 3) k: the number of required top items; 4) $bsize$: the number of data items in the buckets of BuckTop. The default value for n is 2M items. Unless otherwise specified, m is 5, k is 50, and $bsize$ is 20. In our tests,the default database is the synthetic uniform database.

In the experiments, we measure the following metrics:

- **Cloud top-k time:** the time required by the cloud provider to find the set that includes the top-k results, *i.e.*, the set Y.
- **Response time:** the total time elapsed between the time when the query is sent to the cloud and the time when the k decrypted results are returned to the user. This time includes the cloud top-k time, the filtering, and the result post-processing in the client (*e.g.*, decryption).
- **Filtering rate:** the number of false positives eliminated by the filtering algorithm in the cloud.

We performed our experiments using a node with 16 GB of main memory and Intel Core i7-5500 @ 2.40Ghz as processor.

3.6.2 Results

In this section, we reports all the experiment results that we obtained.

Effect of the Number of Data Items: In this experiment, we compare the performance of TA over plaintext data with BuckTop and OPE over encrypted data, while varying the number of data items, *i.e.*, n.

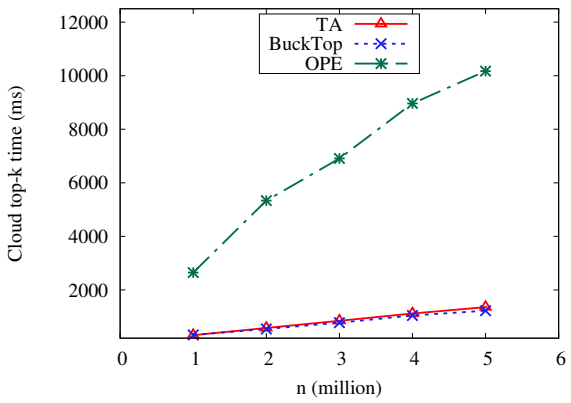

Figure 3.3 – Cloud top-k time vs. number of database tuples

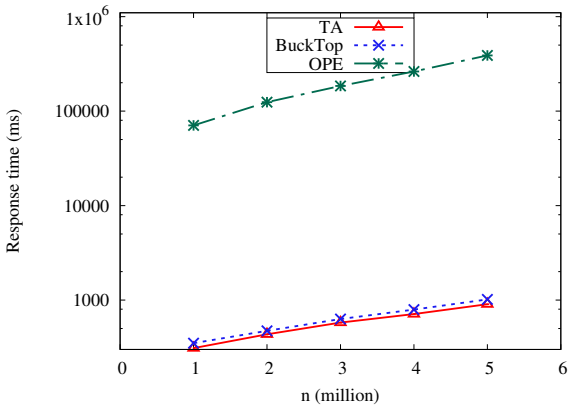

Figure 3.4 – Response time vs. number of database tuples

Figure 3.3 shows how cloud top-k time evolves, with increasing n, and the other parameters set as default values described in Section 3.6.1. The cloud top-k time of all

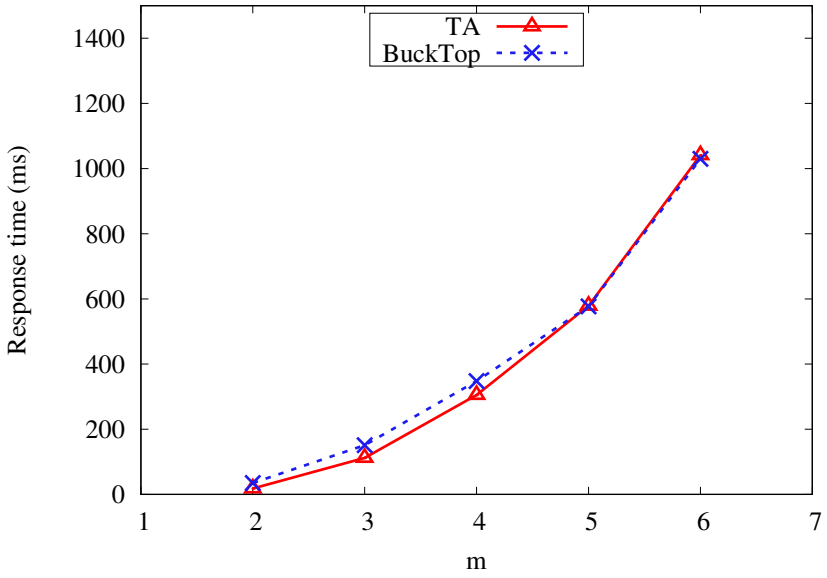

Figure 3.5 – Response time vs. m

approaches increases with n. But, OPE takes more time than the two other approaches, because it stops deeper in lists, and thus reads more data.

Figure 3.4 shows the total response time of BuckTop, OPE and TA while varying n, and the other parameters set as default values. Note that the figure are is in logarithmic scale. TA does not need to decrypt any data, so its response time is almost the same as its cloud time. The response time of BuckTop is slightly higher than its cloud top-k time, as in addition to top-k query processing it performs the filtering in the cloud and also needs to decrypt at least k data items. We see that the response time of OPE is much higher than its cloud top-k time. The reason is that OPE returns to the trusted client a lot of false positives, which should be decrypted, and removed from the final result set. But, this is not the case for BuckTop as its filtering algorithm removes almost all the false positives in the cloud, thus there is no need to decrypt them.

Effect of the Number of Queried Attributes: Figure 3.5 reports the server runtime of TA and BuckTop when varying m (i.e., the number of attributes in the scoring function), and the other parameters set as default values. We observe that when $m \leq 5$, the two algorithms have almost the same server runtime. But when m is more than four, BuckTop performs better than TA.

Effect of k: Figure 3.6 shows the total response times of BuckTop with increasing k, and the other parameters set as default values. We observe that with increasing k the response time increases. The reason is that Bucktop needs to go deeper in the lists to find the top-k results. In addition, increasing k augments the number of data items that the trusted client needs to decrypt (because at least k data items are decrypted by the trusted client).

Effect of Bucket Size: Figure 3.7 reports the response time of BuckTop when varying

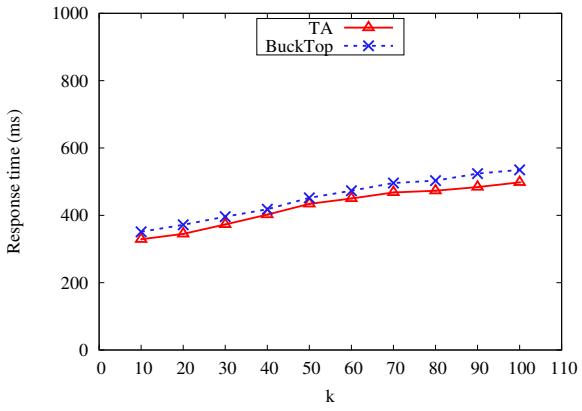

Figure 3.6 – Response time vs. k

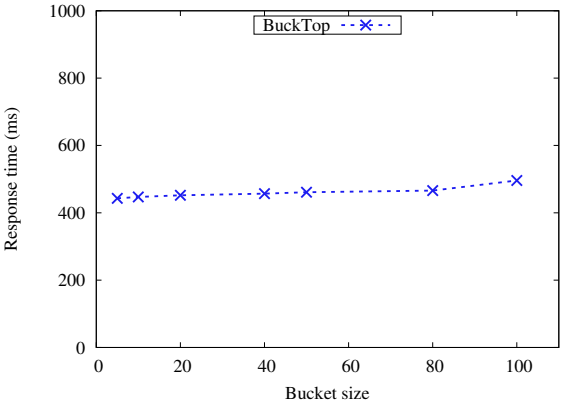

Figure 3.7 – Response time vs. bucket size

the size of buckets, and the other parameters set as default values. We observe that the response time increases when the bucket size increases. The reason is that the top-k query processing algorithm of Bucktop reads more data in the lists, because the data are read bucket by bucket. In addition, increasing the bucket size increases the number of false positives to be removed by the filtering algorithm, and eventually decrpting the none eliminated false positives in the client side.

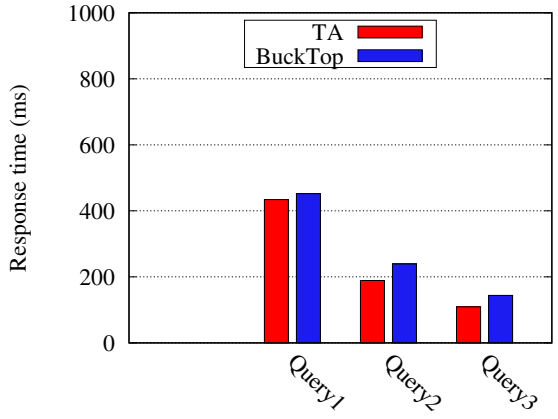

Figure 3.8 – Response time using different queries

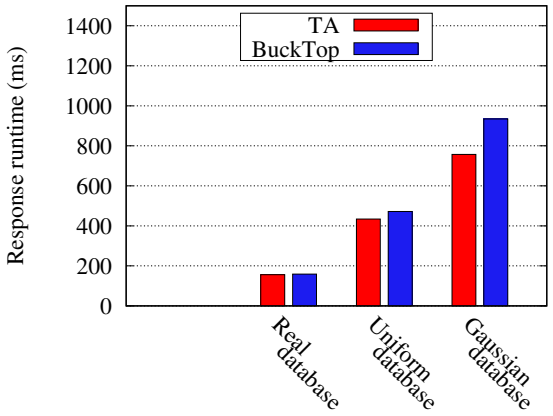

Figure 3.9 – Response time using different datasets

Effect of Queries: We evaluated the effect of queries and their scoring functions on the performance of our approach. For this, we tested three different queries with different scoring functions. In the first query, noted as Q_1, the scoring function is $sf_1(s_1 + s_2 + ... + s_n) = s_1 + s_2 + ... + s_n$. In this query, we have the same coefficient (impact) for all scoring attributes. In the 2nd and 3rd queries, there is a higher skew in the coefficients: in Q_2, we set $sf_2(s_1 + s_2 + ... + s_n) = 1 \times s_1 + 2 \times s_2 + ... + n \times s_n)$, and in Q_3 we set

Database size (M)	1	2	3	4	5	6
Rate of eliminated false positives	100%	100%	100%	99.99%	99.99%	100%

A: over Uniform dataset

Database size (M)	1	2	3	4	5	6
Rate of eliminated false positives	99.98%	99.99%	99.99%	99.99%	99.99%	99.99%

B: over Real dataset

Database size (M)	1	2	3	4	5	6
Rate of eliminated false positives	99.94%	99.96%	99.97%	99.98%	99.98%	99.98%

C: over Gaussian dataset

Table 3.1 – False positive elimination by filtering algorithm over different datasets

$sf_3(s_1 + s_2 + ... + s_n) = 2^1 \times s_1 + 2^2 \times s_2 + ... + 2^n \times s_n$.

Figure 3.8 shows the response time of TA and BuckTop using the three queries. Note that in our experiments the default query is Q_1. We observe that for the query Q_3, BuckTop performs better than the other queries. The reason is that in Q_3, only one or two attributes are the dominating factors of the scoring function (*i.e.*, those with very high coefficients). In this case, the top-k processing algorithm takes less time to stop, and this is why the response time of BuckTop with Q_3 is lower than the other queries.

Performance over Different Datasets: We study the effect of the datasets on the performance of BuckTop and TA using different datasets: synthetic datasets with uniform and Gaussian distributions, and real dataset (Gowalla). Figure 3.9 shows the response time of TA and BuckTop over different datasets, while other parameters are set as default values. We see that over the uniform and real datasets, BuckTop and TA have approximately the same response times. Over the Gaussian dataset, the response time of BuckTop is a little higher than TA. The reason is that over this dataset the number of false positives is higher than the other datasets, thus more encrypted data should be decrypted by the trusted client.

Effect of the Filtering Algorithm: BuckTop's filtering algorithm is used to eliminate/reduce the false positives in the cloud. We study the filtering rate by increasing the size of the dataset. For the uniform synthetic dataset, the results are shown in Table 3.1-*A*. For datasets with up to three million data items, the filtering method eliminates 100% of the false positives, and the cloud returns to the trusted client only the k data items that are the result of the query. For larger datasets, BuckTop filters up to 99,99% of the false positives. By using the Gaussian dataset, we obtain the results shown in Table 3.1-*C*. We see that around 99,94% of false positives are eliminated.

Over the real dataset, Table 3.1-*B* shows the filtering rate. We observe that the filtering algorithm eliminates 99,99% of false positives. Thus, the filtering algorithm is very efficient over all the tested datasets. However, there is a little difference in the filtering rate for different datasets because of the local score distributions. For example, in the

Gaussian distribution, the local scores of many data items are very close to each other, thus the filtering rate decreases in this dataset.

3.7 Conclusion

In this chapter, we proposed a novel system, called BuckTop, designed to encrypt sensitive data items, outsource them to a non-trusted cloud, and answer top-k queries. It uses the bucketization technique to manage the encrypted data in the cloud. BuckTop has a top-k query processing algorithm that is executed over encrypted data, and returns a set containing the top-k results, without decrypting the data in the cloud. It also comes with a powerful filtering algorithm that eliminates significantly the false positives from the result set.

We validated our system through experimentation over synthetic and real datasets. The experimental results show excellent performance gains for BuckTop compared to OPE over encrypted data, and TA algorithm over original (plaintext) data. They illustrate that the overhead of using BuckTop for top-k processing over encrypted data is very low, because of efficient top-k processing and false positive filtering.

Chapter 4

Top-k Query Processing over Distributed Encrypted Data

4.1 Introduction

In this chapter, we consider the case of distributed datasets where a dataset (*e.g.*, a relation) is vertically fragmented and distributed across multiple nodes of a cloud data center. The user data are encrypted (for privacy reasons) and distributed (for performance reasons) across multiple nodes. In this context, we address the problem of privacy-preserving top-k query processing.

Privacy preserving top-k query processing is critical for many distributed applications that outsource sensitive data. For example, consider a university that outsources the students database in a public cloud, in Infrastructure-as-a-Service (IaaS) mode, with non-trusted nodes. The database is vertically partitioned (for performance reasons) and encrypted. Then, an interesting top-k query over the encrypted distributed data is the following: return the k students that have the worst averages in some given courses.

The problem of top-k query processing over distributed data has been yet addressed over plaintext data, *e.g.*, [15]. However, the proposed approaches assume the existence of local scores of the data items (*i.e.*, their attribute values) in plaintext, and there is no efficient solution capable of evaluating efficiently top-k queries over encrypted data in distributed environments.

In this chapter, we address the problem of evaluating top-k queries over encrypted data distributed across multiple nodes of a cloud data center. We first present two simple TA-based approaches, called Remote-TA and Block-TA, that are coordinated by the client. These two approaches can correctly find top-k results from the encrypted data, but may need many round-trips between the client and the nodes, and also need to decrypt many encrypted data in the client side. Then, we propose two efficient systems, called SDB-TOPK (Secure Distributed Bucket based TOP-K) [55] and SD-TOPK (Secure Distributed TOPK) [54, 53]. They are executed in the nodes of the cloud data center. Both SDB-TOPK and SD-TOPK include efficient algorithms to process top-k query over distributed encrypted data and a powerful filtering algorithm that filters the false positives as much

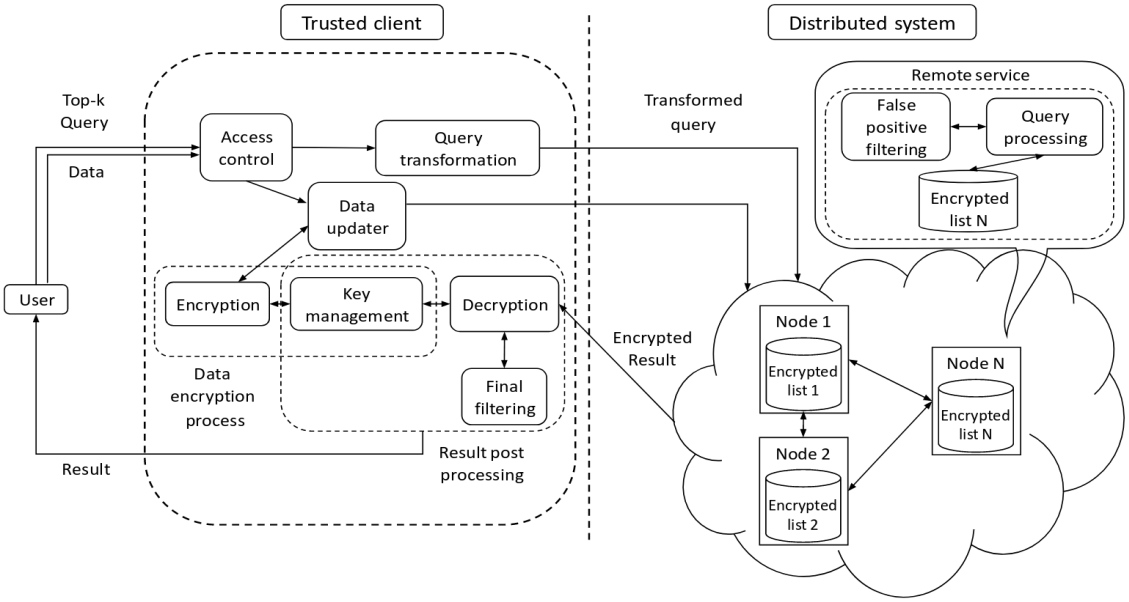

Figure 4.1 – SDB-TOPK and SD-TOPK architecture

as possible in the nodes. As a result, they return a small set of encrypted data that will be decrypted in the client side. We theoretically prove the correctness of the query processing algorithms. We analyze the security of our approaches by measuring the amount of data that can be revealed to the adversary, and propose efficient strategies for enforcing it.

The rest of this chapter is organized as follows. Section 4.2 formally defines the problem we address. In Section 4.3, we propose basic approaches based on the TA algorithm called Remote-TA and Block-TA. In Sections 4.4 and 4.5, we propose our two main solutions called SDB-TOPK and SD-TOPK. Section 4.6 analyzes the security of the proposed solutions. Section 4.8 describes our performance evaluation. And Section 4.9 concludes.

4.2 Problem Definition

In this section, we present the properties that we specify for the Our system and the data presentation, then we define the problem that we address.

4.2.1 Database Distribution and Systems Architecture

We suppose that the sorted lists database is fragmented over a number of nodes of *a cloud data center*. The architecture of both SDB-TOPK and SD-TOPK systems is shown in Figure 4.1. It is composed of:

- *Trusted client.* It is responsible for encrypting the user data, decrypting the results and controlling the user accesses

- *Remote service.* It is installed in the nodes of a cloud data center, and is responsible for storing the encrypted data, executing the queries provided by the trusted client, and returning the result.

Formally, let P be the set of the nodes of the cloud data center. Each sorted list L_i is stored over a node $p \in P$. We call p the *owner* of L_i. Each node $p \in P$ is the owner of at least one list of the database $E(D)$. Each list owner can perform sorted and random access only in the lists that it keeps, not in the other lists.

We consider the *honest-but-curious* adversary model for the nodes of a cloud data center.

4.2.2 Problem Statement

The problem we attack in this chapter is top-k query processing over encrypted and distributed data across the nodes of a cloud data center.

Let D be a database composed of n data items, and represented by m sorted lists. We want to encrypt the data items contained in the lists of D, and store the encrypted lists in the nodes of a cloud data center. Then, our goal is to develop a distributed algorithm A that given any top-k query q (including a scoring function f) returns the k data items that have the highest overall scores with regard to f. This should be done without decrypting the data items in the nodes, while minimizing the response time and the communication cost of the query execution.

4.3 TA-Based Approaches

In this section, we present two approaches based on the TA algorithm for top-k query processing over encrypted data in distributed environments: Remote-TA and Block-TA. In these approaches, the top-k query processing is coordinated in the trusted client.

4.3.1 Data Storage

To be able to execute TA-based algorithms over encrypted data, the trusted client stores the database in the nodes of the cloud data center as follows. It encrypts the pairs $\langle d, s_i(d)\rangle$ of the sorted lists using two encryption schemes: 1) *deterministic* to encrypt data identifier d; 2)*probabilistic* to encrypt local score of the data, *i.e.*, $s_i(d)$. The encrypted pairs $\langle E(d), E(s_i(d))\rangle$ in the lists are sorted in the same order as their initial order.

After encryption, the trusted client sends each encrypted sorted list to one node of the cloud data center nodes, which is called the *owner* of the list. The Remote-TA and Block-TA are based on the TA algorithm. The coordination is done in the trusted client, not in the nodes.

4.3.2 Remote-TA

Given a top-k query containing a number k and a scoring function f, Remote-TA proceeds as follows:

1. The trusted client asks the list owners for the encrypted pairs (encrypted data id and score) which are in position j of the lists (initially $j = 1$). The list owners return the asked data.

2. The trusted client decrypts the received encrypted scores and calculates a threshold TH by applying the scoring function on the decrypted scores.

3. Let S be the set of encrypted data items returned from the position j of the lists. The trusted client demands the list owners to return the encrypted scores of data items in S. Each list owner does random access in its list to find the encrypted scores of each data item in S, then sends them to the trusted client.

4. The coordinator collects the data such that, as a result, for each data item in S it has all its encrypted scores in all the lists .i.e for each $E(d_i)$ where $1 \leq i \leq size(S)$, the coordinator has $E(s_1(d_i)), E(s_2(d_i)), ..., E(s_m(d_i))$. it sends the data collected to the trusted client.

5. Trusted client decrypts each returned data item d and calculates its overall score $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$. Then, it checks if among the yet received data items there are at least k data items that have an overall score greater than or equal to TH. If this is the case, then it stops the algorithm and returns the k received data items that have the highest overall scores to the user. Otherwise, it increases j by one and restarts from step 1.

Example. We use the database shown in Table 4.1 composed of three lists, $list_1$, $list_2$, and $list_3$ stored respectively in nodes s_1, s_2 and s_3. The user asks for the top-4 data items in the database with SUM as a scoring function. As a first step of Remote-TA, the trusted client asks for the data items in the first position in each node. The list owner returns the asked data items to the trusted client who decrypts them and calculates the threshold $th = 30 + 29 + 27 = 86$. Then, the client asks the list owners to return the encrypted score of the data items $E(d3)$ and $E(d6)$. The client calculates the overall scores: $ov(d3) = 30 + 29 + 25 = 84$, $ov(d6) = 26 + 28 + 27 = 81$. Now, there are two data items that have an overall score greater than or equal to the threshold, thus it continues and asks the data items in the next position. Finally, in the fifth position, there are at least k data items with overall scores greater than or equal to TH: $ov(d1) = 71$, $ov(d2) = 63$, $ov(d3) = 84$, $ov(d5) = 61$, $ov(d6) = 81$. The trusted client stops asking for the data items. It returns to the user the result of the top-k query which is the set of data items $d1, d2, d3$ and $d6$.

List 1		List 2		List 3	
encrypted data item	encrypted local score	encrypted data item	encrypted local score	encrypted data item	encrypted local score
$E(d3)$	$E(30)$	$E(d3)$	$E(29)$	$E(d6)$	$E(27)$
$E(d1)$	$E(27)$	$E(d6)$	$E(28)$	$E(d3)$	$E(25)$
$E(d6)$	$E(26)$	$E(d2)$	$E(26)$	$E(d2)$	$E(22)$
$E(d5)$	$E(24)$	$E(d1)$	$E(24)$	$E(d5)$	$E(21)$
$E(d8)$	$E(20)$	$E(d7)$	$E(21)$	$E(d1)$	$E(20)$
$E(d2)$	$E(15)$	$E(d4)$	$E(19)$	$E(d9)$	$E(18)$
$E(d4)$	$E(14)$	$E(d5)$	$E(16)$	$E(d8)$	$E(17)$
$E(d7)$	$E(12)$	$E(d9)$	$E(13)$	$E(d7)$	$E(14)$
$E(d9)$	$E(11)$	$E(d8)$	$E(10)$	$E(d4)$	$E(11)$
...	...	...	...	...	...

Table 4.1 – Example of an encrypted database

4.3.3 Block-TA

Block-TA is an improvement of the Remote-TA algorithm where the encrypted data items are read block by block. Indeed, in order to minimize the communication cost, in the first step of Block-TA, the trusted client asks blocks of predefined size from the list owners. After decrypting the data items of the block, the trusted client computes the threshold by applying the scoring function on the scores of the last data items in the blocks, and stops if among yet received data items there are at leas k data items with overall scores higher than or equal to the threshold. Otherwise it retrieves the next block, and so on.

Example. Consider the same previous example presented in Table 4.1, the same top-4 query and the SUM function.

Let us run Block-TA on the database of Table 4.1 by using blocks of size 4. The trusted client asks the list owners to return the 4 first data items in each list. The owner of the list L_1 returns $d3, d1, d6$ and $d5$. It also returns $E(24)$ which is the encrypted score of the last data item in the block. This is used later to calculate the threshold. The owner of L_2 returns $d3, d6, d2, d1$ and $E(24)$. The owner of L_3 returns $d6, d3, d2, d5$ and $E(21)$. Then, the trusted client decrypts the received data and calculates the threshold $TH = 24 + 24 + 21 = 69$. The client asks for the encrypted score of the returned data items $d1, d2, d3, d5$ and $d6$. When it gets the asked scores, it decrypts them and calculates the overall score of each data item: $ov(d1) = 71$, $ov(d2) = 63$, $ov(d3) = 84$, $ov(d5) = 61$, $ov(d6) = 81$. The client finds that only $d1, d3$ and $d6$ have an overall score greater than or equal to the threshold TH, so it asks the next block of four data items in each list. The owner of L_1 returns $d8, d2, d4, d7$ and $E(12)$. The owner of L_2 returns $d7, d4, d5, d9$ and $E(13)$, and that of L_3 returns $d1, d9, d8, d7$ and $E(14)$. The trusted client calculates the threshold $TH = 12 + 13 + 14 = 39$. Then, the client asks each

node to return the encrypted score of the data items and calculates their overall score: $ov(d4) = 44, ov(d7) = 47, ov(d8) = 47, ov(d9) = 42$. The trusted client finds that there are at least 4 data items with overall scores greater than or equal to TH. Thus, it stops communicating with the list owners, and returns to the user the data items $d1, d2, d3, d6$ that have the biggest overall score among the data items received from the list owners.

4.4 SDB-TOPK System

In this section, we present our first contribution called SDB-TOPK; we describe our technique for encrypting the data and outsourcing them to the nodes of the cloud. Then, we propose our SDB-TOPK top-k query processing and false positive filtering algorithms.

4.4.1 Data Encryption and Outsourcing

Before outsourcing a database, SD-TOPK creates sorted lists for all important attributes, *i.e.*, those that may be used in the top-k queries. Then, each sorted list is partitioned into buckets. Let b_1, b_2, ..., b_t be the created buckets for a sorted list L_j. Each bucket b_i has a lower bound, denoted by $min(b_i)$, and an upper bound, denoted by $max(b_i)$. A data item d is in the bucket b_i, if and only if its local score (attribute value) in the list L_j is between the lower and upper bounds of the bucket, *i.e.*, $min(b_i) \leq s_j(d) < max(b_i)$.

We use two types of encryption schemes for encrypting the data item ids and the local scores of the sorted lists: *deterministic* and *probabilistic*. The *deterministic* scheme is used to encrypt the ID of the data items. We use the *probabilistic* scheme to encrypt the local scores (attribute values) of data items.

After encrypting the data IDs and local scores of each list L_i, the trusted client puts them in their bucket (chosen based on the local score). Then, it sends the buckets of each sorted list to one node in the cloud. The buckets are stored in the nodes according to their lower bound order. However, there is no order for the data items inside each bucket, *i.e.*, the position of the data items inside each bucket is chosen randomly.

4.4.2 Top-k Query Processing

The main idea behind top-k query processing algorithm of SDB-TOPK is to use the bucket boundaries and a new technique to decide when to stop reading the encrypted data from the buckets.

For each top-k query, one of the nodes of the cloud performs the coordination between the nodes to execute the query. We call this node as *coordinator*. The coordinator may be the node that initially receives the user's query or be randomly chosen among the system nodes.

Given a top-k query with a number k and a scoring function f, SDB-TOPK chooses a node as query coordinator. The coordinator sets a variable $j = 1$, and then the following steps are performed to answer the query (see pseudo-code in Algorithm1):

Algorithm 1: SDB-TOPK Query Processing Algorithm

Input: The top-k query Q including k and the scoring function f; Encrypted database partitioned vertically over m nodes $N_1, \ldots, N_m$;

Output: Set Y containing encrypted top-k data items

```
1  begin
2      j = 1;
3      Let N_0 be the coordinator node;
4      while !stop do
5          N_0 asks N_1 to N_m in parallel to return their j^th bucket;
6          N_1 ... N_m return to N_0 the asked buckets and their lower bound;
7          N_0 calculates θ = f(min(b_1), min(b_2), ..., min(b_m)); where min(b_i) is the
             lower bound of the bucket received from N_i.
8          N_0 sets Y = {all encrypted data items contained in the received buckets};
9          N_0 sends Y to N_1 to N_m in parallel;
10         N_1 ... N_m : foreach d ∈ Y do
11             do random access in the list of buckets, find the bucket that contains d,
                 and return to N_0 the lower bound of the bucket;
12         N_0 : foreach d ∈ Y do
13             calculate ov_min(d) = f(min(b_1(d)) + min(b_2(d)) + ... + min(b_m(d)),
                 where where min(b_i(d)) is the lower bound of the bucket that
                 contains d in the list L_i.
14         N_0 :if (there are at least k data items in Y with ov_min ≥ θ) then
15             stop=true;
16         else
17             j = j + 1;
```

1. The coordinator asks each node to return the j^{th} bucket of its list.

2. The nodes return the asked buckets, including the encrypted local score and encrypted id of the data items contained in the buckets. They also return the lower bound of the asked buckets.

3. The coordinator calculates a threshold θ by applying the scoring function f on the lower bound of the returned buckets: $\theta = f(min(b'_1), min(b'_2), ..., min(b'_m))$ where $b'_1, ..., b'_m$ are the buckets returned from the nodes keeping the lists $L_1, ..., L_m$ respectively.

4. Let Y be the set of data items contained in the buckets received from the nodes. The coordinator sends Y to all the nodes. For each data item $d \in Y$, the nodes perform random access in their lists to find the bucket b_i containing d (by using the encrypted id of d). Each node returns to the coordinator a set containing the lower and upper bounds of the buckets containing the data items contained in Y.

5. For each data item d in Y, the coordinator calculates a minimum overall score $ov_{min}(d) = f(min(b_1), min(b_2), ..., min(b_m))$, where $min(b_i)$ is the lower bound of the bucket where d is found in the list L_i.

6. If among the encrypted data items received so far there are at least k data items with minimum overall scores equal or greater than θ, then the algorithm ends. Otherwise, the coordinator increments j, *i.e.*, sets $j = j + 1$, and goes to Step 1.

When the above algorithm ends, the set of data items contained in Y is called the set of *candidate data items*. The coordinator sends the candidate data items to the trusted client, where they are decrypted, their overall score is calculated, false positives are eliminated, and the top-k data items are returned to the user.

The following theorem shows that the output of SDB-TOPK contains the encrypted top-k data items.

Theorem 6. *Given a top-k query with a monotonic scoring function f, the output of SDB-TOPK contains the encrypted top-k results.*

Proof. Let Y be output of the SDB-TOPK algorithm, i.e. the set that contains all the encrypted data items seen under sorted access when the algorithm ends. We show that each data item d that is not in Y ($d \notin Y$), has an overall score that is less than or equal to the overall score of at least k data items in Y. For each list L_i, let s_i be the local score of d in the list L_i. Let b'_i be the last bucket seen under sorted access in the list L_i by its owner. Since d is not in Y, it has not been sent to the coordinator by the nodes. Thus, it has not been seen under sorted access in the lists. Thus, its involving buckets are after the last buckets seen under sorted access by the nodes in the lists. Therefore, we have $s_i < min(b'_i)$ for $1 \leq i \leq m$, *i.e.*, the local score of d in each list L_i is less than the lower bound of the last bucket read under sorted access in L_i. Thus since the scoring function

is monotonic, we have $f(s_1, ..., s_m) < f(min(b'_1), min(b'_2), ..., min(b'_m)) = \theta$. Thus, the overall score of d is less than the threshold used in the BuckTop algorithm. When the algorithm stops, there are at least k data items in Y whose minimum overall scores are greater than or equal to the threshold. Thus, their overall scores are at least the threshold. Therefore, their overall score is greater than or equal to that of the data item d. □

4.4.3 False Positive Filtering

The set of candidate data items returned by SDB-TOPK may involve false positives which should be decrypted and eliminated in the trusted client. Below, we propose a filtering algorithm to eliminate most of the false positives in the nodes of the cloud. This algorithm improves significantly the response time of the queries because the eliminated false positives do not need to be communicated to the trusted client and be decrypted there.

Given the set of candidate data items Y, the filtering algorithm done by the coordinator proceeds as follows:

- Calculate the *minimum overall score* of all candidate data items contained in Y, sort them according to their minimum overall score, and take the k^{th} *minimum overall score* denoted by δ_2.

- Calculate the *maximum overall score* of all candidate data items contained in Y, and eliminate those with *maximum overall score* less than $< \delta_2$. The *maximum overall score* of a data item d is computed as follows: $ov_{max}(d) = f(max(b_1) + max(b_2) + ... + max(b_m))$, where $max(b_i)$ is the upper bound of the bucket b_i that contains d in the list L_i.

At the end of the filtering algorithm, the candidate data items that remain in Y are sent to the trusted client to be decrypted, and the top-k items be extracted.

The following theorem shows that the filtering algorithm works correctly, i.e., the removed data are only false positives.

Theorem 7. *Given a top-k query with a monotonic scoring function f, then any data item removed by the filtering algorithm cannot belong to the top-k results.*

Proof. The proof can be done by considering the fact that any removed data item d has a maximum overall score that is lower than the minimum overall score of at least k data items. Thus the overall score of d is less than or equal to that of at least k data items that are not removed. Therefore, we can eliminate d safely. □

Example To illustrate the top-k query processing and filtering algorithms in SDB-TOPK, we use the database shown in Table 4.4, with a top-4 query and the scoring function as SUM. We suppose that the nodes n_1, n_2, and n_3 keep the lists L_1, L_2 and L_3 respectively. Assume that the coordinator is the node n_0. In the first step, n_0 asks the other nodes to return the encrypted data items which are in the first bucket of their lists, as well as the lower bound (min) of the bucket. The node n_1 return $(\{E(d1), E(d3), E(d6)\}, 24.6)$. The

List 1			List 2			List 3		
bucket ID	enc data item	enc local score	bucket ID	enc data item	enc local score	bucket ID	enc data item	enc local score
B_{11}	$E(d1)$	$E(27)$	B_{21}	$E(d6)$	$E(28)$	B_{31}	$E(d2)$	$E(22)$
B_{11}	$E(d3)$	$E(30)$	B_{21}	$E(d3)$	$E(29)$	B_{31}	$E(d3)$	$E(25)$
B_{11}	$E(d6)$	$E(26)$	B_{21}	$E(d2)$	$E(26)$	B_{31}	$E(d6)$	$E(27)$
B_{12}	$E(d2)$	$E(15)$	B_{22}	$E(d1)$	$E(24)$	B_{32}	$E(d5)$	$E(21)$
B_{12}	$E(d8)$	$E(20)$	B_{22}	$E(d7)$	$E(21)$	B_{32}	$E(d1)$	$E(20)$
B_{12}	$E(d5)$	$E(24)$	B_{22}	$E(d4)$	$E(19)$	B_{32}	$E(d9)$	$E(18)$
B_{13}	$E(d4)$	$E(14)$	B_{23}	$E(d5)$	$E(16)$	B_{33}	$E(d8)$	$E(17)$
B_{13}	$E(d9)$	$E(11)$	B_{23}	$E(d9)$	$E(13)$	B_{33}	$E(d7)$	$E(14)$
B_{13}	$E(d7)$	$E(12)$	B_{23}	$E(d8)$	$E(10)$	B_{33}	$E(d4)$	$E(11)$
...	...	...	...	...	...	...	...	...

Table 4.2 – Encrypted database, with 3 data items in each bucket. The encrypted scores inside buckets are not sorted. The boundaries (minimum and maximum) of buckets are shown below

List 1			List 2			List 3		
bucket ID	min	max	bucket ID	min	max	bucket ID	min	max
B_{11}	24.6	32	B_{21}	25.5	31	B_{31}	21.9	28
B_{12}	14.8	24.1	B_{22}	18	24.1	B_{32}	17.7	21.5
B_{13}	10.7	14.2	B_{23}	9	16.5	B_{33}	10	17.3

Table 4.3 – Bucket boundaries

Table 4.4 – Example of an encrypted database, and the information about its buckets.

node n_2 returns $(\{E(d6), E(d3), E(d2)\}, 25.5)$, and n_3 returns $(\{E(d2), E(d3), E(d6)\}, 21.9)$. After receiving the buckets from n_1, n_2 and n_3, the coordinator calculates the threshold $\theta = 24.6 + 25.5 + 21.9 = 72$. Then, it asks the nodes to return the lower bound of the buckets that contain the returned data items. Then, the coordinator calculates for each data item, its minimum overall score using its $min(b_i)$ in each list. We have $ov_{min}(E(d1)) = 24.6 + 18 + 17.7 = 60.3$, $ov_{min}(E(d2)) = 62.2$, $ov_{min}(E(d3)) = 72$, and $ov_{min}(E(d6)) = 72$. The coordinator finds that only for two data items, the minimum overall score ov_{min} is greater than or equal to θ. Thus it sets $j = 2$ and asks each node to return data items that are in the second bucket in addition to the lower bound of the bucket. After receiving the asked information, the coordinator calculates the new threshold $\theta = 14.8 + 18 + 17.7 = 50.5$ and asks the nodes n_1, n_2 and n_3 for the lower and upper bounds of the buckets that contain the returned data items. After that, it calculates ov_{min} of the returned data items, *i.e.*, $E(d4), E(d5), E(d7), E(d8), E(d9)$. Thus, it calculates $ov_{min}(E(d4)) = 38.7$, $ov_{min}(E(d5)) = 41.5$, $ov_{min}(E(d7)) = 38.7$, $ov_{min}(E(d8)) = 33.8$ and $ov_{min}(E(d9)) = 37.4$. Now, the coordinator finds that four (i.e., k) data items have minimum overall scores greater than or equal to θ. Thus, it stops retrieving the buckets from the nodes. The set of data items received by the coordinator until now is called the set of candidate items.

Then, the coordinator executes the filtering algorithm on the candidate items as follows. It sets $\delta_2 = 60.3$ (the 4^{th} minimum overall score). For each candidate data item, the coordinator calculates its maximum overall score: $ov_{max}(E(d1)) = 77.6$, $ov_{max}(E(d2)) = 83.1$, $ov_{max}(E(d3)) = 91$, $ov_{max}(E(d4)) = 55.6$, $ov_{max}(E(d5)) = 62.1$, $ov_{max}(E(d6)) = 91$, $ov_{max}(E(d7)) = 55.6$, $ov_{max}(E(d8)) = 57.9$, $ov_{max}(E(d9)) = 52.2$. The coordinator finds that $E(d4), E(d7)$, $E(d8)$ and $E(d9)$ have a maximum overall score less than δ_2, so it eliminates them from the candidate set and returns the remained data items to the trusted client with their encrypted scores. The trusted client decrypts each returned data item and calculates its overall score: $ov(d1) = 71, ov(d2) = 63, ov(d3) = 84, ov(d5) = 61, ov(d6) = 81$. It finds that the top-4 data items are $d1, d2, d3$ and $d6$, and returns them to the user.

4.5 SD-TOPK System

In this section, we present the SD-TOPK system, designed for efficient processing top-k queries over encrypted and distributed data across the nodes of the cloud data center. It is coordinated in the nodes of the cloud. SD-TOPK needs much less communication between the cloud and the trusted client than SDB-TOPK. It is also much more efficient in terms of response time.

In SD-TOPK, the data encryption and outsourcing is done in the same way as that of SDB-TOPK, *i.e.*, by creating buckets, and using deterministic and probabilistic methods for encryption (see Section 4.4.1).

The rest of this section is organized as follows. We first introduce the top-k query processing algorithm of SD-TOPK. Then, we present an algorithm for removing the false

positives from the results of the top-k query processing algorithm, without decrypting the data.

4.5.1 Top-k Query Processing

Algorithm 2: SD-TOPK Query Processing Algorithm

Input: Encrypted database partitioned vertically over m nodes:
$E(D) = \{L_1, L_2, ..., L_m\}$
Output: Set containing top-K data items

begin
- N_0 asks $N_1 ... N_m$ to return buckets containing the k first data items and their lower bounds
- N_0: **foreach** *returned data item d* **do**
  - calculates $ov_{min}(d) = f(v_1(d) + v_2(d) + ... + v_m(d))$
- N_0 sets δ = the k^{th} ov_{min}
- N_0 calculates $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$
- N_0 sends θ to $N_1 ... N_m$
- $N_1 ... N_m$ return to N_0 all the data items which are in buckets with $max(b_j) \geq \theta$
- N_0 sets Y = {candidate data items}
- N_0 asks $N_1 ... N_m$ to return the encrypted local scores of each data item $d \, in Y$
- N_0 returns to the client Y with the returned encrypted scores

In our algorithm, we assume that the scoring function is in the class of linear functions with positive coefficients (denoted as LFPC), i.e. $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$ where each coefficient $a_i \geq 0$ for $1 \leq i \leq m$. Many functions such as SUM, COUNT, AVG and MAX are in the class of LFPC functions. Note that every LFPC function is monotonic, but there exist functions that are monotonic but not LFPC.

In SD-TOPK, for each query, one of the nodes of the cloud performs the coordination between the list owner nodes. We call this node as *coordinator*. As in SDB-TOPK, the coordinator may be the node that initially receives the user's query or be randomly chosen among the list owner nodes.

Let us describe the SD-TOPK algorithm. Given a top-k query with a number k and a scoring function f, the cloud chooses a node as query coordinator, and then the following steps are performed to answer the query (see Algorithm 2:

1. The coordinator broadcasts to the list owners the number k, and asks each list owner to return the buckets that contain the k first data items in the list. With each bucket, the list owner returns the encrypted identifier of the data items involved in the bucket, as well as the lower bound of the buckets. Formally, let B be the set of buckets containing the first k data items of the list. For each bucket $b_i \in B$, the owner returns $min(b_i)$ and $E(d)$ if $d \in b_i$.

2. For each returned data item d, the coordinator calculates its *minimum overall score* defined as follows: $ov_{min}(d) = f(v_1(d) + v_2(d) + ... + v_m(d))$ where v_i is the minimum value of the bucket that contains d in the list L_i if d is returned to the coordinator by the owner of L_i, otherwise $v_i = 0$. In other words, for calculating the minimum overall score of a data d, we apply the scoring function on the minimum value of the buckets that contain d in the lists. If the data d has not been sent by the owner of a list, then we use zero in the function instead of the minimum value of the bucket.

3. The coordinator sorts the received data items according to their minimum overall score, and chooses the data item d' that has the k^{th} minimum overall score denoted by δ. Then, it uses the minimum overall score of d' to calculate a threshold θ as follows: $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$ where $a_1, \ldots, a_m$ are the coefficients in the scoring function. After computing the threshold θ, the coordinator sends it to all list owners.

4. Each list owner returns to the coordinator the list of the data items that are in the buckets with maximum values greater than or equal to θ.

5. Let Y be the set of all data items that are sent to the coordinator by at least one list owner. We call Y the set of *candidate items*. The coordinator sends the encrypted id of all data items involved in Y to the list owners, and they return the encrypted score of each data item contained in Y.

6. Finally, the coordinator returns to the trusted client the candidate items and their encrypted local scores.

When the trusted client receives the candidate items, it decrypts them using the secret keys. Then, it calculates for each candidate d its overall score, extracts the k data items that have the highest overall scores, and returns them to the user.

The following theorem shows that the output of SD-TOPK contains the encrypted top-k data items.

Theorem 8. *Given a top-k query with a scoring function f that is linear with positive coefficients. Then, the output of SD-TOPK contains the encrypted top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$. Let Y be the output of the algorithm, *i.e.*, the set of candidate items. To prove the theorem, it is sufficient to show that each data item d that has not been sent to the coordinator in the 4th step of the algorithm, has an overall score that is less than or equal to the overall score of at least k data items in Y. Let θ be the threshold value that is to the list owners in the third step of the algorithm. For each list L_i, let s_i be the local score of d in the list L_i. The overall score of d is computed as $ov(d) = a_1s_1 + \cdots + a_ms_m$. Since d has not been sent to the coordinator, from the 4th step of the algorithm, we know that $s_i < \theta$. Thus, we have $ov(d) < a_1 \times \theta + \cdots + a_m \times \theta = \sum_{i=1}^{m} a_i \times \theta$. From the 3rd step of the algorithm, we

know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have $ov(d) < \delta$. In other words, the overall score of d is less than the minimum overall score of the data item d' that is the k^{th} data item found in the 3rd step of the algorithm. Therefor, the overall score of d is less than at least k data items found by the SD-TOPK algorithm, so d cannot be among the top-k results. □

4.5.2 False Positive Filtering

The set of candidate data items returned by SD-TOPK may involve many false positives which should be decrypted and eliminated by the trusted client. Below, we propose a filtering algorithm to eliminate most of the false positives in the cloud. It improves significantly the response time of the queries because the eliminated false positives do not need to be communicated to the trusted client and be decrypted there.

Given the set of candidate data items Y, the filtering algorithm done by the coordinator proceeds as follows:

- Calculate the *minimum overall score* of all candidate data items, sort them according to their minimum overall score, and take the k^{th} *minimum overall score* denoted by δ_2.
- Calculate the *maximum overall score* of all candidate data items, and eliminate those with *maximum overall score* less than $< \delta_2$. The *maximum overall score* of a data item d is computed as follows: $ov_{max}(d) = f(v_1(d) + v_2(d) + ... + v_m(d))$ where v_i is the maximum value of the bucket that contains d in the list L_i if d is returned to the coordinator by the owner of L_i, otherwise v_i is equal to the minimum value of the last bucket received from the owner of L_i.

At the end of the filtering algorithm, the candidate data items that remain in Y are sent to the trusted client to be decrypted, and the top-k items be extracted.

The following theorem shows that the filtering algorithm works correctly, i.e., the removed data are only false positives.

Theorem 9. *Given a top-k query with a scoring function f that is linear with positive coefficients, then any data item removed by the filtering algorithm cannot belong to the top-k results.*

Proof. The proof can be done by considering the fact that any removed data item d has a maximum overall score that is lower than the minimum overall score of at least k data items. Thus, according to Lemmas 1 and 2 the overall score of d is less than or equal to that of at least k data items that are not removed. Therefore, we can eliminate d. □

Example To illustrate SD-TOPK, we use the database shown in Table 4.4 with a top-4 query and SUM as scoring function. We suppose that a node n_0 is the coordinator. It sends a messages to all list owners (e.g., n_1, n_2, n_3) and asks for the 4 first data items in each list (since $k = 4$), and the minimum of the buckets in which they are. The node n_1 returns $\langle d1, 24.6\rangle$ $\langle d3, 24.6\rangle$ $\langle d6, 24.6\rangle\langle d2, 14.8\rangle$. The node n_2 returns $\langle d6, 25.5\rangle$ $\langle d3, 25.5\rangle$

$\langle d2, 25.5\rangle$ $\langle d1, 18\rangle$ and the node n_3 returns $\langle d2, 21.9\ \rangle$ $\langle d3, 21.9\rangle$ $\langle d6, 21.9\rangle$ $\langle d5, 17.7\rangle$. The coordinator calculates the minimum overall score of the returned data items by using the minimum of their buckets. It finds that $ov_{min}(d1) = 24.6 + 18 = 42.6$, $ov_{min}(d2) = 14.8 + 25.5 + 21.9 = 62.2$, $ov_{min}(d3) = 24.6 + 25.5 + 21.9 = 72$, $ov_{min}(d5) = 17.7$ and $ov_{min}(d6) = 72$. After sorting the minimum overall scores, the coordinator finds that the 4th minimum overall score is 42.6 (that of $d3$), so it sets $\delta = 42.6$. Then, it calculates $\theta = \delta/3 = 14.2$. Afterwards, it asks each node to return the encrypted id (bucket boundaries) of the data items that are in buckets b_i such that $max(b_i) \geq \theta$. The data items received from list owners are called candidate data items. The coordinator calculates for the candidate items their minimum overall score: $ov_{min}(d1) = 60.3$, $ov_{min}(d2) = 62.2$, $ov_{min}(d3) = 72$, $ov_{min}(d4) = 38.7$, $ov_{min}(d5) = 41.5$, $ov_{min}(d6) = 72$, $ov_{min}(d7) = 38.7$, $ov_{min}(d8) = 33.8$ and $ov_{min}(d9) = 37.4$. It also calculates $\delta_2 = 60.3$, that is the kth minimum overall score among candidate data items (defined in the filtering algorithm). Then, it calculates the maximum overall scores of the candidate data items: $ov_{max}(d1) = 77.6$, $ov_{max}(d2) = 83.1$, $ov_{max}(d3) = 91$, $ov_{max}(d4) = 55.6$, $ov_{max}(d5) = 62.1$, $ov_{max}(d6) = 91$, $ov_{max}(d7) = 55.6$, $ov_{max}(d8) = 57.9$ and $ov_{max}(d9) = 52.2$. According to the filtering algorithm, the coordinator eliminates the data items that have a maximum overall score less than 60.3. Then, it remains five data items in the set of candidate items $Y = \{d1, d2, d3, d5, d6\}$. The coordinator asks the list owners to return all encrypted scores of the candidate items and sends them to the trusted client. When the client receives the data items, it decrypts them and calculates their real overall score: $ov(d1) = 71, ov(d2) = 63, ov(d3) = 84, ov(d5) = 61, ov(d6) = 81$. Finally, the trusted client finds that the top-4 data items are $d1, d2, d3$ and $d6$. It returns them to the user who had issued the query.

4.6 Security Analysis and Improvement

In this section, first, we propose a technique that allows to perturb the bucket boundaries in order to improve and reinforce the security of our proposed systems SDB-TOPK and SD-TOPK. Note that this technique is also applicable on BuckTop system proposed in chapter 3. Then, we analyze the different types of information that can be leaked to the adversary (the nodes of the cloud data center). Finally, we propose some techniques to reduce the risk of disclosing sensitive data.

4.6.1 Obfuscating Bucket Boundaries

To strengthen the security of our systems SDB-TOPK and SD-TOPK, we propose a method to obfuscate the bucket boundaries, thus the adversary cannot learn the limits of the data items in the buckets. For this, we change the bucket limits as follows. We choose two random numbers a and c. These numbers must be kept secret in the trusted client. Before sending the database to the cloud, the lower and upper bounds of each

bucket b_i are obfuscated (modified) as follows:

$$min(b_i) := min(b_i) \times a + c \tag{4.1}$$

$$max(b_i) := max(b_i) \times a + c \tag{4.2}$$

Thus, the trusted client multiplies the lower (upper) bounds by the secret number a, and then adds the secret number c to the result. These obfuscated bucket limits are sent to the cloud data center nodes together with the encrypted IDs and scores.

By the above strategy, we hide the limits of the buckets from the cloud nodes. But, a question remains to answer: do SDB-TOPK and SD-TOPK work correctly if it uses the changed lower/upper bounds? The answer to this question is positive. The intuition is that the stop condition of SDB-TOPK and SD-TOPK remains valid, if we multiply and add all bucket limits to the same positive numbers.

Now, we present the theorems proving that SDB-TOPK and SD-TOPK works correctly if they use the obfuscated lower bounds.

Theorem 10. *Given a top-k query with a monotonic scoring function f. If we change the lower bound of the buckets by using Equation 4.1, then the output of SDB-TOPK will contain the top-k results.*

Proof. Let Y be the output of the SDB-TOPK top-k query processing algorithm. We show that each data item d that is not in Y ($d \notin Y$), has an overall score that is less than or equal to the overall score of at least k data items contained in Y. Let $Y' \subseteq Y$ be the k data items whose minimum overall score is higher than θ when the algorithm ends. Let $d' \in Y'$ be the data item that has the smallest overall score among the data items contained in Y'. In each list L_i, let b'_i be the bucket that contains d' in L_i, and thus $min(b'_i) * a + c$ is the new (modified) lower bound of b'_i. Let $b_1, ..., b_m$ be the last buckets seen by the algorithm before it ends. Then, from the stop condition of SDB-TOPK, we have: $\theta = f(min(b_1)*a+c, ..., min(b_m)*a+c) \leq f(min(b'_1)*a+c, ..., min(b'_m)*a+c)$

Since f is monotonic and the numbers a and c are positive, we have:

$$f(min(b_1), ..., min(b_m)) \leq f(min(b'_1), ..., min(b'_m)) \tag{4.3}$$

Before changing the lower bounds, the local score of d' in each list is higher than or equal to the lower bound of its bucket. Thus, we have:

$$f(min(b'_1), ..., min(b'_m)) \leq f(s'_1, ..., s'_m) \tag{4.4}$$

By comparing Equations 4.3 and 4.4, we have:
$f(min(b_1), ..., min(b_m)) \leq f(s'_1, ..., s'_m) = ovl(d')$

In the right hand side of the above equation, we have the overall score of d'. Now, we show that the left hand side of the above equation is higher than or equal to the overall score of a data d that has not been seen by the algorithm. Let s_i be the (plaintext) local score of d in the list L_i. Since d has not been seen by the algorithm, its bucket in L_i is after

b_i that is the last bucket seen by the algorithm. Thus, we have $s_i \leq min(b_i)$ for $1 \leq i \leq m$. Therefore, since f is monotonic, we have: $f(s'_1, ..., s'_m) \leq f(min(b_1), ..., min(b_m))$

In other words, $ov(d) \leq ov(d')$. Thus, the overall score of any unseen data item d is less than or equal to that of at least k data items contained in Y. Therefore, Y contains the top-k results, and the proof is done. □

The following theorem shows that the filtering algorithm of SDB-TOPK works correctly, if we change the bucket boundaries using Equations 4.1 and 4.2.

Theorem 11. *Assume a top-k query with a monotonic scoring function f. If we modify the lower bound of the buckets by using Equations 4.1 and 4.2, then the filtering algorithm of SDB-TOPK does not remove any top-k result.*

Proof. Let Y be the output of SDB-TOPK algorithm, and $Y' \subseteq Y$ be the k data items in Y that have the highest *min_ovl* scores. Let d' be the data item in Y'. In each list L_i, let b'_i and s'_i be the bucket and local score of d'_i in the list.

We do the proof by contradiction. We choose a data item d that is a top-k result and has been removed by the filtering algorithm. We show that this assumption yields to a contradiction. In each list L_i, let b_i and s_i be the bucket and local score of d_i in the list. Since, d has been removed from the list, its maximum overall score using the modified boundaries is lower than or equal to that of d'. Thus, we have:

$f(max(b_1) \times a + b, ..., max(b_m) \times a + b) \leq f(min(b'_1) \times a + c, ..., min(b'_1) \times a + c)$

Since the coefficients a and c are positive, the monotonicity of f implies that:

$f(max(b_1), ..., max(b_m)) \leq f(min(b'_1), ..., min(b'_1))$.

Therefore, we have: $max\_ovl(d) \leq min\_ovl(d')$. Then by using Lemmas2 and 3, we have: $ovl(d) \leq ovl(d')$. In other words, the overall score of d is less than that of any data item contained in Y'. Thus, d is not a top-k result and can be removed. □

Let us now prove that our second system, *i.e.*, SD-TOPK, works correctly if we obfuscate bucket boundaries.

Theorem 12. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Equation 4.1, then the output of SD-TOPK will involve the top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \cdots + a_mx_m$. Let Y be the output of SD-TOPK algorithm, *i.e.*, the set of candidate items. We show that each data item d that has not been sent to the coordinator by the list owners, has an overall score that is less than or equal to the overall score of at least k data items involved in Y. Let b_i be the bucket that contains the data d in the list L_i, and thus $max(b_i) * a + c$ is the new (modified) upper bound of b_i. From the 4^{th} step of SD-TOPK, we know that in each list L_i we have $max(b_i) * a + c < \theta$. Thus, we have $a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c) < \sum_{i=1}^{m} a_i \times \theta$. We know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have the following equation:

$$a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c) < \delta \tag{4.5}$$

Now, let d' be the data item that has the k^{th} minimum overall score in the 3rd step of SD-TOPK. In each list L_i, let b'_i be the bucket that contains d' in L_i, and thus $min(b'_i)*a+c$ is the new (modified) lower bound of b'_i. From the 3rd step of the algorithm, we know that the minimum overall score of d' (computed by using the obfuscated buckets) is equal to δ. Thus, we have $a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) = \delta$. Thus, we have the following equation:

$$a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) = \delta \tag{4.6}$$

By comparing Equations 4.5 and 4.6, we have: $a_1 \times (min(b'_1) * a + c) + \cdots + a_m \times (min(b'_m) * a + c) > a_1 \times (max(b_1) * a + c) + \cdots + a_m \times (max(b_m) * a + c)$. Since the numbers a and c are positive, we can write: $a_1 \times (min(b'_1) + \cdots + a_m \times min(b'_m) > a_1 \times max(b_1) + \cdots + a_m \times max(b_m)$. This means that the minimum overall score of the data item d' is higher than the maximum overall score of d. In other words, the data item d could not be among the top-k results.

□

The following theorem shows that the filtering algorithm works correctly for SD-TOPK, if it uses the obfuscated bucket limits.

Theorem 13. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Equation 4.1, then the filtering algorithm of SD-TOPK does not remove any top-k result.*

Proof. Let Y be the output of SD-TOPK algorithm, and d' be the data item that has the k^{th} minimum overall score among the data items in the 3rd step of SD-TOPK. In each list L_i, let b'_i and s'_i be the bucket and local score of d'_i in the list.

We do the proof by contradiction. We assume a top-k data item d has been removed by the filtering algorithm, and show that this assumption yields to a contradiction. Let b_i and s_i be the bucket and local score of d_i in the list L_i. Since, d has been removed from the list, its maximum overall score using the modified limits is lower than or equal to minimum overall score of d'. Thus, we have: $a_1 \times (max(b_1) \times a + c), ..., a_m \times (max(b_m) \times a + c) \leq a_1 \times (min(b'_1) \times a + c), ..., a_m \times (min(b'_1) \times a + c)$. Since the parameters a and c are positive, we have: $a_1 \times max(b_1), ..., a_m \times max(b_m) \leq a_1 \times min(b'_1), ..., a_m \times min(b'_1)$. This means that the maximum overall score of the data item d is lower than the minimum overall score of d'. Thus, d cannot be a top-k result.

□

4.6.2 Security Analysis

Partial Order Leakage: In SDB-TOPK and SD-TOPK, we use the bucketization technique for managing the data in the cloud nodes. The bucket boundaries are obfuscated, so they disclose no information to the adversary. Inside the buckets, no information is leaked because the data items are not ordered and the local scores are encrypted using a probabilistic scheme.

But a partial order is leaked about the data items that are in different buckets (since the buckets are ordered). We point out that the security of SBD-TOPK and SD-TOPK is higher than the TA-based approaches, because with the latter approaches the total order of the encrypted data is leaked to the cloud nodes.

Even a partial order leakage may help the adversary to obtain rough information about the sensitive data of individuals if she has some background information about the data. For example, if the adversary A knows that the age of a target person u is very high, then A may find the bucket containing u in the list corresponding to age (*i.e.*, the first or last bucket of the list). Then, by guessing the ID of u in the bucket (e.g., if the size of the bucket is too small), A may find the bucket of u in the salary's list, and then estimate her salary with some confidence probability. *We show that this probability (i.e., the risk of privacy violation) is very low, when the size of buckets is not small.*

Let u be an individual (data item) in the database, and assume that the adversary A knows the value of u in some attribute a. We want to compute the confidence probability that A finds the bucket containing u's value in a sensitive attribute s. Let us denote this confidence probability by $P(b_{s,u}|a)$. We assume that if A finds the bucket of u in the list representing s, then she can make a good estimation of u's value, *e.g.*, using some background knowledge about the values of attribute s.

To find the bucket of u in the sensitive attribute s, the adversary A needs to perform the following steps: 1) guessing the lists that represent a and s; 2) finding the bucket of u in the list representing a; 3) guessing the ID of u in the found bucket; 4) searching u's ID in the list representing s, and finding its bucket.

Let $P(L_1 = a \wedge L_2 = x)$ be the probability that A guesses correctly the lists representing the attributes a and s. Let m be the number of lists in the database. In our systems, the metadata of sorted lists (*e.g.*, their identification) is encrypted, and they have the same size and format. Thus, the probability of finding the correct list of an attribute is $\frac{1}{m}$. Therefore, the probability of correctly guessing the lists representing both attributes a and s is:

$$P(L_1 = a \wedge L_2 = x) = \frac{1}{m \times (m-1)} \tag{4.7}$$

If the adversary A finds correctly the list representing a, then we assume that A is able to find the bucket containing u by using the background knowledge about the value of u in a (and some statistical information). After finding the bucket, say b, the adversary needs to guess the ID of u. Let $size(b)$ be the number of encrypted values in the bucket b. Then, the probability of finding u' ID in the bucket b, denoted as $P(ID = u)$, is:

$$P(ID = u) = \frac{1}{|size(b)|} \tag{4.8}$$

If A guesses correctly the ID of u in the bucket b, then she can find the bucket containing the ID in the list representing the attribute s (if she guesses correctly the list of s), and then she can roughly estimate the u's value in s.

The following theorem provides a formula that calculates the probability that an adversary finds the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a.

Theorem 14. *Let s be a sensitive attribute, $size(b)$ be the size of the buckets, and m be the number of lists in the cloud. Let $P(b_{s,u}|a)$ be the probability that the adversary detects correctly the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a. Then, $P(b_{s,u}|a)$ is:*

$$P(b_{s,u}|a) \leq \frac{1}{size(b) \times m \times (m-1)} \tag{4.9}$$

Proof. The proof can be done using Equations 4.7 and 4.8. □

The above theorem shows that when the size of the buckets is not small, the probability of privacy violation is very low.

When the bucket size is one (which is equivalent of preserving the total order), the risk of privacy violation is the highest. We advise at least size 10 for the buckets.

Note that choosing very big buckets increases the response time of query processing (see the effect of bucket size on performance in Section 4.8.2). Therefore, the size of the buckets should be taken based on the user privacy requirements (*e.g.*, the maximum acceptable probability of privacy violation) and performance (*e.g.*, response time).

4.6.3 Security Improvements

In this section we propose some improvements to minimize as possible the data leakage to the adversary.

Perturbing the Number of Asked Results: In the basic version of of SDB-TOPK and SD-TOPK, the information about the number of asked results, *i.e.*, k, is disclosed to the nodes. We can perturb this information as follows. The trusted client generates a random integer s between 0 and a predefined (small) value, and adds it to k. Then, it sends $k' = k + s$ to the cloud as the number of required results. After receiving the encrypted results from the cloud, the trusted client filters the result set and sends only k results to the user. This strategy improves the security of our systems, however, we must choose the value of s such that the response time of SDB-TOPK and SD-TOPK will not be affected (see the effect of increasing k on performance in Section 4.8.2).

Perturbing the Database Size: Another information which, is leaked is the database size (*i.e.*, the number of tuples). This leakage can be avoided by adding dummy tuples to the database before sending it to the cloud nodes. But, we have to be careful not to add dummy tuples which could be returned to the user as a result of top-k queries. For this, we can proceed as follows. Let n' be the number of dummy data items that we want to add to the lists. In each list L_i, let s_i be the last local score in the list. We generate n' random data IDs. Then, for each list L_i, we generate n' random scores smaller than s_i, and assign them randomly to the n' data IDs. Afterwards, we add the generated data IDs and their local scores to the sorted lists. Since the local scores of the dummy data items are smaller than any real data item, they have no chance to be returned as a result of top-k queries.

4.7 Update Management

To update a data item in SDB-TOPK and SD-TOPK systems is done by deleting the old data scores (attribute values) and then inserting its new scores in the nodes of the cloud data center. Let d be the data to be updated and the new scores in the lists L_1, ..., L_m are s_1, ..., s_m respectively.

To delete the old encrypted scores of d, it is sufficient to encrypt the ID of d using the key that has been used for encrypting the data IDs, and then asking the nodes to find the encrypted ID in the lists and then remove the pairs $(E(d), E(s_i(d)))$ from the lists.

Inserting the new scores of d is done as follows. The trusted client uses the metadata of the buckets (*i.e.*, the lower and upper bounds), and for each list L_i, it calculates the bucket of the list to which the data score s_i should by stored. Let b_i be the corresponding bucket of s_i. The trusted client encrypts the ID and scores of d by using the encryption schemes that are used for encrypting the ID and scores. Finally, for each list L_i, it creates the pairs $(E(d), E(s_i(d)))$, and asks the owner of the list L_i to put the pair in the bucket b_i.

4.8 Experimental Evaluation

In this section, we evaluate the performance of SDB-TOPK and SD-TOPK using synthetic and real datasets. We study the effect of different parameters such as the number of data items in each lists, the number of the database lists, the number of data items requested, etc.

In the rest of this section, we first describe the experimental setup, and then report the results of our experiments.

4.8.1 Setup

We did our tests on real and synthetic datasets [1]. As in some previous work on privacy (*e.g.*, [46]), we use the Gowalla database, which is a location-based social networking dataset collected from users locations. The database contains 6 million tuples where each tuple represents user number, time, user geographic position, etc. In our experiments, we are interested in the attribute time, which is the second value in each tuple. As in [46], we decompose this attribute into 6 attributes (year, month, day, hour, minute, second), and then create a database with the values of those attributes. In addition to the real dataset, we have also generated random datasets using uniform and Gaussian distributions.

For the data encryption in all the approaches, we use the two following algorithms: XOR [17] to encrypt the identifiers of the database items, and Blowfish [72] to encrypt the local scores of the database items. For Remote-TA and Block-TA, we simply sort the encrypted data items in each list based on their initial order (i.e., their order in plaintext).

[1] The code and tested data are accessible at: `https://github.com/MAHBOUBIsakina/SD_TOPK_approach`

To create the distributed topology, we choose PeerSim [58]. In our tests, the number of nodes is equal to the number of lists, *i.e.,* each node will be owner of one list. In the tested system, the average latency of messages is 50 ms. The coordinator node is one the list owner nodes (randomly chosen).

In our performance evaluation, we study the effect of several parameters: 1) n: the number of data items in the database; 2) m: the number of lists; 3) k: the number of required top items; 4) $bsize$: the number of data items in the buckets (or blocks) in SDB-TOPK, SD-TOPK and Block-TA. The default value for n is 2M items. Unless otherwise specified, m is 5, k is 50, and $bsize$ is 10. In our tests, the default database is the synthetic uniform database.

To evaluate the performance of our approach, we measured the following metrics:

- **Response time:** the total time elapsed between the time when the query is sent to the coordinator and the time when the k decrypted results are returned to the user. This time includes top-k query processing time, filtering time, and the result post-processing time (*e.g.*, decryption).

- **Filtering rate:** the number of false positives eliminated by the filtering algorithm in the cloud nodes.

- **Communication cost:** We measure two metrics: 1) the number of messages communicated between the nodes to answer a top-k query; 2) the total number of bytes communicated to answer a top-k query.

4.8.2 Results

In this section, we reports all the experiment results that we obtained.

Effect of Database Size: In this experiment, we compare the response time of SDB-TOPK, SD-TOPK, Remote-TA and Block-TA, while varying the number of data items, *i.e.*, n.

Figure 4.2 shows how response time evolves, with increasing n, and the other parameters set as default values described in Section 4.8. Note that the results are shown in logarithmic scale. The response time of all approaches increases with increasing the database size. SD-TOPK is the best; its response time is at least two orders of magnitude better than the other algorithms. Also, the results show that SDB-TOPK performs better than TA-based approaches. This high difference between SDB-TOPK and SD-TOPK is mainly due to the difference in the number of communicated messages to find the top-k result. The difference between our proposed systems and TA-based algorithms is due to the high number of encrypted data items that should be decrypted by TA-based algorithms in trusted client, and also the communicated messages. Block-TA performs better than Remote-TA, because of reading the lists in blocks, thus it needs less number of messages.

Effect of the Number of Queried Attributes: Figure 4.3 shows the response time of SDB-TOPK, SD-TOPK and TA-based algorithms when varying m (i.e., the number of attributes in the scoring function), and the other parameters set as default values (except

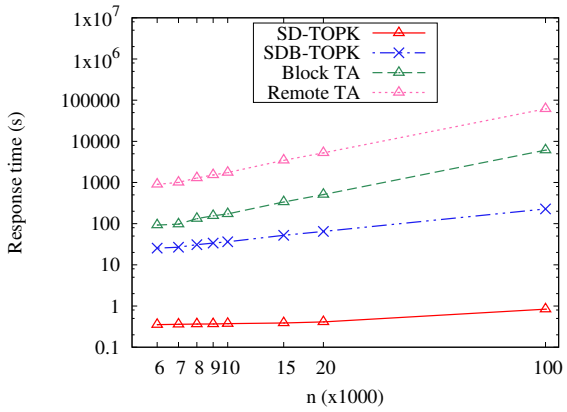

Figure 4.2 – Response time vs. number of database tuples

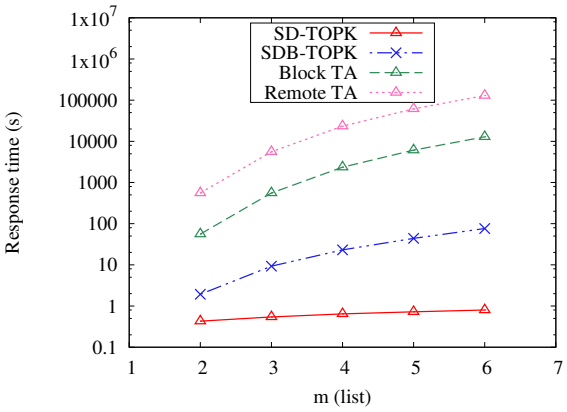

Figure 4.3 – Response time vs. number of queried attributes m

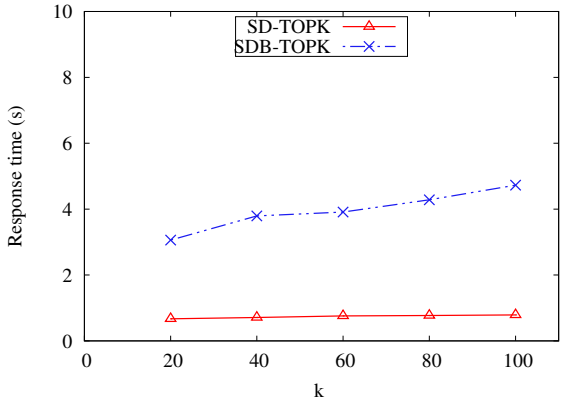

Figure 4.4 – Response time vs. k

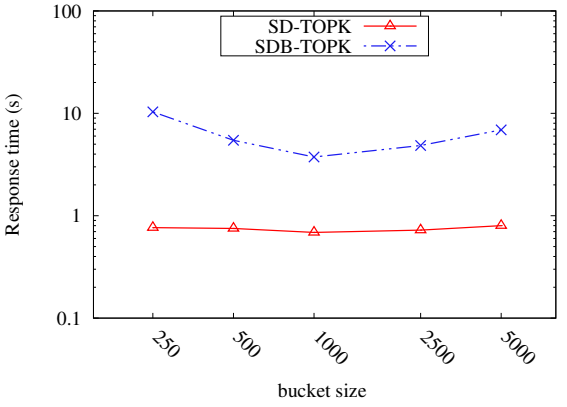

Figure 4.5 – Response time vs. bucket size

n which is fixed to 100K data items). We observe that the response time of SD-TOPK increases slightly comparing to the other approaches when the number of lists increases. The reason is that when we increase the number of lists, more data (sent by the nodes) should be processed by the coordinator for finding the candidate items.

Effect of k: Figure 4.4 shows the response times of SDB-TOPK and SD-TOPK with increasing k, and the other parameters set as default values. We observe that with increasing k the response time of SD-TOPK increases slightly compared to SDB-TOPK. The reason is that when k increases, SD-TOPK needs to get more data items from the list owners in each step. In addition, increasing k augments the number of data items that the trusted client needs to decrypt (because at least k data items are decrypted by the trusted client).

Effect of Bucket Size: Figure 4.5 reports the response time of SDB-TOPK and SD-TOPK when varying the size of buckets, and the other parameters set as default values. We observe that the response time of SD-TOPK is almost constant compared to SDB-TOPK response time. We observe that SDB-TOPK response time decreases when the bucket size is less than 1000 data item, and when the bucket size is greater than this value, the response time increases. The reason is that initially, by increasing the bucket size, SDB-TOPK needs to exchange less messages between the nodes to find the top-k data items, without a significant impact on the number of false positives. But, when the bucket size gets very big, the number of false positives sent to the trusted client increases significantly, thus it needs much more time to decrypt and eliminate them.

Performance over Different Datasets: We study the effect of the datasets on the performance of SDB-TOPK, SD-TOPK, Remote-TA and Block-TA using different datasets: synthetic datasets with uniform and Gaussian distributions, and real dataset (Gowalla). Figure 4.6 shows the response time of the approaches over different datasets by using 100K data items, while other parameters are set as default values. We see that the performance of all approaches over the Gaussian database is better than real and uniform databases. The reason is that with the latter databases, the algorithms need to go deeper into the lists to be sure that they have found the top-k results.

Communication Cost: We measure the communication cost of SDB-TOPK, SD-TOPK, Remote-TA and Block-TA in terms of the total number of messages exchanged between the different nodes and the size of the exchanged data.

Figure 4.7 shows the number of communicated messages while increasing the number of tuples and fixing the other parameters to the default values. We observe that SD-TOPK needs to exchange a small number of messages comparing to the others approaches. The reason is that SD-TOPK runs in only some rounds of communication (that does not depend on the database size). SDB-TOPK and TA-based algorithms exchange messages until finding the top-k elements, thus for them the number of messages depends on the position where they stop in the lists. Results show that SDB-TOPK and Block-TA exchanges the same number of messages because they read the data items in the lists by buckets with same sizes so they stops in the same position. We see that the number of messages exchanged by Block-TA is less than that of SDB-TOPK and Remote-TA, because the latter approaches communicates the data items in blocks not one by one as in

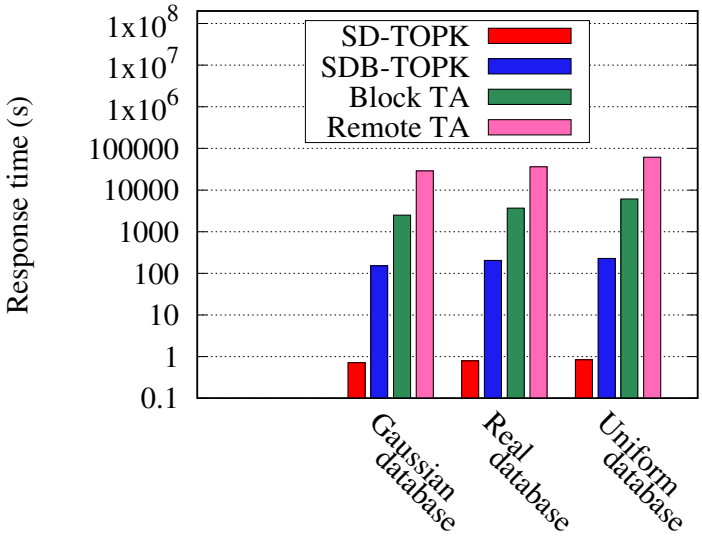

Figure 4.6 – Response time using different databases

Remote-TA.

Figure 4.8 illustrates the size of the communicated data in bytes, while increasing the number of tuples in the database and setting the other parameters to the default values. We note that the size of the communicated data increases with the database size. The amount of data transferred by SDB-TOPK and SD-TOPK is less than that of TA-based approaches. The reason is that SDB-TOPK and SD-TOPK uses the obfuscated bucket boundaries to check the top-k data items and these boundaries have a size less than the encrypted scores used by other algorithms. We observe also that SD-TOPK transfers more data than SDB-TOPK because the threshold calculated by SD-TOPK is generally lower than the one calculated by SDB-TOPK. Thus, the nodes in SD-TOPK needs to go deeper in the lists and return more data compared to SDB-TOPK.

Filtering Rate The filtering algorithm used by SDB-TOPK and SD-TOPK algorithm is used to eliminate/reduce the false positives in the cloud. We study the filtering rate of our approaches by using different datasets. The results are shown in Table 4.5. We observe that the filtering algorithm of SD-TOPK eliminates 100% of the false positives over the uniform database, thus the coordinator returns to the trusted client only the k data items that are the result of the query.

We see that in the real and Gaussian databases, around 99.99% of false positives are eliminated. For SDB-TOPK, we see that the filtering algorithm eliminates between 99.97% and 97.44% of positives over uniform and real databases, and 98.53% over gaussian database. This difference comes from the local score distributions. For example, in the Gaussian distribution, the local scores of many data items are very close to each other, thus the filtering rate decreases in this dataset. The results show that the filtering algorithm is very efficient over all the tested datasets, this is why we need to decrypt in the client side only a small number of data (*i.e.*, only the top-k data and a very small number

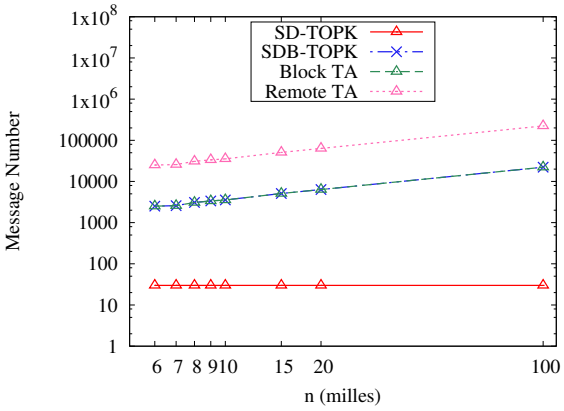

Figure 4.7 – Number of communicated messages vs. number of database tuples

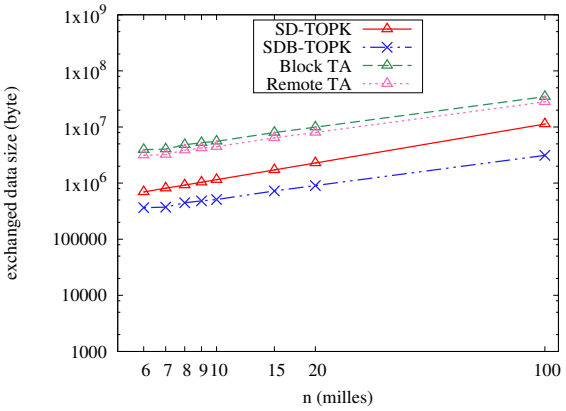

Figure 4.8 – Size of communicated data (in bytes) vs. number of database tuples

System	Uniform dataset	Real dataset	Gaussian dataset
SD-TOPK	100%	99.995%	99.991%
SDB-TOPK	99.97%	97.44%	89.53%

Table 4.5 – False positive elimination by the filtering algorithm of SDB-TOPK and SD-TOPK over different databases.

of false positives, if any).

4.9 Conclusion

In this chapter, we addressed the problem of evaluating top-k queries over encrypted and distributed data across the nodes of a cloud data center. We first presented two TA-based approaches, called Remote-TA and Block-TA, that correctly find top-k results over encrypted data. But, they need many round-trips between the client and the cloud nodes, and also need to decrypt many encrypted data in the client side. Then, we proposed tow efficient Systems, called SDB-TOPK and SD-TOPK, that are coordinated and executed in the nodes of the cloud. Our proposed systems include a top-k query processing algorithm that finds a set of encrypted data that includes the top-k data items. In addition, we proposed a filtering algorithms that filters efficiently the false positives in the nodes. We theoretically proved the correctness of SDB-TOPK and SD-TOPK query processing and filtering algorithms. We also analyzed the security of the two systems, and proposed efficient strategies for enforcing it.

We evaluated the performance of our solutions over synthetic and real databases. The results show excellent response time and communication cost for the SDB-TOPK and SD-TOPK compared to TA-based approaches. The results also show a significant gain in communication cost of SD-TOPK compared to the other algorithms. They also show the

efficiency of the filtering algorithm that eliminates almost all false positives in the cloud nodes.

Chapter 5

Conclusion

In this thesis, we addressed the problem of privacy preserving top-k query processing over encrypted data. Data encryption is the most popular and used technique to protect data and to preserve users privacy. The major challenge is how to execute the user queries over the encrypted data. In this context, we proposed three efficient solutions for executing top-k queries over encrypted data stored in the cloud, for two cases where the database is stored at a single node or distributed across multiple nodes of the cloud data center.

In the rest of this chapter, we summarize and discuss the main contributions made in this thesis. We end with some directions for the future work.

5.1 Contributions

A survey of privacy preserving query processing

We gave a thorough introduction to the existing solutions for privacy preserving query processing over outsourced data. We introduced the data outsourcing in general, and then presented the techniques used to preserve user data privacy, *e.g.*, anonymization, differential privacy and data encryption. The later technique is a common method to assure data privacy in the cloud. We presented the main techniques for processing range queries, k-nearest neighbor (kNN) queries and top-k queries over encrypted data, organized in three categories: 1) special data structure based techniques; 2) order preserving encryption based techniques; 3) bucketization based techniques.

BuckTopk

We first addressed the problem of top-k query processing over encrypted data, in the case where the database can be stored in a single node of a cloud data center. We developed a novel system, called BuckTop, designed to encrypt data items, outsource them to a non-trusted cloud node, and answer top-k queries. BuckTop has a top-k query processing algorithm that is executed over encrypted data, and returns a set containing the top-k

results, without decrypting the data in the cloud node. It also comes with a powerful filtering algorithm that eliminates significantly the false positives from the result set.

We validated our system through experimentation over synthetic and real datasets. We compared its response time with order preserving encryption based approach over encrypted data, and with the TA algorithm over original (plaintext) data. The experimental results show excellent performance gains for BuckTop. They illustrate that the overhead of using BuckTop for top-k processing over encrypted data is very low, because of efficient top-k processing and false positive filtering algorithm that eliminates until 99% of the false positives in the cloud node.

SDB-TOPK and SD-TOPK

We addressed the problem of evaluating top-k queries over encrypted data, in the case where the data is vertically partitioned across multiple nodes of the cloud data center. We first proposed two TA-based approaches, called Remote-TA and Block-TA, that correctly find top-k results over encrypted data. But, they need many round-trips between the client and the cloud, and also need to decrypt many encrypted data in the client side. Then, we presented our main contributions, called SDB-TOPK and SD-TOPK, that are executed in the nodes of the cloud data center. They include a top-k query processing algorithm that finds a set of encrypted data that includes the top-k data items. In addition, we proposed a filtering algorithms for each system that filters efficiently the false positives in the nodes. We theoretically proved the correctness of SDB-TOPK and SD-TOPK top-k processing and filtering algorithms. We also analyzed the security of our systems, and proposed efficient strategies for enforcing them.

We evaluated the performance of our solutions over synthetic and real databases. The results show excellent response time and communication cost for the SDB-TOPK and SD-TOPK compared to TA-based approaches. They show that their response time can be several order of magnitude better than that of TA-based algorithms. This is mainly due to their optimized top-k query processing and filtering algorithms. The results also show significant gains in communication cost of SDB-TOPK and SD-TOPK compared to the other algorithms. They also show the efficiency of the filtering algorithm that eliminates almost all false positives, without decrypting the data.

5.2 Directions for Future Work

Our proposed systems address the problem of top-k query processing over encrypted data. To enrich and extend them we can envisage the following research directions.

Supporting skyline queries

Our work could be extended to consider the execution of *skyline queries* over encrypted data. Like top-k, the skyline query [9, 62] is an important query that is used for reducing

the number of results returned to the users, by eliminating those that are not interesting. Thus, skyline queries can be very useful for reducing the communication cost between the cloud and its users.

Formally, a skyline query is defined as follows. Given a dataset D and a set of m attributes, a skyline query Q over D returns a set of tuples which are not dominated by any other tuple in the given attributes. We say that one tuple X dominates another tuple Y, if on all attributes X is as good as or better than Y and on at least one attribute, X is better than Y. The dominance between two tuples is denoted by $X > Y$. The preferences of the attributes are specified in the skyline query. The main differences between skylines and top-k queries is that the results of top-k queries change with the different score functions, while the results of skyline queries are fixed for a given dataset. In addition, the size of the skyline cannot be controlled by the user and it can be as large as the data size in the worst case.

One solution for executing skyline queries over encrypted data is to use an order preserving encryption that allows us to determine the dominance of tuples in each attribute. However, the challenge is to find more secure solutions that do not disclose the order of encrypted data or disclose it partially.

Using software guard extensions technology

Another future work can be the utilization of Software Guard Extensions (SGX) for top-k query processing over encrypted data in the cloud. SGX, proposed by Intel, is a new hardware technology that protects an application's secrets from malicious software by creating isolated memory regions of code and data called enclaves. These non-addressable memory pages are reserved from the system's physical RAM and then encrypted, allowing the application to access its secrets without fear of exposure. SGX applications are built with two parts: 1) Trusted part which consists of the enclaves. They reside in encrypted memory and are protected by Intel SGX. Enclaves are considered trusted because they cannot be modified after they have been built. If a malicious user or software tries to modify an enclave, this trial will be detected and avoided by the CPU; 2) the untrusted part which is the rest of the application. Any application or memory region not protected by Intel SGX is considered untrusted.

We could use this technology, which is available in some clouds (*e.g.*, Azure), to improve the response time and security of query processing over encrypted data. The idea is to use the trusted enclaves to decrypt some data in the cloud, *i.e.*, when it is necessary. However, the challenge is to optimize the operations to be executed in the enclaves because they can be overloaded quickly as their capacity is limited.

Bibliography

[1] AGRAWAL, R., KIERNAN, J., SRIKANT, R., AND XU, Y. Order preserving encryption for numeric data. In *International Conference on Management of Data (SIGMOD)* (2004), pp. 563–574.

[2] AKBARINIA, R., PACITTI, E., AND VALDURIEZ, P. Best position algorithms for top-k queries. In *Proceedings of the VLDB Endowment (PVLDB)* (2007), pp. 495–506.

[3] ANDRZEJAK, A., AND XU, Z. Scalable, efficient range queries for grid information services. In *International Conference on Peer-to-Peer Computing (P2P)* (2002), pp. 33–40.

[4] ANTONOPOULOS, N., EXARCHAKOS, G., LI, M., AND LIOTTA, A. *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications: Models, Methodologies and Applications*. Information Science Reference (an imprint of IGI Global), 2010.

[5] AUMANN, Y., AND LINDELL, Y. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference (TCC)* (2007), pp. 137–156.

[6] BALDIMTSI, F., AND OHRIMENKO, O. Sorting and searching behind the curtain. In *International Conference on Financial Cryptography and Data Security (FC)* (2015), pp. 127–146.

[7] BOLDYREVA, A., CHENETTE, N., LEE, Y., AND O'NEILL, A. Order-preserving symmetric encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)* (2009), pp. 224–241.

[8] BONEH, D., AND WATERS, B. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference (TCC)* (2007), pp. 535–554.

[9] BORZSONY, S., KOSSMANN, D., AND STOCKER, K. The skyline operator. In *International Conference on Data Engineering (ICDE)* (2001), pp. 421–430.

[10] BOST, R., POPA, R. A., TU, S., AND GOLDWASSER, S. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS)* (2015).

[11] BRAKERSKI, Z., GENTRY, C., AND VAIKUNTANATHAN, V. (leveled) fully homomorphic encryption without bootstrapping. *Transactions on Computation Theory Journal (TOCT) 6*, 3 (2014), 13.

[12] BRUNO, N., GRAVANO, L., AND MARIAN, A. Evaluating top-k queries over web-accessible databases. In *International Conference on Data Engineering (ICDE)* (2002), pp. 369–380.

[13] BURKHART, M., AND DIMITROPOULOS, X. Fast privacy-preserving top-k queries using secret sharing. In *International Conference on Computer Communications and Networks (ICCCN)* (2010), pp. 1–7.

[14] CAO, P., AND WANG, Z. Efficient top-k query calculation in distributed networks. In *ACM Symposium on Principles of Distributed Computing (PODC)* (2004), pp. 206–215.

[15] CAO, P., AND WANG, Z. Efficient top-k query calculation in distributed networks. In *ACM Symposium on Principles of Distributed Computing Conference (PODC)* (2004), pp. 206–215.

[16] CHANG, Y.-C., AND MITZENMACHER, M. Privacy preserving keyword searches on remote encrypted data. In *International Conference on Applied Cryptography and Network Security (ACNS)* (2005), pp. 442–455.

[17] CHURCHHOUSE, R. *Codes and Ciphers*. Cambridge University Press, 2002.

[18] CICERI, E., FRATERNALI, P., MARTINENGHI, D., AND TAGLIASACCHI, M. Crowdsourcing for top-k query processing over uncertain data. *Transactions on Knowledge and Data Engineering Journal (TKDE) 28*, 1 (2016), 41–53.

[19] COLES, C., AND YEOH, J. Cloud adoption practices and priorities survey report. Tech. rep., Cloud Security Alliance report, Jan. 2015.

[20] CURTMOLA, R., GARAY, J., KAMARA, S., AND OSTROVSKY, R. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security (JCS) 19*, 5 (2011), 895–934.

[21] DOMINGO-FERRER, J. A provably secure additive and multiplicative privacy homomorphism. In *International Conference on Information Security (ISC)* (2002), pp. 471–483.

[22] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference (TCC)* (2006), pp. 265–284.

[23] DWORK, C., ROTH, A., ET AL. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science Journal 9*, 3–4 (2014), 211–407.

[24] ELMEHDWI, Y., SAMANTHULA, B. K., AND JIANG, W. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *International Conference on Data Engineering (ICDE)* (2014), pp. 664–675.

[25] FAGIN, R. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences (JCSS) 58*, 1 (1999), 83–99.

[26] FAGIN, R., LOTEM, A., AND NAOR, M. Optimal aggregation algorithms for middleware. In *SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2001), pp. 102–113.

[27] FAGIN, R., LOTEM, A., AND NAOR, M. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences (JCSS) 66*, 4 (2003), 614–656.

[28] GENTRY, C. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing Conference (STOC)* (2009), pp. 169–178.

[29] GHINITA, G., KALNIS, P., KHOSHGOZARAN, A., SHAHABI, C., AND TAN, K.-L. Private queries in location based services: anonymizers are not necessary. In *International Conference on Management of Data (SIGMOD)* (2008), pp. 121–132.

[30] GRAEFE, G. Query evaluation techniques for large databases. *Computing Surveys Journal (CSUR) 25*, 2 (1993), 73–169.

[31] GUPTA, M., GAO, J., YAN, X., CAM, H., AND HAN, J. Top-k interesting subgraph discovery in information networks. In *International Conference on Data Engineering (ICDE)* (2014), pp. 820–831.

[32] HACIGÜMÜŞ, H., IYER, B., LI, C., AND MEHROTRA, S. Executing sql over encrypted data in the database-service-provider model. In *International Conference on Management of Data (SIGMOD)* (2002), pp. 216–227.

[33] HACIGÜMÜS, H., MEHROTRA, S., AND IYER, B. Providing database as a service. In *International Conference on Data Engineering (ICDE)* (2002), pp. 29–40.

[34] HARDT, M., AND TALWAR, K. On the geometry of differential privacy. In *ACM Symposium on Theory of Computing Conference (STOC)* (2010), pp. 705–714.

[35] HORE, B., MEHROTRA, S., CANIM, M., AND KANTARCIOGLU, M. Secure multidimensional range queries over outsourced data. *VLDB Journal 21*, 3 (2012), 333–358.

[36] HORE, B., MEHROTRA, S., AND TSUDIK, G. A privacy-preserving index for range queries. In *Proceedings of the VLDB Endowment (PVLDB)* (2004), pp. 720–731.

[37] HU, H., XU, J., REN, C., AND CHOI, B. Processing private queries over untrusted data cloud through privacy homomorphism. In *International Conference on Data Engineering (ICDE)* (2011), pp. 601–612.

[38] KAMARA, S., PAPAMANTHOU, C., AND ROEDER, T. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS)* (2012), pp. 965–976.

[39] KESARWANI, M., KAUL, A., NALDURG, P., PATRANABIS, S., SINGH, G., MEHTA, S., AND MUKHOPADHYAY, D. Efficient secure k-nearest neighbours over encrypted data. In *International Conference on Extending Database Technology (EDBT)* (2018), pp. 264–275.

[40] KOLAHDOUZAN, M., AND SHAHABI, C. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the VLDB Endowment (PVLDB)* (2004), pp. 840–851.

[41] KOSSMANN, D. The state of the art in distributed query processing. *Computing Surveys Journal (CSUR) 32*, 4 (2000), 422–469.

[42] LEI, J. Differentially private m-estimators. In *Neural Information Processing Systems Conference (NIPS)* (2011), pp. 361–369.

[43] LI, C., HAY, M., MIKLAU, G., AND WANG, Y. A data- and workload-aware query answering algorithm for range queries under differential privacy. *VLDB Journal 7*, 5 (2014), 341–352.

[44] LI, D., CAO, J., LU, X., AND CHEN, K. C. Efficient range query processing in peer-to-peer systems. *Transactions on Knowledge and Data Engineering Journal (TKDE) 21*, 1 (2009), 78–91.

[45] LI, J., AND OMIECINSKI, E. R. Efficiency and security trade-off in supporting range queries on encrypted databases. In *ACM Conference on Data and Applications Security and Privacy (CODASPY)* (2005), pp. 69–83.

[46] LI, R., LIU, A. X., WANG, A. L., AND BRUHADESHWAR, B. Fast range query processing with strong privacy protection for cloud computing. *VLDB Journal 7*, 14 (2014), 1953–1964.

[47] LI, X., KIM, Y. J., GOVINDAN, R., AND HONG, W. Multi-dimensional range queries in sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)* (2003), pp. 63–75.

[48] LIU, D., LIM, E.-P., AND NG, W.-K. Efficient k nearest neighbor queries on remote spatial databases using range estimation. In *International Conference on Scientific and Statistical Database Management (SSDBM)* (2002), pp. 121–130.

[49] LIU, D., AND WANG, S. Programmable order-preserving secure index for encrypted database query. In *International Conference on Cloud Computing (CLOUD)* (2012), pp. 502–509.

[50] MACHANAVAJJHALA, A., GEHRKE, J., KIFER, D., AND VENKITASUBRAMANIAM, M. \ell-diversity: Privacy beyond\kappa-anonymity. In *International Conference on Data Engineering (ICDE)* (2006), p. 24.

[51] MAHBOUBI, S., AKBARINIA, R., AND VALDURIEZ, P. Privacy preserving query processing in the cloud. In *Gestion de données - principes, technologies et applications (BDA)* (2016), pp. 61–62.

[52] MAHBOUBI, S., AKBARINIA, R., AND VALDURIEZ, P. Answering top-k queries over outsourced sensitive data in the cloud. In *International Conference on Database and Expert Systems Applications (DEXA)* (2018), pp. 218–231.

[53] MAHBOUBI, S., AKBARINIA, R., AND VALDURIEZ, P. Distributed privacy-preserving top-k query processing. In *Gestion de données - principes, technologies et applications (BDA)* (2018).

[54] MAHBOUBI, S., AKBARINIA, R., AND VALDURIEZ, P. Privacy-preserving top-k query processing in distributed systems. In *European Conference on Parallel Processing (Euro-Par)* (2018), pp. 281–292.

[55] MAHBOUBI, S., AKBARINIA, R., AND VALDURIEZ, P. Top-k query processing over distributed sensitive data. In *International Database Engineering & Applications Symposium (IDEAS)* (2018), pp. 208–216.

[56] MAMOULIS, N., CHENG, K. H., YIU, M. L., AND CHEUNG, D. W. Efficient aggregation of ranked inputs. In *International Conference on Data Engineering (ICDE)* (2006), pp. 72–72.

[57] MICHEL, S., TRIANTAFILLOU, P., AND WEIKUM, G. Klee: A framework for distributed top-k query algorithms. In *Proceedings of the VLDB Endowment (PVLDB)* (2005), pp. 637–648.

[58] MONTRESOR, A., AND JELASITY, M. Peersim: A scalable p2p simulator. In *International Conference on Peer-to-Peer Computing (P2P)* (2009), pp. 99–100.

[59] ÖZSU, M. T., AND VALDURIEZ, P. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[60] PAGEL, B.-U., SIX, H.-W., TOBEN, H., AND WIDMAYER, P. Towards an analysis of range query performance in spatial data structures. In *SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1993), pp. 214–221.

[61] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)* (1999), pp. 223–238.

[62] PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. An optimal and progressive algorithm for skyline queries. In *International Conference on Management of data (SIGMOD)* (2003), pp. 467–478.

[63] PAPADIAS, D., ZHANG, J., MAMOULIS, N., AND TAO, Y. Query processing in spatial network databases. In *Proceedings of the VLDB Endowment (PVLDB)* (2003), pp. 802–813.

[64] PILOURDAULT, J., LEROY, V., AND AMER-YAHIA, S. Distributed evaluation of top-k temporal joins. In *International Conference on Management of Data (SIGMOD)* (2016), pp. 1027–1039.

[65] PIRAHESH, H., HELLERSTEIN, J. M., AND HASAN, W. Extensible/rule based query rewrite optimization in starburst. In *Sigmod Record* (1992), vol. 21, pp. 39–48.

[66] PITOURA, T., NTARMOS, N., AND TRIANTAFILLOU, P. Replication, load balancing and efficient range query processing in dhts. In *International Conference on Extending Database Technology (EDBT)* (2006), pp. 131–148.

[67] POPA, R. A., LI, F. H., AND ZELDOVICH, N. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy (S&P)* (2013), pp. 463–477.

[68] POPA, R. A., REDFIELD, C., ZELDOVICH, N., AND BALAKRISHNAN, H. Cryptdb: protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles (SOSP)* (2011), pp. 85–100.

[69] ROTH, A., AND ROUGHGARDEN, T. Interactive privacy via the median mechanism. In *ACM Symposium on Theory of Computing Conference (STOC)* (2010), pp. 765–774.

[70] SAHIN, C., ALLARD, T., AKBARINIA, R., ABBADI, A. E., AND PACITTI, E. A differentially private index for range query processing in clouds. In *International Conference on Data Engineering (ICDE)* (2018).

[71] SAMARATI, P., AND SWEENEY, L. Generalizing data to provide anonymity when disclosing information. In *SIGMOD/PODS Conference* (1998), p. 188.

[72] SCHNEIER, B. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption workshop (FSE)* (1994), pp. 191–204.

[73] SHANECK, M., KIM, Y., AND KUMAR, V. Privacy preserving nearest neighbor search. In *International Conference on Data Mining Workshops (ICDMW)* (2006), pp. 541–545.

[74] SHI, E., BETHENCOURT, J., CHAN, T. H., SONG, D., AND PERRIG, A. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy (S&P)* (2007), pp. 350–364.

[75] SHI, J., WU, D., AND MAMOULIS, N. Top-k relevant semantic place retrieval on spatial RDF data. In *International Conference on Management of Data (SIGMOD)* (2016), pp. 1977–1990.

[76] SKOPAL, T., KRÁTKỲ, M., POKORNỲ, J., AND SNÁŠEL, V. A new range query algorithm for universal b-trees. *Information Systems Journal (ISJ) 31*, 6 (2006), 489–511.

[77] SMART, N. P., AND VERCAUTEREN, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography (PKC)* (2010), pp. 420–443.

[78] SONG, D. X., WAGNER, D., AND PERRIG, A. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy (S&P)* (2000), pp. 44–55.

[79] STEHLÉ, D., AND STEINFELD, R. Faster fully homomorphic encryption. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)* (2010), pp. 377–394.

[80] TANG, J., CUI, Y., LI, Q., REN, K., LIU, J., AND BUYYA, R. Ensuring security and privacy preservation for cloud data services. *Computing Surveys Journal (CSUR) 49*, 1 (2016), 13.

[81] VAIDYA, J., AND CLIFTON, C. Privacy-preserving top-k queries. In *International Conference on Data Engineering (ICDE)* (2005), pp. 545–546.

[82] VAIDYA, J., AND CLIFTON, C. W. Privacy-preserving kth element score over vertically partitioned data. *Transactions on Knowledge and Data Engineering Journal (TKDE) 21*, 2 (2009), 253–258.

[83] VAN DIJK, M., GENTRY, C., HALEVI, S., AND VAIKUNTANATHAN, V. Fully homomorphic encryption over the integers. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)* (2010), pp. 24–43.

[84] VAN LIESDONK, P., SEDGHI, S., DOUMEN, J., HARTEL, P., AND JONKER, W. Computationally efficient searchable symmetric encryption. In *Secure Data Management Workshop (SDM)* (2010), pp. 87–100.

[85] WANG, X., ZHANG, Y., ZHANG, W., LIN, X., AND HUANG, Z. SKYPE: top-k spatial-keyword publish/subscribe over sliding window. *VLDB Journal 9*, 7 (2016), 588–599.

[86] WONG, W. K., CHEUNG, D. W.-L., KAO, B., AND MAMOULIS, N. Secure knn computation on encrypted databases. In *International Conference on Management of Data (SIGMOD)* (2009), pp. 139–152.

[87] XIANRUI MENG, H. Z., AND KOLLIOS, G. Top-k query processing on encrypted databases with strong security guarantees. In *International Conference on Data Engineering (ICDE)* (2018).

[88] XIAO, L., AND YEN, I.-L. Security analysis for order preserving encryption schemes. In *Annual Conference on Information Sciences and Systems (CISS)* (2012), pp. 1–6.

[89] XIONG, L., CHITTI, S., AND LIU, L. Topk queries across multiple private databases. In *International Conference on Distributed Computing Systems (ICDCS)* (2005), pp. 145–154.

[90] XU, J., FAN, J., AMMAR, M. H., AND MOON, S. B. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *International Conference on Network Protocols (ICNP)* (2002), pp. 280–289.

[91] YANG, H., CHUNG, C., AND KIM, M. An efficient top-k query processing framework in mobile sensor networks. *Data & Knowledge Engineering Journal (DKE) 102* (2016), 78–95.

[92] YAO, B., LI, F., AND XIAO, X. Secure nearest neighbor revisited. In *International Conference on Data Engineering (ICDE)* (2013), pp. 733–744.

[93] YU, H., LI, H.-G., WU, P., AGRAWAL, D., AND EL ABBADI, A. Efficient processing of distributed top-k queries. In *International Conference on Database and Expert Systems Applications (DEXA)* (2005), pp. 65–74.

[94] ZHANG, J., ZHANG, Z., XIAO, X., YANG, Y., AND WINSLETT, M. Functional mechanism: regression analysis under differential privacy. *VLDB Journal 5*, 11 (2012), 1364–1375.

[95] ZHANG, X., LI, G., AND FENG, J. Crowdsourced top-k algorithms: An experimental evaluation. *VLDB Journal 9*, 8 (2016), 612–623.

[96] ZHU, Y., XU, R., AND TAKAGI, T. Secure k-nn computation on encrypted cloud data without sharing key with query users. In *International Workshop on Security in Cloud Computing (SCC)* (2013), pp. 55–60.