PRIVACY-PRESERVING AND AUTHENTICATED DATA CLEANING ON OUTSOURCED DATABASES

by

Boxiang Dong

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Boxiang Dong, Candidate

ADVISORY COMMITTEE

Wendy Hui Wang, Chairman Date

Yingying Chen Date

David Naumann Date

Antonio Nicolosi Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2016

PRIVACY-PRESERVING AND AUTHENTICATED DATA CLEANING ON OUTSOURCED DATABASES

ABSTRACT

Data cleaning detects and removes errors, missing values and inconsistency from dirty data in order to improve the data quality. It is a labor-intensive and complex process. The data-cleaning-as-a-service (DCaS) paradigm enables users to outsource their data and data cleaning needs to computationally powerful third-party service providers. However, the DCaS framework raises several security issues.

One of the issues is how the data owner can protect the private information in the outsourced data. As the outsourced dataset often involves sensitive personal information, it is essential for the data owner to protect the private information when outsourcing the data to the untrusted data cleaning service provider. In this thesis, we focus on the privacy issue of specific data cleaning tasks, including data deduplication and data inconsistency repair. To protect the data privacy, we design efficient encoding and encryption schemes for the data owner to transform the data before outsourcing. The proposed schemes provide robust privacy guarantee against various inference attacks, while preserve the accuracy of data cleaning result.

Another problem is the authentication of the outsourced data cleaning. It is challenging for the data owner with weak computational power to verify whether the untrusted service provider has executed the data cleaning task faithfully. To address this issue, we design a lightweight authentication framework for the data owner to verify the outsourced data deduplication result with small overhead. Any incorrect or incomplete result can be caught with 100% guarantee.

Author: Boxiang Dong

Advisor: Wendy Hui Wang

Date: December 1, 2016

Department: Computer Science

Degree: Doctor of Philosophy

Dedication

This thesis is dedicated to all Stevens students.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Wendy Wang for providing me the opportunity to start my graduate study at Stevens, for her patience and effort in cultivating me to be a qualified researcher, and for her invaluable guidance in my career development. After doing research with her for more than five years, I take her as my role model. I wish that someday I could be an advisor as good as her.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Yingying Chen, Prof. David Naumann, and Prof. Antonio Nicolosi, for their insightful comments and encouragement. I also want to thank them for letting my defense be an enjoyable moment.

I thank my fellow labmates for the stimulating discussions, for the exciting nights we were working together before deadlines, and for the lessons I have learned from them.

Last but not the least, I would like to thank my family: my parents, my sisters, and my parents-in-law for all of the sacrifices that they have made on my behalf. I also express the special gratitude to my wife for supporting me spiritually throughout my Ph.D study and my life in general.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

It is well known that real-world datasets, particularly those from multiple sources, tend to be *dirty* - *inaccurate*, *incomplete*, and *inconsistent*. Data inaccuracy is usually raised by data entry mistakes like spelling errors. Missing data occurs for many reasons such as data collecting device's malfunction or a participant's failure to respond to certain questions in a poll. Data inconsistency exists when different and conflicting versions of the same data appear in different places [33]. Dirty data is commonly present in both single data collections (e.g., due to misspellings during data entry, missing information or other invalid data) and multiple data sources. Inaccuracies, obsolete and inconsistent information makes achieving the big data utopia a slog for businesses and researchers [1]. In a real-life example shown in [42], an insurance company receives 2 million claims per month with 377 data elements per claim. Even at an error rate of 0.001, the claims data contains more than 754,000 errors per month and more than 9.04 million errors per year. According to a recent survey [42], around 40% of companies have suffered losses, problems, or costs due to data of poor quality. 33% of organizations have delayed or cancelled new IT systems because of poor data quality [85]. 42% of the sales professionals claim that inaccurate contact data is the biggest barrier to multichannel marketing. Many data management and decision making applications, e.g., data warehouses and web-based information systems, require extensive support for data cleaning.

Data cleaning deals with detecting and removing errors, missing values and inconsistencies from data in order to improve the quality of data [101]. There are various specialized techniques to process different data cleaning tasks. For ex-

ample, *data deduplication* aims at eliminating duplicated copies of repeating data; *data inconsistency repair* makes the data consistent by making minimal change to the original data; while *data imputation* replaces missing data with plausible values. In data inconsistency repair, the *integrity constraints* such as *functional dependencies (FDs)* play a crucial role. Informally, a set of attributes X in a dataset D is said to functionally determine another set of attributes Y, (written $X \rightarrow Y$) if, and only if, each X value in D is associated with precisely one Y value. The data inconsistency problem rises when the data content does not comply with the FDs.

Data cleaning is a labor-intensive and complex process [13]. Most data cleaning computations hold a high complexity. For example, the complexity of FD discovery, which is at the root of inconsistency repair, is exponential to the number of attributes in the dataset [62, 125]. Even with the FD knowledge, finding inconsistency repair solution demands intensive user interaction and incurs computational complexity NP-complete in the size of the dataset. Besides, nowadays the data is evolving at a high velocity. During the lifetime of a database, we may witness frequent evolution and violations of the pre-defined integrity constraints. Thus, a rigid monitoring and analysis of the evolving data is demanded to keep up with the transformation. To make data cleaning more complicated, huge amounts of data are generated from a large variety of applications and sources. These data is organized with inconsistency in schema, formats and adherence to integrity constraints. A comprehensive and in-depth expertise is needed to create a unified tool to eliminate the errors and discrepancies from such diverse data. With the fast growth of data volume, the sheer size of todays datasets are increasingly crossing the zetabyte barrier. Considering the intrinsic high complexity, the huge data volume makes the cleaning process prohibitively complex.

Providing a data cleaning solution that is powerful, reliable, and easy to use

is extremely challenging. With corporate data exploding in volume and variety, cleaning of large scale data has grown difficult. Therefore, building in-house tools for data cleaning is very difficult, and may create maintenance problems down the road. Alternatively, purchasing expensive proprietary data cleaning solutions may not be acceptable by those medium- and small-size organizations that have limited budgets. A possible solution to resolve this dilemma is to outsource the data to a third-party data cleaning service provider for efficient data cleaning. This motivates the *data-cleaning-as-a-service* (DCaS) paradigm that enables organizations (clients) with limited computational resources to outsource their data cleaning needs to a third-party service provider (server). Recently, several academic and industrial organizations have started investigating and developing technologies and infrastructure for cloud-based data cleaning services (e.g., OpenRefine [2]). The DCaS paradigm alleviates small- and medium-size organizations the pressure to build in-house data cleaning solutions and hire data science experts to maintain the technology. Instead, these organizations enjoy the state-of-the-art data cleaning techniques by only transmitting the raw and nasty data to the service provider.

It is true that the DCaS paradigm enables data owners with limited resources to improve their data quality. However, outsourcing data cleaning to a third-party service provider raises a few issues. One important issue is *data privacy*. When the outsourced data involves any personal information (e.g., medical or financial details of individuals), the privacy of the information needs to be carefully protected. Indeed, personal information is commonly present in datasets that call for cleaning. Consider the data deduplication process which detects different records that refer to the same entity. To make sure of the high deduplication accuracy, it is necessary to include identifying descriptive properties such as date of birth, social number, and home address. However, these information is extremely sensitive and should

not be leaked to any untrusted party. It is therefore paramount to protect data privacy when outsourcing the data to the untrusted DCaS provider.

The other concern is that the untrusted service provider may cheat on the data cleaning results. The service provider has abundant incentives to do so. For example, the service provider may intensionally execute the computation on a small fraction of the data, in order to save the computational cost and improve the revenue [49]. It is also possible that the service provider returns incorrect results due to software bugs or mis-configuration. As the data cleaning result is so critical to the organization's marketing strategy and investment plan, it is essential to authenticate whether the service provider has performed the data cleaning faithfully, and returned the correct results [26].

This thesis focuses on *privacy-preserving* and *authenticated* outsourced data cleaning. Particularly, we take the data deduplication and data inconsistency repair as the central computations into consideration. In specific, in order to protect the sensitive information in the outsourced data, we encrypt the original data before outsourcing. The client only transmits the encrypted dataset to the server to clean. However, the challenge is how to make sure that the server can execute accurate data cleaning over the ciphertext values. For instance, in data deduplication, the ciphertext values of a pair of similar records may be dramatically different to each other. As a consequence, the server cannot identify these two records as near-duplicates based on the ciphertext values. What makes the problem more challenging is the auxiliary knowledge about the private dataset that the server obtains from the external sources. Take the frequency analysis attack as an example. Previous work [90] demonstrates that the frequency analysis attack can recover more than 60% of the patient records from the encrypted attributes. Considering that the frequency distribution of the data values at attribute level are easily accessible in

real-world applications [108, 41], we need to design efficient encryption schemes to defend against such attacks.

On the other hand, authentication of outsourced data cleaning is far from trivial too. Traditionally, the authentication of outsourced computations can be solved by verifiable computing protocols [49, 98]. In these protocols, before outsourcing, the client converts any polynomial-time algorithm to quadratic programs, and constructs verifiable schemes. The server constructs cryptographic proofs to demonstrate the result correctness. However, this solution is not pratical in the DCaS paradigm. The client with weak computational power cannot afford the setup cost which is equivalent to the computation complexity. Besides, the proof verification cost at the client side is close to doing the data cleaning again. Thus, with the consideration of the asymmetric computational power between the client and server, we face the challenge of designing a lightweight authentication framework with cheap setup and verification cost at the client side.

In this thesis, we make the following contributions.

First, we study the privacy-preserving techniques of outsourced *data deduplication*. To be specific, we design the privacy-preserving data-deduplication-as-a-service (PraDa) system that enables the client to outsource her data as well as data deduplication needs to a potentially untrusted server. PraDa consists of two efficient encoding schemes that transform the records into a set of hash values or Euclidean points. These two approaches guarantee to preserve the similarity between the original records, and flatten the frequency of the encoded data. Therefore, PraDa enables the server to discover near-duplicated records from the encoded data, while provides provable privacy guarantee against the frequency analysis attack. We published our research results in a research paper titled “PraDa: Privacy-preserving Data-Deduplication-as-a-Service” in the Proceed-

ings of ACM International Conference on Information and Knowledge Management (CIKM), November 2014.

Second, we investigate the authentication problem of outsourced data deduplication. The key idea is that besides returning the similar record pairs, the server returns a verification object (VO) that can prove the soundness and completeness of the returned data deduplication results. We design an authentication tree structure named *MB-tree*. When traversing the MB-Tree to find near-duplicates, the server also constructs the VO, which consists of certain MB-tree entries, that can be later utilized by the client to verify the correctness of the results. We further reduce the VO verification cost at the client side by replacing a large amount of expensive record similarity calculation with a small number of cheap Euclidean distance computation. We published our research results in a research paper titled "ARM: Authenticated Approximate Record Matching for Outsourced Databases" in the Proceedings of IEEE International Conference on Information Reuse and Integration (IRI), July 2016.

Third, we inspect the privacy issue in the outsourcing framework of *data inconsistency repair*. We take functional dependencies (FDs) as the auxiliary knowledge that the server posses about the private dataset. Given a dataset, a set of attributes X in a dataset D is said to functionally determine another set of attributes Y, (written $X \rightarrow Y$) if, and only if, each X value in D is associated with precisely one Y value. The server fixes the inconsistency problem by making minimal change to the data to comply with the FDs. We define a flexible security model that allows the client to specify a part of the data to be sensitive. We perform the foundational study of security vulnerabilities by adversarial FDs, and formalize the *FD attack*. We prove the NP-completeness of finding the optimal scheme that encrypts the minimum number of data cells. Following that, we design an effi-

cient algorithm to find the data cells that are necessary to be encrypted to defend against the FD attack. We published our research results in a research paper titled “Secure Data Outsourcing with Adversarial Data Dependency Constraints” in the Proceedings of IEEE International Conference on Big Data Security on Cloud (BigDataSecurity), April 2016.

Finally, we consider the scenario where the client is not aware of the FDs in the dataset, while the server is expected to discover the FDs. The discovered FDs function as the basis to improve the data quality through inconsistency repair when the data is evolving in the future. In this process, the client takes the server with the frequency knowledge into account and intends to keep the original data confidential due to the privacy concern. We design F^2, a frequency-hiding, FD-preserving encryption scheme based on probabilistic encryption for *data inconsistency repair*. F^2 allows the client to encrypt the data without the awareness of any FD in the original dataset. To defend against the frequency analysis attack on the encrypted data, we apply the *probabilistic encryption* in a way that the frequency distribution of the ciphertext values is always flattened. The complexity of F^2 is much cheaper than doing inconsistency repair locally. Furthermore, F^2 guarantees to preserve all FDs in the original dataset, while avoids to introduce any false positive FDs. The results of this project are under review.

The remainder of this thesis is structured as follows: In Chapter 2, we discuss the related work. In Chapter 3, we introduce the preliminaries of our research. We present the privacy-preserving and authentication approaches for outsourced data deduplication in Chapter 4 and 5 respectively. Chapter 6 and 7 detail our progress on privacy-preserving techniques of outsourced data inconsistency repair with different privacy concerns. In Chapter 8, we discuss the next steps of my thesis. We conclude this thesis in Chapter 9.

Chapter 2
Related Work

In this chapter, we review the related work. We categorize the previous work into three types: (1) data cleaning approaches, (2) privacy-preserving outsourced computation, and (3) verifiable computing. Next, we discuss each category in detail.

2.1 Data Cleaning

2.1.1 Data Deduplication

Data deduplication is a data compression technique for eliminating duplicate copies or near-duplicate copies of repeating data. It is closely related to *record linkage* and *similarity search*.

McCallum et al. [87] initiate the study on blocking methods to filter a small set of candidates for similarity search in an efficient way. The key idea is to apply a cheap approximate distance measure to divide the data into multiple overlapping canopies. Each canopy is a cluster of similar records. After that, exact similar matching can be initiated within each canopy to discover the matching pairs. For the first time, Gravano et al. [50] provide the capacity of approximate string similarity join to commercial databases. By constructing an additional table to store the q-grams of the records, they are able to calculate the edit distance between the records in a way that is significantly more efficient than the user-defined fuctions. Silva et al. [111] further improve the similarity join in the database research by presenting multiple transformation rules that enable optimization through the generation of equivalent similarity query execution plans. Jin et al. [68] convert

the expensive similarity measurement on string values into the cheap Euclidean distance calculation between Euclidean points. To achieve that, the strings are first mapped into Euclidean points in a similarity-preserving way. Then the matching is applied directly on the points. As a tradeoff, there exists the accuracy loss. Following this direction, various embedding approaches [47, 79, 60] have been proposed. They have different properties and embedding complexity. Most similarity join approaches first convert records into a set of q-grams, and then estimate the edit distance based on the matching of q-grams. On the contrary, Ed-join [126] is a similarity join framework based on the derivation of edit distance from the mismatching of q-grams. This blocking method can dramatically reduce the number of candidate similar records. Yan et al. [129] propose an adaptive approach to adjusting the parameter values in record linkage to obtain the best performance. The parameters include the attributes used for blocking, window size, and choice of similarity measurement metrics.

2.1.2 Data Inconsistency Repair

Inconsistencies, errors and conflicts in a database often emerge as violations of integrity constraints. The data inconsistency repair aims at finding a minimal change whose incurred cost is the smallest so that after the update, the data is consistent with the constraints. Most often, the FDs and conditional functional dependencies (CFDs) are used as the integrity constraints.

Formally, given a relation R, there is a *positive* FD between a set of attributes X and Y in R (denoted as $X \rightarrow Y$) if for any pair of records r_1, r_2 in R, if $r_1[X] = r_2[X]$, then $r_1[Y] = r_2[Y]$. Otherwise, we say $X \rightarrow Y$ is *negative* (denoted as $X \not\rightarrow Y$). We use $LHS(F)$ ($RHS(F)$, resp.) to denote the attributes at the

left-hand (right-hand, resp.) side of the functional dependency F. As FDs function as the guideline to detect inconsistency, how to discover FDs efficiently becomes a research concentration. Kivinen et al. [71] define various measures to evaluate the error of a FD in a relation. Depending on the measurement in use, the authors propose a sampling approach to find the approximate FDs with small errors. Among the previous work, Huhtala et al. [62] propose a classic FD discovery algorithm, named *TANE*. The authors reveal the relationship between a FD $X \rightarrow Y$ and the partitions of attribute set X and $X \cup Y$. By exploiting the properties of partitions, TANE discovers the FDs based on a lattice structure of the attribute sets. On each lattice level, TANE calculates the partitions of all attribute sets and checks the existence of FDs. In order to improve the efficiency, a pruning procedure is proposed to avoid the calculation on a large fraction of the lattice. In constrast, FastFDs [125] traverses the lattice in a depth-first way. However, the complexity of TANE and FastFDs can still be exponential to the number of attributes in the worst case. Papenbrock et al. [97] experimentally evaluate seven FD discovery algorithms, including TANE, FastFDs, and FD_MINE, with regard to both time performance and memory usage. Differently, Heise et al. [57] focus on the discovery of unique column combinations, which is a set of columns whose projection has no duplicate. This problem is closely related to FD discovery as the FD attributes must be a subset of a non-unique column combination. The proposed approach traverses the lattice in a depth-first and random walk combination.

Conditional functional dependency binds specific values with functional dependencies. An example of CFD is the constraint *[Disease='ovarian cancer']* $\rightarrow$ *[Gender='Female']*. Formally, a CFD [13] on a given dataset D is a pair $\varphi(X \rightarrow Y, T_p)$, where (1) X, Y are sets of attributes from D; (2) $X \rightarrow Y$ is a standard FD; and (3) T_p is a tableau with all attributes in X and Y, where for each A in X or

Y and each record $r \in T_p$, $r[A]$ is a constant in the domain of A, or an unnamed variable "_". A CFD $\varphi(X \rightarrow Y, T_p)$ is called a *constant* CFD if its pattern record r_p consists of constants only, while it is called a *variable* CFD if $T_p[Y] = \_$, i.e., the right-hand side (RHS) of its pattern tuple is the unnamed variable _. For simplicity, we only consider constant CFDs in this thesis.

Bohannon et al. [12] define a new cost evaluation model to measure the cost of a set of value modifications. This model incorporates the techniques from record linkage to measure the difference between the original data and the repaired data. [12] proves that it is a NP-complete problem to find minimal-cost repairs under this model. Following that, a heuristic approach is proposed to merge equivalence classes to resolve the conflict. Bohannon et al. [13] take CFDs as the guidelines to detect inconsistency. The CFDs bring novel challenges as a set of CFDs may have conflict in themselves. They design an approach similar to the Armstrong's axioms to detect the conflicting CFDs. Similarly, Wang et al. [121] develop efficient algorithms to check whether a set of fixing rules is consistent. A fixing rule contains an evidence pattern, a set of negative patterns, and a fact value. Cong et al. [31] propose two algorithms to heuristically find repairs with small cost to resolve the inconsistency with regard to CFDs. One algorithm finds the tuple with the smallest repair cost, and the other suggests the optimal value to be updated. The iteration of these two algorithms yield a clean dataset. When the data is distributed either horizontally or vertically, it is a NP-complete problem to detect the violations of CFDs. Fan et al. [48] extend the heuristic detection approach to the distributed setting to increase the parallelism and reduce data shipment between different sites. Chomicki et al. [25] consider the integrity constraints to be both FDs and inclusion dependencies. The authors aim at deleting the minimum amount of tuples to resolve the inconsistency. However, this may lead to serious information

loss. Beskales et al. [11] motivate the inconsistency repair based on FD repairs rather than data repair. They propose a random sampling approach to fix the inconsistency by modifying the FDs. Mazuran et al. [86] also suggest to update the FDs in the evolving data where the data quality is believed to be high. They propose a way to order the FDs based on the fixing cost, and remove the inconsistency by adding a small amount of attributes to the FDs. Chiang et al. [24] propose a unified model that eliminate the inconsistency by modifying both the data values and the FDs at the same time.

2.2 Privacy-preserving Outsourced Computation

When the data is outsourced to a untrusted third party for computation, the data privacy issue turns to be a concentration of the security community. In order to protect data privacy, data can be encrypted into ciphertext values or encoded into another format. Besides, secure multiparty computation (SMC) can also be applied to guarantee that the server does not learn anything from the original data more than the computed result. Differential privacy considers the data privacy from another angle. In this section, we introduce the related work in the four fields.

2.2.1 Encryption

Data encryption is an effective way to prevent the server from obtaining the sensitive information. However, a major challenge is to enable the server to execute the functions on the encrypted data.

[53] is one of the pioneering works that explores the data encryption for the outsourcing paradigm. They propose an infrastructure to guarantee the security of stored data. Different granularity of data to be encrypted, such as row

level, field level and page level, is compared. Chen et al. [52] develop a framework for query execution over encrypted data in the outsourcing paradigm. In this framework, the domain of values of each attribute are partitioned into some buckets. The bucket ID which refers to the partition to which the plain value belongs serves as an index of ciphertext values. The server returns the query result on the encrypted data, and the client needs to perform post-processing to retrieve the correct result in plaintext. After that, a number of privacy-preserving cryptographic protocols are developed for specific applications. For example, searchable encryption [114] allows to conduct keyword searches on the encrypted data, without revealing any additional information. Homomorphic encryption [113] enables the service provider to perform meaningful computations on the encrypted data. It provides general privacy protection in theory, but it is not practical. Hacigumus et al. [52] enable to execute queries over the encrypted data by using conventional encryption techniques and additionally assigning a bucket id to each attribute value. Agrawal et al. [3] design order-preserving encryption (OPE) functions such that the distribution of the encrypted values follows an arbitrarily chosen target distribution. Curino et al. [34] propose a system that provides security protection. Many cryptographic techniques like randomized encryption, order-preserving encryption and homomorphic encryption are applied to provide adjustable security. CryptDB [100] supports processing queries on encrypted data. It employs multiple encryption functions and encrypts each data item under various sequences of encryption functions in an onion approach. Liu et al. [81] propose a secure outsourced cross-user data deduplication framework based on additively homomorphic encryption and the *password authenticated key exchange* protocol. The server obtains no information about the private data, while the deduplication accuracy is reasonable. ENKI [55] aims at protecting the data privacy when multiple users store the private

data on the same server. The private data is protected from the untrusted server, as well as the other users, based on encryption and key management.

In the scenario where the server may posses auxiliary knowledge about the private dataset, it is possible that the server exploits the knowledge to infer the sensitive information from the encrypted data. In [90], experiment results suggest that the inference attack on determinisitc or order-preserving encrytion schemes can reveal up to 99% of all the data. To block inference channels such as frequency analysis attack, Kerschbaum [70] designs a frequency-hiding order-preserving encryption scheme that provides indistinguishability under frequency-analyzing ordered chosen plaintext attack.

2.2.2 Encoding

To protect the sensitive data from the service provider, the client may transform her data so that the server cannot access the actual content of the data outsourced to it. Most of the encoding techniques for secure data cleaning focus on data deduplication. Churches et al. [27] propose a three-party data deduplication protocol based on hash values of q-grams. In particular, for each string, its power set (i.e. the subsets of the original bigram set) is constructed. Each subset of the power set is hashed using a common secret key shared among database owner A and B. Party A and B transmit tuples containing the hash values, the number of q-grams in the hashed subset, the total number of q-grams and an encryption of the identifiers to a third party C. Party C computes the similarity based on the hash values. To prevent the frequency analysis attack, Churches et al. [27] proposes to: (1) use an additional trusted party, thereby extending the number of parties to four; and (2) recommend to use chaffing and winnowing [105] to insert dummy records to

obfuscate the frequency information in the hash values. A trusted party may not exist in practice. Furthermore, besides the substantial increase of computational and communication costs, inserting dummy records is still prone to frequency attacks on the hashes of the q-gram subsets with just one q-gram [119]. Schnell et al. [108] and Durham et al. [41] propose to store the q-grams in the Bloom Filters (BFs); the BFs of two similar strings (i.e., many q-grams in common) have a large number of identical bit positions set to 1. Kuzu et al. [73] formalize the frequency attack on the Bloom filters as a constraint satisfaction problem, and point out that the probability of mapping BF values to original strings is high even when only the frequency of samples is used for the attack. To fix this problem, Durham et al. [41] propose composite BFs by which the attribute-level bloom filter values are to be combined into a single bloom filter for the entire record. Storer et al. [115] consider the duplicates as exact identical contents. They use convergent encryption that uses a function of the hash of the plaintext of a chunk as the encryption key, so that any identical plaintext values will be encrypted to identical ciphertext values. This one-to-one encryption cannot defend against the frequency analysis attack. Scannapieco et al. [107] focus on the three-party scenario and use embedding techniques to transform original strings into a Euclidean space by using the $SparseMap$ method [59]. The embedding guarantees that the distance of the embedded Euclidean points approximates the actual distance of the strings. Both parties then send the embedded strings to a third party which determines their similarity.

Hung et al. [63] design a privacy-preserving way for the schema contributors to upload their schemas to the repository so that the other users can directly reuse the existing schemas and avoid from designing new schema. They allow the contributors to specify the privacy constraints in the schema. The algorithm pro-

duces the anonymized schema by removing some attributes while preserving the schema utility as much as possible. There is little work to protect the data privacy in inconsistency repair as the repair problem is extremely complicated in itself.

There are also numerous encoding-based approaches to solve the privacy issue for other computation tasks. Cheng et al. [23] investigate the privacy attacks based on background knowledge in the outsourced graph database. K-isomorphism, through which anonymization is realized by building k pairwise isomorphic subgraphs, is proved to be a sufficient condition to protect the privacy. Stavros et al. [94] propose a method to protect a query's privacy in a location-based database service. The scheme ensures that a *k-nearest-neighbor* (KNN) query's location can not be distinguished by the service provider from any other location in the data space. A private range query framework [77] is proposed to provide index indistinguishability under the chosen keyword attack. To achieve the goal, a secure indexing structure based on *bloom filters* is utilized to prevent an adversary from deducing the plaintext values.

2.2.3 Secure Multiparty Computation

Given a set of participants, each holding a collection of private data, secure multiparty computation (SMC) allows them to collaborate to get an output from the whole data but learn nothing beyond the output. Clifton et at. [120, 69] propose privacy-preserving association rule mining methods based on SMC when the data is distributed either horizontally or vertically. Emekçi et al. [45] concentrate on learning a decision tree in a privacy-preserving manner. To overcome the limitation by the *partially homomorphic encryption schemes* which only support a specific type of query, Wong et al. [124] adapt the SMC model to the outsourcing

paradigm to provide *data interoperability*, via which a wide range of queries can be executed on the encrypted data. Atallah et al. [9] design a SMC protocol based on homomorphic encryption for record linkage based on edit distance similarity. Ravikumar et al. [103] propose a SMC protocol for two-party record matching using TF-IDF distance and Euclidean distance. Since we consider a client-server framework, we aim to design privacy-preserving encoding methods that are more efficient than SMC computations. [64, 65, 66] propose a three-party protocol that first runs the *blocking* step by utilizing anonymized datasets to accurately match or mismatch a large portion of record pairs. Then the blocking step is followed by the SMC step, where unlabeled record pairs are labeled using cryptographic techniques. Talukder et al. [117] consider a scenario where one party owns the private data quality rules (e.g., functional dependencies) and the other party owns the private data. These two parties wish to cooperate by checking the quality of the data in one party's database with the rules discovered in the other party's database. They require that both the data and the quality rules need to remain private. They propose a cryptographic approach for FD-based inconsistency detection in private databases without the use of a third party. The quadratic algorithms in the protocol may incur high cost on large datasets [117]. Liu et al. [80] propose a SMC protocol, given a trajectory path from a user, for the server to compute the similarity to all paths in a database. The proposed protocol is secure under the semi-honest adverary model.

2.2.4 Differential Privacy

Differential privacy aims to provide means to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records.

Chen et al. [21] propose a sanitization scheme for *set-valued data* such as transaction data and click streams. The transformed data achieves *differential privacy* and supports a wide class of counting queries. Bonomi et al. [15] propose a new transformation method that integrates embedding methods with differential privacy. Its embedding strategy projects the original data on a base formed by a set of frequent grams of variable length. The privacy of the gram construction procedure is guaranteed to satisfy differential privacy. Inan et al. [65] extend [64] by integrating differential privacy with the blocking step in record linkage. McSherry et al. [88] design *PINQ* as a platform for privacy-preserving data analysis. It supplies the functionality of differential privacy to both analysts and the service providers. Applying differential privacy on a set of histogram queries may lead to inconsistent aggregate results. Hay et al. [56] resolve the problem by adding a post-processing phase to guarantee the final result is both differentially-private and consistent. Allard et al. [5] protects the highly-sensitive personal time-series data by combining encryption with differential privacy. The clustering quality due to the smoothing of the impact from differentially private perturbation. To remove the noise from differential privacy, Tramer et al. [118] take the prior distribution into consideration to generate a more reasonable perturbed result. Thus, a balance between utility and privacy is reached. Chen et al. [22] improves the data utility by discovering the dependencies between the attributes in a high-dimensional dataset. The released dataset preserves the join distribution of the original dataset.

2.3 Verifiable Computing

In the outsourcing paradigm, verifiable computing (VC) enables the client to verify the correctness of results returned by the untrusted server. Some verifiable com-

puting protocols are general-purpose, i.e., the protocol can verify the results of any computation. While others are function-specific. In other words, they can only verify the results of a particular function, but at a cheaper cost. Next, we discuss these two flavors of work.

2.3.1 General-purpose Verifiable Computing

Yao [132, 133] proposes the garbled circuit construction protocol that converts any function to a garbled circuit. By combining Yao's protocol with the 1-out-of-2 oblivious transfer protocol [46], we not only obtain a secure multiparty computation protocol, but also achieve a "one-time" verifiable computing protocol [78]. This is because as long as the server learns one of the two output keys by executing the computation correctly once, he can just return it to the client in future computations. The client is not able to detect the cheating behavior. Gennaro, Gentry, and Parno [49] achieves non-interactive verifiable computing by applying homomorphic encryption to make Yao's protocol multi-time verifiable. The key idea is that for each wire, the client randomly generates a pair of master keys for homomorphic encryption. In each round of computation, a new pair of session keys is associated with each wire. Thus, the server cannot use previous output keys for future verification. Setty et al. [109] build a general-purpose verification approach to support a wide range of computation including floating-point fractions and inequality comparisons. Parno et al. [99] propose to construct a VC scheme with public delegation and public verifiability from any attribute-based encryption scheme. *PEPPER* was proposed in [110]. It dramatically reduces the verification cost by using an argument system in which the verifier queries the linear probabilistically checkable proofs in an inexpensive way. *PEPPER* also reduces the prover's over-

head so that its total work is not significantly more than the cost of executing the computation. Groth [51] constructs non-interactive zero knowledge arguments that have sub-linear size and can be efficiently verified by the public. Braun et al. [16] propose a proof-based verification framework that supports the computations that interact with RAM and disk. This work greatly extend the reachability of verifiable computations by incorporating MapReduce jobs. *Pinocchio* [98] is the first practical general-purpose verifiable computation scheme. The authors make substantial improvement in the implementation to reduce the computational cost of the underlying crypto functions. For some computation tasks such as matrix multiplication, the verficiation cost can be marginally cheaper than local native execution.

2.3.2 Function-specific Verifiable Computing

Considering the asymmetric computational power between the client and the server, it is necessary to exploit the unique nature of specific function to design more practical verifiable computing protocols.

In the last decade, intensive efforts have been devoted to ensure the correctness and completeness of the query result in the database-as-a-service paradigm [91, 52]. Authentication data structures [89, 76, 91] are utilized to build a signature of the outsourced database. For example, Li et al. [76] present the Merkle B-Tree as the authenticated index structure and use two boundary nodes in the tree to prove the completeness. Mykletun et al. [89, 91] propose to use the Merkle Hash Tree to facilitate the authentication of query replies. Besides, Sion [112] proposes a method to let the server provide *challenge tokens* for every batch of client's queries to prove that the queries are executed correctly on the entire data. Xie et al. [127] design a probabilistic integrity audit method by inserting a small

amount of fake tuples into the outsourced database. The integrity of a query result is checked by analyzing the fake tuples in the reply. Hu et al. [61] propose a way for the service provider to deliver the location-based services in an authenticable manner. Indexing structures like B-Tree and R-tree and cryptographic proofs are utilized by the server to prepare a query proof. To authenticate a wide range of queries on outsourced streaming data, Papadopoulos et al. [95] design authentication mechanisms for *dynamic vector sums* and *dot products*. These mechanisms are adapted to support a range of linear algebraic queries in stream authentication. Chen et al. [20] develop a novel authentication code called homomorphic secret sharing seal that can aggregate the inputs from individual sources faithfully by the untrusted server for future query authentication. Two authenticated indexing structure are designed based on the authentication code to process queries on multi-dimensional data.

The verification problem becomes more challenging when the outsourced computing replies on complex data mining algorithms. Wong et al. [123] initiate the study by proposing a result verification scheme for outsourced frequent itemset mining. They consider a curious server without background knowledge about the dataset. A set of fake items are inserted into the original dataset to construct fake (in)frequent itemsets. The returned result is checked against the fake (in)frequent itemsets to verify the correctness and completeness. Following that, Dong et al. [36] design verification techniques by constructing evidence (in)frequent itemsets from real items. Both these approaches give the client probabilistic guarantee about the result's integrity. [38, 37] adapt the *secure intersection verification protocol* to check the result integrity of outsourced frequent itemset mining to provide the client with a deterministic guarantee. The key idea is that the server returns cryptographic proofs to demonstrate the correctness and completeness. Liu et al.

[82, 83, 84] design efficient verification approaches for some other data mining tasks. In all these work, three types of untrusted servers (*sloppy, lazy and malicious*) are formally defined according to the behavior. In the outlier detection, Liu et al. [82] design a probabilistic verification approach by injecting a set of artificial tuples which consist of artificial outliers and artificial non-outliers. Any incorrect/incomplete result can be caught with high confidence. Liu et al. [83] propose both deterministic and probabilistic approaches to verify the correctness of outsourced k-means clustering tasks. The deterministic approach applies the Voronoi diagram to pick a small portion of centroids for verification. The probabilistic approach inserts artificial tuples that are far away from the original tuples. These artificial tuples are clustered independently from the original data. The client can check the result correctness by comparing it with the artificial clusters. In [84], Liu et al. introduce three probabilistic verification approaches for the client to check the integrity of outsourced Bayesian network structure learning tasks with small computational overhead. Papadopoulos et al. [93] design an efficient answer-verification process for authenticated pattern matching. An authenticated data structure named *suffix-tree* is proposed to store the accumulation values of the data. A nice property of this approach is that the proof includes 10 accumulation values at most, which saves the bandwidth for proof transmission. Armknecht et al. [7] propose a way for the client to verify the actual storage space after the server launches the data deduplication.

Chapter 3
Preliminaries

3.1 Data Cleaning

In an ideal world, data would go into the database accurately, completely and consistently; in other words, cleanly. Unfortunately, in reality, that is rarely possible. In real-world applications, data tend to be dirty - inaccurate, incomplete and inconsistent. *Data cleaning*, also known as *data cleansing* or *scrubbing*, aims at detecting and removing errors, duplications, redundancies and fixing the missing and inconsistent data in order to improve data quality. There are various data cleaning tasks.

- *Data deduplication* is a data compression technique for eliminating duplicate copies or near-duplicate copies of repeating data.
- *Data inconsistency repair* modifies the data to make it consistent with the integrity constraints.
- *Data imputation* is the process of replacing the missing data with substituted values.
- *Data smoothing* identifies outliers and smoothes out noisy data.

In this thesis, we focus on data-deduplication and data inconsistency repair. In the following subsections, we introduce more details about these two data cleaning tasks.

3.1.1 Data Deduplication

Data deduplication aims to identify the records that refer to the same entity. It effectively reduces the storage needs by ensuring only one unique instance of an entity is actually retained on the storage media. A concept that is highly related to data deduplication is *record linkage*, which shares the same goal as data deduplication as finding duplicated records in the datasets. There are two types of duplicated records: identical duplicates and *near-duplicates*. Most of the existing data deduplication work considers near-duplicates that have small differences in data values. In this thesis, we consider the problem of finding near-duplicates from string databases.

String Similarity Metric. Record linkage techniques are used to link together records that relate to the same entity (e.g. patient or customer) in one or more datasets where a unique identifier is not available. The key of most record linkage techniques is to measure the similarity of records. Many record linkage methods treat each attribute value as a string (e.g., [50]). In this thesis, we consider approximate string matching. There are a number of approximate string similarity measurement methods in the literature, e.g., q-grams, edit distance, and Euclidean distance (See [72] for a good tutorial.) In this thesis, we consider two similarity measurement metrics, i.e., the *edit distance* and the *jaccard* similarity based on q-grams.

- The edit distance between a pair of strings S_1 and S_2 quantifies the minimum number of removal, insertion, or substitution of characters to transform S_1 into S_2. It is widely applied in error corrcting, pattern recognition, and other related applications.

  One efficient way to compute the edit distance is dynamic programming. Let

$S[i]$ denote the i-th character of string S, then the edit distance between S_1 and S_2 is $DST(S_1, S_2) = d_{\ell_1,\ell_2}$, where ℓ_1 (ℓ_2) is the length of S_1 (S_2), and $d_{i,j}$ is defined in Equation 3.1.

$$d_{i,j} = \begin{cases} 0 & : i = 0 \\ 0 & : j = 0 \\ d_{i-1,j-1} & : S_1[i] = S_2[j] \\ min\{d_{i-1,j-1}, d_{i-1,j}, d_{i,j-1}\} + 1 & : S_1[i] \neq S_2[j] \end{cases} \tag{3.1}$$

The value $d_{|S_1|,|S_2|}$ is the edit distance $DST(S_1, S_2)$. We say two strings S_1 and S_2 are *similar*, denoted as $S_1 \simeq_E S_2$, if $DST(S_1, S_2) \leq \theta$, where θ is a user-specified similarity threshold. Otherwise, we say S_1 and S_2 are *dissimilar* (denoted as $S_1 \not\simeq_E S_2$).

- To calculate the jaccard similarity, each string is decomposed into a set of tokens of length q (called q-*grams*). Then for any two strings S_1 and S_2,

$$jaccard(S_1, S_2) = \frac{|G(S_1, q) \cap G(S_2, q)|}{|G(S_1, q) \cup G(S_2, q)|}, \tag{3.2}$$

where $G(S_1, q)$ ($G(S_2, q)$, resp.) is the set of q-grams of S_1 (S_2, resp.). We say two strings S_1 and S_2 are δ-*similar* regarding the Jaccard metrics, denoted as $S_1 \simeq_J S_2$, if $jaccard(S_1, S_2) \geq \delta$. Otherwise, S_1 and S_2 are dissimilar (denoted as $S_1 \not\simeq_J S_2$).

3.1.2 Data Inconsistency Repair

Integrity constraints serve as restrictions on data semantics. Inconsistencies, errors and conflicts in a database often emerge as violations of integrity constraints.

Due to careless maintenance or the negligence to obey data quality rules, inconsistencies are commonly present in a large database. Therefore, integrity constraints like functional dependencies have been widely studied to capture errors from semantically related values. For example, we show a dataset containing the conference information in Figure 3.1. In Figure 3.1, the records r_1, r_2 and r_3, r_5 fail to comply with the FD $\{Country\} \rightarrow \{Capital\}$.

TID	Conference	Year	Country	Capital	City
r_1	SIGMOD	2007	China	Beijing	Beijing
r_2	ICDM	2014	China	Shanghai	Shenzhen
r_3	KDD	2014	U.S.	Washington D.C.	New York City
r_4	KDD	2015	Australia	Canberra	Sydney
r_5	ICDM	2015	U.S.	New York City	Atlantic City

Figure 3.1: An Example of Data Inconsistency

Given a set of integrity constraints, data inconsistency repair aims at detecting the inconsistencies and finding a minimal change with which the repaired dataset is consistent with the constraints. Functional dependencies (FDs) are commonly used as the integrity constraints. For any FD $F : X \rightarrow Y$ such that $Y \subseteq X$, F is considered as *trivial*. In this thesis, we only consider non-trivial FDs. It is well known that for any FD $F : X \rightarrow Y$ such that Y contains more than one attribute, F can be decomposed to multiple functional dependency rules, each having a single attribute at the right-hand side. For the following discussions, we assume the FD rules only contain one single attribute at the right-hand side. We assume that the client is not aware of any FD in her dataset. We also assume that the data is consistent with the FDs. Given the original dataset D and its encoded version $\hat{D}$, we say the FD F is a *false positive* if F holds in $\hat{D}$ but not in D. We will show that our encoding scheme guarantees that the server can discover all true FDs in D, and no false positives.

There exists monotone property among positive and negative FDs. In particular, the *upward closure* specifies that for any positive FD $F : X \rightarrow Y$, the FD $X' \rightarrow Y$ must also hold for any attribute set X' such that $X \subseteq X'$. And the *downward closure* states that for any negative FD $X \not\rightarrow Y$, $X' \not\rightarrow Y$ must also hold for any attribute set X' such that $X' \subseteq X$.

We use the *edit distance* to measure the distance between the original data and the repaired data. In particular, the edit distance between two strings S_1 and S_2 represents the minimal number of insertions, deletions and substitutions to transform S_1 into S_2. The edit distance between two datasets is the summation of the distances in every data cell. It has been proved that the problem of finding a inconsistency repair with minimal cost is NP-complete [31]. Follow that, various heuristic algorithms [12, 13, 121, 31] have been designed to find the repair with small cost to make the data consistent with the integrity constraints.

3.2 The Data-Cleaning-as-a-Service (DCaS) Paradigm

Traditional way of data cleaning requires the data owner to be equipped with expensive computational resources and strong expertise. In recent years, the advantage of network technologies and cloud computing enables the data-cleaning-as-a-service (DCaS) paradigm in which the data owner with the need to perform data cleaning on big data but without resources outsources the data cleaning task to a service provider. Typically, the architecture of the DCaS paradigm involves two entities:

The data owner (client) owns the private dataset and outsources his/her data cleaning tasks to the DCaS service provider. To protect the private information in D, the client encodes D to $\hat{D}$, and sends $\hat{D}$ to the server. If it is necessary, the

client also specifies the data cleaning configuration (e.g. the matching rules in data deduplication).

The service provider (server) receives the dataset and the configuration from the client and performs data cleaning according to the client's instruction. The data cleaning result is returned to the client after the cleaning is finished.

3.3 Embedding Methods for Similarity Searching

Given two records S_1 and S_2, normally the complexity of computing edit distance is $O(|S_1||S_2|)$, where $|S_1|$ and $|S_2|$ are the lengths of S_1 and S_2. One way to reduce the complexity of similarity measurement is to map the records into a multi-dimensional Euclidean space, such that the similar strings are mapped to close Euclidean points. The main reason of the embedding is that the computation of Euclidean distance is much cheaper than string edit distance. A few string embedding techniques (e.g., [47, 68, 59, 79, 60]) exist in the literature. These embedding techniques include SparseMap [79, 60], FastMap [47], StringMap [68], and MetricMap [122, 130]. These algorithms have different properties in terms of their efficiency and distortion rate (See [58] for a good survey). In this thesis, we consider an important property named *contractiveness* property of the embedding methods, which requires that for any pair of strings (S_i, S_j) and their embedded Euclidean points (p_i, p_j), $dst(p_i, p_j) \leq DST(S_i, S_j)$, where $dst()$ and $DST()$ are the distance function in the Euclidean space and string space respectively. In this thesis, we use $dst()$ and $DST()$ to denote the Euclidean distance and edit distance. The contractiveness property is important as for any two embedded points p_i and p_j that are θ-dissimilar (i.e. $dst(p_i, p_j) > \theta$), their original strings S_i and S_j must be θ-dissimilar. We use the *SparseMap* method [59] for the string embedding.

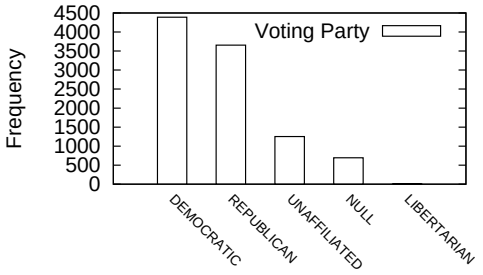
SparseMap preserves the contractiveness property. We will show how to leverage the contractiveness property to improve the verification performance in Chapter 5. Note that the embedding methods may introduce false positives, i.e. the embedding points of dissimilar strings may become close in the Euclidean space.

3.4 Inference Attack

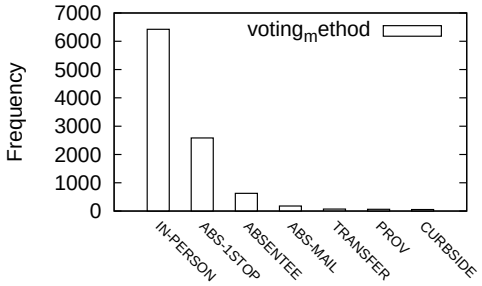
We assume the server is *curious-but-honest* that it follows the DCaS outsourcing protocols honestly (i.e., no cheating on storage and faithful execution of data analysis), but he is curious to extract additional information from the received dataset. Since the server is potentially untrusted, the client sends the encrypted data to the server. The client considers the true identity of every cipher value as the sensitive information that should be protected.

The server may posses some background knowledge about the dataset and the encryption techniques and tries to infer more information from the encrypted dataset. In this thesis, we consider that the server may have three types of prior knowledge and is able to initiate the following three attacks.

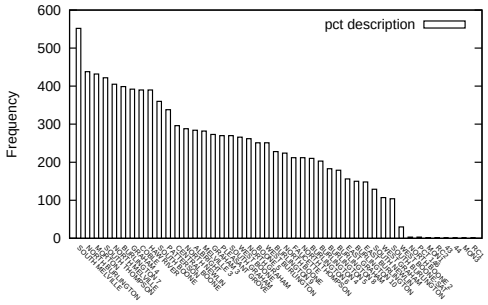
Frequency Analysis Attack. We assume that the attacker may know the domain values in the original dataset D. We also assume that the attacker may possess the frequency knowledge of plain values in D. As shown previously [108], such adversary knowledge can be easily obtained in real-world applications. In reality, the attacker may possess approximate knowledge of the value frequency in D. However, in order to make the analysis robust, we adopt the conservative assumption that the attacker knows the exact frequency of every plain value in D. The attacker can launch the frequency analysis attack by matching the encoded data with the original ones of the same frequency.



(a) “Voting Party” attribute



(b) “Voting Methods” attribute



(c) “PCT Description” attribute

Figure 3.2: Frequencies of Different Attributes

Figure 3.2 shows an example of frequency distributions of various attributes the North Carolina voter registration (NCVR) dataset[1]. We select three attributes namely “Voting Party”, “Voting Methods”, and “PCT Description” and count their frequency over 10k tuples. It is easy to observe, the three attributes have different frequency distributions. For example, in Figure 3.2 (a), there are 4 unique values: “Democratic Party”, “Republican Party”, “Libertarian Party” and “Unaffiliated” that illustrated the party the voter voted for. A “Null” value shows that the field is not filled with any value. From the chart, we can easily observe that the Democratic has the highest support while the Libertarian only receives few votes. On the other hand, the frequency distribution of voting methods (Figure 3.2 (b)) is much skewer. The frequency of “In-person” values is more than 6400 while other values’

[1]Available at `ftp://www.app.sboe.state.nc.us/data`

frequencies are much smaller. The attacker can utilize such information to help decrypting the dataset. The last attribute “PCT description” also shows few values have distinguishable frequency (Figure 3.2 (c)). This example drawn from the real dataset shows that the frequency distribution of different attributes is different and can be used to help decrypting the dataset. Moreover, recent work [90] shows how effective is the frequency analysis attack at uncovering sensitive data encrypted by a deterministic scheme. By launching attacks on real-world data from hospitals, the frequency analysis attack is able to recover the *Mortality Risk* and *Disease Severity* attribute for almost all patients. Besides, it can also reveal the *Primary Payer* and *Admission Type* attribute for more than 60% of the patients.

Known-scheme Attack. We assume the attacker knows the details of the encoding methods that are used by the client. He may try to break the encoding scheme by utilizing the knowledge of the encoding methods.

NM	SEX	AGE	DC	DS	SR
Alice	F	53	CPD5	Breast cancer	N
David	M	30	VPI8	HIV	Y
Ela	F	24	VPI8	HIV	N

(a) The original dataset D

NM	SEX	AGE	DC	DS	SR
Alice	F	53	CPD5	α	N
David	M	30	VPI8	HIV	Y
Ela	F	24	VPI8	γ	N

(b) The unsafe encoded dataset $\bar{D}$

Figure 3.3: An Instance of Customer Information

FD-based Attack. In some real-word applications, the adversary may be aware of the integrity constraints in the data. In this thesis, we consider the integrity constraints that take the form of functional dependencies (FDs). Informally, a $FD : X \rightarrow Y$ constraint indicates that attribute set X uniquely determines attribute set Y. For example, the FD *Zipcode*$\rightarrow$*City* indicates that all tuples of the same zipcode values always have the same *City* values. It is possible that FDs can serve as an important piece of adversary knowledge and bring security vul-

nerabilities. For example, consider a hospital that stores its patients' information, including patient's name (NM), gender (SEX), age (AGE), hospital-wide disease code (DC) and disease (DS), in a dataset. The dataset also has the SR attribute indicating whether the patients are willing to disclose his/her disease information to any third-party (*SR* = "Y"/"N" means *can be disclosed*/*no disclosure*). Consider an example of the instance in Figure 3.3 (a). Assume it has the FD rule: F: DC $\rightarrow$ DS. Before outsourcing the dataset to a third-party service provider, the hospital encodes its data according to the patients' settings. The encrypted instance is shown in Figure 3.3 (b). Consider Ela's (encrypted) disease value. Since David has the same disease code as Ela's, the attacker can easily infer Ela's disease if he knows the FD rule F.

Chapter 4
Privacy-preserving Outsourced Data Deduplication

In this chapter, we discuss the problem of how to protect the data privacy in the outsourced data deduplication computation. We consider that the client has a private dataset D and would like to find out all record pairs in D that are near-duplicates. However, she is not willing to reveal any private information in D to the server. Therefore, to protect the private information in D, she transforms D to $\hat{D}$, and sends $\hat{D}$ together with the matching decision rule based on the Jaccard similarity of q-grams to the server. When the server receives $\hat{D}$, it executes the data deduplication on $\hat{D}$ to find all data pairs that are similar (e.g., their distance is less than the client's threshold), and returns the results $\hat{R^S}$ to the client. This work has been published in the proceedings of ACM International Conference on Information and Knowledge Management (CIKM), November 2014.

4.1 Preliminaries

4.1.1 Utility of Privacy-preserving Techniques

We define *precision* and *recall* to measure the impact of privacy-preserving techniques to the accuracy of data deduplication. Formally, given a dataset D, let R_S be the similar records in D, and $\hat{R_S}$ be the pairs of similar records (possibly in encoded format) returned by the service provider. The *precision* is measured as

$$Pre = \frac{|R_S \cap \hat{R_S}|}{|\hat{R_S}|}. \tag{4.1}$$

And the *recall* is measured by

$$Rec = \frac{|R_S \cap \hat{R}_S|}{|R_S|}. \tag{4.2}$$

Intuitively, the precision measures the fraction of near-duplicates by the encoding method that are correct, while the recall measures the fraction of original near-duplicates that are preserved by the encoding methods.

4.1.2 Locality Sensitive Hash (LSH) Function

LSH is a set of hash functions that map objects into several buckets such that similar objects share a bucket with high probability, while dissimilar ones do not. Formally, let $\mathcal{P}$ be the domain of objects, and $dist()$ be the distance measure between two objects. Then,

Definition 1. [(d_1, d_2, $prob_1$, $prob_2$)-sensitive hashing] A function family $\mathcal{H}$ is called *(d_1, d_2, $prob_1$, $prob_2$)-sensitive* if for any function $h \in \mathcal{H}$, and for any two objects $p_1, p_2 \in \mathcal{P}$: (1) If $dist(p_1, p_2) < d_1$, then $Pr_{\mathcal{H}}[h(p_1) = h(p_2)] \geq prob_1$; (2) If $dist(p_1, p_2) > d_2$, then $Pr_{\mathcal{H}}[h(p_1) = h(p_2)] \leq prob_2$, where $0 \leq d_1 < d_2 \leq 1$, and $0 \leq prob_1, prob_2 \leq 1$.

A family is interesting when $prob_1 > prob_2$. Intuitively, any two similar objects will have the same LSH value with high probability, while any two dissimilar objects will have the same LSH value with a low probability. As we consider the q-gram based Jaccard similarity, we consider the distance of any two objects p_1 and p_2

$$dist(p_1, p_2) = 1 - jaccard(p_1, p_2). \tag{4.3}$$

To date, several LSH families have been discovered for different distance

metrics. In this thesis, we consider the q-gram based Jaccard metric and use the MinHash [29] as the LSH function family. The family of MinHash functions is a $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive family for any d_1 and d_2, where $0 \leq d_1 < d_2 \leq 1$. Our method can be applied to other distance metrics such as Hamming distance and edit distance. The basic idea of MinHash is to first pick a random permutation π on the ground universe of bit vectors. Then, apply this random permutation to re-order indices of a sparse binary vector. Define

$$h_\pi(p) = min\{\pi(v)|v \in p\}. \tag{4.4}$$

In other words, the hash value of the object p equals to the first index that contains "1". It is not hard to prove that the probability

$$Pr_\pi[h_\pi(p_1) = h_\pi(p_2)] = jaccard(p_1, p_2). \tag{4.5}$$

We follow [74] to divide the MinHash signature matrix into μ bands consisting of ν rows each. We require that $\mu\nu = \xi$, where ξ is the length of the MinHash signatures. For each band, there is a hash function that takes vectors of r integers (the portion of one column within that band) and hashes them to some large number of buckets. We use a separate bucket array for each band, so the columns of the same vector in different bands will not hash to the same bucket. Then the client outsources the MinHash signatures and the threshold δ to the server. The server will examine the signatures; those signatures agree on at least δ fractions of bands are considered as similar candidates.

4.1.3 Attack Model

We consider that the server may know the domain values in the original dataset D. We assume that the server may posses knowledge of the value frequency in D. Besides the frequency knowledge, the server may know the details of our encoding schemes and try to utilize such knowledge to break the cipher. Based on the attacker's knowledge, we consider two types of attacks, namely *frequency analysis attack* and *known-scheme attack*.

4.1.4 Our Approaches in a Nutshell

We propose two approaches: (1) Locality-Sensitive Hashing based (LSHB) approach, and (2) embedding & homophonic substitution (EHS) approach.

LSH-based Approach. The basic idea of LSHB scheme is to use a set of hash functions to map objects into several buckets such that similar objects share a bucket with high probability, while dissimilar ones do not [67]. Following this property, the client uses the LSH functions to transform her data, and send LSH values to the server for data deduplication. To defend against the frequency-based attack, identical strings are split into multiple *split copies*, so that the LSH values of these split copies are different. In addition, duplicates of split copies may be inserted to ensure that the frequency of LSH values follows a uniform distribution. We control the number of split copies so that the encoded dataset provides provable privacy guarantee with acceptable amounts of accuracy loss of data deduplication. More details of the LSH-based approach can be found in Section 4.2.

Embedding & Homophonic Substitution (EHS) Approach. This approach first maps all strings in D to an Euclidean space; each unique string is mapped to a point in the Euclidean space. To defend against the frequency-based attack, for

any plaintext string of frequency $f > 1$, its corresponding point in the Euclidean space is encoded as f different points in the same space. The client sends the encoded Euclidean points to the server for data deduplication. Compared with the LSH approach, EHS does not insert any new record, but requires more expensive communication overhead. More details of the EHS approach can be found in Section 4.3.

For both approaches, we assume the transformation is applied at the cell level; each individual attribute value is transformed to either a LSH value or an EHS encoding.

4.2 Locality-Sensitive Hashing Based (LSHB) Approach

In this section, we propose two locality-sensitive hashing based (LSHB) approaches, namely the *duplication* approach (Section 4.2.1) and the *split-and-duplication* approach (Section 4.2.2). For both approaches, we present their details, and analyze their privacy guarantee as well as the impact to the performance of data deduplication. The key idea of the two LSHB methods is to map the strings to locality-sensitive hashing (LSH) values. Intuitively, sending LSH values guarantees privacy as it does not involve any data. However, the LSH values are vulnerable against the frequency analysis attack by which the attacker tries to map the LSH values to strings based on their frequency. Formally, given n_S unique strings $\mathcal{S} = \{S_1, \ldots, S_{n_S}\}$ and $n_L \leq n_S$ LSH values of these strings, let f_i and l_i be the frequency of the string $S_i (1 \leq i \leq n_S)$ and the LSH value $L_i (1 \leq i \leq n_L)$ respectively. Then for each LSH value $L_i (1 \leq i \leq n_L)$, the attacker tries to find its matching candidates $\mathcal{S}' \subseteq \mathcal{S}$ s.t. (1) for all $S_i, S_j \in \mathcal{S}'$, $S_i \simeq S_j$, and (2) $\sum_{S_j \in \mathcal{S}'} f_j = l_i$. In other words, the attacker will try to find those similar strings that are mapped to the same

LSH value based on their frequency. Then the attacker's probability that the LSH value $L_i (1 \leq i \leq n_L)$ is mapped to a specific string $S_j \in \mathcal{S}'$ is $prob(L_i \rightarrow S_j) = \frac{1}{|\mathcal{S}'|}$. For any LSH value that has few matching candidates (e.g., those strings have few similar others), it can be mapped to the correct string with high probability (possibly 100%).

4.2.1 Duplication Approach

To defend against the frequency analysis attack on the LSH-based approach, we define α-*privacy* to quantify the desired privacy guarantee:

Definition 2. [α-privacy] Given a set of unique strings $\mathcal{S}$ $\{S_1, \ldots, S_{n_S}\}$ and their LSH values $\mathcal{L}$ $\{L_1, \ldots, L_{n_L}\}$, we say $\mathcal{L}$ is α-*private* if for each $L_i \in \mathcal{L}$, the probability that maps it to string S_j satisfies that $prob(L_i \rightarrow S_j) \leq \alpha$.

Intuitively, lower α provides higher privacy guarantee.

Next, we present our duplication-based approach that constructs α-private LSH values. It consists of two steps:

Step 1: Grouping. First, we construct the LSH values of all strings in $\mathcal{S}$. Second, we sort LSH values by their frequency in descending order. Third, starting from the most frequent LSH value, we repeat grouping $k = [\frac{1}{\alpha}]$ adjacent values together into one group (called an α-group), until all LSH values are grouped. The last group, if less than k in size, is merged with its previous group.

Step 2: Frequency homogenization. For each group, let l_{max} be the maximum frequency of its LSH values. Then for each LSH value L_i in the group, we add $l_{max} - l_i$ copies of L_i, where l_i is the frequency of L_i.

After the two steps, all LSH values in the same group have the same frequency, regardless their frequency in the original dataset.

Privacy Analysis of Duplication Approach

Frequency Analysis Attack. The duplication-based approach guarantees that there are always at least $[\frac{1}{\alpha}]$ LSH values that are of the same frequency. Therefore, even for the worst case that these values have no other similar strings, the probability of associating any LSH value with its original string based on their frequency is at most α. In other words, the duplication approach guarantees α-privacy.

Known-scheme Attack. When the attacker knows the details of the duplication approach, he can apply the following two-step attack trying to find the mapping between LSH values and strings. First, the attacker tries to find the mapping between string groups and LSH groups. He can execute the grouping step of the duplication approach over the string values to get string groups; the string groups are expected to be similar to the LSH groups of the strings. Then for each string group, he will try to find out the corresponding LSH values based on their frequency. From the knowledge of the encoding algorithm, he knows that with high probability all LSH values of the same frequency belong to the same group. Furthermore, for any string S and its LSH value L, it must be true that $freq(S) \leq freq(L)$. Based on these information, he can map LSH groups to the string groups based on their frequency. Note that all LSH values in the same group have the same frequency, while the strings in the same group do not have to. Apparently, for any string groups of k unique and dissimilar strings, its LSH group must contain exactly k unique LSH values. Furthermore, since it is highly likely that a string group only contains dissimilar strings, the maximum frequency of the LSH group must be no larger than the maximum frequency of its string group. The attacker can use these knowledge to map LSH groups to string groups. Under the worst case, there is only one such mapping. Second, the attacker tries to find the mapping between specific strings

and LSH values based on the group mapping result. Consider a mapping that consists of k strings and k LSH values. The attacker constructs all possible mappings between k LSH values and k strings, with each LSH value mapped to exactly one string. There are $k!$ such mappings. Among these mappings, $(k-1)!$ mappings map a specific LSH value $L_j(1 \leq j \leq k)$ to its correct string $S_i(1 \leq i \leq k)$ correctly. Therefore, the probability

$$prob(L_j \rightarrow S_i) = \frac{(k-1)!}{k!} = \frac{1}{k}. \tag{4.6}$$

Since $k = [\frac{1}{\alpha}]$, the duplication-approach is α-private against the known-scheme attack.

Complexity Discussion

The duplication approach requires $O(n_S)$ to group and add duplicates at the client side, where n_S is the number of unique strings in D. The complexity of data deduplication at the server side is $O((|D|+n_d)^2)$, where $|D|$ is the total number of strings in D ($|D| \gg n_S$), and n_d is the total number of duplicated LSH values. In particular, the number of duplicated LSH values that are added to the string S_i is

$$f_i' = (f_{max} - f_i). \tag{4.7}$$

The total number n_d of duplicated LSH values added to D is

$$n_d = \sum_{i=1}^{n_S} f_i'. \tag{4.8}$$

It can be large, especially for the datasets of skewed frequency distribution. This will hurt the performance of data deduplication in terms of both accuracy and time performance. Next, we discuss our *split-and-duplication* approach that can improve the duplication-based approach by reducing the number of added LSH copies, while still providing provable guarantee against both the frequency analysis attack and the known-scheme attack.

4.2.2 Split-and-duplication (SD) Approach

The main idea of the split-and-duplication (SD) approach is to add a *split* step between the grouping step and the frequency homogenization step of the duplication-based approach. The aim of the split step is to convert those LSH values of large frequency to several different ones of smaller frequency, so that the number of added LSH values by the frequency homogenization step can be reduced. Apparently the split copies of the same string cannot have the same LSH values. A naive approach is, for each LSH value L of high frequency, we make several copies of L, so that the frequencies of these copies accumulate to the same frequency. Then for each copy of L, we randomly modify a number of its bits, so that all split copies of L will be different after modification. Though simple, this approach may lead to high amounts of accuracy lost due to the fact that the LSH values of similar strings may not share the same bucket anymore. Therefore, instead of making split copies of LSH values, we propose to make split copies of the strings, and construct LSH values of these string split copies. We ensure that for each similar string pair, at least one pair of their split copies share the same LSH value with high probability. Therefore, the server still can identify the similar strings from the received LSH values. Next, we present the details of the split step.

Our split step is based on the *permutation* mechanism. Formally, let S and S' be equal-size strings. We say S is a *permutation* of S' if there exists a bijection π such that for each i, $S[i] = \pi(S'[i])$. For instance, the string "cacb" is a permutation of the string "abac" as there exists the bijection $\pi(a) = c$, $\pi(b) = a$, and $\pi(c) = b$. We call π a permutation function. For any given string S of frequency f, the client defines sc unique permutation functions, and constructs sc unique split copies of S by applying the sc permutation functions on S. Each permutation generates a split copy of frequency $[\frac{f}{sc}]$. Continuing our example, consider the string "abac", and two permutation functions π_1 and π_2 such that $\pi_1(a) = c$, $\pi_1(b) = a$, and $\pi_1(c) = b$, as well as $\pi_2(a) = b$, $\pi_2(b) = c$, and $\pi_2(c) = a$, it has two split copies "cacb" and "bcba". We have the following theorem to show that permutation preserves Jaccard similarity.

Lemma 1. *Given any two strings S_1 and S_2, let π be a permutation function. Let S'_1 and S'_2 be the strings after applying π on S_1 and S_2 respectively. Then it must be true that $jaccard(S_1, S_2) = jaccard(S'_1, S'_2)$.*

Proof. Let Σ be the alphabet. Since π is a bijection, for any S_1, $S_2 \in \Sigma$, $\pi(S_1) = \pi(S_2)$ if and only if $S_1 = S_2$. Therefore, for any q-gram $W \in G(S_1, q) \cap G(S_2, q)$, it must be true that $\pi(W) \in G(S'_1, q) \cap G(S'_2, q)$. Also for any q-gram $W \in G(S'_1, q) \cap G(S'_2, q)$, it must be true that $\pi^{-1}(W) \in G(S_1, q) \cap G(S_2, q)$. Therefore, $|G(S_1, q) \cap G(S_2, q)| = |G(S'_1, q) \cap G(S'_2, q)|$. Similarly, we can prove that $|G(S_1, q) \cup G(S_2, q)| = |G(S'_1, q) \cup G(S'_2, q)|$. Therefore, $jaccard(S'_1, S'_2) = jaccard(S_1, S_2)$.

Following Lemma 1, we have the following theorem to show that for any two (dis)similar strings, their permutations using the same permutation function are always (dis)similar. Indeed, this applies to not only Jaccard similarity measurement but also other distance metrics such as Hamming distance and edit distance.

Theorem 1. *Given any two strings S_1 and S_2 s.t. $S_1 \simeq S_2$ ($S_1 \not\simeq_{Eucl} S_2$ resp.). Let S_1' and S_2' be the strings after applying the permutation function π on S_1 and S_2 respectively. Then it must be true that $S_1' \simeq S_2'$ ($S_1' \not\simeq_{Eucl} S_2'$ resp.).*

Theorem 1 shows that the permutation-based split procedure will lead to neither *false negatives* (i.e., the split copies of two similar strings turn dissimilar) nor *false positives* (i.e., the split copies of two dissimilar strings become similar).

Next, we discuss how to decide the total number of split copies. Our goal is to ensure that the total amounts of duplicated LSH values by the SD approach is no larger than that of the duplication approach. First, we calculate the total number of the duplicated values that are to be added by the SD approach. For each group that consists of k strings $\{S_1, \ldots, S_k\}$, let f_i be the frequency of the string $S_i (1 \leq i \leq k)$. Assume that the string S_i is split into sc_i copies. Apparently, the frequency of each split copy of S_i is $[\frac{f_i}{sc_i}]$. Let $\overline{f_{max}}$ be the maximum frequency of the split copies in the group. Then for each string S_i, its number $\bar{f}_i$ of duplicated LSH values is

$$\bar{f}_i = sc_i(\overline{f_{max}} - [\frac{f_i}{sc_i}]). \tag{4.9}$$

To find out when the SD approach incurs smaller amounts of duplicate values than the duplication approach, we compare f_i' (Eqn. 4.7) and $\bar{f}_i$ (Eqn. 4.9). Apparently $\bar{f}_i = sc_i(\overline{f_{max}} - [\frac{f_i}{sc_i}]) = sc_i\overline{f_{max}} - f_i$. Compared with $f_i' = f_{max} - f_i$, the SD approach wins the duplication approach if $f_{max} > sc_i\overline{f_{max}}$. Note that the split copy of the string that has $\overline{f_{max}}$ may not be the same string of f_{max}. Therefore, for each string, we always pick sc_i, the number of split copies, in the way that ensures $f_{max} > sc_i\overline{f_{max}}$.

Privacy Analysis of SD Approach

Frequency Analysis Attack. The SD approach guarantees that for each LSH value, there are always at least $[\frac{1}{\alpha}] - 1$ other LSH values that are of the same frequency. Thus, it guarantees α-privacy against the frequency analysis attack.

Known-scheme Attack. If the attacker knows the details of the SD approach, he can launch the following two-step attack trying to decode the received LSH values. By the first step, the attacker tries to find mapping between string and LSH groups based on the frequency of strings and LSH values. From the analysis of the SD algorithm, he knows that all LSH values of the same frequency belong to the same group with high probability. Furthermore, for any string S and its split copy S', $freq(S) \geq freq(S')$. Apparently, for any string groups of k unique strings, its LSH groups must contain at least k unique LSH values. Furthermore, the maximum frequency of the LSH group must be no larger than the maximum frequency of its string group. The attacker can use these two facts to map LSH groups with string groups. In the worst case, there is only one such mapping. Then by the second step, the attacker tries to find the mapping between LSH values and strings. Consider a mapping that consists of k strings and $k' \geq k$ LSH values, in which LSH values are of the same frequency while the strings may have different frequency. The attacker tries to construct all possible mappings of k' LSH values to k strings. This is equivalent to the problem of distributing k' distinguishable balls into k distinguishable boxes where $k' > k$ and no box is empty. The number of such mapping equals:

$$M(k', k) = \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^{k'}. \tag{4.10}$$

Among these mappings, there are $M(k'-1,k)+M(k'-1,k-1)$ correct mappings between a LSH value $L_j(1 \leq j \leq k')$ and its string $S_i(1 \leq i \leq k)$. Therefore, the probability of finding the correct mapping $L_j \rightarrow S_i$ is

$$prob(L_j \rightarrow S_i) = \frac{M(k'-1,k)+M(k'-1,k-1)}{M(k',k)} = \frac{1}{k}. \tag{4.11}$$

Since $k \geq \frac{1}{\alpha}$, the SD approach can provide α-privacy against the known-scheme attack. Interestingly, varying the number of split copies will not impact the privacy guarantee.

Complexity Discussion

The SD approach requires $O(\overline{sc}n_S)$ to construct all split copies at the client side, where n_S is the number of unique strings in D, and $\overline{sc}$ is the average number of split copies of all strings. The complexity of the data deduplication at the server side is $O((|D|+n_d)^2)$, where $|D|$ is the total number of strings in D, and n_d is the total number of duplicates. Note that $|D| \gg n$. Thus the computational effort at the client side is much cheaper than that at the server side.

4.3 Embedding & Homophonic Substitution (EHS) Approach

In this section, we present a different approach called Embedding & Homophonic Substitution (EHS) that constructs the encoding of the input data without adding any additional data. In a nutshell, EHS has two steps: (1) conversion of categorical data to Euclidean space, and (2) 1-to-many homophonic substitution of Euclidean points. The EHS approach can be easily adapted to numerical data for which the conversion step is not necessary, while the substitution step will be applied on the

original data.

For the first step, many mapping techniques (e.g., $FastMap$ [47] and $StringMap$ [68]) can be used to map strings into a multidimensional Euclidean space. The mapping functions guarantee that the similar strings are mapped to close Euclidean points. In this paper, we use $FastMap$ [47] for string conversion. By $FastMap$, the string similarity threshold δ will be adjusted to a new similarity threshold δ_E for Euclidean points. $FastMap$ is not reversible [47], thus the original strings cannot be reconstructed from the embedded Euclidean points. The complexity of the embedding step is $O(d^2|D|)$, where d is the number of dimensions of the Euclidean points, and $|D|$ is the number of strings in D.

$FastMap$ embedding is one-to-one, simply mapping strings to Euclidean points fails to defend against the frequency analysis attack. Therefore, the second step of EHS is to apply a homophonic substitution encoding scheme on the Euclidean points to defend against the frequency analysis attack. Homophonic substitution schemes are one-to-many, meaning that multiple encoded symbols can map to one plaintext symbol. Next, we will first discuss our basic homophonic substitution encoding scheme and its weakness against both the frequency analysis and the known-scheme attacks (Section 4.3.1). We then present our grouping-based homophonic substitution (GHS) scheme that can defend against these attacks with provable guarantee (Section 4.3.2).

4.3.1 Basic Approach

By the basic approach, any Euclidean point P of frequency f is transformed to f unique points $E_1, \ldots, E_f$ in the same Euclidean space, each point of frequency 1. Therefore, the frequency distribution of the Euclidean points is always uniform, re-

gardless of the frequency distribution of the original data. The substitution scheme is one-to-many as one string maps to multiple Euclidean points. To preserve similarity, for each point P in the d-dimension Euclidean space, we construct a d-sphere H_P^R of radius R centered at P, where R is a user-specified parameter. Let f be the frequency of P. Then we pick f points from H_P^R in a uniformly random manner, with each point of frequency 1. These f points are the encoded versions of P. The distance between the constructed Euclidean points satisfies the following theorem.

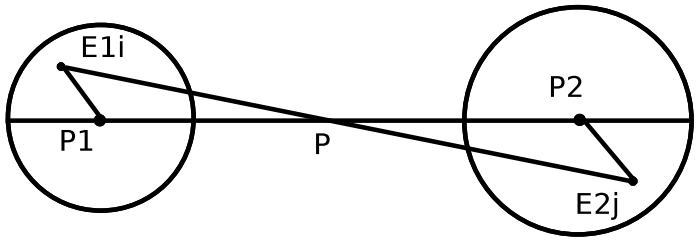


Figure 4.1: Illustration of 1-to-many Substitution Encoding

Theorem 2. *Given any two original Euclidean points P_1 and P_2, let $\{E_1^1, \ldots, E_1^{f_1}\}$ and $\{E_2^1, \ldots, E_2^{f_2}\}$ be the Euclidean points of P_1 and P_2 constructed by the above procedure (f_1 and f_2 are not necessarily the same). Then for any pair of Euclidean points E_1^i and E_2^j ($i \in [1, f_1], j \in [1, f_2]$), $dist(E_1^i, E_2^j) \leq dist(P_1, P_2) + R_1 + R_2$, where $dist()$ is the Euclidean distance function, and R_1, R_2 are the length of the radius of the corresponding d-spheres of P_1 and P_2.*

Proof. Given any two points P_1 and P_2, we pick a point P between these two points. Figure 4.1 illustrates the positions of P_1, P_2 and P. Apparently, it must be true that $dist(P_1, E_1^i) \leq R_1 (1 \leq i \leq f_1)$. Similarly, $dist(P_2, E_2^j) \leq R_2$ $(1 \leq j \leq f_2)$. According to triangle inequality,

$$dist(E_1^i, P) \leq dist(P_1, E_1^i) + dist(P_1, P) \leq R_1 + dist(P_1, P), \tag{4.12}$$

and

$$dist(E_2^j, P) \leq dist(P_2, E_2^j) + dist(P_2, P) \leq R_2 + dist(P_2, P). \tag{4.13}$$

It is easy to infer that

$$\begin{aligned} dist(E_1^i, E_2^j) &= dist(E_1^i, P) + dist(E_2^j, P) \\ &\leq R_1 + R_2 + dist(P_1, P) + dist(P_2, P) \\ &= R_1 + R_2 + dist(P_1, P_2). \end{aligned} \tag{4.14}$$

Based on Theorem 2, we adjust the similarity threshold of the encoded Euclidean points accordingly. We say two encoded values E_i and E_j are similar if $dist(E_i, E_j) \leq \delta_E + R_i + R_j$, where δ_E is the distance threshold after embedding, R_i and R_j are the length of the radius of spheres of P_i and P_j, the corresponding original Euclidean point of E_i and E_j.

Privacy Analysis

Frequency Analysis Attack. Given n_S strings, each of frequency f_i, and n_E encoded Euclidean points, each of frequency f ($f = 1$ for the basic approach), we define $sf = \sum_{i=1}^{n} f_i$. With further computation, the probability $prob_F$ that the encoded point E_j is mapped to its string S_i by the frequency analysis attack is

$$prob_F(E_j \rightarrow S_i) = \frac{\binom{f_i}{f}}{\binom{sf}{f}}. \tag{4.15}$$

As an example, consider two strings S_1 and S_2 of frequency 10000 and 10. These two strings are encoded as 10000 and 10 Euclidean points respectively by the basic approach. Then for any encoded Euclidean point E_j of S_1, $prob_F(E_j \rightarrow S_1) =$

$\binom{10000}{1}/\binom{10010}{1} = \frac{10000}{10010} \approx 1$.

Known-scheme Attack. If the attacker possesses the details of the basic approach, he knows that all Euclidean points of the same string must fall in a circle of a small radius in the Euclidean space. Given n_S unique strings and $n_E \geq n_S$ Euclidean points, he can apply the following 2-step attack to find the mapping between the Euclidean points and the strings: By the first step, it uses a clustering algorithm (e.g., k-means clustering) to group n_E Euclidean points to n_S groups based on their closeness, so that the groups are of low intra-group distances and high inter-group distances. By the second step, it maps n_S clusters of Euclidean points to n_S unique strings based on their similarities. In particular, the attacker maps the n_S cluster centroids to n strings in the way that for any three centroids ct_i, ct_j and ct_k, their corresponding strings S_i, S_j and S_k satisfy that if $dist(ct_i, ct_j) \leq dist(ct_i, ct_k)$, then $jaccard(S_i, S_j) \geq jaccard(S_i, S_k)$. After the two-step attack, some Euclidean points of the same string may be covered by multiple clusters (i.e., mapped to multiple strings). Given n_S unique strings and $n_E \geq n_S$ Euclidean points by the basic approach, the probability $prob_S$ that the encoded point E_j is mapped to its corresponding string S_i by the known-scheme attack is

$$prob_S(E_j \rightarrow S_i) = \frac{1}{t}, \tag{4.16}$$

where t is the number of clusters that include E_j. If $t = 1$, the mapping $E_j \rightarrow S_i$ is broken with 100% certainty.

We simulate the known-scheme attack on the 1990 Census dataset of frequently occurring surnames[1]. We used the female first name data set (4000 names) and applied the basic approach on them. Then we executed the known-

[1] http://www.census.gov/genealogy/www/data/1990surnames /names_files.html.

scheme attack on the encoded data. Figure 4.2 illustrates the clustering result of the Euclidean points of five names. The Euclidean points of the same color and the same shape belong to the same string. We observe that some Euclidean points (e.g., the red square ones) only belong to one cluster. These Euclidean points are mapped to their string with 100% probability.

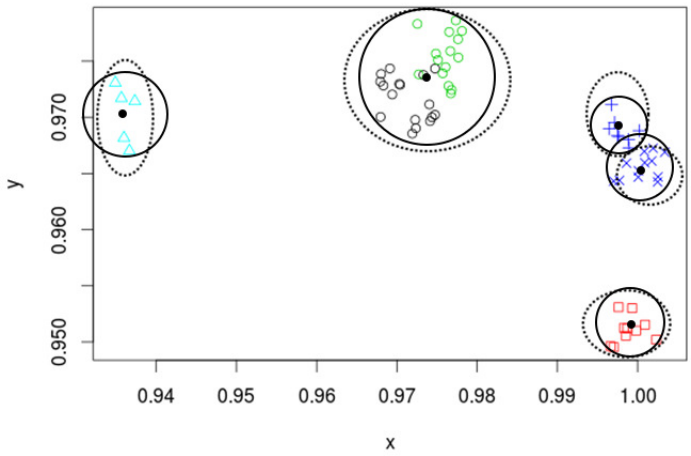


Figure 4.2: Simulation of similarity-based known-scheme attack
(dotted lines represent the real clusters, while solid lines represent the clusters constructed by the known-scheme attack)

Now we are ready to define the privacy model that can defend against both the frequency analysis attack and the known-scheme attack.

Definition 3. [(α_1, α_2)-privacy] Given n_S strings $\mathcal{S}$ $\{S_1, \ldots, S_{n_S}\}$ and n_E encoded Euclidean points $\mathcal{E}$ $\{E_1, \ldots, E_{n_E}\}$, we say $\mathcal{E}$ satisfies (α_1, α_2)-*privacy* if for any Euclidean point $E_i \in \mathcal{E}$ (whose corresponding string is $S_j \in \mathcal{S}$), $prob_F(E_j \rightarrow S_i) \leq \alpha_1$, and $prob_S(P_i \rightarrow S_i) \leq \alpha_2$, where $prob_F$ and $prob_S$ are measured by Eqn. 4.15 and Eqn. 4.16 respectively.

As we have shown, the basic approach may fail to meet the (α_1, α_2)-privacy requirement.

4.3.2 Grouping-based Homophonic Substitution (GHS) Scheme

To address the privacy weakness of the basic approach, we design the grouping-based homophonic substitution encoding scheme (GHS) that satisfies (α_1, α_2)-privacy. The GHS takes n_S Euclidean points that are transformed from the original n_S strings and outputs n_E ($n_E \geq n_S$) transformed Euclidean points, each of frequency 1. It satisfies that $\sum_{i=1}^{n_S} f_i = n_E$, where $\sum_{i=1}^{n} f_i$ is the sum of the frequency of all Euclidean points. Before we explain the details of GHS method, we first define (α_1, α_2)-*bounded clusters*.

Definition 4. [(α_1, α_2)-bounded clusters] For any given cluster CT and two user-specified thresholds α_1, α_2, CT is (α_1, α_2)-*bounded* if it satisfies: (1) for each point $P \in CT$, $\frac{f_P}{\sum_{Q \in CT} f_Q} \leq \alpha_1$, where f_P (f_Q, resp.) is the frequency of point P (point Q, resp); and (2) CT contains at least $[\frac{1}{\alpha_2}]$ points.

The GHS procedure consists of 2 steps. First, the Euclidean points are grouped into (α_1, α_2)-bounded clusters. For each cluster CT, a d-sphere H is constructed to cover all points of CT, where d is the dimension of the Euclidean space. Second, for each sphere H that covers the points $\{P_1, \cdots, P_k\}$, $f_1 + \cdots + f_k$ points in total are constructed within H in a uniform random fashion, where f_i ($1 \leq i \leq k$) is the frequency of the point P_i. Each constructed point is of frequency 1. Theorem 3 below shows how to update the similarity threshold for the Euclidean points constructed by the GHS approach.

Theorem 3. *Given any two original Euclidean points P_1 and P_2 that are similar according to the threshold δ', let $\{E_1^1, \ldots, E_1^{f_1}\}$ and $\{E_2^1, \ldots, E_2^{f_2}\}$ be the Euclidean points of P_1 and P_2 constructed by the above procedure (f_1 and f_2 are not necessarily the same), and R_1 and R_2 be the top-2 maximum radius length of all*

spheres. Then for any pair of Euclidean points E_1^i *and* E_2^j *(*$i \in [1, f_1], j \in [1, f_2]$*),* $dist(E_1^i, E_2^j) \leq \delta' + 2R_1 + 2R_2$*, where* $dist()$ *is the Euclidean distance function.*

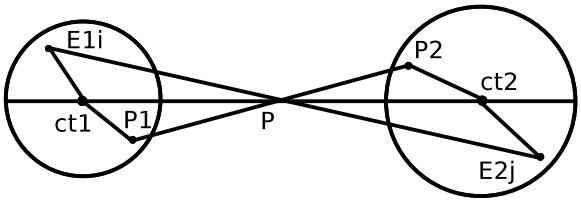


Figure 4.3: Similarity Preservation of Euclidean Points

Proof. Given any two Euclidean points P_1 and P_2, assume that they are in the different clusters whose centers are ct_1 and ct_2. Let P be the intersection point of the line (P_1, P_2) and (ct_1, ct_2). By triangle inequality,

$$dist(P_1, P) + dist(ct_1, P_1) \geq dist(ct_1, P), \tag{4.17}$$

and

$$dist(P_2, P) + dist(ct_2, P_2) \geq dist(ct_2, P). \tag{4.18}$$

Combining Equation 4.17 and 4.18 we have:

$$\begin{aligned} dist(P_1, P_2) &= dist(P_1, P) + dist(P_2, P) \\ &\geq dist(ct_1, P) + dist(ct_2, P) - dist(ct_1, P_1) - dist(ct_2, P_2) \\ &\geq dist(ct_1, ct_2) - R_1 - R_2. \end{aligned} \tag{4.19}$$

Since $dist(ct_1, P_1) \geq R_1$ and $dist(ct_2, P_2) \geq R_2$,

$$dist(ct_1, ct_2) \leq dist(P_1, P_2) + R_1 + R_2. \tag{4.20}$$

On the other hand, for any points E_1^i constructed for P_1 and any points E_2^j constructed for P_2, we have

$$dist(E_1^i, P) \leq dist(ct_1, P) + dist(ct_1, E_1^i), \tag{4.21}$$

and

$$dist(E_2^j, P) \leq dist(ct_2, P) + dist(ct_2, E_2^j). \tag{4.22}$$

Combining Equation 4.21 and 4.22 we have:

$$dist(E_1^i, E_2^j) \leq dist(ct_1, ct_2) + dist(ct_1, E_1^i) + dist(ct_1, E_2^j). \tag{4.23}$$

Given the fact that $dist(ct_1, E_1^i) \leq R_1$, $dist(ct_2, E_2^j) \leq R_2$, and $dist(P_1, P_2) \leq \delta'$, we have:

$$\begin{aligned} dist(E_1^i, E_2^j) &\leq dist(ct_1, ct_2) + dist(ct_1, E_1^i) + dist(ct_2, E_2^j) \\ &\leq dist(P_1, P_2) + R_1 + R_2 + R_1 + R_2 \\ &\leq \delta' + 2R_1 + 2R_2. \end{aligned} \tag{4.24}$$

Note that different from Theorem 2 that considers the distance between the centers of spheres, Theorem 3 considers the distance between any two points within the spheres. Following Theorem 3, the new similarity threshold for the points constructed by GHS approach is

$$\delta_{new} = \delta_E + 2R_1 + 2R_2. \tag{4.25}$$

Next, we discuss how to construct (α_1, α_2)-bounded clusters.

Our algorithm consists of the following steps. First, each point is initialized as a cluster. Second, for each cluster CT_i that is not (α_1, α_2)-bounded, we look for other cluster(s) to merge with C_i. In particular, for each cluster candidate CT_j, we compute the radius length of the sphere by merging CT_j with CT_i, and pick the candidate that leads to the smallest radius length. We design an efficient algorithm that computes the *approximate* radius length in $O(1)$ time. In particular, given two clusters CT_i, CT_j of centers ct_i and ct_i respectively, let R_i and R_j be the radius length of the spheres of CT_i and CT_j, and $ct_i[z]$ be the value of the z-th dimension of the vector ct_i. The new center ct_{new} of the sphere that covers all points in $CT_i \cup CT_j$ is computed as

$$ct_{new}[z] = \frac{ct_i[z]st_i + ct_j[z]st_j}{st_i + st_j}, \forall z \in [1, d], \tag{4.26}$$

where st_i and st_j are the number of points of CT_i and CT_j, and d is the dimension of the Euclidean space. Then the approximate radius length of the sphere that covers all points in $CT_i \cup CT_j$ is computed as

$$R_{new} = max(dist(ct_i, ct_{new}) + R_i, dist(ct_j, ct_{new}) + R_j). \tag{4.27}$$

We look for the cluster CT_j whose merge with CT_i will lead to the smallest R_{new}. After that, we check whether $CT_i \cup CT_j$ is a (α_1, α_2)-bounded cluster. If not, we repeat the merging procedure, until all clusters are (α_1, α_2)-bounded.

Privacy analysis

The GHS approach provides (α_1, α_2)-privacy against the frequency analysis attack, as each point belongs to a (α_1, α_2)-cluster. Next, we discuss the robustness against

the known-scheme attack. If the attacker knows the details of the GHS method, he can group the received Euclidean points into clusters of high intra-cluster similarity. Next, he tries to map Euclidean points to the strings based on their frequency. In particular, a cluster of f points is highly likely to be mapped to the strings $\{S_1, \ldots, S_t\}$ if $\sum_{i=1}^{t} f_i = f$. It is possible that the attacker can find a unique mapping between the clusters and the strings. After that, for each cluster CT_i that consists of st Euclidean points $\{E_1, \ldots, E_{st}\}$ (each of frequency $f = 1$), assume CT_i is mapped to $k \leq st$ strings $\{S_1, \ldots, S_k\}$. He can apply the frequency analysis attack (Section 4.3.1) on CT_i. Since each cluster is (α_1, α_2)-bounded, the probability $prob_F$ that the encoded point E_j is mapped to its corresponding string S_i by the frequency analysis attack equals to $prob_F(E_j \rightarrow S_i) = \binom{f_i}{f} / \binom{fn_E}{f} = \frac{f_i}{n_E f} \leq \alpha_1$. The attacker also computes the probability of mapping each Euclidean point to a specific string by trying all possibilities. The reasoning of the attack probability is similar to the attack as described in Section 4.2.2, where $M(st, k) = \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^{st}$, and $prob_S(E_j \rightarrow S_i) = \frac{M(st-1,k)+M(st-1,k-1)}{M(st,k)} = \frac{1}{k}$. Since $k \geq [\frac{1}{\alpha_2}]$, the probability $prob_S \leq \alpha_2$. Therefore, the GHS approach can provide (α_1, α_2)-privacy guarantee.

Complexity analysis

The complexity of constructing (α_1, α_2)-bounded clusters is $O(n * n_{iter})$, where n is the number of Euclidean points, and n_{iter} is the number of required iterations. Our empirical study shows that n_{iter} is normally a small portion of n (at most half of n). The complexity of constructing the spheres for each cluster is $O(1)$, and the complexity of constructing all transformed Euclidean points is $O(n)$. Thus the total complexity of GHS is $O(n * n_{iter})$. The complexity of data deduplication at the server side is $O(|D|^2)$, where $|D|$ is the total number of strings of D. Since identical string

values are always mapped to the same Euclidean point, $|D| \gg n$. Therefore, the computations at the client side are much cheaper than that at the server side.

Improvement of Precision

Our GHS approach guarantees a 100% recall, but a possibly low precision. The low precision comes from the fact that the radius of (α_1, α_2)-bounded clusters may be much larger than the original Euclidean similarity threshold δ_E. This leads to a too relaxed similarity threshold and thus large amounts of false positives. A possible solution to bound the precision that GHS is to control the radius of the clusters. In particular, given the set of Euclidean points $\mathcal{P}$ and a set of clusters $\mathcal{CT}$ that is constructed from $\mathcal{P}$ by our GHS approach, we say $\mathcal{CT}$ provides γ-*precision* if the precision of $\mathcal{CT}$ is never below γ. We propose the following procedure to find γ-precision clusters. First, we pick a subset of points from $\mathcal{P}$ as the sample, and measure the pair-wise distance between all pairs of the sample. The sample's size is normally much smaller than that of $\mathcal{P}$; thus the computational efforts for pairwise distance measurement at the client side is acceptable. Second, we sort the pairwise distances in the ascending order. From the sorted list, we find the index of the distance that is smaller than δ_E. Let the found index be j. Then, we calculate $k = [\frac{1-\gamma}{\gamma} j]$, and find the distance dst at the $(j+k)^{th}$ location in the sorted list. We use dst as the new similarity threshold δ_{new}. Third, we group all points in $\mathcal{P}$ into clusters whose radius R satisfies that $R \leq (dst - \delta_E)/4$. Though those clusters can provide bounded precision loss, they may not satisfy the (α_1, α_2)-bounded requirement. There exists the trade-off between the precision and privacy. High privacy may lead to large loss of precision.

4.4 Post-Processing

After the server returns the near-duplicates, the client maps the received results to strings, then eliminates the false positives by measuring the similarity of all near-duplicate pairs. Assume that the similarity of each near-duplicate pair can be measured in constant time. Then the complexity of the post-processing is $O(|\hat{R}^S|)$, where $|\hat{R}^S|$ is the number of near-duplicates that are returned by the server. Compared with the complexity of data deduplication of the original dataset D (quadratic to $|D|$), the post-processing is much cheaper than running the data deduplication locally (note that $|\hat{R}^S| \ll |D|$).

4.5 Experiments

4.5.1 Setup

Experiment Environment. We implemented both of our LSHB and EHS approaches in Java. All experiments were executed on a PC with a 2.4GHz Intel Core i5 CPU and 8GB memory running Linux.

Datasets. We use two real datasets[2] from US Census Bureau in our experiments: (1) the *female first name* dataset (*FEMALE* for short) that contains 4275 unique female first names and their frequencies, and (2) the *last name* dataset (*LAST* for short) that contains 88799 last names and their frequencies. We also use subsets of *LAST* dataset of various sizes for scalability measurement. To study the impact of pairwise distance distribution to the performance, we also prepared two types of datasets: (1) the *thick* dataset in which a large potion of strings are similar to each other; and (2) the *sparse* dataset in which most strings are dissimilar.

[2]available at `http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html`.

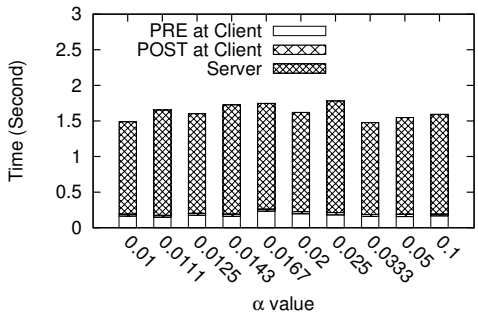
Parameters. We use $q = 2$ for q-gram setting and set the jaccard similarity threshold to be 0.4 for both *FEMALE* and *LAST* datasets. For LSHB approach, we apply 100 random permutations to generate the MinHash signatures.

4.5.2 Performance of LSHB Approach

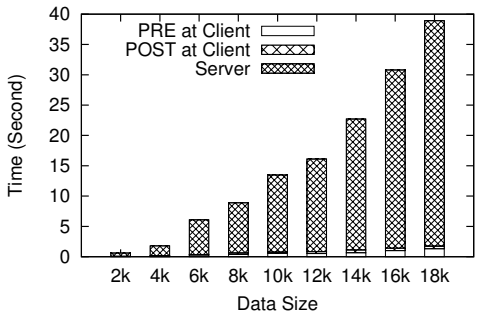
Time performance

First, we measure the impact of various α values (for α-privacy) to the time performance of LSHB approach (Figure 4.4 (a)). We measure preparation time at the client side, data deduplication time at the server side, and post-processing time at the client side. First, the preparation time at the client side is stable. This is because the complexity of encoding is decided by the number of unique strings, which is not affected by α value. The post-processing time at client side is also stable, because different α values do not change the number of similar string pairs. Second, the time performance at the server side is also stable. The reason is the complexity at the server side only relies on the number of unique LSH values (and thus the number of unique strings), thus changing α values does not affect the running time at server side. Third, the computational efforts at the server side dominate the whole data deduplication process (including both at the client and the server side). This observation supports our claim that the LSHB approach suits the outsourcing paradigm very well.

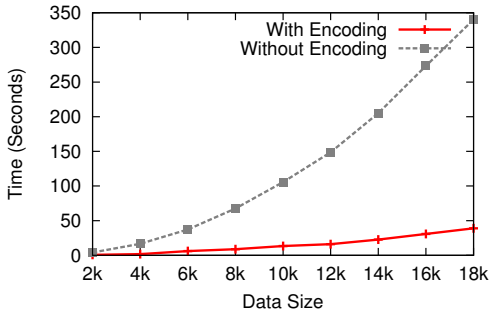
Second, we measure the scalability of LSHB approach on datasets of various sizes (Figure 4.4 (b)). We observe that the preparation time at the client side increases slowly with the growth of the data size, while the post-processing time at the client side and the time performance at the server side increase sharply. This is because the complexity of preparing LSH values at the client side is linear



(a) Various α values (*FEMALE* dataset)



(b) Various data sizes (*LAST* dataset)

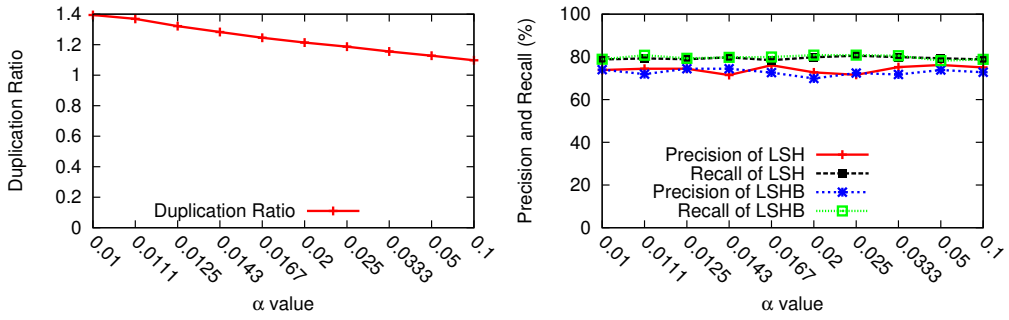


(c) Time overhead by encoding (*LAST* dataset)

Figure 4.4: Time Performance of LSHB Approach

to the data size, while the complexity of deduplication and data post-processing is quadratic to data size. Second, compared to the time performance at the server side, the post-processing time at the client side is very small, as the server has filtered many dissimilar string pairs. We also observe that the time performance of preparation at the client side is much cheaper than that of the server side. Indeed, the LSH construction time at client side never exceeds 2 seconds. We also observe that the difference between the performance at the client and the server sides becomes more significant with the growth of data size. This is because the complexity of preparation at the client side is linear to the data size, while the complexity at the server side is quadratic to the data size. This proves the advantage of the outsourcing paradigm that handles large datasets most of the time.

Third, we measure the impact of LSHB encoding to the time performance



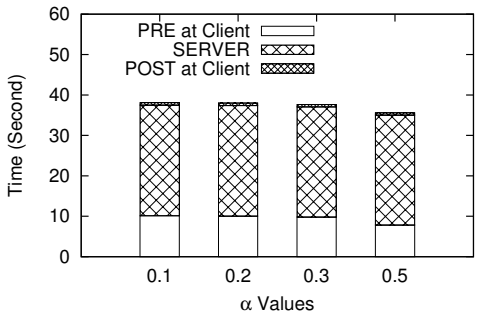
(a) Amounts of added copies (b) Precision & recall

Figure 4.5: Impact of LSHB Encoding to Accuracy

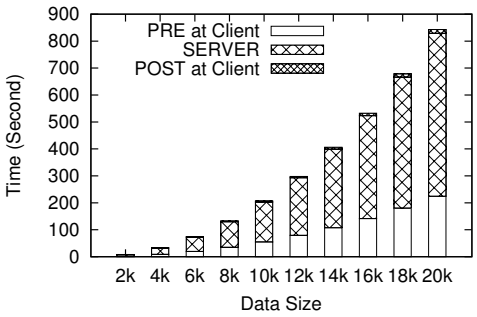
of data deduplication. We compare the time of the LSHB approach (including the time at the client and the server side) with the time of the data deduplication on the original data (Figure 4.4 (c)). We observe that the time performance of both scenarios increases with the growth of data size. However, the time performance of the LSHB encoding is always cheaper than that of the original data. This speed-up is mainly due to the use of LSH values. This shows that although the privacy-preserving methods may bring computational overhead in general, picking a right encoding method (e.g., LSHs) may enable faster data deduplication than the brute force approach of measuring the pairwise similarity of all string pairs.

Impact to Accuracy of Data Deduplication

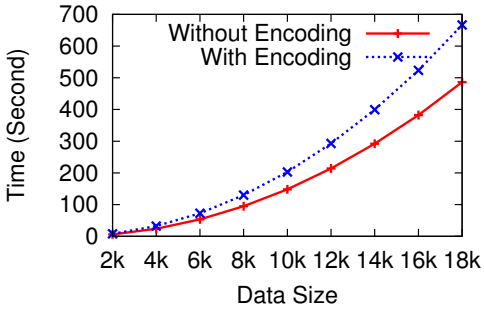
Storage Overhead. We measure the amounts of the added LSH copies by using *duplication ratio*. We define the duplication ratio as $\frac{|\hat{D}|}{|D|}$, where $|D|$ and $|\hat{D}|$ are the number of strings in original D and after applying LSHB respectively. In Figure 4.5 (a), the ratio increases with the decrease of α (i.e., higher privacy requirement). The reason is that smaller α requires larger groups, and thus more inserted copies to make their frequency homogenized. This is the price we need to pay for higher privacy.



(a) Various α values ($FEMALE$ dataset)



(b) Various data sizes ($LAST$ dataset)



(c) Time overhead by encoding ($LAST$ dataset)

Figure 4.6: Time Performance of EHS Approach

Precision/Recall We compare the precision/recall of applying our LSHB approach with the precision/recall of directly applying LSH hashing over the original data (Figure 4.5 (b)). In all the experiments, the precision varies around 75%, and the recall is around 80%. Our LSHB approach is able to provide comparable accuracy to LSH. This convinces us the good utility of the LSHB approach. We note that after post-processing, we can ensure 100% precision.

4.5.3 Performance of EHS Approach

Time performance

First, we measure the time performance of EHS encodings for various α_1 and α_2 settings of (α_1, α_2)-privacy (Figure 4.6 (a)). We use the similarity threshold $\delta_E = 0.2$

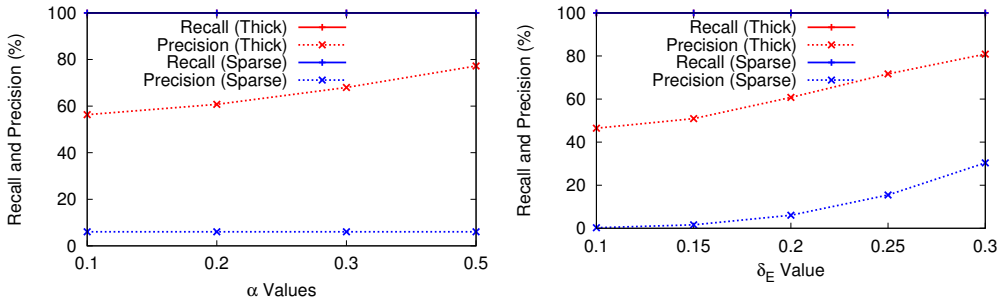
and various α values ($\alpha = \alpha_1 = \alpha_2$). We measure: (1) the preparation time at the client side, (2) the data deduplication time at the server side, and (3) the post-processing time at the client side. First, we observe that the preparation time decreases when α increases. The reason is that larger α value requires smaller clusters and points of lower frequency, leading to less merge during the (α_1, α_2)-bounded cluster. Second, the data deduplication time is stable for all α values, because the complexity at the server side is decided by the dataset size, which is unchanged for these settings. Third, the post-processing time decreases when α increases. That is when α value is larger, the clusters get smaller, which leads to more tightened similarity threshold and thus fewer false positives. For all experiments, the post-processing time is always less than 0.6 seconds. Similar to LSHB approach, we observe that the server's computation efforts are much higher than that of the client.

Second, we measure the scalability of EHS approach for various data sizes. As shown in Figure 4.6 (b), the preparation time increases with the growth of the data size. That is because larger dataset requires more time to construct (α_1, α_2)-bounded clusters. On the other hand, the post-processing time increases because the size of $\hat{R^S}$ increases with the size of data. We also compare the time performance at both the client and the server side. We observe that the client requires much less time than the server. When the size of the dataset increases, the running time at the server side grows faster than the client side. This observation shows that our EHS approach suits the outsourcing paradigm well.

Third, we compare the time performance of data deduplication by our EHS encoding with that on the original data (Figure 4.6 (c)). For EHS encoding, we add up the preparation time, the post-processing time, and the data deduplication time. We observe that the processing time of both cases increases with the size of the

data. However, applying EHS encoding only adds a small potion of time overhead compared with the performance of the original data.

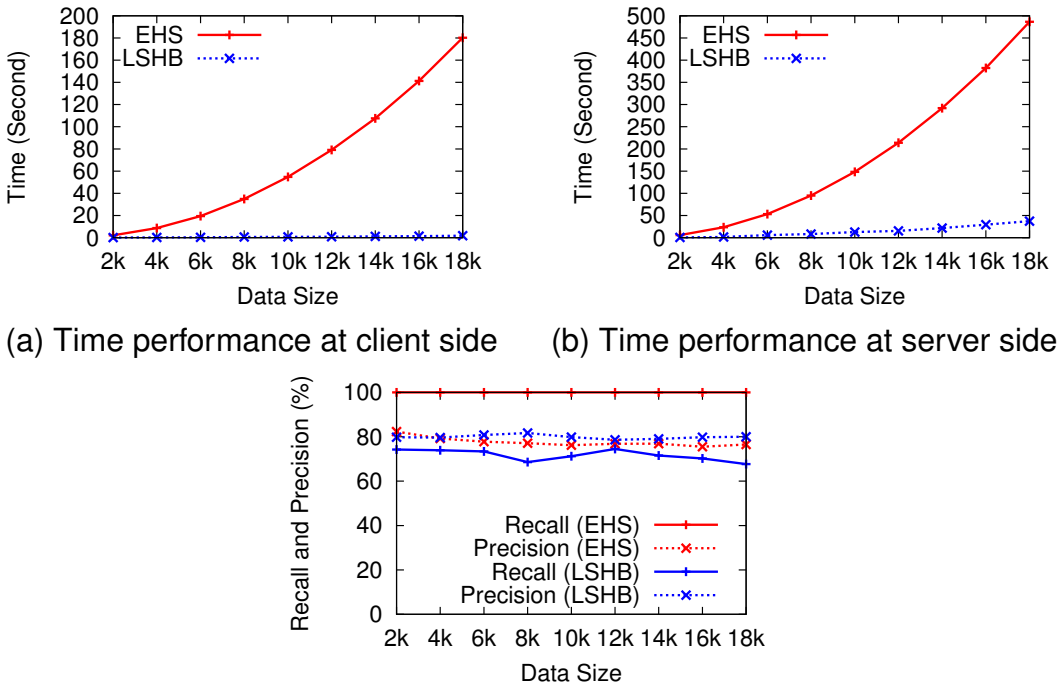
Impact to Accuracy of Data Deduplication



(a) Various α values ($\delta_E = 0.2$) (b) Various δ_E values ($\alpha = 0.2$)

Figure 4.7: Impact of EHS Encoding to Accuracy of Deduplication (*FEMALE* dataset, $d = 10$)

We measure the precision and recall of our EHS approach for various α values ($\alpha = \alpha_1 = \alpha_2$) and various δ_E values (Figure 4.7). First, we observe that recall is always 100%. In other words, all the near-duplicates in the original dataset are returned. Second, from Figure 4.7 (a), we observe that precision increases with α value. This is because larger α value leads to smaller clusters, which yield more tightened δ_{new}. One interesting observation is, the precision on the *sparse* dataset increases much slower than that of the *thick* dataset. That is because the δ_{new} value of the *sparse* dataset is large; thus the change of δ_{new} (from 1.00 to 0.89) makes less impact than the change of δ_{new} on the *thick* dataset (from 0.23 to 0.21). As shown in Figure 4.7 (b), the precision raises with δ_E, because large δ_E results in more similar pairs. Thus, the ratio of false positive decreases. In all experiments, we observe that the precision of the *thick* dataset (at least 0.4) is much better than the *sparse* dataset. To understand why, we measured the average pair-wise



(a) Time performance at client side

(b) Time performance at server side

(c) Precision and recall

Figure 4.8: Comparison of LSHB Approach and EHS Approach (*LAST* dataset, $r = 0.4$)

distances of both datasets. It turned out that the average pair-wise distance of the *thick* and the *sparse* datasets are 0.27 and 0.31 respectively. We use 0.2 as the value of δ_E, and measure the new similarity threshold δ_{new}. For the *thick* dataset, δ_{new} is less than 0.23, which is less than 15% change of the similarity threshold. While for the *sparse* dataset, δ_{new} is changed to be between 0.9 and 1, which is too loose compared with the original threshold and thus concludes almost all pairs as similar. Therefore, the (α_1, α_2)-bounded clustering method introduces fewer false positives on the *thick* dataset than the *sparse* dataset.

4.5.4 LSHB Versus EHS

First, we compare the time performance of encoding at client side for both approaches (Figure 4.8 (a)). We observe that the LSHB approach is much faster than the EHS approach. This is because constructing $LSHs$ is much faster than the $FastMap$ embedding and (α_1, α_2)-bounded clustering. Second, we compare the time performance of data deduplication at the server side. The result (Figure 4.8 (b)) shows that although LSHB adds additional values to the outsourced dataset, it is again faster than EHS, due to the fact that dealing with LSH values is much faster than the computation of Euclidean distances. Third, we compare the precision and recall of the two approaches (Figure 4.8 (c)). We observe that LSHB yields comparable precision with EHS but EHS yields much higher recall (100%). In summary, there exists the trade-off between the time performance of the privacy-preserving encoding methods and the accuracy of data deduplication on encoded data. The use of LSH values enables faster time performance at both the client and the server sides, but it introduces lower recall rate, while EHS guarantees 100% recall.

Chapter 5
Efficient Authentication of Outsourced Data Deduplication

In this chapter, we discuss the authentication of outsourced data deduplication. Record matching, which targets on finding similar and near-duplicate record pairs from a dataset, is the fundamental computation for data deduplication. Without loss of generality, we focus on the authentication of outsourced record matching. We design two efficient authentication approaches named VS^2 and E-VS^2 to verify the integrity of record matching results returned by an untrusted server. The key idea is that besides returning the matching records, the server also constructs *verification objects (VOs)* to demonstrate the correctness of the result. This work has been published in the proceedings of IEEE International Conference on Information Reuse and Integration (IRI), July 2016.

5.1 Preliminaries

5.1.1 Record Similarity Measurement

Record matching is a fundamental problem in many research areas, e.g., information integration, database joins, and more. In general, the evaluation that whether two records match is mainly based on the string similarity of the attribute values. There are a number of string similarity functions, e.g., Hamming distance, n-grams, and edit distance (see [72] for a good tutorial). In this thesis, we mainly consider *edit distance*, one of the most popular string similarity measurements that have been used in a wide spectrum of applications. Informally, the edit distance of two strings s_1 and s_2, denoted as $DST(s_1, s_2)$, measures the minimum number of inser-

tion, deletion and substitution operations to transform s_1 to s_2. We say two strings s_1 and s_2 are *similar*, denoted as $s_1 \simeq_E s_2$, if $DST(s_1, s_2) \leq \theta$, where θ is a user-specified similarity threshold. Otherwise, we say s_1 and s_2 are *dissimilar* (denoted as $s_1 \not\simeq_E s_2$). Without loss of generality, we assume that the dataset D only contains a single attribute. Thus, in the following sections, we use record and string interchangeably. Our methods can be easily adapted to the datasets that have multiple attributes.

5.1.2 Authenticated Data Structure

To enable the client to authenticate the correctness of mining results, the server returns the results along with some supplementary information that permits result verification. Normally the supplementary information takes the format of *verification object* (VO). In the literature, VO generation is usually performed by an authenticated data structure (e.g., [76, 131, 96]). One of the popular authenticated data structures is *Merkle tree* [104]. In particular, a Merkle tree is a tree T in which each leaf node N stores the digest of a record r: $h_N = h(r)$, where $h()$ is a one-way, collision-resistant hash function (e.g. SHA-1). For each non-leaf node N of T, it is assigned the value $h_N = h(h_{C_1}|| \ldots ||h_{C_k})$, where $C_1, \ldots, C_k$ are the children of N. The root signature sig is generated by signing the digest h_{root} of the root node using the private key of a trusted party (e.g., the data owner). The VO enables the client to re-construct the root hash value.

5.1.3 B^{ed}-Tree for String Similarity Search

A number of compact data structures (e.g., [6, 19, 75]) are designed to handle edit distance based similarity measurement. In this thesis, we consider B^{ed}-*tree* [134]

due to its support for external memory and dynamic data updates. B^{ed}-tree is a B^+-tree based index structure that can handle arbitrary edit distance thresholds. The tree is built upon a *string ordering* scheme which is a mapping function φ to map each string to an integer value. To simplify notation, we say that $s \in [s_i, s_j]$ if $\varphi(s_i) \leq \varphi(s) \leq \varphi(s_j)$. Based on the string ordering, each B^{ed}-tree node N is associated with a string range $[N_b, N_e]$. Each leaf node contains $f \geq 1$ strings $\{s_1, \ldots, s_f\}$, where $s_i \in [N_b, N_e]$, for each $i \in [1, f]$. Each intermediate node N (with range $[N_b, N_e]$) contains multiple children nodes, where for each child, its range $[N'_b, N'_e] \subseteq [N_b, N_e]$. We say these strings that stored in the sub-tree rooted at N as the strings that are *covered* by N.

We use $DST_{min}(s_q, N)$ to denote the *minimal* edit distance between a string s_q and any string s that is covered by a B^{ed}-tree node N. A nice property of the B^{ed}-tree is that, for any string s_q and node N, the string ordering φ enables to compute $DST_{min}(s_q, N)$ efficiently by computing $DST(s_q, N_b)$ and $DST(s_q, N_e)$ only, where N_b, N_e refer to the string range values of N. Based on this, we define the *B^{ed}-tree candidate node*.

Definition 5. Given a string s_q and a similarity threshold θ, a B^{ed}-tree node N is a *candidate* if $DST_{min}(s_q, N) \leq \theta$. Otherwise, N is a *non-candidate*. □

B^{ed}-tree has an important monotone property.

Property 5.1.1. Monotone Property of B^{ed}-tree : *Given a node N_i in the B^{ed}-tree and any child node N_j of N_i, for any string s_q, it must be true that $DST_{min}(s_q, N_i) \leq DST_{min}(s_q, N_j)$. Therefore, for any non-candidate node, all of its children must be non-candidates. This monotone property enables early termination of search on the branches that contain non-candidate nodes.*

For any given string s_q, the string similarity search algorithm starts from the root of the B^{ed}-tree, and iteratively visits the candidate nodes, until all candidate nodes are visited. The algorithm does not visit non-candidate nodes as well as their descendants. An important note is that for each candidate node, some of its covered strings may still be dissimilar to s_q. Therefore, given a string s_q and a candidate B^{ed}-tree node N, it is necessary to compute $DST(s_q, s)$, for each s that is covered by N.

5.1.4 Security Model

To catch the cheating on the matching results, we formally define the integrity of the record matching result from two perspective: *soundness* and *completeness*. Let M be the set of matching pairs in the outsourced database D, and M^S be the result returned by the server. We define the *precision* of M^S as $pre = \frac{|M \cap M^S|}{|M^S|}$ (i.e., the percentage of returned pairs that are similar to each other), and the *recall* of M^S as $rec = \frac{|M \cap M^S|}{|M|}$ (i.e., the percentage of matching pairs that are returned). We say the result M^S is incorrect if $pre < 1$ and incomplete if $rec < 1$. The soundness verification is straightforward. For each record pair $(s_i, s_j) \in M^S$, the client checks whether $s_i \not\approx_E s_j$. On the other hand, the completeness verification requires to verify for each record pair $(s_i, s_j) \notin M^S$, it is true that $s_i \not\approx_E s_j$. A naive method for the client to verify the result integrity (for both soundness and completeness) is to re-compute the similarity of all record pairs, which is prohibitively costly. In this thesis, we design an efficient verification approach for the client to verity both the soundness and completeness of matching results.

5.1.5 Attack Model

We assume that the third-party server is not fully trusted as it could be compromised by the attacker (either inside or outside). The server may alter the received dataset D and return any matching result that does not exist in D. It may also tamper with the matching results. For instance, the server may return *incomplete* results that omit some legitimate similar pairs in the matching results [92]. Note that we do not consider privacy protection for the outsourced data. This issue can be addressed by privacy-preserving record linkage [39] and is discussed in Chapter 4.

5.1.6 Our Approaches in a Nutshell

We consider the outsourcing model that involves three parties - a data owner who possesses a dataset D that contains n records, the user (client) who requests for approximate record matching services on D, and a third-party service provider (server) that executes the record matching services on D. The data owner outsources D to the server. The server provides storage and record matching as services. The client can be the data owner or an authorized party. To facilitate authenticated record matching, the data owner generates some auxiliary information of D, and sends D together with the auxiliary information to the server. Upon finishing record matching on D, the server returns the matching result M^S, as well as the verification object VO, to the client. The client checks the correctness of M^S via VO. Given the fact that the client may not possess D, we require that the availability of D is not necessary for the authentication procedure.

5.2 VO Construction for A Single Target Record

In this section, we consider a set of similar record pairs $\{(s_q, s_1), (s_q, s_2), \ldots, (s_q, s_t)\}$ for a single target record s_q. We first present our basic verification approach named VS^2 (Section 5.2.1). Then we present our E-VS^2 method with improved verification cost (Section 5.2.2).

5.2.1 Basic Approach: VS^2

We design a verification method of similarity search (VS^2) approach. VS^2 constructs a *proof* that proves $M^S = \{s_1, s_2, \ldots, s_t\}$ include all strings (records) that are similar to s_q (completness), and all strings (records) in M^S is indeed similar to s_q (correctness), where θ is the similarity threshold. VS^2 consists of three phases: (1) the *pre-processing* phase in which the data owner constructs the authenticated data structure T of the dataset D. Both D and T are outsourced to the server, while only the root signature sig of T is transmitted to the legitimate clients; (2) the *matching* phase in which the server executes the approximate matching on D to find similar strings for s_q, and constructs the *verification object* (VO) of the search results M^S. The server returns both M^S and VO to the client; and (3) *verification* phase in which the client verifies the integrity of M^S by leveraging VO. Next we explain the details of these three phases.

Pre-Processing

In this one-time phase, the data owner constructs the authenticated data structure of the dataset D before outsourcing D to the server. We design a new authenticated data structure named the *Merkle B^{ed} tree* (MB-tree). Next, we explain the details of MB-tree.

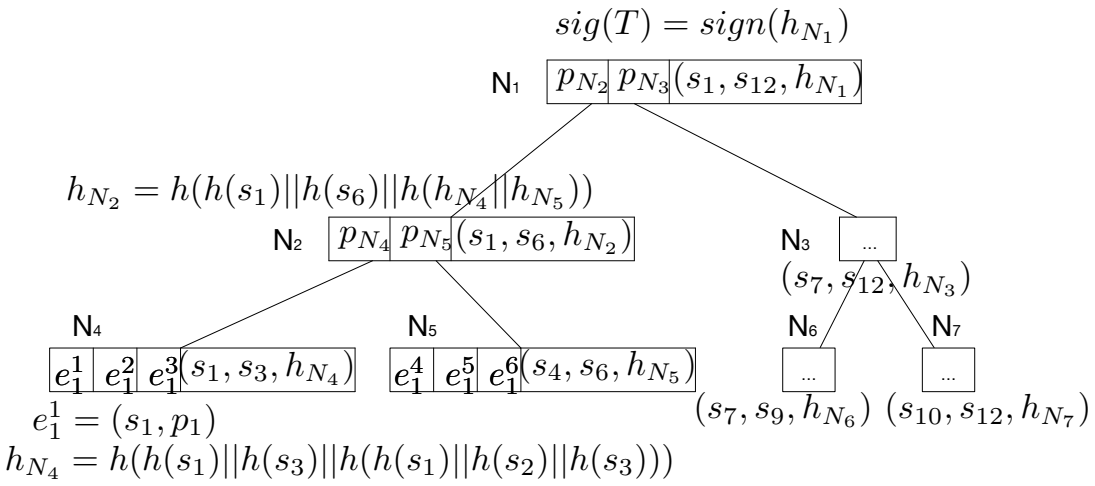


Figure 5.1: An example of Merkle B^{ed} tree

The MB-tree is constructed on top of the B^{ed} tree by assigning the digests to each B^{ed} node. In particular, every MB-tree node contains a triple (N_b, N_e, h_N), where N_b, N_e correspond to the string range values associated with N, and h_N is the digest value computed as $h_N = h(h(N_b)||h(N_e)||h^{1\to f})$, where $h^{1\to f} = h(h_{C_1}||\ldots||h_{C_f})$, with $C_1, \ldots, C_f$ being the children of N. If N is a leaf node, then $C_1, \ldots, C_f$ are the strings $s_1, \ldots, s_f$ covered by N. Besides the triple, each MB-tree node contains multiple entries. In particular, for any leaf node N, assume it covers f strings. Then it contains f entries, each of the format (s, p), where s is a string covered by N, and p is the pointer to the disk block that stores s. For any intermediate node, assume that it has f children nodes. Then it contains f entries, each entry consisting of a pointer to one of its children nodes.

The digests of the MB-tree T can be constructed in the bottom-up fashion, starting from the leaf nodes. After all nodes of T are associated with the digest values, the data owner signs the root with her private key. The signature can be created by using a public-key cryptosystem (e.g., RSA). An example of the MB-tree structure is presented in Figure 5.1. The data owner sends both D and T to

the server. The data owner keeps the root signature of T locally, and sends it to any client who requests for it for authentication purpose.

Following [30, 76], we assume that each node of the MB-tree occupies a disk page. For the constructed MB-tree T, each entry in the leaf node occupies $|s| + |p|$ space, where $|p|$ is the size of a pointer, and $|s|$ is the maximum length of a string value. The triple (N_b, N_e, h_N) takes the space of $2|s| + |h|$, where $|h|$ is the size of a hash value. Therefore, a leaf node can have $f_1 = [\frac{P-2|s|-|h|}{|p|+|s|}]$ entries at most, where P is the page size. Given n unique strings in the dataset, there are $[\frac{n}{f_1}]$ leaf nodes in T. Similarly, for the internal nodes, each entry takes the space of $|p|$. Thus each internal node can have at most $f_2 = [\frac{P-2|s|-|h|}{|p|}]$ entries (i.e., $[\frac{P-2|s|-|h|}{|p|}]$ children nodes). Therefore, the height ht of T is $ht \geq log_{f_2}[\frac{n}{f_1}]$.

VO Construction

Upon receiving the dataset D and the threshold θ from the data owner, the server calculates the edit distance between all the string pairs and distills the similar pairs. For each target string s_q, the server constructs a VO to show that the matching result M^S is both sound and complete.

First, we define *false hits*. Given a target string s_q and a similarity threshold θ, the *false hits* of s_q, denoted as F, are all the strings that are dissimilar to s_q. In other words, $F = \{s | s \in D, s_q \not\approx_E s\}$. Intuitively, to verify that M^S is sound and complete, the VO includes both similar strings M^S and false hits F. Apparently including all false hits may lead to a large VO, and thus high network communication cost and the verification cost at the client side. Therefore, we aim to reduce the VO size of F.

Before we explain how to reduce VO size, we first define C-strings and NC-

strings. Apparently, each false hit string is covered by a leaf node of the MB-tree T. Based on whether a leaf node in MB-tree is a candidate, the false hits F are classified into two types:

- C**-strings**: the strings that are covered by *candidate* leaf nodes; and
- NC**-strings**: the strings that are covered by *non-candidate* leaf nodes.

Our key idea to reduce VO size is to use *representatives* of NC-strings in VO instead of individual NC-strings. The representatives of NC-strings take the format of *maximal false hit subtrees* (MFs). Formally, given a MB-tree T, we say a subtree T^N that is rooted at node N is a *false hit subtree* if N is a *non-candidate* node. We say the false hit subtree T^N rooted at N is *maximal* if the parent of N is a candidate node. The MFs can root at leaf nodes. Apparently, all strings covered by the MFs must be NC-strings. And each NC-string must be covered by a MF node. Furthermore, MFs are disjoint (i.e., no two MFs cover the same string). Therefore, instead of including individual NC-strings into the VO, their MFs are included. As the number of MFs is normally much smaller than the number of NC-strings, this can effectively reduce VO size. Now we are ready to define the VO.

Definition 6. Given a dataset D and a target string s_q, let M^S be the returned similar strings of s_q. Let T be the MB-tree of D, and NC be the strings that are covered by non-candidate nodes of T. Let $\mathcal{M}$ be a set of MFs of NC. Then the VO of s_q consists of: (i) string s, for each $s \in D - NC$; and (ii) a pair $(N, h^{1\to f})$ for each $MF \in \mathcal{M}$ that is rooted at node N, where N is represented as $[N_b, N_e]$, with $[N_b, N_e]$ the string range associated with N, and $h^{1\to f} = h(h_{C_1}||\ldots||h_{C_f})$, with $C_1, \ldots, C_f$ being the children of N. If N is a leaf node, then $C_1, \ldots, C_f$ are the strings $s_1, \ldots, s_f$ covered by N. Furthermore, in VO, a pair of brackets is added

around the strings that share the same parent in T.

Intuitively, in VO, the similar strings and C-strings are present in the original string format, while NC-strings are represented by the MFs (i.e., in the format of $([N_b, N_e], h_N)$).

Example 5.2.1. Consider the MB-tree T in Figure 5.1, and the string s_q to construct VO. Assume the similar strings are $M^S = \{s_2\}$. Also assume that node N_3 of T is the only non-candidate node. Then NC-strings are $NC = \{s_7, \ldots, s_{12}\}$, and C-strings are $\{s_1, s_3, s_4, s_5, s_6\}$. The set of MFs $\mathcal{M}$ =$\{N_3\}$. Therefore,

$VO = \{(((s_1, s_2, s_3), (s_4, s_5, s_6)), ([s_7, s_{12}], h^{7\rightarrow 12}))\}$,
$where\ h^{7\rightarrow 12} = h(h_{N_6}||h_{N_7}), h_{N_6} = h(h(s_7)||h(s_9)||h(h(s_7)||h(s_8)||h(s_9)))$,
$and\ h_{N_7} = h(h(s_{10})||h(s_{12})||h(h(s_{10})||h(s_{11})||h(s_{12})))$.

For each MF, we do not require that $h(N_b)$ and $h(N_e)$ appear in VO. However, the server must include both N_b and N_e in VO. This is to prevent the server from cheating on the non-candidate nodes by including incorrect $[N_b, N_e]$ in VO. More details of the security analysis of our authentication procedure can be found in Section 7.3.

We present the details of the verification objects in Algorithm 5.1. In Algorithm 5.1, we build the VO starting from the MB^{ed}-tree's root. In Algorithm 5.2, for each node N in the MB-tree, we first check if it is a MF node. If it is, we add its triple to VO (Line 3 to 4). Otherwise, if N is a non-leaf node, we go through all of its children nodes to prepare the VO (Line 6 to 8); if not, we only need to add the enclosed strings to VO as it is a leaf node (Line 9 to 11).

Require: the target record s_q, the MB-tree T
Ensure: The verification object to authenticate the query result $M^S(s_q)$.
1: set ht to be the root's height in T
2: $VO = DFS\_VO(s_q, T.root, ht)$
3: **return** VO

Algorithm 5.1: $VSS\_VO\_Construction(s_q, T)$)

Require: the target string s_q, a node N in MB-tree T and its height ht
1: $VO =' ('$
2: Let $N = (N_b, N_e, h_N)$
3: **if** $DST_{min}(s_q, N_b, N_e) > \theta$ **then**
4: $VO = VO + (N_b, N_e, h_N)$
5: **else**
6: **if** $ht > 0$ **then**
7: **for all** pointer $p \in N$ **do**
8: $VO = VO + DFS\_VO(s_q, *p, ht - 1)$
9: **end for**
10: **else**
11: **for all** entry $(s, p) \in N$ **do**
12: $VO = VO + s$
13: **end for**
14: **end if**
15: **end if**
16: $VO = VO+')'$
17: **return** VO

Algorithm 5.2: $DFS\_VO(s_q, N, ht)$)

Authentication Phase

For a given target string s_q, the server returns VO together with the matching set M^S to the client. The verification procedure consists of three steps. In Step 1, the client re-constructs the MB-tree from VO. In Step 2, the client re-computes the root hash value h'_{root}, and compares h'_{root} with the hash value decrypted from sig. In Step 3, the client re-computes the edit distance between s_q and a *subset* of strings in VO. Next, we explain the details.

Step 1: Re-construction of MB-tree. First, the client sorts the strings and string ranges (in the format of $[N_b, N_e]$) in VO by their mapping values according to the

string ordering scheme. String s is put ahead of the range $[N_b, N_e]$ if $s < N_b$. It returns a total order of strings and string ranges. If there exists any two ranges $[N_b, N_e]$ and $[N'_b, N'_e]$ that overlap, the client concludes that the VO is not correct. If there exists a string $s \in M^S$ and a range $[N_b, N_e] \in VO$ such that $s \in [N_b, N_e]$, the client concludes that M^S is not sound, as s indeed is a dissimilar string (i.e., it is included in a non-candidate node). Second, the client maps each string $s \in M^S$ to an entry in a leaf node in T, and each pair $([N_b, N_e], h_N) \in VO$ to an internal node in T. The client re-constructs the parent-children relationships between these nodes by following the matching brackets () in VO.

Step 2: Re-computation of root hash. After the MB-tree T is re-constructed, the client computes the root hash value of T. For each string value s, the client calculates $h(s)$, where $h()$ is the same hash function used for the construction of the MB^{ed}-tree. For each internal node that corresponds to a pair $([N_b, N_e], h^{1 \to f})$ in VO, the client computes the hash h_N of N as $h_N = h(h(N_b)||h(N_e)||h^{1 \to f})$. Finally, the client re-computes the hash value of the root node, namely h'_{root}. The client recovers the original MB-tree's root hash value h_{root} by decrypting sig using the data owner's public key. The client then compares h'_{root} with h_{root}. If $h'_{root} \neq h_{root}$, the client concludes that the server's results are not sound.

Step 3: Re-computation of necessary edit distance. First, for each string $s \in M^S$, the client re-computes the edit distance $DST(s_q, s)$, and verifies whether $DST(s_q, s) \leq \theta$. If all strings $s \in M^S$ pass the verification, then the client concludes that M^S is *sound*. Second, for each C-string $s \in VO$ (i.e., those strings appear in VO but not M^S), the client verifies whether $DST(s_q, s) > \theta$. If it is not (i.e., s is a similar string indeed), the client concludes that the server fails the completeness verification. Third, for each range $[N_b, N_e] \in VO$, the client verifies whether $DST_{min}(s_q, N) > \theta$, where N is the corresponding MB-tree node associated with

the range $[N_b, N_e]$. If it is not (i.e., node N is indeed a candidate node), the client concludes that the server fails the completeness verification.

Example 5.2.2. Consider the MB-tree in Figure 5.1 as an example, and the target string s_q. Assume the similar strings $M^S = \{s_2\}$. Consider the VO shown in Example 5.2.1. The C-strings are $C = \{s_1, s_3, s_4, s_5, s_6\}$. After the client re-constructs the MB-tree, it re-computes the hash values of strings $M^S \cup C$. It computes the root hash value h'_{root} from these values. It also performs the following distance computations: (1) for $M^S = \{s_2\}$, compute $DST(s_q, s_2)$; (2) for $C = \{s_1, s_3, s_4, s_5, s_6\}$, compute the edit distance between s_q and every string in C; and (3) for the pair $([s_7, s_{12}], h^{7\rightarrow 12}) \in VO$, compute $DST_{min}(s_q, N_3)$. Compared with the record matching locally which requires 12 edit distance calculations, VS^2 only computes 7 edit distances.

Security Analysis

Given a target string s_q and a similarity threshold θ, let M (F, resp.) be the similar strings (false hits, resp.) of s_q. Let M^S be the similar strings of s_q returned by the server. An untrusted server may perform the following cheating behaviors to generate M^S: (1) *tampered values*: some strings in M^S do no exist in the original dataset D; (2) *soundness violation*: the server returns $M^S = M \cup FS$, where $FS \subseteq F$; and (3) *completeness violation*: the server returns $M^S = M - SS$, where $SS \subseteq M$.

The tampered values can be easily caught by the authentication procedure, as the hash values of the tampered strings are not the same as the original strings. This leads to that the root hash value of MB-tree re-constructed by the client different from the root value of original dataset. The client can catch the tampered

values by Step 2 of the authentication procedure. Next, we mainly focus on the discussion of how to catch soundness and completeness violations.

Soundness. The server may deal with the VO construction of $M^S = M \cup FS$ in two different ways:

Case 1. The server constructs the VO of the correct result M, and returns $\{M^S, VO\}$ to the client;

Case 2. The server constructs the VO of M^S, and returns $\{M^S, VO\}$ to the client.

Note that the strings in FS can be either NC-strings or C-strings. Next, we discuss how to catch these two types of strings for both Case 1 and 2.

For Case 1, for each NC-string $s \in FS$, s must fall into a MF-tree node in VO. Thus, there must exist an MF-tree node whose associated string range overlaps with s. The client can catch s by Step 1 of the authentication procedure. For each C-string $s \in FS$, s must be treated as a C-string in VO. Thus the client can catch it by computing $DST(s, s_q)$ in Step 3.

For Case 2, the C-strings in FS will be caught in the same way as Case 1. Regarding NC-strings in FS, they will not be included in any MF-tree in V'. Therefore, the client cannot catch them by Step 1 of the authentication procedure (as Case 1). However, as these strings are included in M^S, the client still can catch these strings by computing the edit distance of s_q and any string in M^S in Step 3.

Completeness. To deal with the VO construction of $M^S = M - SS$, we again consider the two cases as for the discussion of soundness violation.

For Case 1, where VO is constructed from the correct result M, any string $s \in SS$ is a C-string. These strings can be caught by re-computing the edit distance between the target string and any C-string (i.e., Step 3 of the authentication procedure).

For Case 2, where VO is constructed from the M^S, any string $s \in SS$ is

either a NC-string or a C-string in V'. For any C-string $s \in SS$, it can be caught by re-computing the edit distance between the target string and the C-strings (i.e., Step 3 of the authentication procedure). For any NC-string $s \in SS$, it must be included into a non-candidate MB-tree node. We have the following theorem.

Theorem 4. Given a target string s_q and a non-candidate node N of range $[N_b, N_e]$, including any string s' into N such that $s' \not\approx_E s_q$ will change N to be a candidate node.

The proof is straightforward. It is easy to see that $s' \notin [N_b, N_e]$. Therefore, including s' into N must change the range to be either $[s', N_e]$ or $[N_b, s']$, depending on whether $s' < N_b$ or $s' > N_e$. Now it must be true that $DST_{min}(s_q, N) \leq \theta$, as $DST(s', s_q) \leq \theta$.

Following Theorem 4, for any string $s \in SS$ that is considered a NC-string in VO, the client can easily catch it by verifying whether $DST_{min}(s_q, N) > \theta$, for any non-candidate node (i.e., Step 3 of the authentication procedure).

5.2.2 String Embedding Authentication: E-VS^2

One weakness of VS^2 is that if there exists a significant number of C-strings, the verification cost might be expensive, as the client has to compute the string edit distance between the target record and the C-strings. Our goal is to further reduce the VO verification cost at the client side by decreasing the amount of edit distance computation. We observe that although C-strings are not similar to the target record, they may be similar to each other. Therefore, we design E-VS^2, a computation-efficient method on top of VS^2. The key idea of E-VS^2 is to construct a set of *representatives* of C-strings based on their similarity, and only include the representatives of C-strings in VO. To construct the representatives of C-strings,

we first apply a similarity-preserving mapping function on C-strings, and transform them into the Euclidean space, so that the similar records are mapped to the close points in the Euclidean space. Then C-strings are organized into a small number of groups called *distant bounding hyper-rectangles (DBHs)*. DBHs are the representatives of C-strings in VO. In the verification phase, the client only needs to calculate the Euclidean distance between s_q and DBHs. Since the number of DBHs is much smaller than the number of C-strings, and Euclidean distance calculation is much faster than that of edit distance, the verification cost of the E-VS^2 approach is much cheaper than that of VS^2. Next, we explain the details of the E-VS^2 approach. Similar to the VS^2 approach, E-VS^2 consists of three phases: (1) the *pre-processing* phase at the data owner side; (2) the VO *construction* phase at the server side; and (3) the *authentication* phase at the client side.

Pre-Processing

Before outsourcing the dataset D to the server, similar to VS^2, the data owner constructs the MB-tree T on D. In addition, the data owner maps D to the Euclidean space E via a similarity-preserving embedding function $f : D \rightarrow E$ denote the embedding function. We use SparseMap [59] as the embedding function due to its contractive property.The complexity of the embedding is $O(cdn^2)$, where c is a constant value between 0 and 1, d is the number of dimensions of the Euclidean space, and n is the number of strings of D. We agree that the complexity of string embedding is comparable to the complexity of similarity search over D. However, such embedding procedure is performed only once. Its cost will be amortized over the authentication of all the future similarity search queries.

The data owner sends D, T and the embedding function f to the server.

The server constructs the embedded space of D by using the embedding function f. The function f will also be available to the client for result authentication.

VO Construction

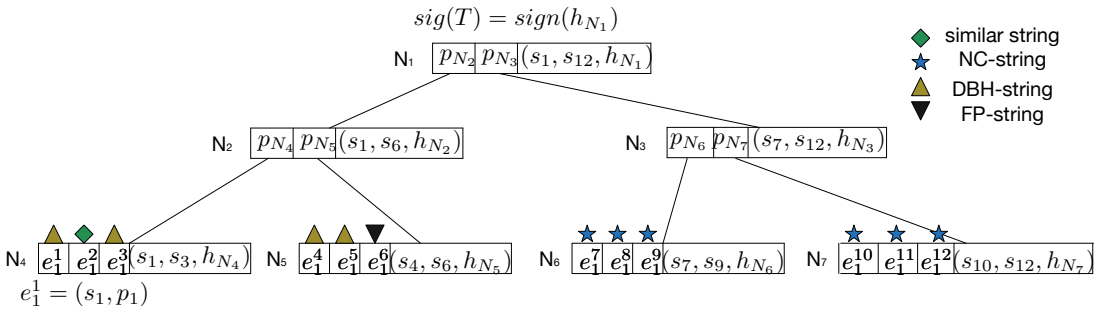
Given the query (s_q, θ) from the client, the server applies the embedding function f on s_q, and finds its corresponding node P_q in the Euclidean space. Then the server finds the result set M^S of s_q. To prove the soundness and completeness of M^S, the server builds a verification object VO. First, similar to VS^2, the server searches the MB-tree to build MFs of NC-strings. For the C-strings, the server constructs a set of *distant bounding hyper-rectangles* (DBHs) from their embedded nodes in the Euclidean space. Before we define DBH, first, we define the minimum distance between an Euclidean point and a hyper-rectangle. Given a set of points $\mathcal{P} = \{P_1, \ldots, P_t\}$ in a d-dimensional Euclidean space, a hyper-rectangle $R(\langle l_1, u_1\rangle, \ldots, \langle l_d, u_d\rangle)$ is the *minimum bounding hyper-rectangle* (MBH) of $\mathcal{P}$ if $l_i = min_{k=1}^{t}(P_k[i])$ and $u_i = max_{k=1}^{t}(P_k[i])$, for $1 \leq i \leq d$, where $P_k[i]$ is the i-dimensional value of P_k. For any point P and any hyper-rectangle $R(\langle l_1, u_1\rangle, \ldots, \langle l_d, u_d\rangle)$, the minimum Euclidean distance between P and R is

$$dst_{min}(P, R) = \sqrt{\sum_{1 \leq i \leq d} m[i]^2},$$

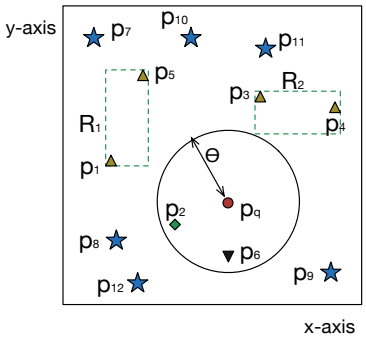
where $m[i] = max\{l_i - p[i], 0, p[i] - u_i\}$. Intuitively, if the node P is inside R, the minimum distance between P and R is 0. Otherwise, we pick the length of the shortest path that starts from P to reach R. We have:

Lemma 2. Given a point P and a hyper-rectangle R, for any point $P' \in R$, the Euclidean distance $dst(P', P) \geq dst_{min}(P, R)$.

The proof of Lemma 2 is trivial. We omit the details due to the space limit.



(a) The MB-tree

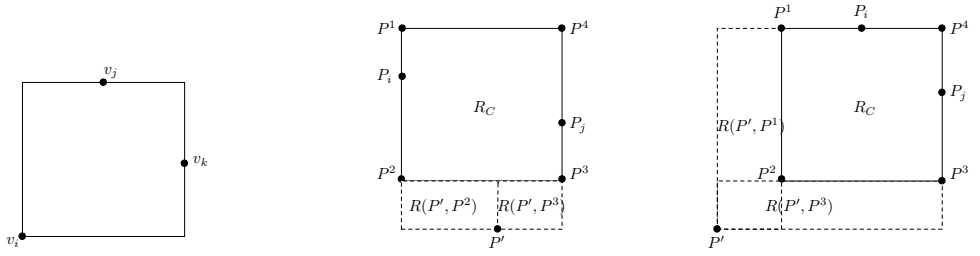


(b) The embedded Euclidean space

Figure 5.2: An example of VO construction by E-VS^2 method

Now we are ready to define *distant bounding hyper-rectangles* (DBHs). Given a target record s_q, let P_q be its embedded point in the Euclidean space. For any hyper-rectangle R in the same space, R is a *distant bounding hyper-rectangle* (DBH) of P_q if $dst_{min}(P_q, R) > \theta$.

Given a DBH R, Lemma 2 guarantees that $dst(P_q, P) > \theta$ for any point



(a) Base case (b) Case 2 of induction (c) Case 3 of induction

Figure 5.3: Illustration: Proof of Theorem 7

$P \in R$. Recalling the contractive property of the SparseMap method, we have $dst(P_i, P_j) \leq DST(s_i, s_j)$ for any string pair s_i, s_j and their embedded points P_i and P_j. Thus we have the following theorem:

Theorem 5. Given a target record s_q, let P_q be its embedded point. Then for any record s, s must be dissimilar to s_q if there exists a DBH R of P_q such that $P \in R$, where P is the embedded point of s.

Based on Theorem 5, to prove that the C-strings are dissimilar to the target record s_q, the server can build a number of DBHs to the embedded Euclidean points of these C-strings. We must note that not all C-strings can be included into DBHs. This is because the embedding function may introduce *false positives*, i.e., there may exist a false hit string s of s_q whose embedded point P becomes close to P_q, where P_q is the embedded point of s_q. Given a target record s_q, we say a C-string s of s_q is an *FP-string* if $dst(P, P_q) \leq \theta$, where P and P_q are the embedded Euclidean points of s and s_q. Otherwise (i.e. $dst(P, P_q) > \theta$), we call s a *DBH-string*. We have:

Theorem 6. Given a target record s_q, for any DBH-string s, its embedded point P must belong to a DBH.

The proof of Theorem 6 is straightforward. For any DBH-string whose embedded point cannot be included into a DBH with other points, it constructs a hyper-rectangle H that only consists of one point. Obviously H is a DBH.

Therefore, given a target record s_q and a set of C-strings, first, the server classifies them into FP- and DBH-strings, based on the Euclidean distance between their embedded points and the embedded point of s_q. Apparently the embedded points of FP-strings cannot be put into any DBH. Therefore, the server only considers DBH-strings and builds DBHs. The VO of DBH-strings is computed from DBHs. In order to minimize the verification cost at the client side, the server aims to minimize the number of DBHs. Formally,

$MDBH$ *Problem*: Given a set of DBH-strings $\{s_1, \ldots, s_t\}$, let $\mathcal{P} = \{P_1, \ldots, P_t\}$ be their embedded points. Construct a minimum number of DBHs $\mathcal{R} = \{R_1, \ldots, R_k\}$ such that: (1) $\forall R_i, R_j \in \mathcal{R}$, R_i and R_j do not overlap; and (2) $\forall P_i \in \mathcal{P}$, there exists a DBH $R \in \mathcal{R}$ such that $P_i \in R$.

Next, we present the solution to the $MDBH$ problem. We first present the simple case for the 2-D space (i.e. $d = 2$). Then we discuss the scenario when $d > 2$. For both settings, consider the same input that includes a query point P_q and a set of Euclidean points $\mathcal{P} = \{P_1, \ldots, P_t\}$ which are the embedded points of a target record s_q and its DBH-strings respectively.

When $d = 2$. We construct a graph $G = (V, E)$ such that for each point $P_i \in \mathcal{P}$, it corresponds to a vertex $v_i \in V$. For any two vertices v_i and v_j that correspond to two points P_i and P_j, there is an edge $(v_i, v_j) \in E$ if $dst_{min}(P_q, R) > \theta$, where R is the MBH of P_i and P_j. We have:

Theorem 7. Given the graph $G = (V, E)$ constructed as above, for any clique C in G, let R be the MBH constructed from the points corresponding to the vertice in C. Then R must be a DBH.

Proof. We prove it by induction. Let $|C|$ denote the number of vertices in the clique. It is trivial to show that the theorem holds if $|C| < 3$. Next, we mainly discuss $|C| \geq 3$.

Base case. When $|C| = 3$, let v_i, v_j and v_k denote the vertices in the clique C. Let R_{ij}, R_{jk}, and R_{ik} be the MBHs constructed from the pairs (v_i, v_j), (v_j, v_k), and (v_i, v_k) respectively. Let R_{ijk} be the MBH constructed from v_i, v_j, and v_k. Apparently $R_{ijk} = R_{ij} \cup R_{jk} \cup R_{ik}$. Given the fact that $dst_{min}(p, R_{ij}) > \theta$, $dst_{min}(p, R_{ik}) > \theta$, and $dst_{min}(p, R_{jk}) > \theta$, it must be true that
$dst_{min}(p, R_{ijk}) > \theta$. Therefore, R must be a DBH.

Induction step. If we add v' into C, we get a new clique C'. Let $R_{C'}$ be the MBH constructed from C'. Next, we prove that $R_{C'}$ is always a DBH. We prove this for three cases: (1) $P' \in R_C$, (2) P' falls out of the range of R_C at one dimension, and (3) P' falls out of the range of R_C at both dimensions.

Case 1. $P' \in R_C$. This case is trivial as it is easy to see that $R_{C'} = R_C$. So $R_{C'}$ must be a DBH.

Case 2. At exactly one dimension, P' falls out of R_C. Then it must be true that either $P'[i] < l_i^C$ or $P'[i] > u_i^C$, for either $i = 1$ or $i = 2$. Without loss of generality, we define the four boundary nodes of R_C as P^1, P^2, P^3, and P^4 (as shown in Figure 5.3 (b)). Also we assume that $P'[1] \in [l_1^C, u_1^C]$ and $P'[2] < l_2^C$. It is easy to see that $R_{C'} = (< l_1^C, u_1^C >, < P'[2], u_2^C >) = R_C \cup R(P', P^2) \cup R(P', P^3)$. Apparently, R_C is covered by $R_{C'}$.

Before we prove that $R_{C'}$ is a DBH, we present a lemma.

Lemma 3. *Given two rectangles* $R_1(< l_1^1, u_1^1 >, \ldots, < l_d^1, u_d^1 >)$ *and* $R_2(< l_1^2, u_1^2 > , \ldots, < l_d^2, u_d^2 >)$ *in the same Euclidean space, if* R_1 *is covered by* R_2*, i.e.* $l_i^2 \leq l_i^1 \leq u_i^1 \leq u_i^2$ *for any* $i = 1, \ldots, d$*, then* $dst_{min}(P, R_1) \geq dst_{min}(P, R_2)$ *for any point* P*.*

Lemma 3 states that if R_1 is covered by R_2, for any point P, its minimum distance to R_1 is no less than the minimum distance to R_2.

Next, let's consider $R_{C'}$ that is constructed from adding the point P' to the existing DBH R_C. We pick a point $P_i \in R_{C'}$ ($1 \leq i \leq t$) s.t. $P_i[1] = l_1^C$ and $P_i[2] \in [l_1^C, u_1^C]$. Apparently, $R(P', P^2)$ is covered by $R(P', P_i)$. Because there is an edge between v' and v_i in the graph G, it must be true that $dst_{min}(P_q, R(P', P_i)) > \theta$. Following Lemma 3, we can infer that $dst_{min}(P_q, R(P', P^2)) > \theta$. Similarly, there must be a point P_j with $P_j[1] = u_1^C$ and $P_j[2] \in [l_1^C, u_1^C]$. Because $R(P', P^3)$ is covered by $R(P', P_j)$ and $dst_{min}(P_q, R(P', P_j)) > \theta$, we can prove that $dst_{min}(P_q, R(P', P^3)) > \theta$. Thus, we prove that $dst_{min}(P_q, R_{C'}) > \theta$ and $R_{C'}$ is a DBH.

Case 3. On both dimensions, P' falls out of R_C. Formally, either $p'[i] < l_i^C$ or $P'[i] > u_i^C$, for both $i = 1, 2$. Without loss of generality, we assume that $P'[0] \in [l_1^C, u_1^C]$ and $P'[2] < l_2^C$. It is easy to see that $R_{C'} = (< P'[1], u_1^C >, < P'[2], u_2^C >) = R_C \cup R(P', P^1) \cup R(P', P^3)$. There must exists a $P_i (1 \leq i \leq t)$ s.t. $P_i[2] = u_2^C$ and $P_i[1] \in [l_1^C, u_1^C]$. In other words, $R(P', P^1)$ is covered by $R(P', P_i)$. As $dst_{min}(P_q, R(P', P_i)) > \theta$, it must be true that $dst_{min}(P_q, R(P', P^1)) > \theta$. Similar to Case 2, we can prove that $dst_{min}(P_q, R(P', P^3)) > \theta$ based on P_j. Thus, we prove that $dst_{min}(P_q, R_{C'}) > \theta$ and $R_{C'}$ is a DBH. □

Based on Theorem 7, the *MDBH* problem is equivalent to the well-known *clique partition problem*, which is to find the smallest number of cliques in a graph such that every vertex in the graph belongs to exactly one clique. The clique partition problem is NP-complete. Thus, we design the heuristic solution to our *MDBH*

problem. Our heuristic algorithm is based on the concept of *maximal cliques*. Formally, a clique is maximal if it cannot include one more adjacent vertex. The maximal cliques can be constructed in polynomial time [44]. It is shown that every maximal clique is part of some optimal clique-partition [44]. Based on this, finding a minimal number of cliques is equivalent to finding a number of maximal cliques. Thus we construct maximal cliques of G iteratively, until all the vertices belong to at least one clique.

We have the pseudocode in Algorithm 5.3. To solve the $MDBH$ problem, we first construct the graph G (Line 1). Then we initialize a clique with a single node whose degree is the minimum (Line 4-5). This is because the vertex of small degree is difficult to put into a large clique. Thus we start with the most difficult vertex and build a maximal clique for it. In Line 6 and 7, we define U_1 and U_2, the union of which is the set of vertices that are not in any partition yet. Specifically, U_1 contains the set of vertices that have the potential to be added to the current clique C, because every vertex in U_1 is connected with all the nodes in C in the graph G. In Line 17, from all the vertices in U_1, we pick the one of smallest deg_{U_2} as it is the one that is most difficult to be grouped into other cliques. After we extend the clique C by one vertex, we update the set U_1 and U_2 accordingly (Line 20). We repeatedly add vertices to C until it is maximal, i.e., not extendable. We repeat the steps to find maximal cliques until all vertices are in a clique. The DBHs can be constructed by building MBHs for the points of the vertices in every clique.

There is a special case where the $MDBH$ problem can be solved in polynomial time: when the embedded points of all DBH-strings lie on a single line, we can construct a minimal number of DBHs in the complexity of $O(\ell)$, where ℓ is the number of DBH-strings. Let $\mathcal{L}$ be the line that the embedded points of DBH-strings lie on. We draw a perpendicular line from P_q to $\mathcal{L}$. Let $dst(P_q, \mathcal{L})$ be the distance

between P_q and $\mathcal{L}$. Depending on the relationship between θ and $dst(P_q, \mathcal{L})$, there are two cases:

Case 1: $dst(P_q, \mathcal{L}) > \theta$. We construct the MBH of all the embedded points of DBH-strings.

Case 2: $dst(P_q, \mathcal{L}) \leq \theta$. The perpendicular line splits all points of DBH strings into two subsets, P_L and P_R, where P_L includes the embedded points that are at one side of $\mathcal{L}$, and P_R be the points at the other side. A special case is that P_q lies on $\mathcal{L}$. For this case, P_q still splits all points on $\mathcal{L}$ into two subsets, P_L and P_R. It is possible that P_L or P_R is empty. For each non-empty P_L or P_R, we construct a corresponding MBH.

We have the following theorem.

Theorem 8. *The MBH constructed following the above approach must be a DBH.*

Proof. For Case 1, if $dst(P_q, \mathcal{L}) > \theta$, for any point P located on $\mathcal{L}$, it must be true that $dst(P_q, P) \geq dst(P_q, \mathcal{L}) > \theta$. For Case 2, if P_L is non-empty, then $dst_{min}(P_q, P_L) \geq min\{dst(P_q, P) | P \in P_L\} > \theta$. So P_L must be a DBH. The same reasoning holds for P_R. □

When $d > 2$**.** Unfortunately, Theorem 7 can not be extended to the case of $d > 2$. There may exist MBHs of the pairs (v_i, v_j), (v_i, v_k), and (v_j, v_k) that are DBHs. However, the MBH of the triple (v_i, v_j, v_k) is not a DBH, as it includes a point w such that w is not inside $R(v_i, v_j)$, $R(v_i, v_k)$, and $R(v_j, v_k)$, but $dst(P_q, w) < \theta$.

To construct the DBHs for the case $d > 2$, we slightly modify the clique-based construction algorithm for the case $d = 2$. In particular, when we extend a clique C by adding an adjacent vertex v, we check if the MBH of the extended clique $C' = C \cup \{v\}$ is a DBH. If not, we delete the edge (u, v) from G for all $u \in C$

(Line 9-20 of Algorithm 5.3). This step ensures that if we merge any vertex into a clique C, the MBH of the newly generated clique is still a DBH.

For both cases $d = 2$ and $d > 2$, the complexity of constructing DBHs from DBH-strings is $O(n_{DS}^3)$, where n_{DS} is the number of DBH-strings.

Require: The query point P_q, all the TP-points $\{p_1, \ldots, p_t\}$
Ensure: The set of DBHs $\mathcal{R}$
1: Construct $G = (V, E)$ from $\{p_1, \ldots, p_t\}$
2: $\mathcal{C} = \emptyset$
3: **while** $V \neq \emptyset$ **do**
4: Find the node v with smallest degree $deg(v)$
5: Set a new clique $C = \{v\}$
6: $U_1 = \{u | \forall u \in V$, $\forall C \in \mathcal{C}$, $u \notin C$ and $\forall v \in C$ $(u, v) \in E\}$
7: $U_2 = \{u | \forall u \in V$, $\forall C \in \mathcal{C}$, $u \notin C$ and $\exists v \in C$ $(u, v) \notin E\}$
8: **while** $U_1 \neq \emptyset$ **do**
9: **if** $d > 2$ **then**
10: **if** $|C| \geq 2$ **then**
11: **for all** $u \in U_1$ **do**
12: $C' = C \cup \{u\}$
13: **if** $DST_{min}(rect(C'), p) \leq \theta$ **then**
14: $U_1 = U_1 - \{u\}$
15: $U_2 = U_2 \cup \{u\}$
16: Remove (u, v) from E for each $v \in C$
17: **end if**
18: **end for**
19: **end if**
20: **end if**
21: Pick $u \in U_1$ with smallest $deg_{U_2}(u) = |\{v | v \in U_2$ and $(u, v) \in E\}|$
22: $V = V - \{u\}$
23: $C = C \cup \{u\}$
24: Update U_1 and U_2
25: **end while**
26: $\mathcal{C} = \mathcal{C} \cup C$
27: **end while**
28: $\mathcal{R} = \{MBH(C) | C \in \mathcal{C}\}$
29: **return** $\mathcal{R}$

Algorithm 5.3: Build a small number of DBHs to cover TP-strings

Now we are ready to describe VO construction by the E-VS^2 approach.

Given a dataset D and a target record s_q, let M^S and F be the similar strings and false hits of s_q respectively. VS^2 groups F into C-strings and NC-strings. E-VS^2 further groups C-strings into FP-strings and DBH-strings. Then E-VS^2 constructs VO from M^S, NC-strings, FP-strings, and DBH-strings. Formally,

Definition 7. Given a target record s_q, let M^S be the returned similar strings of s_q. Let NC be the NC-strings, and DS the DBH-strings. Let T be the MB-tree, MF be the maximum false hit trees of NC. Let $\mathcal{R}$ be the set of DBH constructed from DBH. Then the VO of s_q consists of: (i) string s, for each $s \in D - NC - DS$ (i.e., s is either a similar string or a FP-string); (ii) a pair $(N, h^{1\to f})$ for each non-leaf MF that is rooted at node N, where N takes the format of $[N_b, N_e]$, with $[N_b, N_e]$ the string range associated with N, and $h^{1\to f} = h(h_{C_1}||\ldots||h_{C_f})$, with $C_1, \ldots, C_f$ being the children of N; (iii) $\mathcal{R}$; and (iv) a pair (s, p_R) for each $s \in DS$, where p_R is the pointer to the DBH in $\mathcal{R}$ that covers the Euclidean point of s; Furthermore, in VO, a pair of square bracket is added around the strings that share the same parent in T.

Example 5.2.3. Consider the MB-tree in Figure 5.2 (a). The target record is s_q. The similar strings $M^S = \{s_2\}$. The NC-strings $NC = \{s_7, \ldots, s_{12}\}$. The C-strings $C = \{s_1, s_3, s_4, s_5, s_6\}$. Consider the embedded Euclidean space shown in Figure 5.2 (b). Apparently s_6 is a FP-string as $dst(p_q, p_6) < \theta$. So the DBH-strings are $\{s_1, s_3, s_4, s_5\}$. The DBHs of these DBH-strings are shown in the rectangles in Figure 5.2 (b). The VO of target record s_q is

$$VO = \{(((((s_1, p_{R_1}), s_2, (s_3, p_{R_2})), ((s_4, p_{R_2}), (s_5, p_{R_1}), s_6)), ([s_7, s_{12}], h^{7\to 12})), \{R_1, R_2\}\},$$
$$where\ h^{7\to 12} = h(h_{N_6}||h_{N_7}).$$

Phase	Measurement	VS^2	E-VS^2
Pre-processing	Time	$O(n)$	$O(cdn^2)$
	Space	$O(n)$	$O(n)$
VO construction	Time	$O(n)$	$O(n + n_{DS}^3)$
	VO Size	$(n_R + n_C)\sigma_S + n_{MF}\sigma_M$	$(n_R + n_C)\sigma_S + n_{MF}\sigma_M + n_{DBH}\sigma_D$
Verification	Time	$O((n_R + n_{MF} + n_C)C_{Ed})$	$O((n_R + n_{MF} + n_{FP})C_{Ed} + n_{DBH}C_{El})$

Table 5.1: Complexity comparison between VS^2 and E-VS^2
(n: # of strings in D; c: a constant in [0, 1]; d: # of dimensions of Euclidean space; σ_S: the average length of the string; σ_M: Avg. size of a MB-tree node; σ_D: Avg. size of a DBH; n_R: # of strings in M^S; n_C: # of C-strings; n_{FP}: # of FP-strings; n_{DS}: # of DBH-strings; n_{DBH}: # of DBHs; n_{MF}: # of MF nodes; C_{Ed}: the complexity of an edit distance computation; C_{El}: the complexity of Euclidean distance calculation.)

Authentication Phase

After receiving VO from the server, the client extracts M^S and uses VO to verify if M^S is sound and complete. The verification of E-VS^2 consists of four steps. The first three steps are similar to the three steps of the VS^2 approach. The fourth step is to re-compute a set of Euclidean distance. Next, we discuss the four steps in details.

Step 1 & 2: These two steps are exactly the same as Step 1 & 2 of VS^2.

Step 3: Re-computing necessary edit distance. Similar to VS^2, first, for each $s \in M^S$, the client verifies $DST(s, s_q) \leq \theta$. Second, for each range $[N_b, N_e] \in VO$, the client verifies whether $DST_{min}(s_q, N) > \theta$, where N is the corresponding MB-tree node associated with the range $[N_b, N_e]$. The only difference of the E-VS^2 approach is that for each FP-string s, the client verifies if $DST(s_q, s) > \theta$. If not, the client concludes that M^S is incomplete.

Step 4: Re-computing of necessary Euclidean distance. Step 3 only verifies the dissimilarity of FP- and NC-strings. In this step, the client verifies the dissimilarity of DBH-strings. First, for each pair $(s, p_R) \in VO$, the client checks if $P_s \in R$,

where P_s is the embedded point of s, and R is the DBH that p_R points to. If all pairs pass the verification, the client ensures that the DBHs in VO covers the embedded points of all the DBH-strings. Second, for each DBH $R \in VO$, the client checks if $dst_{min}(P_q, R) > \theta$. If it is not, the client concludes that the returned results are not correct.

Note that we do not require to re-compute the edit distance between any DBH-string and the target record. Instead we only require the computation of the Euclidean distance between a set of DBHs and the embedded points of the target record. Since the Euclidean distance computation is much faster than that of the edit distance, E-VS^2 saves much verification cost compared with VS^2. More comparison of VS^2 and E-VS^2 can be found in Section 5.2.3.

Example 5.2.4. Following the running example in Example 5.2.3, after calculating the root hash h'_{root} from VO and comparing it with the one decrypted from the signature sig received from the data owner, the client performs the following computations: (1) for $M^S = \{s_2\}$, compute $DST(s_q, s_2)$; (2) for NC-strings $NC = \{s_7, \ldots, s_{12}\}$, compute $DST_{min}(s_q, N_3)$; (3) for FP-strings $FP = \{s_6\}$, compute $DST(s_q, s_6)$; and (4) for DBH-strings $DS = \{s_1, s_3, s_4, s_5\}$, compute $dst_{min}(P_q, R_1)$ and $dst_{min}(P_q, R_2)$. Compared with the VS^2 approach which computes 7 edit distances, E-VS^2 computes 3 edit distances, and 2 Euclidean distances. Note that the Euclidean distances computation is much cheaper than the edit distance computation.

Security Analysis

Similar to the security discussion for the VS^2 approach (Sec. 7.3), the server may perform three types of cheating behaviors, i.e., tampered values, soundness

violation, and completeness violation. E-VS^2 can catch the cheating behaviors of tampered values by re-computing the root hash value of MB-tree (i.e., Step 2 of the authentication procedure). Next, we mainly focus on how to catch the soundness and completeness violation by the E-VS^2 approach.

Soundness. To violate soundness, the server returns $M^S = M \cup FS$, where $FS \subseteq F$ (i.e., FS is a subset of false hits). We consider two possible ways that the server constructs VO: (Case 1.) the server constructs the VO of the correct result M, and returns $\{M^S, VO\}$ to the client; and (Case 2.) the server constructs the VO of M^S, and returns $\{M^S, VO\}$ to the client. Note that the strings in FS can be NC-strings, FP-strings, and DBH-strings. Next, we discuss how to catch these three types of strings for both cases.

For Case 1, for any NC-string $s \in FS$, s can be caught in the same way as by VS^2 approach (i.e., Step 1 of authentication). For any FP-string $s \in FS$, s can be caught by re-computing the edit distance $DST(s, s_q)$ (i.e., Step 3 of authentication). For any DBH-string $s \in FS$, let P_s be the embedded point of s. Since the VO is constructed from the correct M and F, there must exist a DBH in VO that includes P_s. Therefore, s can be caught by verifying whether there exist any DBH that includes the embedded points of s (Step 4 of the authentication procedure).

For Case 2, the NC-strings and FP-strings in FS can be caught in the same way as in Case 1. The DBH-strings cannot be caught by Step 4 now, as: (1) the DBHs constructed from a subset of DBH-strings are still DBHs, and (2) no string in FS is included in a DBH in the VO. However, these DBH-strings are treated as FP-strings (i.e., not included in any DBH), and thus can be caught by Step 3.

Completeness. To violate the completeness requirements, the server returns returns $M^S = M - SS$, where $SS \subseteq M$. Let V and V' be the VO constructed from M and M^S respectively. We again consider the two cases as for the discussion of

soundness violation.

For Case 1, where the VO is constructed from the correct result M, any string $s \in SS$ is a FP-string, as s is not included in M. Then by calculating $DST(s, s_q)$, (i.e., Step 3 of the authentication procedure), the client discovers that s is indeed similar to s_q and thus catch the incomplete results.

For Case 2, where the VO is constructed from M^S, any string $s \in SS$ is either a FP-string or a DBH-string. If s is treated as a FP-string, it can be caught by recomputing of edit distance (Step 3 of the authentication procedure). If s is a DBH-string, then its Euclidean point P_s must be included in a DBH R. We have the following theorem.

Theorem 9. Given a target record s_q and a DBH R (i.e., $dst_{min}(P_q, R) > \theta$), then for any string s such that $s \not\simeq_E s_q$, adding the embedded point of s to R must change R to be a non-DBH.

Proof. The proof of Theorem 9 is straightforward. Let R' be the hyper-rectangle after adding P_s to R. It must be true that $dst_{min}(P_q, R) \leq dst(P_q, P) \leq dst(s_q, s) \leq \theta$. Then R' cannot be a DBH. □

Following Theorem 9, for any string $s \in SS$, including its embedded point into any DBH R will change R to be a non-DBH. Then the client can easily catch it by re-computing the euclidean distance (Step 4 of verification).

5.2.3 Complexity Analysis

In this section, we compare VS^2 and E-VS^2 approaches in terms of the time and space of the pre-processing, VO construction, and verification phases. The comparison results are summarized in Table 5.1. Regarding the VO construction overhead at the server side, the DBH construction introduces $O(n_{DS}^3)$ complexity. This

makes the VO construction complexity of E-VS^2 higher than that of VS^2. However, in our empirical study, n_{DS} is around a tenth of n. Besides, the computing capability at the server side is solid. Thus the overhead $O(n + n_{DS}^3)$ of the E-VS^2 approach is still acceptable.

Regarding the VO size, the VO size of the VS^2 approach is calculated as the sum of two parts: (1) the total size of the similar strings and C-strings (in string format), and (2) the size of MF nodes. Note that $\sigma_M = 2\sigma_S + |h|$, where $|h|$ is the size of a hash value. In our experiments, it turned out that $\sigma_M/\sigma_S \not\approx_E 10$. The VO size of the E-VS^2 approach is calculated as the sum of three parts: (1) the total size of the similar strings and C-strings (in string format), (2) the size of MF nodes, and (3) the size of DBHs. Our experimental results show that σ_D is small, and the VO size of E-VS^2 is comparable to that of VS^2.

Regarding the complexity of verification time, note that $n_C = n_{FP} + n_{DS}$, where n_C, n_{FP} and n_{DS} are the number of C-strings, FP-strings, and DBH-strings respectively. Usually, $n_{DBH} << n_{DS}$ as a single DBH can cover the Euclidean points of a large number of DBH-strings. Also note that C_{Ed} (i.e., complexity of an edit distance computation) is much more expensive than C_{El} (i.e, the complexity of Euclidean distance calculation). Our experiments show that the time to compute one single edit distance can be 20 times of computing one Euclidean distance. Therefore, compared with VS^2, E-VS^2 significantly reduces the verification overhead at the client side. We admit that it increases the overhead of pre-processing at the data owner side and the VO construction at the server side. We argue that, as the pre-processing phase is a one-time operation, the cost of constructing the embedding function can be amortized by a large number of queries from the client. And as the server is equipped with strong computing resources, it is of the highest priority to reduce the verification complexity at the client side.

5.3 VO Construction for Multiple Target Records

So far we have discussed the authentication of record matching of a single target string (record). To authenticate the matching result for multiple target strings (records) $S = \{s_1, \ldots, s_\ell\}$, a straightforward solution is to create VO for each string $s_i \in S$ and its similarity result $M^S(s_i)$. Apparently this solution may lead to VO of large sizes in total. Thus, we aim to reduce the size of VOs. Our VO optimization method consists of two main strategies. We discuss both schemes in details.

5.3.1 Optimization by Triangle Inequality

It is well known that the string edit distance satisfies the *triangle inequality*, i.e. $|DST(s_i, s_k) - DST(s_j, s_k)| \leq DST(s_i, s_j) \leq DST(s_i, s_k) + DST(s_j, s_k)$. Therefore, consider two strings s_i, $s_j \in D$, assume the the server has executed the approximate matching of s_i and prepared the VO of the results. Then consider the authentication of the search results of string s_j, for any string $s \in D$ such that $DST(s, s_i) - DST(s_i, s_j) > \theta$, there is no need to prepare the proof of s showing that it is a false hit for s_j. A straightforward method is to remove s from the MB-tree T for VO construction for the string s_j. Now the question is whether removing s always lead to VO of smaller sizes. We have the following theorem.

Theorem 10. Given a string s and a MB-tree T, let $N \in T$ be the corresponding node of s, and N_P be the parent of N in T. Let $C(N_P)$ be the set of children of P in T, and N'_P be the node constructed from $C(N_P) \backslash N$ (i.e., the children nodes of N_p excluding N). Then if N_P is a non-candidate, it must be true that N'_P must be a non-candidate.

Proof. Let $[N_b, N_e]$ be the range of N_P. The proof of Theorem 10 considers two

cases: (1) $s \neq N_b$ and $s \neq N_e$. Removing s will not change N_b and N_e, which results in that both N'_p and N_p have the same range $[N_b, N_e]$. (2) $s = N_b$ or $s = N_e$. Removing s from $C(N_P)$ changes the range of N_P to be $[N'_b, N'_e]$. Note that it must be true $[N'_b, N'_e] \subseteq [N_b, N_e]$. Therefore, N'_P must be a non-candidate. □

Based on Theorem 10, removing the MB-tree nodes that correspond to any dissimilar string does not change the structure of the MB-tree. Therefore, we can optimize the VO construction procedure by removing those strings covered by the triangle inequality from the MB-tree.

Another possible optimization is that for any two target strings s_i and s_j, for any string $s \in D$ such that $DST(s, s_i) + DST(s_i, s_j) \leq \theta$, the client does not need to re-compute $DST(s, s_j)$ to prove that s_i is a similar string in the results. In Algorithm 5.4 we show the pseudocode of VO constuction at the server side. Intuitively, $M[i, j] = k$ if and only if $DST(s_i, s_k) + DST(s_j, s_k) \leq \theta$, while $N[i, j] = k$ if and only if $|DST(s_i, s_k) - DST(s_j, s_k)| > \theta$. $R^{TRI}(s_i)$ ($FH^{TRI}(s_i)$ resp.) includes the set of similar (dissimilar resp.) strings to s_i which can be verified by the triangle inequality. From Line 9 to 20, the server constructs $R^{TRI}(s_i)$ and $FH^{TRI}(s_i)$ based on the matrix M and N. In Line 21 to 32, the server updates M and N using the similarity search result of s_i. In Algorithm 5.5, we display the pseudocode for the client to do verification. Basically, for any string s_i, the client does not need to check the verification objects for $R^{TRI}(s_i)$ and $FH^{TRI}(s_i)$. Instead, the verification can be accomplished by using an easy arithmetic based on the triangle inequality.

5.3.2 Optimization by Overlapped Dissimilar Strings

Given multiple-string search query, the key optimization idea is to merge the VOs of individual query strings. This is motivated by the fact that any two query strings

s_i and s_j may share a number of dissimilar strings. These shared dissimilar strings can enable to merge the VOs of s_i and s_j. Note that simply merging all similar strings of s_i and s_j into one set and constructing MB-tree of the merged set is not correct, as the resulting MB-tree may deliver non-leaf MFs that are candidates to both s_i and s_j. Therefore, given two query strings s_i and s_j such that their false hits overlap, let NC_i (NC_j, resp.) and DBH_i (DBH_j, resp.) be the NC-strings and DBH-strings of s_i (s_j resp.), the server finds the overlap of NC_i and NC_j, as well as the overlap between DBH_i and DBH_j. Then the server constructs VO that shares the same data structure on these overlapping strings. In particular, first, given the overlap $O_1 = NC_i \cap NC_j$, the server constructs the non-leaf MFs from O_1, and include the constructed MFs in the VOs of both s_i and s_j. Second, given the overlap $O_2 = DBH_i \cap DBH_j$, the server constructs the $DBHs$ from O_2, and include the constructed $DBHs$ in the VOs of both s_i and s_j.

5.4 Experiments

In order to evaluate the efficiency of our string similarity search authentication approaches, we run an extensive set of experiments on real-world datasets. In this section, we report the experiment results.

5.4.1 Experiment Setup

Datasets and queries. We use two real-world datasets : (1) the *Actors* dataset from IMDB [1] that contains 260K last names. The maximum length of a name is 63, while the average is 14.627; and (2) the *Authors* dataset from DBLP [2] including 1,000,000 researcher names. The maximum length of a name is 99 while the

[1] http://www.imdb.com/interfaces
[2] http://dblp.uni-trier.de/xml/

Require: the set of search strings $\mathcal{S} = \{s_1, s_2, \ldots, s_\ell\}$, the string dataset $D = \{s_1, s_2, \ldots, s_n\}$, the similarity threshold value θ
Ensure: The set of verification objects for $\mathcal{S}$
1: let M, N be two $\ell * n$ matrix
2: **for** $i \in [1, \ell]$ **do**
3: **for** $j \in [1, n]$ **do**
4: $M[i, j] = -1$
5: $N[i, j] = -1$
6: **end for**
7: **end for**
8: **for all** $s_i \in \mathcal{S}$ **do**
9: **for all** $s_j \in M^S(s_i)$ **do**
10: **if** $M[i, j] \neq -1$ **then**
11: $M^S_{TRI}(s_i) = M^S_{TRI}(s_i) \cup \{(s_j, M[i, j])\}$
12: $M^S(s_i) = M^S(s_i) - \{s_j\}$
13: **end if**
14: **end for**
15: **for all** $s_j \in FH(s_i)$ **do**
16: **if** $N[i, j] \neq -1$ **then**
17: $FH_{TRI}(s_i) = FH_{TRI}(s_i) \cup \{(s_j, N[i, j])\}$
18: $FH(s_i) = FH(s_i) - \{s_j\}$
19: **end if**
20: **end for**
21: **for all** $s_j, s_k \in M^S(s_i)$ and $j \neq k$ **do**
22: **if** $M[j, k] = -1$ AND $DST(s_i, s_j) + DST(s_i, s_k) \leq \theta$ **then**
23: $M[j, k] = i$
24: $M[j, k] = i$
25: **end if**
26: **end for**
27: **for all** $s_j \in FH(s_i)$, $s_k \in FH(s_i) \cup R(s_i)$ **do**
28: **if** $N[j, k] = -1$ AND $|DST(s_i, s_j) - DST(s_i, s_k)| > \theta$ **then**
29: $N[j, k] = i$
30: $N[k, j] = i$
31: **end if**
32: **end for**
33: construct $VO(S_i)$ for $M^S(s_i)$ and $FH(s_i)$
34: $VO(s_i) = VO(s_i) \cup M^S_{TRI}(s_i) \cup FH_{TRI}(s_i)$
35: **end for**
36: **return** the set of VOs for Q

Algorithm 5.4: $Multi\_VO\_Construct(\mathcal{S}, D, \theta)$)

average length is 14.732. As both datasets are very large, it is too time-consuming to execute approximate matching for all records in these two datasets. Thus, we

Require: the set of search strings $\mathcal{S} = \{s_1, s_2, \ldots, s_\ell\}$, the set of VOs $\mathcal{VO} = \{VO(s_1), VO(s_2), \ldots, VO(s_\ell)\}$, the similarity threshold value θ
Ensure: the correctness of the result

```
for all s_i ∈ S do
    for all (s_j, k) ∈ M^S_TRI(s_i) do
        if DST(s_i, s_k) + DST(s_j, s_k) > θ then
            catch server's soundness cheating
        end if
    end for
    for all (s_j, k) ∈ FH_TRI(s_i) do
        if |DST(s_i, s_k) − DST(s_j, s_k)| ≤ θ then
            catch server's completeness cheating
        end if
    end for
end for
return the set of VOs for S
```

Algorithm 5.5: $Multi\_VO\_Verify(\mathcal{S}, \mathcal{VO}, \theta))$

picked 10 target strings for approximate matching. For the following results, we report the average performance of matching of the 10 target strings.

Experimental environment. We implement the VS^2 and E-VS^2 approaches in C++. The hash function we use is the *SHA256* function from the OpenSSL library. We execute the experiments on a machine with 2.4 GHz CPU and 48 GB RAM, running Linux 3.2.

Baseline. As the baseline method, we implement the brute-force approximate string matching algorithm that calculates the edit distance between every string in the dataset and the target string s_q.

Parameter setup. The parameters include: (1) string edit distance threshold θ, and (2) the fanout f of MB-tree nodes (i.e., the number of entries that each node contains). The details of the parameter settings can be found in Table 5.2.

In the discussions, we use the following notations: (1) n: the number of strings in the dataset, (2) n_R: the number of similar strings, (3) n_{MF}: the number of MFs, and (4) n_C: the number of C-strings. We use σ_S to indicate the average

Parameter	setting
Similarity threshold θ	2,3,4,5,6,7
MB-tree fanout f	10, 100, 1000, 10000

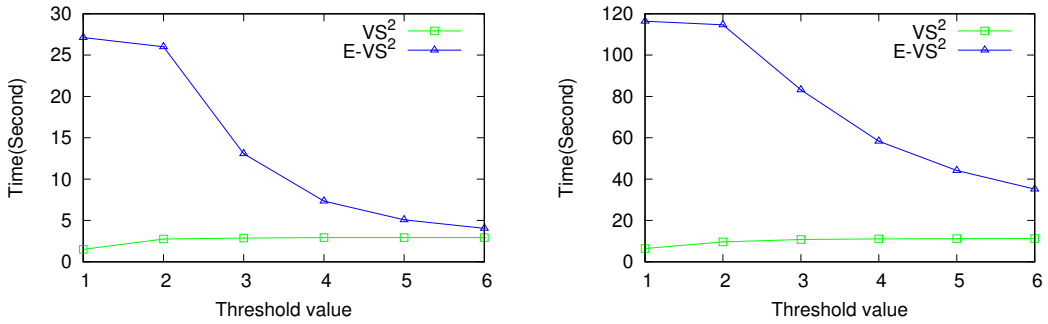
Table 5.2: Parameter settings

string size, and σ_M as the size of MB-tree node.

5.4.2 VO Construction Time

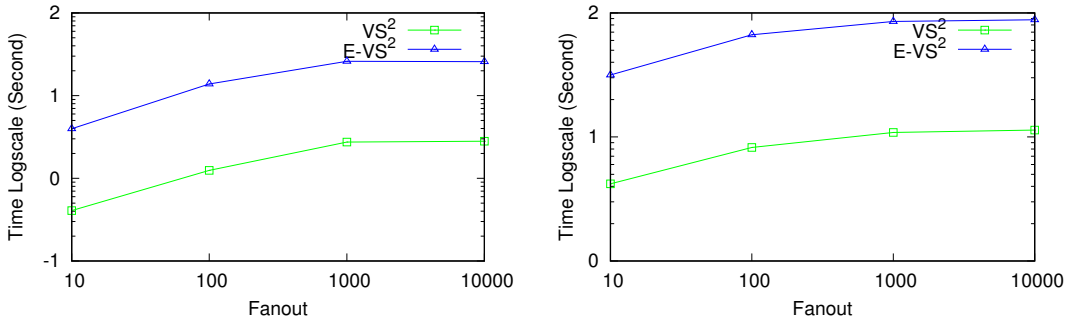
First, we measure the impact of distance threshold θ on the VO construction time. In Figure 5.4, we show the VO construction time with regard to different θ values on both datasets. First, we observe that E-VS^2 consumes more time on constructing VOs than VS^2, especially when θ is a small value. This is because besides traversing the MB-tree like VS^2, E-VS^2 constructs DBHs for the DBH-strings. The number of DBH-strings n_{DS} is large when θ is small. For example, on the $Authors$ dataset, when $\theta = 1$, $n_{DS} = 510,185$; while when $\theta = 6$, $n_{DS} = 111,129$. As a result, the DBH construction complexity is higher when the threshold is small. Second, with the increase of θ, there is a slight increment in the VO construction time of VS^2. The reason lies in the decrease in the number of MFs. When θ is small, a node N may be a MF, because $DST_{min}(s_q, N) > \theta$. While with a larger θ values, the node N becomes a candidate. For instance, on the *Actors* dataset, when $\theta = 1$, $n_{MF} = 247$; when $\theta = 2$, $n_{MF} = 22$. Consequently, VS^2 needs to search deep in the MB-tree to find the MF nodes, if there is any. However, compared with n, the increase of n_{MF} is not significant. Third, there is a dramatic decrease in VO construction time of E-VS^2 when θ increases. The reason lies in the sharp reduction in n_{DS}. Since the complexity of VO construction is cubic to n_{DS}, the total VO construction time decreases intensively when n_{DS} decreases. Fourth, the VO construction time on the *Authors* dataset is larger than that on the *Actors* dataset.

This is because with the same fanout f, the MB-tree's size of the *Authors* dataset is larger, due to the greater data size.



(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 5.4: VO construction time w.r.t. distance threshold θ



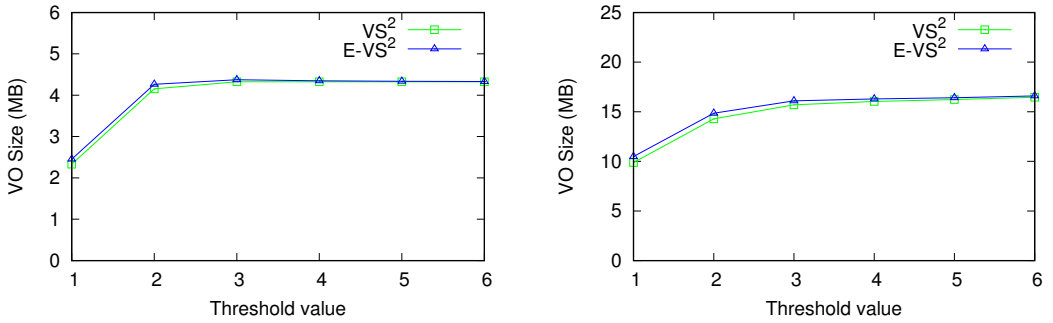
(a) The *Actors* dataset ($\theta = 2$) (b) The *Authors* dataset ($\theta = 3$)

Figure 5.5: VO construction time w.r.t. MB-tree fanout f

We also measured the impact of the MB-tree structure on the VO construction time. From Figure 5.5, we observe that VS^2 requires smaller VO construction time than E-VS^2 regardless of the fanout of the MB-tree. The underlying reason resides in the DBH construction cost of E-VS^2. Besides, in all the experiments, the VO construction time increases with the fanout value f. Even though the total number of nodes in the MB-tree decreases with larger fanout values, the decrease in the number of MFs n_{MF} is more intense. For example, on the *Authors* dataset, when $f = 100$, $n_{MF} = 5,420$; while when $f = 1,000$, $n_{MF} = 57$. The reason behind

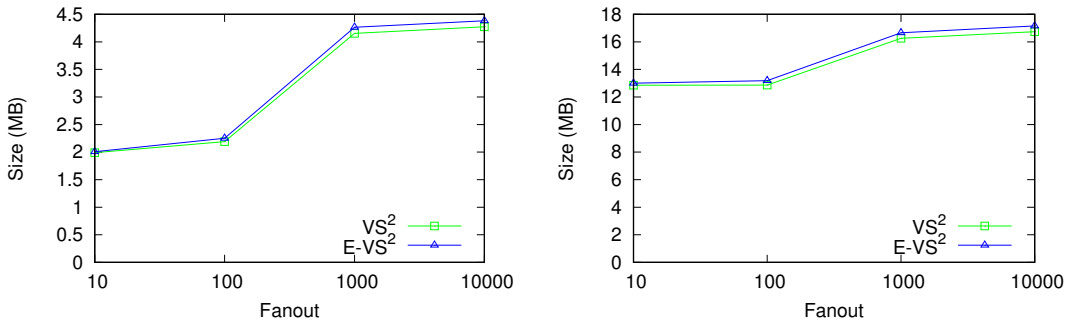
the significant reduction in n_{MF} is that when f is large, $DST_{min}(s_q, N)$ becomes a loose lower-bound for any MB-tree node N. This leads to a significant portion of the MB-tree nodes to become candidates. The observation from Figure 5.5 suggests that small fanout values fit our VS^2 and E-VS^2 methods better.

5.4.3 VO Size



(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 5.6: VO size w.r.t. distance threshold θ



(a) The *Actors* dataset ($\theta = 2$) (b) The *Authors* dataset ($\theta = 3$)

Figure 5.7: VO size w.r.t. MB-tree fanout f

We measure the impact of the distance threshold θ on the VO size. The results are shown in Figure 5.6. We notice that the VO size of VS^2 and E-VS^2 is very close. The reason is two manifold. On the one hand, the size of each

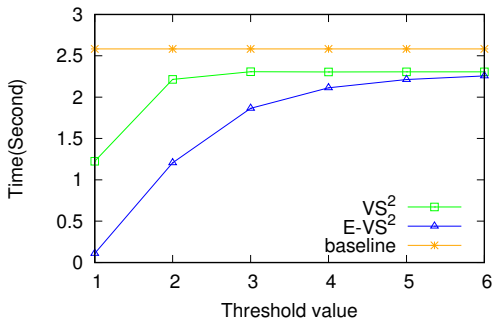
DBH is very small. On the other hand, the number of DBHs n_{DBH} is also very small. In the experiments, $n_{DS}/n_{DBH} \not\approx_E 26$. In other words, each DBH covers the Euclidean points of 26 DBH-strings. Due to these two reasons, even though E-VS^2 introduces DBHs additionally to the VO, the increment in VO size is small, and the overall VO size is roughly the same as that of VS^2. Another observation is that the VO size increases with the growth of θ value, especially when θ increases from 1 to 2. Apparently, larger θ values lead to more similar strings (i.e., larger n_R), fewer MFs (i.e., smaller n_{MF}), and fewer C-strings (i.e., smaller n_C). In VS^2, the VO size is decided by $(n_R + n_C)\sigma_S + n_{MF}\sigma_M$, where $\frac{\sigma_M}{\sigma_S} \not\approx_E 20$. When we increase θ, even though n_{MF} reduces, the intensive growth of n_R and n_C leads to the increase of VO size. For example, on the *Actors* dataset, when $\theta = 1$, $n_R + n_C = 137,655$, $n_{MF} = 247$. While when $\theta = 3$, $n_R + n_C = 249,373$, $n_{MF} = 22$.

Furthermore, we measure the impact of the MB-tree fanout f on the VO size. Following the same reason as before, we notice in Figure 5.7 that VS^2 and E-VS^2 produce VO of similar size. For both approaches, the VO size increases with the growth of f. Recall that in VS^2, the VO size is $(n_R + n_C)\sigma_S + n_{MF}\sigma_M$. When f increases, n_R is unchanged. Meanwhile, n_C dramatically increases with f (e.g., on the *Actors* dataset, when f increases from 100 to $1,000$, n_C increases by $122,515$). Moreover, when f increases, n_{MF} decreases by a small degree (when f changes from 100 to $1,000$, n_{MF} decreases by $2,937$). Considering that $\frac{\sigma_M}{\sigma_S} \not\approx_E 20$, the intensive increase of n_C leads to a larger VO size.

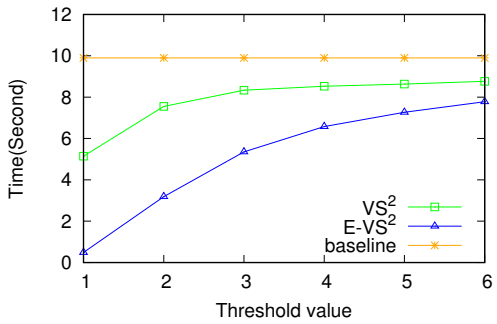
5.4.4 VO Verification Time

In this section, we measure the VO verification time at the client side. We compare the VO verification time with the performance of our baseline algorithm, aiming to

show that the client's verification effort is less than by doing string matching locally.

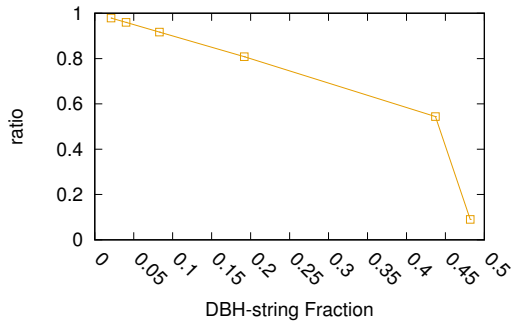


(a) The *Actors* dataset ($f = 1,000$)

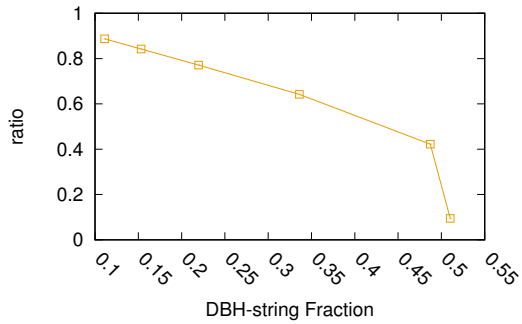


(b) The *Authors* dataset ($f = 1,000$)

Figure 5.8: VO verification time w.r.t. distance threshold θ

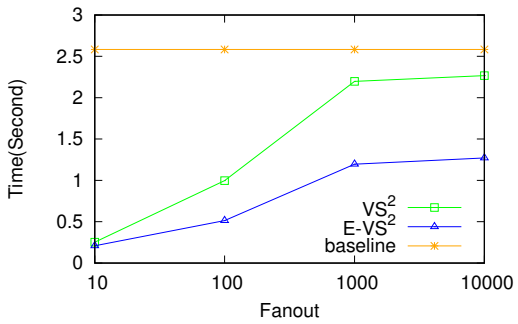


(a) The *Actors* dataset ($f = 1,000$)

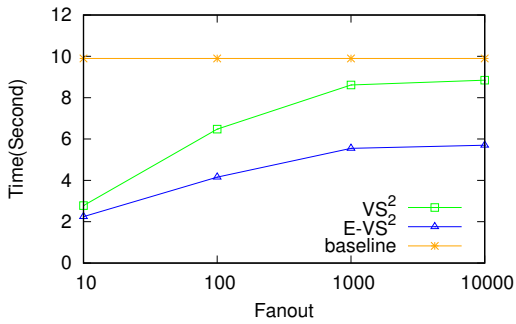


(b) The *Authors* dataset ($f = 1,000$)

Figure 5.9: VO verification time comparison of VS^2 and E-VS^2



(a) The *Actors* dataset ($\theta = 2$)



(b) The *Authors* dataset ($\theta = 3$)

Figure 5.10: VO verification time w.r.t. MB-tree fanout f

In Figure 5.8, we report the VO verification time of VS^2 and E-VS^2 with regard to various distance threshold values. First, we observe that the VO verification time is always smaller than the baseline. In particular, it can save up to 53% computational effort at the client side if she verifies the approximate matching result using VS^2 rather than using the baseline. E-VS^2 can save even 96% of the client's verification effort. This proves that our verification method is suitable for the outsourcing setting. Second, the verification time of E-VS^2 is always much smaller than that of VS^2, which proves the efficiency of E-VS^2. Third, the VO verification time increases when θ increases. Recall that the complexity of VO verification is $O((n_R + n_C + n_{MF})C_{Ed})$ for VS^2 and $O((n_R + n_{MF} + n_{FP})C_{Ed} + n_{DBH}C_{El})$ for E-VS^2. When θ increases, n_{MF} decreases, while n_R, n_C and n_{FP} increase. As n_R, n_C and n_{FP} increase faster than the decrease of n_{MF}, it leads to more edit distance calculations. Therefore, the total verification time increases.

We observe that when θ is smaller, E-VS^2 wins more over VS^2. We conduct deeper study to explore the reason. We find that rather than θ, the key factor to verification time is the fraction of DBH-strings (denoted as p_{DS}), where $p_{DS} = \frac{n_{DS}}{n}$. In Figure 5.9, we compare the VO verification time of VS^2 and E-VS^2 with regard to various p_{DS} values. In particular, let t_1 and t_2 be the verification time of of VS^2 and E-VS^2 respectively. We report the ratio $r = \frac{t_2}{t_1}$ under different p_{DS} values. Intuitively, smaller r values show the effectiveness of E-VS^2 in saving the verification time at the client side. The main observation is that r is smaller for larger p_{DS} value. This is because with E-VS^2, the client replaces the edit distance calculation for DBH-strings by efficiently computing the Euclidean distance of a small amount of DBHs. The larger p_{DS} is, the larger fraction of edit distance calculations that the client can avoid. In particular, E-VS^2 can save up to 91% verification cost at the client side than VS^2 ($r = 9\%$). This demonstrates the strength of E-VS^2 in reducing the VO

verification complexity.

We also measure the VO verification time with various MB-tree fanout f. On both datasets, the VO verification is always more efficient than the baseline method. In particular, when the fanout reaches 10, VS^2 and E-VS^2 can save up to 90% and 92% computational effort at the client side respectively. Besides, the verification time increases when f grows. This is straightforward as larger f results in the growth of the number of C-strings for VS^2 and F-strings for E-VS^2. This requires more edit distance calculations.

In sum, the experiment results suggest that VS^2 and E-VS^2 are efficient authentication approaches to verify the result of outsourced record matching computations. In particular, E-VS^2 alleviates the verification effort at the client side better, but demands more VO construction endeavor at the server side.

Chapter 6

Privacy-preserving Outsourced Data Inconsistency Repair with Adversarial FD Knowledge

In this chapter, we discuss how to protect the data privacy when outsourcing the data inconsistency repair tasks to a server with functional dependency (FD) knowledge about the original dataset. We allow the client to express content-based security granularity (i.e., the sensitive information) in the format of *security constraints* (SCs). We consider the server tries to infer the sensitive information via the FD knowledge. We study the security threats by the adversarial functional dependencies and design efficient partial encryption schemes to defend against the FD-based attack with small overhead. This work has been published in the proceedings of IEEE International Conference on Big Data Security on Cloud (BigDataSecurity), April, 2016.

6.1 Preliminaries

In this section, we introduce the background knowledge.

6.1.1 Query Types

We consider D as a relational table that consists of m attributes and n records. We use $r[X]$ to specify the value of record r on the attribute(s) X. Since D may contain sensitive information and the server is not fully trusted, the data owner encrypts D to $\hat{D}$ using symmetric encryption algorithms like AES [35] and DES [32], and sends $\hat{D}$ to the server. Only authorized users have the key to decrypt $\hat{D}$. It is worth noting

that it is possible that only a portion of D is sensitive. The non-sensitive data of D is allowed to be accessible by any third party, including the server.

NM	SEX	AGE	DC	DS
Alice	F	53	CPD5	HIV
Carol	F	30	VPI8	Breast Cancer
Ela	F	24	VPI8	Breast Cancer

(a) The original dataset D

NM	SEX	AGE	DC	DS
Alice	F	53	CPD5	α
Carol	F	30	VPI8	Breast Cancer
Ela	F	24	VPI8	γ

(b) The unsafe encrypted dataset $\bar{D}$

Figure 6.1: An example of data instance (FD: DC $\rightarrow$ DS)

After the server receives the outsourced dataset $\hat{D}$, it accepts SQL queries from the users, and evaluate the queries on $\hat{D}$. The data owner can be one of the data users. We consider two types of query Q:

- *Point query*: Q consists of the value-based constraint that is specified in the format $\cup_{1\leq i\leq m} A_i = a_i$, where A_i is an attribute of D, and a_i is a specific value of A_i. Considering Figure 6.1, an example of point query is "SELECT NM FROM D WHERE $AGE = 30$". This query returns all patients whose age is 30;

- *Range query*: Q consists of the value-based constraint in the format $\cup_{1\leq i\leq m} A_i \in [a_i, b_i]$, where A_i is an attribute of D, and a_i, b_j are two specific values of A_i. Again, according to the table in Figure 6.1, an example of range query is "SELECT NM FROM D WHERE $AGE \in [20, 40]$". This query returns all patients whose age is between 20 and 40.

For both types of queries, the server returns the records that satisfy Q.

6.1.2 Security Constraints

We propose *security constraints* (SCs) as a means for the data owner to specify the data that she intends to protect from the untrusted service provider. We adapt the *selection-projection queries* [17] as the format of the SCs. Formally, the *security constraint* is defined in the format $\Pi_Y \sigma_c$, where

- Π_Y denotes the projection of a relation on attributes Y;
- σ_c denotes the selection condition in which C is a conjunction of equalities of format $A = B$ or $A = a$ such that A, B are attributes and a is a constant.

For any SC $S : \Pi_Y \sigma_c$, we use $Proj(S)$ to denote the projection attribute list Y of Π_Y, and $Sel(S)$ to denote the list of the attributes in its selection condition σ_c. We allow $Sel(S)$ to be empty. For example, consider the security constraint S: $\Pi_B \sigma_{C=c_1}$, then $Proj(S) = \{B\}$, and $Sel(S) = \{C\}$.

Selection-projection queries enable SCs to be expressed in SQL statements so that the data owner can easily enforce the SCs (by executing the SQL statements) over her dataset. For a given dataset D and a SC S, we use $S(D)$ to denote the *sensitive* cells that is required to be protected. We consider encryption as the way to enforce SCs.

For a given dataset D and a set of SCs $\mathcal{S}$, we define the *basic scheme* as following. We use $\cup_{S \in \mathcal{S}}(S(D))$ to denote the union of sensitive cells of all SCs.

Definition 6.1.1. [Basic Scheme] Given a dataset D and a set of security constraints $\mathcal{S}$, the *basic scheme* of enforcing $\mathcal{S}$ on D is to encrypt $\cup_{S \in \mathcal{S}}(S(D))$.

Example 6.1.1. *Consider the dataset D in Figure 7.1 (a). Apparently, there is a FD $A \rightarrow B$ in D. Consider the security constraint SC $\Pi_B \sigma_{C=c_1}$. The basic encryption scheme $\bar{D}$ according to the SC is shown in Figure 7.1 (b).*

We focus on the deterministic encryption mechanism (i.e., the same plaintext is always encrypted as the same ciphertext for a given key). In particular, we use AES as the deterministic encryption algorithm. The algorithm is applied at cell level, i.e., the cells in the relation D are encrypted individually. The advantage of using cell-level encryption is to support execution of point queries on the encrypted data. The execution of range queries can be supported by order-preserving encryption (e.g., [4, 14]) or homomorphic encryption [100]. We use these two encryption algorithms to support efficient query evaluation, especially range queries. We are aware that both the deterministic encryption and order preserving encryption can lead to data leakage [90]. This is the trade-off between security and efficiency.

6.1.3 Attack Model

In this thesis, we consider the server that can be compromised and thus acts as the attacker. We assume the server is semi-honest, i.e., honestly follow the protocol but curious to obtain the sensitive information. Thus it may try to infer the plaintext values from the ciphertext. We only focus on FDs as the adversary knowledge. Other types of adversary knowledge (e.g., frequency distribution of the plaintext data) is out of scope of this section.

6.1.4 Our Approaches in a Nutshell

We perform the foundational study of security vulnerabilities by adversarial FDs, and formalize the *FD attack*. Following this, we design an efficient algorithm to find the data cells that are necessarily to be encrypted to defend against the FD attack. We prove that finding the optimal scheme that encrypts the minimum number of

data cells is NP-complete. Therefore, we design efficient greedy algorithms to find a small amount of non-sensitive data cells to be encrypted (together with sensitive data cells), under the presence of one or multiple FDs. In order to facilitate the query processing over the partially encrypted data, we design a secure query rewriting scheme that enables the service provider to answer point and range SQL queries on the encrypted data with provable security guarantee.

6.2 Difference between Our Approach and the Existing Work

In this section, we briefly introduce some existing work in eliminating inference channel from the dataset and discuss their problems in our setting.

The problem of using the association between attributes to infer sensitive information has been widely explored in the last two decades. One solution is to block the inference channel by increasing the security level of certain attributes [116, 17]. The other is to split the data into multiple fragments to break the sensitive associations between attributes [28, 128]. Next, we compare our approach with these two methods.

Inference Control The problem of inference channel via FDs has been investigated in the context of multilevel secure relational database management system (MDB) [116, 17]. The existing work can be classified into two types: (1) at query time; and (2) at design time. The security mechanisms that enforce protection at query time [17] mainly reject or modify the queries that may incur security violations. Such mechanisms are not suitable for the outsourcing paradigm as the service provider may be compromised and fail to perform the protection when it evaluates the queries. On other hand, the at-design mechanisms (e.g. [116]) detect and eliminate the inference channels via encryption during database design

time. However, the encryption is applied at an attribute level (i.e., all values of the attribute are encrypted). Such coarse encryption granularity may bring tremendous encryption overhead. In Example 7.2.4, we will show that by encrypting a small amounts of non-sensitive data besides the sensitive one can disable the inference based on adversarial FDs with lightweight encryption overhead.

TID	A	B	C
1	a_1	b_1	c_1
⋮	⋮	⋮	⋮
999	a_1	b_1	c_2
1000	a_1	b_1	c_2
1001	a_1	b_1	c_3
1002	a_2	b_2	c_3
⋮	⋮	⋮	⋮
2000	a_2	b_2	c_3

(a) Original table D ($FD : A \rightarrow B$, $S_1 : \Pi_B \sigma_{C=c_1}$, $S_2 : \Pi_B \sigma_{C=c_2}$)

TID	A	B	C
1	a_1	$\boldsymbol{\beta_1}$	c_1
⋮	⋮	⋮	⋮
999	a_1	$\boldsymbol{\beta_1}$	c_2
1000	a_1	$\boldsymbol{\beta_1}$	c_2
1001	a_1	b_1	c_3
1002	a_2	b_2	c_3
⋮	⋮	⋮	⋮
2000	a_2	b_2	c_3

(b) Basic encryption $\bar{D}$ (not safe against FD attack)

TID	A	B	C
1	$\boldsymbol{\alpha_1}$	$\boldsymbol{\beta_1}$	c_1
⋮	⋮	⋮	⋮
999	$\boldsymbol{\alpha_1}$	$\boldsymbol{\beta_1}$	c_2
1000	$\boldsymbol{\alpha_1}$	$\boldsymbol{\beta_1}$	c_2
1001	$\boldsymbol{\alpha_1}$	b_1	c_3
1002	$\boldsymbol{\alpha_2}$	b_2	c_3
⋮	⋮	⋮	⋮
2000	$\boldsymbol{\alpha_2}$	b_2	c_3

(c) State-of-the-art [116] (overhead: 2000)

TID	A	B	C
1	a_1	$\boldsymbol{\beta_1}$	c_1
⋮	⋮	⋮	⋮
999	a_1	$\boldsymbol{\beta_1}$	c_2
1000	a_1	$\boldsymbol{\beta_1}$	c_2
1001	$\boldsymbol{\alpha_1}$	b_1	c_3
1002	a_2	b_2	c_3
⋮	⋮	⋮	⋮
2000	a_2	b_2	c_3

(d) Our approach (overhead: 1)

Figure 6.2: An example that compares the existing work with our approach (encrypted cells are shown in bold; encrypted non-sensitive cells are shown in boxes)

Example 6.2.1. *Consider the dataset D in Figure 6.2 (a). Assume it has FD $F : A \rightarrow B$. Consider two security constraint rules $S_1 : \Pi_B \sigma_{C=c_1}$ and $S_2 : \Pi_B \sigma_{C=c_2}$ on D, which specify that for all records whose value of attribute C is c_1 or c_2, their*

value of attribute B is sensitive. The encryption scheme $\bar{D}$ that enforces S_1 and S_2 is shown in Figure 6.2 (b). Apparently, by the reasoning of F and the (a_1, b_1) pair of Tuple 1001 in $\bar{D}$, the attacker can infer that the cipher value β_1 of Tuple 1, . . . , 1000 is indeed mapped to the plaintext value. To fix this, state-of-the-art approach [116] encrypts the A attribute of all the tuples (cells in Figure 6.2 (c) that are put in rectangles), which leads to 2000 non-sensitive cells to be encrypted. Our approach (Figure 6.2 (d)) only encrypts the A attribute of Tuple 1001, leading to one single non-sensitive data cell to be encrypted. The encryption overhead (i.e., the number of non-sensitive tuples being encrypted) of our approach is only 0.05% of the existing methods [17, 116].

Example 7.2.4 shows that there exists a trade-off between the amounts of encryption overhead and the data owner's computational efforts. The existing methods for the MDB model (e.g., [116, 17]) do not require the client to traverse the data, with the price of high encryption overhead, while our approach reduces the encryption overhead dramatically but with the owner's efforts to find appropriate non-sensitive data to encrypt. We argue that such efforts at the client side are indeed affordable for some outsourcing models, for example, the cloud computing paradigm, in which organizations may actually possess much computational resources [54]. Besides, since our approach introduces much smaller encryption overhead, the client would enjoy a great improvement in data utility and significant saving of decryption cost.

Data Fragmentation An alternative to protecting data with the existence of adversary data associations is to decompose the data into smaller fragments [28, 128]. The fragmentation scheme is decided by the confidentiality constraints, which state the set of attributes that should not appear together. Compared with this work, first,

our security constraints are more flexible as they allow arbitrary associations between attributes and/or data values. Second, the fragmented data introduces additional overhead for processing queries that need to join multiple fragments. Third, consider a cost model that measures the query processing overhead, the problem of finding an optimal fragmentation scheme with minimum cost is NP-hard. As pointed out in [128], the size of the search space for the problem exponentially grows with m, where m is the number of attributes.

6.3 FD Attack

The basic idea of the FD attack is to find the *sensitive* and *evidence* records. We first define the sensitive/evidence records and sensitive cells.

Definition 6.3.1. [Sensitive Cells/Records, Evidence Records] Given the dataset D and a set of security constraints $\mathcal{S}$, let $\bar{D}$ be the basic scheme of enforcing $\mathcal{S}$ on D. Let $F : X \rightarrow Y$ be a FD of D. Then for each record $r \in \bar{D}$, if $r[Y]$ is encrypted, we call r a *sensitive record*, and $r[Z]$ a *sensitive cell*, for any Z that is a single attribute of Y. Given a sensitive record r, for each record $r' \neq r$ that none of $r'[X]$ and $r'[Y]$ in $\bar{D}$ is encrypted, but $r'[X] = r[X]$, we call r' an *evidence record* of $r[Y]$.

As an example, consider the table D and its basic scheme $\bar{D}$ in Figure 7.1 (a) and (b). The record r_1 is a sensitive record, with $r_1[B]$ as a sensitive cell. The records r_2 and r_3 are the evidence records of $r_1[B]$. It is worth noting that a sensitive record can contain both sensitive and non-sensitive cells. Take r_1 as an example. It contains the sensitive cell $r_1[B]$, and non-sensitive cells $r_1[A]$ and $r_1[C]$.

Given an FD $F : X \rightarrow Y$, consider a record $r \in \bar{D}$ such that $r[Y]$ is encrypted but $r[X]$ is not. The attacker can launch the FD attack to break the encryption on $r[Y]$. In particular, for any sensitive (and encrypted) value $r[Y] \in \bar{D}$, the FD attack

TID	A	B	C
r_1	a_1	b_1	c_1
r_2	a_1	b_1	c_2
r_3	a_1	b_1	c_3
r_4	a_2	b_2	c_3

(a) Original table D ($FD : A \rightarrow B$, SC: $\Pi_B \sigma_{C=c_1}$)

TID	A	B	C
r_1	a_1	$\boldsymbol{\beta_1}$	c_1
r_2	a_1	b_1	c_2
r_3	a_1	b_1	c_3
r_4	a_2	b_2	c_3

(b) Basic encryption $\bar{D}$ (not robust)

TID	A	B	C
r_1	$\boxed{\boldsymbol{\alpha_1}}$	$\boldsymbol{\beta_1}$	c_1
r_2	a_1	b_1	c_2
r_3	a_1	b_1	c_3
r_4	a_2	b_2	c_3

(c) Solution 1 $\hat{D}_1$ (overhead: 1)

TID	A	B	C
r_1	a_1	$\boldsymbol{\beta_1}$	c_1
r_2	a_1	$\boxed{\boldsymbol{\beta_1}}$	c_2
r_3	$\boxed{\boldsymbol{\alpha_1}}$	b_1	c_3
r_4	a_2	b_2	c_3

(d) Solution 2 $\hat{D}_2$ (overhead: 2)

Figure 6.3: An example of FD attack and two fix approaches (encrypted cells are shown in bold; encrypted non-sensitive cells are shown in boxes)

on $r[Y]$ consists of the following steps:

Step 1: Search for the record $r' \in \bar{D}$ such that: (1) both $r'[X]$ and $r'[Y]$ are not encrypted, and (2) $r'[X] = r[X]$. Obviously these records are the evidence of $r[Y]$;

Step 2: If any evidence record exists, replace $r[Y]$ (in ciphertext format) with $r'[Y]$ (in plaintext format).

As an example, consider the base table D in Figure 7.1 (a) that has the FD $F : A \rightarrow B$. Also consider the security constraint $\Pi_B \sigma_{C=C_1}$ and the basic scheme $\bar{D}$ shown in Figure 7.1 (b). The attacker can easily find tuple r_2 as the evidence of $r_1[B]$, and replaces $r_1[B]$ in $\bar{D}$ (Figure 7.1 (b)) to be b_1 accordingly. We say a record r is *FD-compromised* if any of its encrypted sensitive cells can be replaced with a plaintext value via the FD attack. We say an encrypted dataset $\bar{D}$ is *robust* against the FD attack if no record in $\bar{D}$ is FD-compromised. Otherwise $\bar{D}$ is *unsafe* against the FD attack.

6.4 Defending Against FD Attack

In this section, we discuss how to design robust schemes against the FD attack.

6.4.1 Robustness Checking of Basic Encryption

The basic scheme encrypts all sensitive cells according to the given security constraints. First, we discuss how to check whether the basic scheme is robust against the FD attack. A naive method is to traverse the dataset to find whether there exist an encrypted cell in the basic scheme that has any evidence record. The complexity is $O(tn)$, where t is the number of sensitive cells and n is the size of D. Though correct, this approach can be costly for large datasets. Therefore, we design a method that can decide whether the basic scheme is robust by using FDs and SCs only, without traversing the dataset.

Our method relies on the sufficient condition of FD attack. [116] presents a *sufficient* condition for preventing the security leakage by FD inference as to ensure that for any FD $F : X \rightarrow Y$, the security level of X should be no lower than that of Y . We argue that this is not a complete set of sufficient conditions. We have the following lemma to show our complete sufficient conditions.

Lemma 4. Consider a dataset D and its FD $F : X \rightarrow Y$. An scheme $\bar{D}$ is robust against the FD attack if for each record in $\bar{D}$ such that $r[Y]$ is sensitive, it satisfies one of the two conditions below:

- Condition 1. There exists at least one attribute $A \in X$ such that $r[A]$ is encrypted;
- Condition 2. $r[X]$ is plaintext, and there does not exist a record $r' \neq r \in \bar{D}$ such that both $r'[X]$ and $r'[Y]$ are plaintext and $r'[X] = r[X]$.

Condition 1 of Lemma 4 is equivalent to the sufficient condition of [116], requiring that both $r[X]$ and $r[Y]$ must be encrypted at the same time. It is not necessary that all attributes of $r[X]$ must be encrypted; at least one attribute being encrypted is sufficient. Condition 2 addresses the case that $r[X]$ can remain to be plaintext as long as there is no evidence tuple of $r[Y]$ in $\bar{D}$. This condition is not covered by [116].

Following Lemma 4, we design the robustness checking method for the basic encryption scheme. First, we consider the case that there is a single security constraint (SC). We have the following theorem to check if the basic scheme is robust against the FD attack when one single SC is present.

Theorem 11. Given a dataset D, a security constraint S, and an FD $F : X \rightarrow Y$, the basic scheme $\bar{D}$ of D is robust against the FD attack w.r.t. F if one of the following conditions is met: (a) $X \cap Proj(S) \neq \emptyset$; (b) $Y \not\subseteq Proj(S)$; and (c) $Sel(S) \subseteq X \cup Y$.

Proof. We prove the correctness by showing that the three conditions in the theorem makes the two conditions in Lemma 4 being satisfied. In particular, Condition (a) enforces Condition 1 of Lemma 4, as at least one attribute of X will be considered as sensitive and thus encrypted. Condition (b) basically requires that Y values are not considered as sensitive. Condition (c) enforces Condition 2 of Lemma 4, since all records on which F holds are sensitive and thus encrypted. □

Example 6.4.1. *Consider the base table in Figure 7.1 (a), with FD $F : A \rightarrow B$ and SC $S : \Pi_B \sigma_{C=c_1}$. To be explicit, $X = \{A\}$, $Y = \{B\}$, $Proj(S) = \{B\}$, $Sel(S) = \{C\}$. The basic scheme (Figure 7.1 (b)) does not satisfy any of the three conditions in Theorem 11. Therefore it is not robust against the FD attack. As another example, assume we have a dataset D as in Figure 6.4 (a), with FD $F : \{A, B\} \rightarrow C$ and SC*

TID	A	B	C	D
r_1	a_1	b_1	c_1	d_1
r_2	a_1	b_1	c_1	d_2
r_3	a_1	b_2	c_2	d_2
r_4	a_2	b_1	c_1	d_3
r_5	a_2	b_1	c_1	d_1

(a) Original table D (FD: $\{A, B\} \rightarrow C$, SC: $\Pi_C \sigma_{B=b_1}$)

TID	A	B	C	D
r_1	a_1	b_1	γ_1	d_1
r_2	a_1	b_1	γ_1	d_2
r_3	a_1	b_2	c_2	d_2
r_4	a_2	b_1	γ_1	d_3
r_5	a_2	b_1	γ_1	d_1

(b) Basic encryption $\bar{D}$ (robust)

Figure 6.4: An example illustrating Theorem 11

$S : \Pi_C \sigma_{B=b_1}$. *In this case,* $X = \{A, B\}$, $Y = \{C\}$, $Proj(S) = \{C\}$, $Sel(S) = \{B\}$. *The third condition of Lemma 11 is satified. As the selection condition is* $B = b_1$, *the attribute value of* C *of records* $\{r_1, r_2, r_4, r_5\}$ *are sensitive and encrypted. It is true that for the four records, Condition 1 of Lemma 4 is not satisfied. However, there does not exist any record whose attribute values of FD are not encrypted and whose attribute values of* $\{A, B\}$ *matches any of the four records. In other words, Condition 2 of Lemma 4 is met. Thus, the basic encryption scheme is robust aginst FD attack.*

The reasoning of robustness checking for one single SC can be easily extended to multiple SCs. We have the following theorem.

Theorem 12. Given a dataset D, a set of security constraints $\mathcal{S}$, and an FD $F : X \rightarrow Y$ of D, the basic scheme $\bar{D}$ of D is robust against the FD attack w.r.t. F if one of the following conditions is met: (1) $\forall S \in \mathcal{S}$, $X \cap Proj(S) \neq \emptyset$; (2) $Y \not\subseteq \cup_{S \in \mathcal{S}} Proj(S)$; (3) $\cup_{S \in \mathcal{S}} Sel(S) \subseteq X \cup Y$.

6.4.2 Encryption Against FD attack

If the basic scheme is not robust against the FD attack, we would fix the basic encryption. The key idea of our fix method is to encrypt a set of non-sensitive cells besides the sensitive cells that have to be encrypted. In this section, we explain how to find those non-sensitive cells.

One Single SC. The success of the FD attack is the existence of evidence records. Therefore, we aim to find the evidence records and fix the encryption on them. First, we explain how to find the sensitive/evidence records efficiently. For a given security constraint $S : \Pi_Y \sigma_c$, in which Y is the RHS of the given FD F, first, we re-write S to be $S' : \Pi_{(X,Y)} \sigma_c$, where X and Y are LHS and RHS of F. Second, we apply S' on D, and bucketize $S'(D)$ by its values. All (X, Y) pairs of the same values are put into the same bucket. Each bucket represents a unique sensitive cell and its associated X value. Third, for each bucket $B(X = x, Y = y)$, we construct two selection-projection queries $S_S : \Pi_{TID} \sigma_{(X=x,Y=y) \cup c}$, where c is the selection condition in S, and $S_E : \Pi_{TID} \sigma_{(X=x,Y=y)}$. After applying them on D, apparently $S_S(D)$ contains sensitive records of $Y = y$ with $X = x$, and $S_E(D)$ contains sensitive and evidence records of $Y = y$ where $X = x$.

Example 6.4.2. *Consider the base table D in Figure 6.4 (a). Assume it has a FD $\{A, B\} \rightarrow C$. Consider the security constraint $S : \Pi_C \sigma_{D=d_1}$. After applying $S(D)$, we get two buckets, i.e., $B_1(a_1, b_1, c_1)$ and $B_2(a_2, b_1, c_1)$. For the bucket $B_1(a_1, b_1, c_1)$, we construct the selection-projection queries $S_S : \Pi_{TID} \sigma_{(A=a_1,B=b_1,C=c_1,D=d_1)}$ and $S_E : \Pi_{TID} \sigma_{(A=a_1,B=b_1,C=c_1)}$. The sensitive record of the bucket $B_1(A = a_1, B = b_1, C = c_1)$ is r_1, as it is in $S_S(D)$. From $S_E(D)$, we find two records $\{r_1, r_2\}$, where r_2 is the evidence record of the bucket $B_1(A = a_1, B = b_1, C = c_1)$.*

Let n_S be the number of records in $S_S(D)$ for the bucket $B(X = x, Y = y)$,

and n_E be the size of $S_E(D)$. Apparently if $n_S = n_E$, then there is no evidence record of the sensitive cell $Y = y$ that are associated with $X = x$ (i.e., it is robust against the FD attack). Otherwise, the basic scheme has to be fixed. Next, we describe two fix approaches:

- Scheme 1: for each record r in the $S_S(D)$, randomly pick one attribute $A \in X$ and encrypt $r[A]$. We allow that different A is picked for different records.
- Scheme 2: for each record r' in $S_E(D) - S_S(D)$ (i.e., r' in $S_E(D)$ but not in $S_S(D)$), encrypt $r'[X]$ or $r'[Y]$.

We have the following theorem.

Theorem 13. Consider a dataset D, an FD $F : X \rightarrow Y$, and a set of SC $\mathcal{S}$, applying one of the two schemes above always delivers a robust scheme against the FD attack.

Both fix schemes require to encrypt some *non-sensitive* cells. We measure the number of those non-sensitive cells as the *encryption overhead* against the FD attack. Formally,

Definition 6.4.1. [Encryption Overhead] Given a dataset D and a set of SCs $\mathcal{S}$, let $\bar{D}$ be the basic scheme, and $\hat{D}$ be the dataset that is robust against the FD attack. Let e and e' be the number of encrypted cells in $\bar{D}$ and $\hat{D}$. Then the *encryption overhead* against the FD attack on D is $o = e' - e$.

Our goal is to find an scheme that is robust against the FD attack with minimal encryption overhead. Apparently, for each sensitive cell, the encryption overhead of Scheme 1 is n_S, while the overhead of Scheme 2 is $n_E - n_S$. We pick the

scheme whose overhead equals

$$min_o = min(n_S, n_E - n_S) \tag{6.1}$$

to decide the scheme of less encryption overhead. We do this for each sensitive cell. Due to the fact that the sensitive/evidence records of different sensitive cells do not overlap, the final scheme is guaranteed to return the minimal encryption overhead. The complexity is $O(n^2)$, where n is the size of D.

Example 6.4.3. *In Figure 7.1, apparently, $\bar{D}$ is not robust against the FD attack. Figure 7.1 (c) & (d) show two solutions to fix $\bar{D}$ by our two fix schemes aforementioned. In particular, Scheme 1 (Figure 7.1 (c)) encrypts the a_1 value of the sensitive record r_1, with the encryption overhead 1, while Scheme 2 (Figure 7.1 (d)) encrypts the B attribute of the evidence record r_2 and the A attribute of the evidence record r_3, , with the encryption overhead 2. We pick Scheme 1 due to its smaller overhead.*

Multiple SCs. When there are $k > 1$ SCs $\mathcal{S} = \{S_1, \ldots, S_k\}$, the brute-force solution is to enumerate all possible encryption solutions, and pick the one of the smallest encryption overhead. There are 2^{tk} schemes in total, where t is the average number of sensitive cells of each SC. As each encryption solution needs $O(n^2)$ complexity to locate the sensitive and evidence records, the complexity of the brute-force method is $O(2^{tk}n^2)$, where n is the size of the dataset. This could be prohibitive if k is large.

A seemly straightforward method is to pick *local optimal* scheme for each SC $S_i \in \mathcal{S}$. However, the local optimality cannot guarantee the global optimal solution for the k SCs in terms of encryption overhead. We use the following example to illustrate that.

Example 6.4.4. *Consider a dataset D that has an FD $X \rightarrow Y$ and two SCs S_1 and S_2. Assume S_1 considers the Y attribute of records r_1 and r_2 as sensitive, while S_2 considers the Y attribute of records r_3, r_4, r_5 as sensitive. For S_1, the evidence records are r_3, r_4, and r_5, while for S_2, the evidence records are r_1, r_2, and r_3. The local optimal solution of S_1 is to encrypt the X attribute of $\{r_1, r_2\}$, while the local optimal solution of S_2 is to encrypt the X attribute of $\{r_4, r_5\}$. The total encryption overhead is 4. However, the global optimal solution is to encrypt the X attribute of $\{r_1, r_2, r_3\}$ or $\{r_3, r_4, r_5\}$. The encryption overhead is 3.*

Next, we prove that the problem of finding the optimal scheme of the minimal encryption overhead with multiple SCs is NP-complete.

Theorem 14. Given a dataset D and $k > 1$ SCs $\mathcal{S}$, the problem of finding the optimal robust scheme that enforces $\mathcal{S}$ on D against the FD attack with the minimum encryption overhead is NP-complete.

Proof. For each SC S_i, and each of its sensitive cell $v_{i,j}$, let $s_{i,j}$ and $e_{i,j}$ be its set of sensitive records and evidence records. According to our scheme (Section 6.4.2), it requires to encrypt either $s_{i,j}$ or $e_{i,j}$. Based on this, we can state our problem of finding the scheme with minimal encryption overhead for multiple security constraints (MSC) below.

Problem MSC: Given a set of SCs $\mathcal{S} = \{S_1, \ldots, S_n\}$, in which each S_i is associated with a set of pairs $\{(s_{i,1}, e_{i,1}), \ldots, (s_{i,k_i}, e_{i,k_i})\}$, where $s_{i,j}$ ($e_{i,j}$, resp.) is a set of sensitive records (resp. evidence records), find a robust scheme with the minimal overhead.

We can map this problem to the well-known weighted vertex cover (WVC) problem.

Problem WVC: Let G be an undirected graph, V a set of nodes, W be the

weighte associated with the nodes, and E a set of edges defined over those nodes. Find the minimum-weight subset of nodes $V' \subset V$ such that $\forall e = \{u, v\} \in E, u \in V'$ or $v \in V'$.

Next we prove that the problem of finding the smallest weighted vertex cover G' is equivalent to finding a robust scheme with the minimal overhead. Given a set of SCs $\mathcal{S} = \{S_1, \ldots, S_n\}$, in which each S_i is associated with a set of pairs $\{(s_{i,1}, e_{i,1}), \ldots, (s_{i,k_i}, e_{i,k_i})\}$, we construct a graph G: (1) for each $s_{i,j}$, we construct a corresponding node with weight $|s_{i,j}|$ in G, (2) for each $e_{i,j}$, we construct a corresponding node with weight $|e_{i,j}|$ in G, and (3) for each $(s_{i,j}, e_{i,j})$, we construct an edge that connects the corresponding nodes of $s_{i,j}$ and $e_{i,j}$. Clearly this transformation can be finished in polynomial time.

First, let V' be the minimal vertex cover of the constructed G. Our scheme is as follows. For any $v \in V'$, if there exists an $e(v, v') \in E$, we pick Scheme 1 (Section 6.4.2) for v. Otherwise, we pick Scheme 2 (Section 6.4.2) for v. Clearly this scheme is robust. Since the encryption overhead equals to the total weight of V', it is minimal.

Next, assume that we have a robust scheme for MSC with the minimal overhead. Let R' be the records that are encrypted by the scheme. We can construct V' in the following way: for each node $v \in V$, if its corresponding records are a subset of R', we insert v to V'. V' a vertex cover because each edge $e(v, v') \in E$ corresponds to a pair $(s_{i,j}, e_{i,j})$. Either $s_{i,j}$ or $e_{i,j}$ must exist in R'. Therefore, V' is a valid vertex cover of the graph G. The total weight of V' is also minimum as R' contains minimum number of records.

This transformation can be finished in polynomial time. The constructed G' is a minimal cover of G. From the above proof, we are able to reduce the well-known *Weighted Vertex Cover* problem to our *MSC* problem in polynomial time.

Therefore, the *MSC* problem is also NP-complete.

□

Require: the set of potentially unsafe security constraints $\mathcal{S}$, the FD $X \rightarrow Y$
Ensure: The scheme to ensure no information leakage based on FD-attack.
1: **for all** $S_i \in \mathcal{S}$ **do**
2: Find the set of sensitive cells $\{v_{i,j}\}$
3: **for all** $v_{i,j}$ **do**
4: Find the sensitive records $s_{i,j}$ and the evidence records $e_{i,j}$
5: **end for**
6: Let $H_i = \{(v_{i,j}, s_{i,j}, e_{i,j})\}$
7: **end for**
8: **while** $H_i \neq \emptyset \; \forall i \in 1, \ldots, k$ **do**
9: Let $min_c = min_{\forall v_{i,j} \in H_i} min(|s_{i,j}|, |e_{i,j}|)$
10: Let min_v be the sensitive cell that delivers min_c
11: Let s_v and e_v be the sensitive and evidence records of min_v
12: **if** $|s_v| < |e_v|$ **then**
13: $min_{se} = s_v$
14: Pick a random attribute $A \in X$, encrypt attribute A in all records of s_v
15: **else**
16: $min_{se} = e_v$
17: Pick a random attribute $A \in X \cup Y$, encrypt attribute A in all records of e_v
18: **end if**
19: **for all** H_i **do**
20: **for all** $(v_{i,j}, s_{i,j}, e_{i,j}) \in H_i$ **do**
21: **if** $v_{i,j} = min_v$ **then**
22: $s_{i,j} = s_{i,j} - min_{se}$
23: $e_{i,j} = e_{i,j} - min_{se}$
24: **if** $s_{i,j} = \emptyset$ or $e_{i,j} = \emptyset$ **then**
25: Remove $(v_{i,j}, s_{i,j}, e_{i,j})$ from H_i
26: **end if**
27: **end if**
28: **end for**
29: **end for**
30: **end while**

Algorithm 6.1: $GMM(\mathcal{S} = \{S_1, S_2, \ldots, S_k\}, X \rightarrow Y)$

Given the NP-completeness result, we design an efficient greedy algorithm (*GMM*) to find an scheme to defend against the FD attack with small encryption overhead. The pseudo code of the *GMM* algorithm is shown in Algorithm 6.1.

The key idea of *GMM* algorithm is to repeatedly pick minimum number of sensitive/evidence records, until the picked records cover all SCs. In particular, given k SCs $\mathcal{S} = \{S_1, \ldots, S_k\}$ for which the basic scheme is not robust, for each $S_i \in \mathcal{S}$ ($1 \leq i \leq k$), it is associated with a set of triples H_i = $\{(v_{i,j}, s_{i,j}, e_{i,j})\}$, where $v_{i,j}$ is the j-th sensitive cell of S_i, and $s_{i,j}$ ($e_{i,j}$, resp.) is a set of sensitive records (resp. evidence records) of $v_{i,j}$. We store the record IDs in $s_{i,j}$ and $e_{i,j}$ (Line 1 to 7 in Algorithm 6.1). We use $|s_{i,j}|$ ($|e_{i,j}|$, resp.) to denote the number of records in $s_{i,j}$ ($e_{i,j}$, resp.). For a given H_i, we define $min_c = min_{\forall v_{i,j} \in H_i} min(|s_{i,j}|, |e_{i,j}|)$. We use min_v to denote the sensitive cell of min_c, and min_{se} to denote the set of sensitive/evidence records of min_v that deliver min_c (Line 9 and 10). For instance, consider H_i = $\{((x_1, y_1),\{r_1, r_2\}, \{r_3, r_4, r_5\}), ((x_2, y_2), \{r_3, r_4\}, \{r_5\})\}$. Then $min_c = 1$, $min_v = \{(x_2, y_2)\}$, and $min_{se} = \{r_5\}$. Let $\mathcal{H}$= $\{H_1, \ldots, H_k\}$. If min_{se} consists of min_v's sensitive records, we apply the encryption based on Scheme 1. Otherwise, we pick a random attribute following Scheme 2 (Line 11 to 18). Initially we mark every H_i in $\mathcal{H}$ as *uncovered*. First, we select $H_i \in \mathcal{H}$ whose min_c is the smallest among all uncovered H_i ($1 \leq i \leq k$). Let min_s and min_e be the set of sensitive records and evidence records of the picked min_v. Second, for all triples $(v_{i,j}, s_{i,j}, e_{i,j}) \in \mathcal{H}$, if $v'_{i,j} = min_v$, we update $s_{i,j} = s_{i,j} - min_{se}$, and $e_{i,j} = e_{i,j} - min_{se}$. If there exist any H_i such that for each triple in H_i, $s_{i,j}$ or $e_{i,j}$ becomes empty, we mark H_i as *covered* (Line 19 to 28). We repeat these two steps until all H_is in $\mathcal{H}$ are covered. The complexity of the *GMM* algorithm is $O(ktn^2)$, where k is the number of SCs, t is the average number of sensitive cells, and n is the number of records in D.

Example 6.4.5. *Consider the base table in Figure 6.5 (a) with a FD* $DC \rightarrow DS$. *Suppose there exist two SCs,* $S_1 = \Pi_{DS}\sigma_{AGE<30}$ *and* $S_2 = \Pi_{DS}\sigma_{SEX=F}$. *After ba-*

TID	NM	SEX	AGE	DC	DS
r_1	Joe	M	28	CPD5	HIV
r_2	Alice	F	24	CPD5	HIV
r_3	Maggy	F	33	CPD5	HIV
r_4	Phil	M	43	CPD5	HIV
r_5	Peter	M	39	CPD5	HIV
r_6	Ray	M	52	CPD5	HIV
r_7	Steve	M	31	CPD5	HIV

(a) Orignial table D

FD: $DC \rightarrow DS$

$S_1 : \Pi_{DS}\sigma_{AGE<30}$

$S_2 : \Pi_{DS}\sigma_{SEX=F}$

TID	NM	SEX	AGE	DC	DS
r_1	Joe	M	28	β	α
r_2	Alice	F	24	β	α
r_3	Maggy	F	33	CPD5	α
r_4	Phil	M	43	CPD5	HIV
r_5	Peter	M	39	CPD5	HIV
r_6	Ray	M	52	CPD5	HIV
r_7	Steve	M	31	CPD5	HIV

(c) D_1 after one iteration

$H_1 = \emptyset$

$H_2 = \{((CPD5, HIV), \{r_3\}, \{r_4, r_5, r_6, r_7\})\}$

TID	NM	SEX	AGE	DC	DS
r_1	Joe	M	28	CPD5	α
r_2	Alice	F	24	CPD5	α
r_3	Maggy	F	33	CPD5	α
r_4	Phil	M	43	CPD5	HIV
r_5	Peter	M	39	CPD5	HIV
r_6	Ray	M	52	CPD5	HIV
r_7	Steve	M	31	CPD5	HIV

(b) Basic encryption $\bar{D}$

(not robust)

$H_1 = \{((CPD5, HIV), \{r_1, r_2\}, \{r_4, r_5, r_6, r_7\})\}$

$H_2 = \{((CPD5, HIV), \{r_2, r_3\}, \{r_4, r_5, r_6, r_7\})\}$

TID	NM	SEX	AGE	DC	DS
r_1	Joe	M	28	β	α
r_2	Alice	F	24	β	α
r_3	Maggy	F	33	β	α
r_4	Phil	M	43	CPD5	HIV
r_5	Peter	M	39	CPD5	HIV
r_6	Ray	M	52	CPD5	HIV
r_7	Steve	M	31	CPD5	HIV

(d) D_2 after two iterations

(Robust against FD attack)

Figure 6.5: An example of GMM

sic encryption, we have $H_1 = \{((CPD5, HIV), \{r_1, r_2\}, \{r_4, r_5, r_6, r_7\})\}$ *and* $H_2 = \{((CPD5, HIV), \{r_2, r_3\}, \{r_4, r_5, r_6, r_7\})\}$. *In the first iteration of* GMM*, we find* $min_c = 2$, $min_v = \{((CPD5, HIV))\}$, *and* $min_{se} = \{r_1, r_2\}$. *After applying Scheme 1, we get a table in Figure 6.5 (c), which* $H_1 = \emptyset$ *and* $H_2 = \{((CPD5, HIV), \{r_3\}, \{r_4, r_5, r_6, r_7\})\}$. *In the second round,* $min_c = 1$, $min_v = \{((CPD5, HIV))\}$, *and* $min_{se} = \{r_3\}$. *After applying Scheme 1 on* r_3*, we get a table in Figure 6.5 (d), which is secure against FD attack.*

6.5 Multiple FDs

There may exist multiple FDs in a given dataset. It is well known that the FDs can interact; new FDs can be inferred from the existing ones. Formally, given a set of FDs $\mathcal{F}$, one can use Armstrong's axioms [8] to derive the closure $\mathcal{F}^+$ that includes both $\mathcal{F}$ and the new FDs that can be inferred from $\mathcal{F}$. Intuitively, we have to consider the security leakage by the FD attack based on $\mathcal{F}^+$. However, it has been proven that if $\mathcal{F}$ is safe against the FD attack, then $\mathcal{F}^+$ is safe against the FD attack [116]. Therefore, for the following discussions, we only consider $\mathcal{F}$ but not $\mathcal{F}^+$.

The robustness checking of the basic scheme against multiple FDs can be achieved by checking whether the basic scheme is robust against each FD (Theorem 12). For the following discussions, we only consider FDs that may enable potential leakage on the basic scheme.

As shown in [116], for the SCs that are specified at the attribute-level granularity, finding an optimal scheme of minimal encryption overhead is an NP-complete problem. Thus given the SCs that support arbitrary granularity, our problem is also NP-complete. Therefore, we aim to design an efficient heuristic approach that generates the robust scheme against the FD attack with small encryption overhead. Next, we present our heuristic solution named GMM-MFD algorithm.

Before deciding which records to be encrypted, we determine on which set of attributes we apply encryption. Intuitively, to reduce encryption overhead, we prefer this attribute set to be small. Formally,

Definition 6.5.1. [Minimum Attribute Cover (MAC)] We say a set of attributes $\mathcal{A}$ *covers* an FD $F : X \rightarrow Y$ if $\mathcal{A} \cap (X \cup Y) \neq \emptyset$. An attribute cover $\mathcal{A}$ is the *minimum attribute cover* (MAC) if any proper subset of $\mathcal{A}$ cannot cover F.

Our problem is the following: given a set of FDs $\mathcal{F}$, find a MAC $\mathcal{A}$ that covers $\mathcal{F}$.

Require: The set of FDs $\mathcal{F} = \{F_1, \ldots, F_f\}$
Ensure: Find a minimum attribute cover of $\mathcal{F}$
1: $\mathcal{A} = \emptyset$;
2: **for all** $A \in R$ **do**
3: $A.weight = 0$
4: **end for**
5: **for all** $F \in \mathcal{F}'$ **do**
6: **for all** $A \in LHS(F)$ **do**
7: $A.weight++$
8: **end for**
9: $RHS(F).weight++$
10: **end for**
11: **while** $\mathcal{F}' \neq \emptyset$ **do**
12: Take A in R with the largest weight
13: **for all** $F \in \mathcal{F}'$ such that $A \in LHS(F)$ OR $A \in RHS(F)$ **do**
14: **for all** $A' \in F$ **do**
15: $A'.weight--$
16: **end for**
17: $\mathcal{F}' = \mathcal{F}' - \{F\}$
18: $\mathcal{A}.add(A)$
19: **end for**
20: **end while**
21: **return** $\mathcal{A}$

Algorithm 6.2: $FindMAC(\mathcal{F})$

We can map the problem of finding the MAC to the well-known hitting set problem. In particular, we construct a bipartite graph $G(V_1 \cup V_2, E)$, in which each node in V_1 corresponds to a unique attribute in $\mathcal{F}$, each node in V_2 corresponds to an FD $F \in \mathcal{F}$, and each edge $(u, v) \in E$ denotes the fact that the attribute that node u corresponds to is included in the FD that the node v corresponds to. Given the fact that each node in V_1 must be connected to at least a node in V_2, and the fact that $|V_1| \geq |V_2|$, the problem of finding the MAC is equivalent to the problem of finding the minimal hitting set of G (i.e., a minimum cardinality subset of V_1 which covers all vertice in V_2). Finding a minimum hitting set is a NP-hard problem.

Therefore, we adapt the greedy algorithm for the hitting set problem to our setting. Algorithm 6.2 shows the procedure of finding such a minimum attribute cover. The basic idea is to assign a weight to each attribute where the weight equals to the number of FDs in $\mathcal{F}$ that include the attribute. We sort the attributes in $\mathcal{F}$ by their weights, and pick the attributes of the highest weight. After that we delete all FDs in $\mathcal{F}$ that contain the picked attribute, and update the weights of the remaining attributes accordingly. We repeat this procedure until $\mathcal{F}$ becomes empty. The time complexity to find a minimum attribute cover is $O(f^2)$, where f is the number of FDs.

Our *GMM-MFD* algorithm works as following. First, we discover the MAC of the given set of FDs $\mathcal{F}$. Then for each attribute $A \in MAC$, we identify the set of FDs $\mathcal{F}' \subseteq \mathcal{F}$ that contain A. For each FD $F \in \mathcal{F}'$, we apply the *GMM* scheme to find and encrypt necessary cells on attribute A. The complexity of this heuristics approach is $O(f^2 + n^2ktf)$, where n is the number of records in D, f is the number of FDs, t is the average number of sensitive cells, and k is the number of SCs.

6.6 Secure Query Evaluation

In this section, first, we discuss the basic query rewriting method. In order to support range queries, we use order-preserving ecnryption. Then we show the security weakness of the basic approach, and present the improved approach that provides provable security guarantee.

The server receives the client's queries and execute them on $\hat{D}$. In this thesis, we consider the queries that take SQL format. The SQL queries can be specified in the format of $\Pi_Y \sigma_c$, where Π_Y denotes the projection of a relation on attributes Y, and σ_c denotes the value-based constraints of Q. As $\hat{D}$ is partially

encrypted, all the original queries from the authorized clients with access to the sensitive data require rewriting to be executed on $\hat{D}$. Given any query Q, we use $C(Q)$ to denote the value-based constraints in σ_c. For example, consider $Q = \Pi_X \sigma_{(Y=y_1 \ and \ Z=z_1)}$, then $Proj(Q) = \{X\}$, $Sel(Q) = \{Y, Z\}$, and $C(Q) = \{Y = y_1, Z = z_1\}$.

6.6.1 Basic Query Rewriting Approach

Point query. For any point query Q, assume that it contains $g \geq 1$ equality constraints in σ_c, denoted as $C(Q) = \{A_1 = a_1, \ldots, A_g = a_g\}$. Since the values of $A_i (1 \leq i \leq g)$ may be partially encrypted in $\hat{D}$, Q is translated to $\hat{Q}$ by the following:

$$A_i = a_i \rightarrow A_i = a_i \ or \ A_i = \alpha_i, \forall i \in [1, g],$$

where $\alpha_i (1 \leq i \leq g)$ is the ciphertext of a_i. To ensure that the client rewrites the queries in the way consistent with how D is encrypted, she uses the same encryption key obtained from the data owner (or the authorized third-party) to encrypt the plaintext values in Q.

Range query. The rewriting procedure of range queries is similar to that of point queries. For any range query Q, assume that it contains $g \geq 1$ range-based constraints in σ_c, denoted as $C(Q) = \{A_1 \in [a_1, b_1], \ldots, A_g \in [a_g, b_g]\}$, Q is translated to $\hat{Q}$ as following:

$$A_i \in [a_i, b_i] \rightarrow A_i \in [a_i, b_i] \ or \ A_i \in [\alpha_i, \beta_i], \forall i \in [1, g],$$

where α_i and $\beta_i (1 \leq i \leq g)$ are the ciphertext values of a_i and b_i respectively.

For both point and range queries, it is easy to see that the query rewriting

procedure guarantees that $Q(D) = Decrypt(\hat{Q}(\hat{D}))$, for any given query Q, where $Decrypt$ is the decryption function.

Security weakness. The basic query procedure guarantees the correctness of query evaluation. However, it has potential security weakness. In particular, the server may observe a series of queries, and find out that most queries are of the same format. For example, for the point queries, the server may observe that $\hat{Q}$ is always of the format of a disjunction of value-based constraints with the mixture of plaintext and ciphertext values of the same attribute (i.e., $A_i = a_i \; or \; A_i = \alpha_i$). Then the server may guess that α_i is the ciphertext of a_i. Therefore, even without the knowledge of the encryption key, the server learns the mapping between cipher values and plain values.

6.6.2 Secure Query Rewriting Approach

First, we define our security model regarding query rewriting. Formally,

Definition 6.6.1. [θ-security] Given any original query Q and its rewritten query $\hat{Q}$, we say $\hat{Q}$ satisfies *θ-security* if the probability that the server can correctly map a cipher value E to its original value P by observing Q

$$Prob(E \rightarrow P|\hat{Q}) \leq \theta,$$

where θ is a pre-defined value between 0 and 1.

Next, we discuss how to achieve θ-security for both cases that there is a single query and multiple queries.

Single Query

We design the following query rewriting method that can provide θ-security. We define $\ell = [\frac{1}{\theta}]$.

Point queries. For any query Q, and for any value-based constraint $A_i = a_i$ in $C(Q)$, the user randomly picks $\ell - 1$ values $\{a_i^1, \ldots, a_i^{\ell-1}\}$ from the domain of A_i. We say $F = \{a_i, a_i^1, \ldots, a_i^{\ell-1}\}$ is the *plain set* of a_i. Then the value-based constraint $A_i = a_i$ is rewritten as

$$A_i = a_i \; or \; A_i = a_i^1 \ldots \; or \; A_i = a_i^{\ell-1} \; or \; A_i = \alpha_i,$$

where α_i is the ciphertext value of a_i. For any ciphertext value α_i, it always can be mapped to one of the plaintext values in F. Therefore, the server's probability $Prob(\alpha_i \rightarrow a_i)$ must be no larger than θ. It is easy to see that for any point query that consists of g value-based constraints, its rewritten query consists of $g(\ell + 1)$ constraints.

Range queries. The range queries are translated in the similar way. For each constraint $A_i \in [a_i, b_i]$, we pick $\ell - 1$ unique values from the domain of A_i, and call these values as a_i's *plain set*, denoted as $P(a_i)$. Similarly we pick b_i's plain set $P(b_i)$. Let $P(a_i)^-$ be the minimum value of $P(a_i)$ and $P(b_i)^+$ be the maximum value of $P(b_i)$. Then $A_i \in [a_i, b_i]$ is rewritten as $A_i \in [P(a_i)^-, P(b_i)^+] \; or \; A_i \in [\alpha_i, \beta_i]$, where α_i and β_i are the ciphertext values of a_i and b_i respectively. The query rewriting scheme satisfies θ-security. Besides, as $[a_i, b_i] \subseteq [P(a_i)^-, P(b_i)^+]$, the answer of the rewritten query is a superset of the answer of the original answer. For any range query that consists of g value-based constraints, its rewritten query consists of $2g$ constraints.

Multiple Queries

The secure query rewriting method may be vulnerable against a sequence of queries. For example, consider that the user issues two queries, Q_1 and Q_2, both of which contains the constraint $A = a_1$. The user rewrites Q_1 as $\hat{Q_1}$: $\{A = a_1 \ or \ A = a_2^1 \ldots or \ A = a_\ell^1 \ or \ A = \alpha_1\}$. Similarly, Q_2 is translated into $\hat{Q_2}$: $\{A = a_1 \ or \ A = a_2^2 \ldots A = a_\ell^2 \ or \ A = \alpha_1$. The server can observe that $\hat{Q_1}$ and $\hat{Q_2}$ contain the same ciphertext value $A = \alpha_1$. Then the server can intersect the two plain sets $\{a_1, a_2^1, \ldots, a_\ell^1\}$ and $\{a_1, a_2^2, \ldots, a_\ell^2\}$. Let us denote the intersection as I. Then the probability $Prob(\alpha_1 \rightarrow a_1) = \frac{1}{|I|}$. As $|I| \leq \ell$ (possibly equals one), neither $\hat{Q_1}$ nor $\hat{Q_2}$ satisfies θ-security anymore.

Therefore, to guarantee θ-*security* with the presence of a sequence of queries, we require that for each specific plaintext value, its plain set is always fixed. To accomplish that, for any attribute A whose domain size is $|A|$, its domain is partitioned into $[\frac{|A|}{\ell}]$ buckets. For each value a_i of A, the bucket that a_i belongs to is picked as its plain set. To enable the client to perform such rewriting, the client has to acquire the knowledge about the domain values from the data owner. This is a one-time initialization phase. Once the domain knowledge is acquired, it will be used repeatedly for query rewriting.

6.6.3 Query Post-processing at Client Side

Apparently $Q(D) \subseteq \hat{Q}(\hat{D})$, i.e., providing θ-security brings additional overhead for query evaluation. Now the client needs to post-process the query answer from the server to get the real answer of her query.

To extract the correct result from $\hat{Q}(\hat{D})$, we require the server to organize the records in $\hat{Q}(\hat{D})$ in the following way. For each record $r \in \hat{Q}(\hat{D})$, we use $r[Sel(Q)]$

to denote the attribute values on the attribute set $Sel(Q)$, and $\hat{Q}(\hat{D})[Sel(Q)]$ to denote the set of unique values on $Sel(Q)$ in the query result. For each value $v \in \hat{Q}(\hat{D})[Sel(Q)]$, the server creates a list L containing every record $r \in \hat{Q}(\hat{D})$ such that $r[Sel(Q)] = v$. We use (v, L) to denote the attribute value and its record list. The server returns $\hat{Q}(\hat{D})$ in the format of $\mathcal{L} = \{(v, L)\}$. Upon receiving $\mathcal{L}$, the client can efficiently check the attribute values v and retrive the record lists of the values v such that v matches $C(\hat{Q})$. In particular, for each $v \in C(\hat{Q})$, the client decrypts v, and only keeps its corresponding L if the decrypted value of v appears in the original query Q.

6.7 Discussion

In this section, we first discuss how to extend our encryption scheme to deal with the updates on the data. Second, we extend our encryption scheme to defend against the attacks based on CFDs. After that, we explain how to support more complex queries. In the last, we discuss in which scenarios our approach is more preferable to existing methods [116, 17].

6.7.1 Data Updates

After initially outsourcing the encrypted data to a third-party DCaS server, the client may perform some updates over the original data D, and sends the updated data (in the encrypted format) to the server. Let ΔD be the update over D. For simplicity, we only consider *pure-insertion* and *pure-deletion* operations. We use $|\Delta D|$ to denote the number of records to be altered. In case of deletions, $|\Delta D|$ is still a positive number.

By deleting a record, the client does not leak any information. The more

interesting problem is data insertion. A straightforward method is to encrypt ΔD in an incremental way: besides encrypting the sensitive cells in ΔD according to SCs, we apply the GMM algorithm on ΔD to encrypt it. However, doing encryption separately on D and ΔD may not be robust against the FD-attack.

TID	A	B	C
1	a_1	b_1	c_1
2	a_1	b_1	c_1
3	a_1	b_1	c_2
4	a_1	b_1	c_3
5	a_2	b_3	c_3

(a) Original table D ($FD: A \rightarrow B$, SC1: $\Pi_B \sigma_{C=c_1}$) SC2: $\Pi_B \sigma_{C=c_5}$)

TID	A	B	C
6	a_1	b_1	c_4
7	a_2	b_3	c_5

(b) Data update ΔD

TID	A	B	C
1	a_1	$\boldsymbol{\beta_1}$	c_1
2	a_1	$\boldsymbol{\beta_1}$	c_1
3	$\boldsymbol{\alpha_1}$	b_1	c_2
4	$\boldsymbol{\alpha_1}$	b_1	c_3
5	a_2	b_3	c_3

(c) Encrypted table $\hat{D}$

TID	A	B	C
6	a_1	b_1	c_4
7	a_2	$\boldsymbol{\beta_3}$	c_5

(d) Encrypted data update $\hat{\Delta D}$

Figure 6.6: An Example of FD Attack if D and ΔD are encrypted separately

Example 6.7.1. *Consider the base table D in Figure 6.6 (a), which has the FD: $A \rightarrow B$. Also consider its update ΔD shown in Figure 6.6 (b). The encrypted version of D and ΔD, notated as $\hat{D}$ and $\hat{\Delta D}$, by applying the GMM algorithm individually is shown in Figure 6.6 (c) and (d) respectively. It is easy to observe that tuple T_6 in $\hat{\Delta D}$ serves as an evidence tuple of tuples T_1 and T_2 in $\hat{D}$, while T_5 in $\hat{D}$ is an evidence tuple of Tuple T_7 in $\hat{\Delta D}$. These evidence tuples will bring the leakage of the encrypted cells in T_1, T_2, and T_5.*

To encrypt the data updates with provable security guarantee, a straightforward approach is to execute the GMM algorithm on $\hat{D}+\Delta D$, aiming to encrypt the necessary cells in ΔD with the minimal overhead. The complexity of this approach is $O(f^2 + (n' + n)^2 kf)$, where f is the number of FDs, k is the number of SCs, and n and n' are the number of records in $\hat{D}$ and ΔD respectively. Therefore, our

GMM algorithm is able to provide comparable encryption overhead as the optimal approach, since the encryption overhead of ΔD is still close to the minimal overhead, as long as $\frac{n'}{n}$ is small.

One weakness of the straightforward approach is that it still needs to access $\hat{D}$, which can be inefficient if $\hat{D}$ is much larger than that of ΔD. To make the process more efficient, we design an alternative method which does not need to revisit $\hat{D}$. The key idea consists of two steps. First, we find the minimum attribute cover $\mathcal{A}$ of $\mathcal{F}$ by using the greedy algorithm in Section 6.5. Second, for each record $r \in \Delta D$, we encrypt $r[\mathcal{A}]$. By this approach, the complexity is reduced to $O(f^2 + n')$, while the encryption overhead is bounded by $n'|\mathcal{A}|$, where $|\mathcal{A}|$ is the number of attributes in $\mathcal{A}$.

6.7.2 Conditional Functional Dependency (CFD) Constraints

Besides FDs, there is another important type of attribute associations named *conditional functional dependency (CFD)*. Conditional functional dependency binds specific values with functional dependencies. An example of CFD is the constraint *[Disease='ovarian cancer'] $\rightarrow$ [Gender='Female']*. Formally, a CFD [13] on a given dataset D is a pair $\varphi(X \rightarrow Y, T_p)$, where (1) X, Y are sets of attributes from D; (2) $X \rightarrow Y$ is a standard FD; and (3) T_p is a tableau with all attributes in X and Y, where for each A in X or Y and each record $r \in T_p$, $r[A]$ is a constant in the domain of A, or an unnamed variable "_". A CFD $\varphi(X \rightarrow Y, T_p)$ is called a *constant* CFD if its pattern record r_p consists of constants only, while it is called a *variable* CFD if $T_p[Y] = \_$, i.e., the right-hand side (RHS) of its pattern tuple is the unnamed variable _. For simplicity, we only consider constant CFDs in this thesis.

CFDs provide an inference channel for the attacker to interpret the sensitive

information in a private dataset, in the similar way to the FD attack. Since the CFD attack is very similar to the FD attack, the way to design robust schemes against the CFD attack is similar to that to prevent the FD attack. In this section, we discuss the encryption algorithms against the CFD attack in terms of their major difference from that against the FD attack.

First, we discuss the data-independent robustness check of the basic scheme against the CFD attack. We use $C(S)$ and $C(F)$ to denote the selection conditions in the SC S and the CFD F respectively. For instance, consider the SC $S = \Pi_X \sigma_{(Y=y_1 \ and \ Z=z_1)}$ and the CFD $F : X = x_1 \rightarrow Y = y_1$, then $C(S) = \{Y = y_1, Z = z_1\}$, and $C(F) = \{X = x_1, Y = y_1\}$. We have the following theorem.

Theorem 15. Given a dataset D, a set of security constraints $\mathcal{S}$, and a CFD $F : X = x \rightarrow Y = y$ of D, the basic scheme of D is robust against the CFD attack w.r.t. F if one of the following conditions is met: (1) $\forall S \in \mathcal{S}$, $X \cap Proj(S) \neq \emptyset$; (2) $Y \not\subseteq \cup_{S \in \mathcal{S}} Proj(S)$; (3) $\forall S \in \mathcal{S}$, $Sel(S) \neq \emptyset$, $\cup_{S \in \mathcal{S}} Sel(S) \subseteq X \cup Y$ and $\cup_{S \in \mathcal{S}} C(S) \cap C(F) = \emptyset$.

Compared with Theorem 12 for the FD attack, Condition (3) of Theorem 15 additionally requires that no sensitive cell should be covered by the CFDs. The robustness checking of the basic scheme can be achieved by checking whether the basic scheme is robust against each CFD.

Unlike the FD attack that requires the existence of evidence records for the inference, $CFDs$ provide the value association information between X and Y. Therefore, there is no need to find evidence records from the dataset for additional encryption. This makes the design of robust scheme much easier than that of the FD attack. Next, we discuss the details of our scheme.

We first discuss the case that there exists only one CFD. Let $F(D)$ be the

set of records that comply with the given CFD $F : X = x \rightarrow Y = y$. Then, for any SC S whose basic scheme is not robust against the CFD attack, for each record r that contains a sensitive cell in $S(D) \cap F(D)$, we randomly pick an attribute $A \in X$, and encrypt $r[A]$. It is not necessary that different records r and r' encrypt at the same attribute. If there exist multiple SCs $\{S_1, \ldots, S_k\}$, we find the records R that contain at least a sensitive cell in $\cup_{i=1}^{k}(S_i(D) \cap F(D))$, and encrypt R in the same way as for the single SC.

If there exist multiple CFDs $\mathcal{F}$, let $L = \cup_{F \in \mathcal{F}} LHS(F)$. First, we identify the minimum attribute cover of L. Let the cover be MAC. Second, we find the set of records R that contain at least a sensitive cell in $\cup F \in \mathcal{F} \cup_{i=1}^{k} (S_i(D) \cap F(D))$. Then for each record $r \in R$, we randomly pick an attribute $A \in MAC$, and encrypt $r[A]$. The complexity of the scheme is $O(f^2 + nfk)$, where f is the number of FDs, n is the number of records in D, and k is the number of SCs.

6.7.3 Complex Queries

Besides the point and range queries, our encryption scheme also provides support for complex queries like *ORDER BY*, *aggregations (MAX, MIN, COUNT)*, and *GROUP BY*. The main challenge is that the query evaluation needs to handle both plaintext and ciphertext values, due to the fact that the data is partially encrypted. In order to support complex queries, we require that the plaintext and ciphertext values do not share the domain. In this way, the queries can be executed on the plaintext and ciphertext values separately at the server side first. After that, the client decrypts the query answer and post-processes it to retrieve the result.

Assume that, in the encrypted dataset $\hat{D}$, on an attribute A, we have a set of plaintext values $\mathcal{P}_A$ and a set of ciphertext values $\mathcal{E}_A$. Next, we will discuss the

query processing for each type of queries.

ORDER BY. Given a query Q with *ORDER BY A*, the client first rewrites it into two queries, $Q_{\mathcal{P}}$ and $Q_{\mathcal{E}}$. $Q_{\mathcal{P}}$ orders the records according to the plaintext values on attribute A, while $Q_{\mathcal{E}}$ orders the records by the ciphertext values on attribute A. The server sends the result $Q_{\mathcal{P}}(\hat{D})$ and $Q_{\mathcal{E}}(\hat{D})$ to the client. The client first decrypts $Q_{\mathcal{E}}(\hat{D})$. Then it uses merge sort to re-order the decryption results with $Q_{\mathcal{P}}(\hat{D})$.

Aggregation. Given a query Q with $MAX(A)$ or $MIN(A)$, the server first retreives the max/min value from $\mathcal{P}_A$ and $\mathcal{E}_A$. The client decrypts $MAX(\mathcal{E}_A)$ (or $MIN(\mathcal{E}_A)$) and compares it with $MAX(\mathcal{P}_A)$ (or $MIN(\mathcal{P}_A)$) to get the maximum (or minimum) attribute value. For a query Q with $COUNT(A)$, the server just needs to execute Q on $\hat{D}$. However, for a query with distinct count *COUNT(DISTINCT A)*, the server first needs to execute the query *Q': SELECT DISTINCT A FROM* $\hat{D}$. Then the client decrypts $Q'(\hat{D})$ to get the number of distinct values.

GROUP BY. Given a query Q with *GROUP BY A*, the server first retrieves $Q(\hat{D})$ and sends it to the client. The client decrypts the values on attribute A. Then it re-groups the results based on decrypted values.

6.8 Experiments

6.8.1 Setup

Experiment environment. We implement our approaches in Java. All the experiments were executed on a machine with 2.4GHz Intel Core i5 CPU and 4GB memory running Mac OS X 10.9. The time performance and encryption overhead are measured as the average of five runs.

Datasets. We execute our experiments on two real-world datasets named *Adult*

dataset[1] and the *Orders* table from the TPC-H benchmark dataset. To measure the scalability of our approaches, we construct a set of datasets of various sizes, by using *Adult* and *Orders* datasets as the seed. The original *Adult* datasets contains $32,561$ records. The datasets containing $64K$, $128K$ and $256K$ records are constructed by repeatedly picking a record from the original dataset in a random fashion and inserting a copy of the picked record into the new dataset, until the data size reaches the requirement. Similarly, the *Orders* dataset originally have 1.5 million records. We build a set of datasets from the *Orders* dataset containing $0.3M$, $0.6M$, $0.9M$, and $1.2M$ records by taking a subset of records from the original $Orders$ data.

Let n be the number of records, m the number of attributes, f the number of functional dependency, t the average number of sensitive cells in each randomly-generated SC, and w the average number of sensitive and evidence records for each sensitive cell. The number t is evaluated as following. For each SC, we count its number of sensitive cells sc. Then $t = \frac{\sum_{i=1}^{k} sc_i}{k}$, where k is the number of SCs. And $w = \frac{\sum_{i=1}^{k}\sum_{j=1}^{sc_i}(|s_{i,j}|+|e_{i,j}|)}{\sum_{i=1}^{k} sc_i}$, where $|s_{i,j}|$ ($|e_{i,j}|$, resp.) is the number of sensitive records (evidence records, resp.) for the j-th sensitive cell of the i-th SC. Intuitively, higher t value leads to larger portion of unique data values that is considered as sensitive and thus must be encrypted, whereas higher w value means each sensitive cell is associated with more evidence records that need to be encrypted. We give the data description in Table 6.7.

There are 78 FDs in the *Adult* dataset. These FDs overlap significantly at the left hand side. In particular, the *fnlwgt* attribute appears in 97.4% of FDs. While in the *Orders* dataset, the FDs overlap greatly on the right hand side. For exam-

[1]Adult dataset is available at UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/index.html.

Data	n	m	f	t	w
Adult_32K	32,000	15	78	3877.4	1.016
Adult_64K	64,000	15	78	3989.7	2.03
Adult_128K	128,000	15	78	3990.4	4.059
Adult_256K	256,000	15	78	3988.9	8.129
Orders_0.3M	300,000	9	10	125.8	126.82
Orders_0.6M	600,000	9	10	191.1	306.51
Orders_0.9M	900,000	9	10	229.7	383.12
Orders_1.2M	1,200,000	9	10	259.6	459.99
Orders_1.5M	1,500,000	9	10	288.7	508.37

Figure 6.7: Dataset Description

ple, the right hand side of the 10 FDs only contain two attributes (*OrderStatus* and *ShipPriority*). In the experiments, we randomly pick a subset of these FDs. An important feature of the *Orders* dataset is that some of its attributes have a small data domain, with each data instance of high frequency. For example, the *OrderStatus* and *OrderPriority* attributes only have 3 and 5 unique values respectively. We will discuss why this feature impacts the encryption performance.

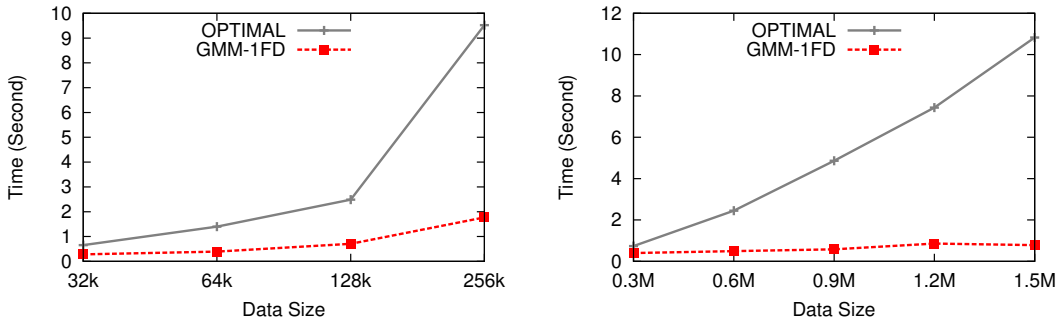
Approaches In the experiments, we implement the following two methods and evaluate them on the datasets.

- **GMM**: our heuristics approach;
- **OPTIMAL**: the exhaustive search algorithm that finds the solution with the minimum encryption overhead.

We take *OPTIMAL* as the baseline methods and compare their performance with our approach.

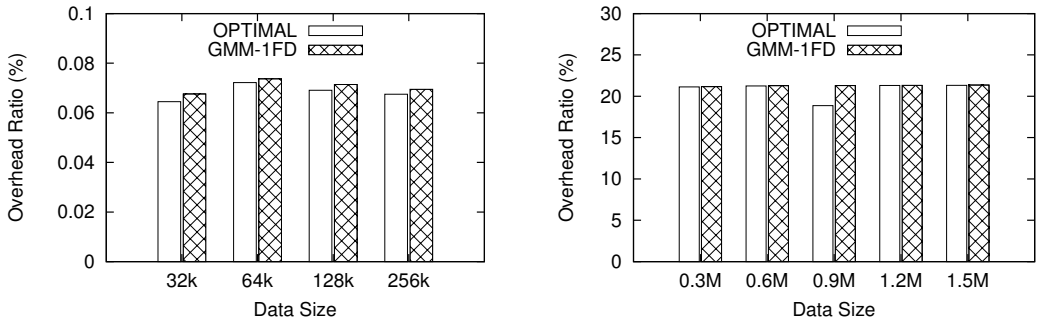
6.8.2 One Single FD

For this set of experiments, we randomly pick a single FD and generate 10 SCs from the two datasets. We follow Theorem 11 to ensure that the generated SCs are potentially unsafe, meaning their basic scheme is not robust against the FD attack.



(a) Adult dataset (b) Orders dataset

Figure 6.8: Time Performance of One Single FD



(a) Adult dataset (b) Orders dataset

Figure 6.9: Encryption Overhead of One Single FD

Time Performance. We measure the time of our *GMM* algorithm on datasets of various sizes. We report the results in Figure 6.8. From the two figures, we observe that our *GMM* approach is much more efficient than the optimal scheme.

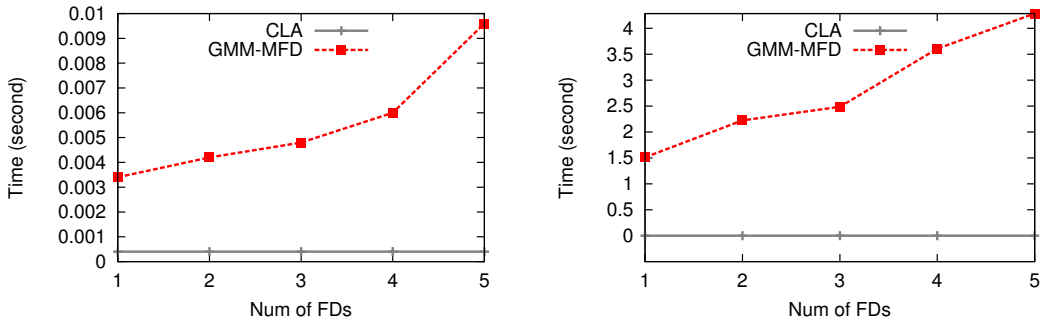
In all circumstances, our *GMM* approach enjoys a ten times speedup compared with *OPTIMAL*. Another observation is that the time performance increases linearly with the data size. This is consistent with our expectation. One interesting observation is that even though the *Orders* datasets are much more larger than the *Adult* datasets, the time performance is comparable. This is mainly because the SCs of *Adult* datasets cover a large number of sensitive cells (close to $4,000$), while the SCs of *Orders* datasets cover a relatively smaller number of sensitive cells (no more than 300). Therefore, even though the *Adult* dataset is smaller, there is no substantial difference in the execution time on the two datasets.

Encryption Overhead. To measure the encryption overhead, we define *encryption overhead ratio* as $o = \frac{h}{n}$, where h is the encryption overhead, and n is the number of records of the dataset. We report the overhead of both our *GMM* approach and the *OPTIMAL* approach in Figure 6.9. First, we observe that our *GMM* approach delivers comparable overhead as *OPTIMAL*. In almost all the cases, the overhead brought by *GMM* is no greater than the optimal overhead by 1% (with only one exception in the *Orders* dataset of 0.9 million records). This encouraging result verifies that our *GMM* approach can efficiently find near-optimal encryption solution to defend against the FD attack. Second, the encryption overhead ratio is small, especially on the *Adult* dataset (no larger than 0.1%). This shows that our approach can be scaled to large datasets with small amounts of encryption overhead. Considering the state-of-the-art solution [116] that simply encrypts all data cells of the RHS of a FD, our *GMM* approach's encryption overhead is significantly smaller. Third, the encryption overhead ratio of the *Adult* dataset is much smaller than that of the *Orders* dataset. The reason why the *Adult* dataset has a small overhead ratio is that each sensitive cell is associated with only a small number (below 10) of sensitive and evidence records. On the contrary, even though the

number of sensitive cells on the *Orders* dataset is smaller, the overhead ratio is larger because its sensitive cells are associated with a large number of sensitive and evidence records.

6.8.3 Multiple FDs

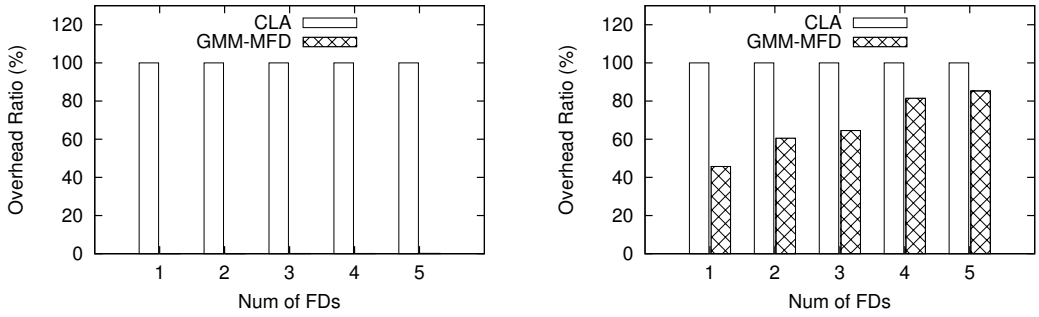
For this set of experiments, first, we pick a number of FDs randomly from the FDs of the two datasets (*Adult*: 78 FDs, *Orders*: 10 FDs). Then for each set of FDs, we randomly generate ten unsafe SCs. We measure the time performance and the encryption overhead for various number of FDs. We do not report the performance of *OPTIMAL* as its execution fails in the presence of multiple FDs due to inadequate memory space. We mainly compare the performance of state-of-the-art *CLA* algorithm with our *GMM-MFD* approach.



(a) Adult dataset (32K) (b) Orders dataset (1.5M)

Figure 6.10: Time Performance of Multiple FDs

Time Performance. We vary the number of FDs from 1 to 5. Our results reported in Figure 6.10 show that the time performance of *CLA* does not change with the number of FDs. This is because that *CLA* mainly relies on the size of minimum attribute covers (MAC). We observe that the picked FDs on *Adult* dataset overlap largely at the left hand side, while the FDs on *Orders* dataset overlap mainly at



(a) Adult dataset (32K) (b) Orders dataset (1.5M)

Figure 6.11: Encryption Overhead of Multiple FDs

the right hand side. This leads to the fact that the discovered MAC is always the same, which only contains the *fnlwgt* attribure for *Adult* dataset and *ShipPriority* attribute for *Orders* dataset. Therefore, the time performance is constant for the *CLA* approach. Regarding our *GMM-MFD* approach, its time performance increases with the rise of the number of FDs, as our algorithm is linear to the number of FDs. Nevertheless, our *GMM-MFD* approach is highly efficient. For example, in the presence of 5 FDs and a dataset of 1.5 million records, the *GMM-MFD* approach finishes within 5 seconds.

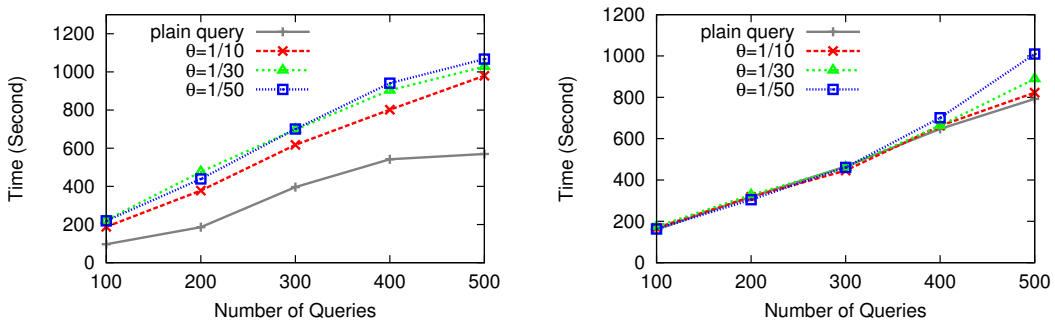
Encryption Overhead. We measure the encryption overhead ratio of both *CLA* and *GMM-MFD* approaches with various number of FDs. The encryption overhead ratio is measured in the same way as for the single FD case (Section 6.8.2). The results are reported in Figure 6.11. The overhead ratio of the *CLA* scheme is always 100%. We did further analysis on this, and found out that the discovered MAC is always the same, which only consists of the *fnlwgt* attribute for *Adult* dataset, and the *ShipPriority* attribute for *Orders* dataset. *CLA* basically encrypts all data values of these attributes, leading 100% overhead ratio. In contrast, the overhead ratio of the *GMM-MFD* scheme is much smaller compared to that of *CLA*

scheme. In particular, the overhead ratio by our approach is negligible for *Adult* dataset, which is at most 0.2% (could not be shown in Figure 6.11 (a)). On the other hand, the encryption overhead ratio of *Orders* dataset is higher (40% - 80%). The reason of higher overhead ratio is due to the fact that in *Orders* dataset of size $1.5M$, each sensitive cell is associated with approximately 500 sensitive and evidence records at average (as shown in Figure 6.7). This means that encrypting one sensitive cell may require to encrypt 500 evidence records in addition for the worst case. Therefore, the encryption overhead relies on the data distribution. We also observe that the encryption overhead ratio increases with the number of FDs. This is not surprising as our approach encrypts more evidence cells for more adversarial FDs.

6.8.4 Query Evaluation Overhead

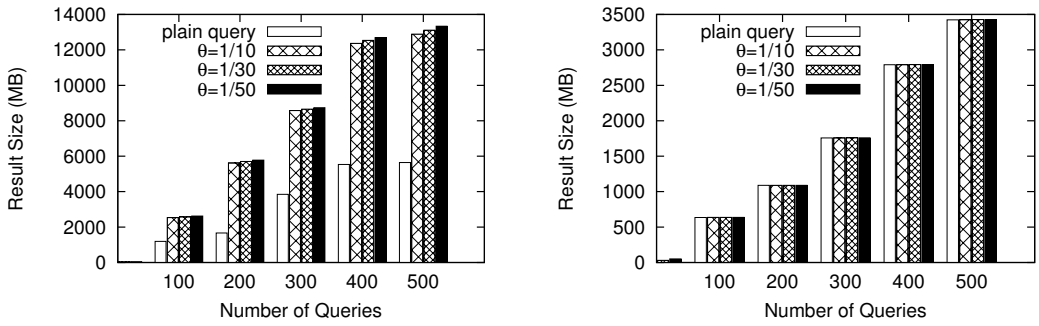
We randomly generate 500 point and range queries on the *Orders* dataset, and measure both the query execution time and result size for various degree of security protection. We vary the security threshold $\theta = \frac{1}{10}, \frac{1}{30}, \frac{1}{50}$. We compare the query execution time and result size with that of evaluating the original query on the dataset before encryption.

Time Performance First, we measure the query execution time on the original dataset and the encrypted dataset with different θ values. The results are displayed in Figure 6.12. Regarding the point query, smaller θ value yields longer execution time. This is expected as intuitively, smaller θ value leads to higher security and thus more query evaluation time. However, there is no sharp increase in query execution time when θ increases. To understand this, we studied the rewritten queries and its answers, and found out that a large portion of plain sets do not



(a) Point query (b) Range query

Figure 6.12: Query evaluation time on *Orders_1.5M*



(a) Point query (b) Range query

Figure 6.13: Query result size on *Orders_1.5M*

have corresponding instances in the dataset. Therefore, the query execution time does not rise much given the fact that these values return empty result.

For range queries, the query execution time (reported in Figure 6.12 (b)) stays relatively stable for different θ values. This is because the *Orders* dataset has two important properties: (1) the data values are of uniformly distributed over the whole domain, and (2) the frequency of data values is also of uniform distribution. We also observe that our original queries are of large range size (93666 at average). Then adding $[\frac{1}{\theta}]$ data values to the query's range does not extend it much. For example, we observe a query with range $[176281, 197721]$, whose rewrit-

ten range is $[176273, 197728]$ for $\theta = \frac{1}{10}$. The original query's range size is 21440, and the rewritten query's range size is 21455. In another word, we only extend the range size by 0.07%. Even though when $\theta = \frac{1}{50}$, the range increase is only 0.4%. Therefore, the query evaluation time does not change much with the increase of θ.

Result Size We report the query result size on the original and encrypted dataset in Figure 6.13. We have the similar observation as the query time performance. For the point query, even if we set the θ value to be 0.02, the query result size is only expanded with a factor of 2 compared to the result from the original data. Regarding the range query, the result size of the rewritten queries is very close to that of the original query. This shows that our query rewriting approach provides provable security with marginal overhead.

Chapter 7
Frequency-hiding Dependency-preserving Encryption for Outsourced Databases

In this chapter, we discuss how to protect the data privacy in the outsourced data inconsistency repair computations when the server has frequency knowledge about the original dataset. We consider the scenario where the data is evolving and may not comply with the functional dependencies (FDs) in the evolvement process. We assume that the client first sends the initial version of the data where the data is consistent with all the FDs. The server discovers the FDs from the initial data, and uses the FDs are the guidelines to repair inconsistency issues when the data is evolving. As the FDs are critical for the server to detect the inconsistency and generate the repair solution, we design a frequency hiding, FD-preserving probabilistic encryption scheme, named F^2, that enables the service provider to discover the FDs from the encrypted dataset and is secure against the frequency analysis attack. This work has been submitted to the International Conference on Data Engineering (ICDE), 2017.

7.1 Preliminaries

7.1.1 Attack and Security Model

We consider the *curious-but-honest* server, i,e., it follows the outsourcing protocols honestly (i.e., no cheating on storage and computational results), but it is curious to extract additional information from the received dataset. Since the server is potentially untrusted, the data owner sends the encrypted data to the server. The data

owner considers the true identity of every cipher value as the sensitive information which should be protected.

We consider two adversarial attacks: (1) the *frequency analysis attack* that is based on frequency distribution information, and (2) the *chosen plaintext attack* that is based on the FD-preserving property of F^2, that try to break the encryption on the outsourced data.

Frequency Analysis Attack (FA)

Frequency analysis attack is one of the most well-known example of an inference attack to break classical ciphers. In this thesis, we assume that the attacker may possess the frequency distribution knowledge of data values in D. In reality, the attacker may possess approximate knowledge of the value frequency in D. However, in order to make the analysis robust, we adopt the conservative assumption that the attacker knows the exact frequency of every plain value in D, and tries to break the encryption scheme by utilizing such frequency knowledge. In particular, let $\mathcal{P}$ and $\mathcal{E}$ be the plaintext and ciphertext values. Consider a ciphertext value e that is randomly chosen from $\mathcal{E}$. Let $freq_{\mathcal{E}}(e)$ be the frequency of e, and $freq(\mathcal{P})$ be the frequency distribution of $\mathcal{P}$. Then e, $freq_{\mathcal{E}}(e)$ and $freq(\mathcal{P})$ are given to the adversary $\mathcal{A}^{freq}$. We formally define the following adversarial experiment Exp^{FA}_{Π} on an encryption scheme Π.

$$\textbf{Experiment } EXP^{FA}_{\Pi}()$$

$$p' \leftarrow A^{freq_{\mathcal{E}}(e),freq(\mathcal{P})}$$

$$Return\ 1\ if\ p' = Decrypt(k, e)$$

$$Return\ 0\ otherwise$$

Intuitively, Exp_{Π}^{FA} returns 1 if the attacker can correctly infer the plaintext of e based on the frequency distribution of plaintext and ciphertext values. Let $Exp_{\Pi}^{FA}(A)$ denote the output of the adversary by FA against the encryption scheme Π. We define the FA *advantage* as

$$Adv_{\Pi}^{FA}(A) = Prob(EXP_{\Pi}^{FA}(A) = 1).$$

Based on the FA advantage, we formally define α-*security*, the security model against FA.

Definition 7.1.1. [α-Security against FA] *An encryption scheme Π is α-secure against FA if for every adversary A it holds that $Adv_{\Pi}^{FA}(A) \leq \alpha$, where $\alpha \in (0, 1]$ is user specified.*

Intuitively, the smaller α is, the stronger security that Π is against the FA. In this thesis, we aim at designing an encryption scheme that provides α-security.

FD-preserving Chosen Plaintext Attack (FCPA)

Indistinguishability under chosen plaintext attack (IND-CPA) is a property of many encryption schemes. Intuitively, if a cryptosystem possesses IND-CPA (and thus semantically secure under chosen plaintext attack), then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. Since F^2 preserves FDs, it cannot provide indistinguishability under the chosen-plaintext attack (IND-CPA). Therefore, we define a new security model named *indistinguishability against FD-preserving chosen plaintext attack* (IND-FCPA). An IND-FCPA encryption scheme reveals the FD, but everything else remains semantically secure. We first introduce the FCPA adversary. The FCPA adversary is designed in the

similar fashion as the left-or-right game [10] for symmetric encryption schemes. Intuitively, the adversary chooses two plaintext datasets m_0 and m_1. We require both m_0 and m_1 have the same size and FDs. The challenger encrypts one of them at random. The adversary tries to guess which one was encrypted by making a sequence of queries $(m_0^1, m_1^1), \ldots, (m_0^q, m_1^q)$ on the encrypted dataset. Specifically, let $LR(., ., b)$ denote the function that on inputs m_0, m_1 returns $m_b (b \in \{0, 1\})$. Given a dataset D with a FD $F : X \rightarrow Y$, for a given symmetric encryption scheme Π, $b \in \{0, 1\}$, and a polynomial-time adversary A, consider the following experiment:

$$\begin{aligned} \textbf{Experiment } & EXP_{\Pi}^{IND-FCPA-b}() \\ & d \leftarrow A^{Encrypt(k,LR(.,.,b)),F} \\ & Return\ d \end{aligned}$$

Let $Exp_{\Pi}^{IND-FCPA-1}(A)$ ($Exp_{\Pi}^{IND-FCPA-0}(A)$, resp.) denote that the adversary A outputs m_1 (m_0, resp.). We define the IND-FCPA advantage as

$$\begin{aligned} Adv_{\Pi}^{FCPA}(A) = & \ |Prob(Exp_{\Pi}^{IND-FCPA-1}(A) = 1) \\ - & \ Prob(Exp_{\Pi}^{IND-FCPA-0}(A) = 1)|. \end{aligned}$$

Now we are ready to define IND-FCPA.

Definition 7.1.2. [Indistinguishability against FD-preserving Chosen Plaintext Attack (IND-FCPA)] *An encryption scheme Π is indistinguishable against FD-preserving chosen plaintext attack if for any polynomial-time adversary A, it holds that the IND-FCPA advantage is negligible in λ, i.e., $Adv_{\Pi}^{FCPA}(A) = negl(\lambda)$, where λ is a pre-defined security parameter.*

Intuitively, IND-FCPA requires that for any polynomial-time adversary that

has oracle access to the encryption function, it cannot differentiate the ciphertext value of a pair of plaintext values with a non-negligible probability.

7.1.2 Our Approaches in a Nutshell

We design F^2, a frequency-hiding, FD-preserving encryption scheme based on probabilistic encryption that allows the data owner to encrypt the data without the awareness of any FD in the original dataset. To defend against the frequency analysis (FA) attack on the encrypted data, we apply the *probabilistic encryption* in a way that the frequency distribution of the ciphertext values is always flattened. The complexity of F^2 is much cheaper than FD discovery locally. To preserve FDs, we discover *maximal attribute sets* (MASs) on which FDs *potentially* exist and apply FD-preserving encoding on each MAS. Multiple MASs may introduce *conflicts* due to encoding on each MAS independently, no matter whether these MASs overlap or not. We provide algorithms for conflict resolution, promising that the incurred amounts of overhead are bounded. Applying probabilistic encryption may introduce *false positive* FDs that do not exist in the original dataset but in the encrypted data. We eliminate such false positive FDs by adding a small amount of artificial records. In this way, F^2 guarantees to preserve all FDs in the original dataset, while avoids to introduce any false positive FDs.

7.2 FD-Preserving Encoding

Our goal is to design efficient *FD-preserving* data encryption methods that enable the FDs in the original dataset to be kept in the encrypted dataset. A naive method is to apply a simple deterministic encryption scheme (i.e. the same plaintext values are always encrypted as the same ciphertext for a given key) on the attributes

ID	A	B	C
r_1	a_1	b_1	c_1
r_2	a_1	b_1	c_2
r_3	a_1	b_1	c_2
r_4	a_1	b_1	c_3
r_5	a_2	b_2	c_3
r_6	a_2	b_2	c_4

(a) Base table D ($FD : A \rightarrow B$, $A \not\rightarrow C$, $B \not\rightarrow C$)

ID	A	B	C
r_1	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_1^1$
r_2	$\hat{a}_1^1$	$\hat{b}_1^2$	$\hat{c}_2^1$
r_3	$\hat{a}_1^2$	$\hat{b}_1^3$	$\hat{c}_2^2$
r_4	$\hat{a}_1^2$	$\hat{b}_1^4$	$\hat{c}_3^1$
r_5	$\hat{a}_2^1$	$\hat{b}_2^1$	$\hat{c}_3^2$
r_6	$\hat{a}_2^2$	$\hat{b}_2^2$	$\hat{c}_4^1$

(b) $\hat{D}_1$: probabilistic encryption on A, B, C individually (ciphertext frequency $>= 1$) $A \rightarrow B$ is not preserved;

ID	A	B	C
r_1	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_1^1$
r_2	$\hat{a}_1^2$	$\hat{b}_1^2$	$\hat{c}_2^1$
r_3	$\hat{a}_1^3$	$\hat{b}_1^3$	$\hat{c}_2^2$
r_4	$\hat{a}_1^4$	$\hat{b}_1^4$	$\hat{c}_3^1$
r_5	$\hat{a}_2^1$	$\hat{b}_2^1$	$\hat{c}_3^2$
r_6	$\hat{a}_2^2$	$\hat{b}_2^2$	$\hat{c}_4^1$

(c) $\hat{D}_2$: probabilistic encryption on A, B, C individually (ciphertext frequency = 1) Introduce false positive FDs $A \rightarrow C$, $B \rightarrow C$

ID	A	B	C
r_1	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_1^1$
r_2	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_2^1$
r_3	$\hat{a}_1^2$	$\hat{b}_1^2$	$\hat{c}_2^2$
r_4	$\hat{a}_1^2$	$\hat{b}_1^2$	$\hat{c}_3^1$
r_5	$\hat{a}_2^1$	$\hat{b}_2^1$	$\hat{c}_3^2$
r_6	$\hat{a}_2^2$	$\hat{b}_2^2$	$\hat{c}_4^1$

(d) $\hat{D}_3$: probabilistic encryption on attribute set $\{A, B\}$ and $\{C\}$ (ciphertext frequency $>= 1$) FD-preserving No false positive FDs

Figure 7.1: Examples of (good and bad) probabilistic encryption schemes

with FDs. For instance, consider the base table D in Figure 7.1 (a) that has a FD: F: $A \rightarrow B$. Suppose that D is encrypted by applying a deterministic encryption scheme on each individual cell of D. Apparently F is preserved in the encrypted dataset. However, this naive method has drawbacks. One of the main drawbacks is that since the encryption preserves the frequency distribution, the deterministic encryption scheme is vulnerable against the frequency analysis attack. The attacker can easily map the ciphertext to the plaintext values based on their frequency.

A straightforward solution to defend against the frequency analysis attack is to use the probabilistic encryption schemes (i.e., the same plaintext values are

encrypted as different ciphertext values) instead of the deterministic encryption schemes, aiming to hide frequency. However, adapting the existing probabilistic encryption schemes to be FD-preserving is far more than trivial. The following example will show that applying the probabilistic encryption scheme in a careless way may either destroy FDs completely or introduce *false positive* FDs (i.e., the FDs that do not exist in D).

Example 7.2.1. Consider the table D shown in Figure 7.1 (a). Figure 7.1 (b) shows an instance $\hat{D}_1$ by applying the probabilistic encryption scheme at the attributes A, B and C individually, without requiring that each plaintext value must be encrypted differently (i.e., the frequency of each ciphertext value can be more than 1, e.g., $\hat{a}_1^1$ in Figure 7.1 (b)). We use the notation $\hat{a}_i^j$ ($\hat{b}_i^j$, $\hat{c}_i^j$, resp.) as the j-th unique ciphertext value of a_i (b_i, c_i, resp.) by the probabilistic encryption scheme. Now the frequency analysis attack fails since the frequency distribution of the plaintext values is dissimilar to that of the ciphertext values. However, the FD $A \rightarrow B$ does not hold on $\hat{D}_1$ anymore, as the same ciphertext value $\hat{a}_1^1$ of attribute A is associated with two different cipher values $\hat{b}_1^1$ and $\hat{b}_1^2$ of attribute B. Now let's consider another encryption scheme $\hat{D}_2$ (Figure 7.1 (c)) that still applies the probabilistic encryption scheme at the attributes A, B and C individually, but requiring that all ciphertext values must be unique (i.e., the frequency of each ciphertext value is always 1). Now $A \rightarrow B$ holds on $\hat{D}_2$. However, since each ciphertext value is unique, there are also two false positive FDs $A \rightarrow C$ and $B \rightarrow C$ in $\hat{D}_2$, which do no exist in D.

Example 7.2.1 shows that the probabilistic encryption scheme, if it is applied on individual attributes, fails to be a FD-preserving solution, regardless of the frequency of the ciphertext values. This raises the first challenge of designing FD-preserving probabilistic encryption schemes: what is the appropriate encryption

granularity on which the probabilistic encryption is applied? If we revisit Example 7.2.1, a correct FD-preserving encryption scheme is to consider FD attributes $\{A, B\}$ together as a *super attribute*, and apply the probabilistic encryption scheme on the super attribute (as shown in Figure 7.1 (d)). FDs are guaranteed to be preserved by this approach. But there lies the challenge of finding those FD attributes that will be considered as the super attribute for encryption. Previous study has shown that FD discovery needs intensive computational efforts [62]. We assume that the data owner may lack of computational resources and/or knowledge to discover FDs by herself. Therefore, we assume that FDs are not available for the encryption. In other word, the data owner does not know which attributes will be considered as a super attribute for FD-preserving encryption.

The key to designing a FD-preserving encryption algorithm is to first identify the set of attributes on which the probabilistic encryption scheme is applied. We have the following theorem to show that for any FD F, if the probabilistic encryption scheme is applied on the attribute set that includes both $LHS(F)$ and $RHS(F)$, F is still preserved.

Theorem 16. *Given a dataset D and any FD F of D, let the attribute set $\mathcal{A}$ be the attribute sets on which the probabilistic encryption scheme is applied on, and let $\hat{D}$ be the encryption result. Then F always holds in $\hat{D}$ if $LHS(F) \cup RHS(F) \subseteq \mathcal{A}$.*

Proof. Assume that there is a FD: $A \rightarrow B$ which holds in D but not in $\hat{D}$. It must be true that there exists at least one pair of records r_1, r_2 such that $r_1[A] = r_2[A]$ and $r_1[B] = r_2[B]$ in D, while in $\hat{D}$, $\hat{r_1}[A] = \hat{r_2}[A]$ but $\hat{r_1}[B] \neq \hat{r_2}[B]$. According to the HS scheme which is applied on an attribute set $\mathcal{A}$ s.t. $LHS(F) \cup RHS(F) \subseteq \mathcal{A}$, there are two cases. First, if r_1 and r_2 are taken as the same instance, then $\hat{r_1}$ and $\hat{r_2}$ have the same value in every attribute. It cannot happen as $\hat{r_1}[B] \neq \hat{r_2}[B]$. Second,

if r_1 and r_2 are taken as different instances, then it must be true that $\hat{r_1}[A] \neq \hat{r_2}[A]$ and $\hat{r_1}[B] \neq \hat{r_2}[B]$ according to Requirement 2 of the HS scheme. So $\hat{r_1}$ and $\hat{r_2}$ cannot break the FD: $A \rightarrow B$ in $\hat{D}$. □

Example 7.2.2. *To continue our running example, consider the base table in Figure 7.1 (a). Figure 7.1 (c) shows an example table $\hat{D}_2$ of applying the probabilistic encryption scheme on the attribute set $\{A, B\}$. Using this scheme, the two instances of (a_1, b_1) are encrypted as $(\hat{a}_1^1, \hat{b}_1^2)$. Now the FD $A \rightarrow B$ still holds on $\hat{D}_2$.*

Our FD-preserving probabilistic encryption method consists of four steps: (1) identifying maximum attribute sets as the *super attribute* for encryption; (2) splitting-and-scaling; (3) conflict resolution; and (4) eliminating false positive FDs. We explain the details of these four steps in the following subsections.

7.2.1 Step 1: Identifying Maximum Attribute Sets

Theorem 16 states that the set of attributes on which the probabilistic encryption scheme is applied should contain all attributes in FDs. Apparently the set of all attributes of D satisfies the requirement. However, it is not a good solution. Let $|\sigma_{\mathcal{A}=r[\mathcal{A}]}(D)|$ denote the number of records in D that have the same value as $r[\mathcal{A}]$, for a specific record r and a set of attributes $\mathcal{A}$. It is highly likely that there does not exist a probabilistic encryption scheme, due to the reason that $f = |\sigma_{\mathcal{A}=r[\mathcal{A}]}(D)|$ is more likely to be 1 when $\mathcal{A}$ contains more attributes. Now the main challenge is to decide the appropriate attribute set $\mathcal{A}$ that will be considered as the *super attribute* on which the probabilistic encryption scheme is be applied on, assuming that the data owner is not aware of the existence of any FD. To address this challenge, we define the *maximum attribute set* on which there exists at least one instance whose frequency is greater than 1. Formally,

Definition 7.2.1. [Maximum Attribute Set (MAS)] *Given a dataset D, an attribute set $\mathcal{A}$ of D is a* maximum attribute set MAS *if: (1) there exists at least an instance a of $\mathcal{A}$ such that $|\sigma_{\mathcal{A}=a}(D)| > 1$; and (2) for any attribute set A' of D such that $\mathcal{A} \subseteq \mathcal{A}'$, there does not exist an instance a' of $\mathcal{A}'$ s.t. $|\sigma_{\mathcal{A}'=a'}(D)| > 1$.*

Our goal is to design the algorithm that finds *all* $MASs$ of a given dataset D. Note that the problem of finding $MASs$ is not equivalent to finding FDs. For instance, consider the base table D in Figure 7.1 (a). Its $MASs$ is $\{A, B\}$ and $\{B, C\}$. But its FD is $A \rightarrow B$. In general, given a set of MASs $\mathcal{M}$ and a set of FDs $\mathcal{F}$, for each FD $F \in \mathcal{F}$, there always exists at least an MAS $M \in \mathcal{M}$ such that $(LHS(F) \cup RHS(F)) \subseteq M$. But it is possible that a MAS does not contain any FD. For example, consider Figure 7.1 (a), there does not exist an FD on the attributes of MAS $\{B, C\}$.

In general, finding $MASs$ is quite challenging given the exponential number of attribute combinations to check. We found out that our MAS is equivalent to the *maximal non-unique column combination* [57]. We adapt the $Ducc$ algorithm [57] to find $MASs$. The complexity of $Ducc$ is decided by the solution set size but not the number of attributes. Due to the space limit, we omit the details of the algorithm here. For each discovered MAS, we find its *partitions* [62]. We say two tuples r and r' are *equivalent* with respect to a set of attributes X if $r[X] = r'[X]$.

Definition 7.2.2. [Equivalence Class (EC) and Partitions] *[62] The* equivalence class *(EC) of a tuple r with respect to an attribute set X, denoted as r_X, is defined as $r_X = \{r' | r[A] = r'[A], \forall A \in X\}$. The* size *of r_X is defined as the number of tuples in r_X, and the* representative value *of r_X is defined as $r[X]$. The set $\pi_X = \{r_X | r \in D\}$ is defined as a* partition *of D under the attribute set X. That is, π_X is a collection of disjoint sets (ECs) of tuples, such that each set has a unique*

representative value of a set of attributes X, and the union of the sets equals D.

As an example, consider the dataset D in Figure 7.1 (a), $\pi_{\{A,B\}}$ consists of five ECs whose representative values are (a_1, b_1), (a_2, b_1), (a_3, b_1), (a_4, b_2), and (a_5, b_2), . Apparently, a MAS is an attribute set whose partitions contain at least one equivalence class whose size is more than 1.

7.2.2 Step 2: Splitting-and-Scaling Encryption

After the $MASs$ and their partitions are discovered, we design two steps to apply the probabilistic encryption scheme on the partitions: (1) grouping of ECs, and (2) splitting and scaling on the EC groups. Next, we explain the details.

Step 2.1. Grouping of Equivalence Classes

To provide α-security, we group the ECs in the way that each equivalence class belongs to one single group, and each group contains at least $k \geq [\frac{1}{\alpha}]$ ECs, where α is the threshold for α-security. We use ECG for the *equivalence class group* in short.

We have two requirements for the construction of $ECGs$. First, we prefer to put ECs of close sizes into the same group, so that we can minimize the number of new tuples added by the next splitting & scaling step (Section 7.2.2). Second, we do not allow any two ECs in the same ECG to have the same value on any attribute of MAS. Formally,

Definition 7.2.3. [Collision of ECs] *Given a MAS M and two equivalence classes C_i and C_j of M, we say C_i and C_j have* collision *if there exists at least one attribute $A \in M$ such that $C_i[A] = C_j[A]$.*

For security reason, we require that all ECs in the same ECG should be collision-free (more details in Section 7.3).

Require: the set of equivalence groups $\overline{C} = \{C_1, \ldots, C_x\}$ in partition π_M, the minimal ECG size k
Ensure: Every equivalence class is grouped into a valid ECG.
1: Sort $C_i (1 \leq i \leq x)$ in descending order by the equivalence class's frequency;
2: **for** $i = 1$ **to** x **do**
3: **if** C_i is not grouped **then**
4: create a new ECG G containing C_i only
5: **for** $j = i + 1$ **to** x **do**
6: **if** C_j is not grouped AND C_j has no collision with any EC in G **then**
7: add C_j into G
8: **if** $G.size \geq k$ **then**
9: **break**
10: **end if**
11: **end if**
12: **end for**
13: **end if**
14: **end for**
15: **for all** ECG G s.t. $G.size < k$ **do**
16: Add *fake* collision-free ECs to make $G.size \geq k$
17: **end for**

Algorithm 7.1: $Grouping(\overline{C}, k)$

Algorithm 7.1 specifies the detailed procedure to construct $ECGs$ that satisfy the aforementioned two requirements. First, we sort the equivalence classes by their sizes in ascending order (Line 1). Second, we group the collision-free ECs with the closest size into the same ECG, until the number of non-collisional ECs in the ECG reaches $k = [\frac{1}{\alpha}]$ (Line 2 to 14). It is possible that for some ECGs, the number of ECs that can be found is less than the required $k = [\frac{1}{\alpha}]$. In this case, we add *fake* collision-free ECs to achieve the size k requirement. The fake ECs only consist of the values that do not exist in the original dataset (Line 15 to 17). The size of these fake ECs is set as the minimum size of the ECs in the same ECG. We must note that the server cannot distinguish the fake values from real

ones.This is because both true and fake values are encrypted before outsourcing.

EC	ID	A	B
C_1	$\{r_2, r_3\}$	a_1	b_1
C_2	$\{r_1\}$	a_2	b_1
C_3	$\{r_4\}$	a_3	b_1
C_4	$\{r_5\}$	a_4	b_2
C_5	$\{r_6\}$	a_5	b_2

(a) ECs in of $MAS = \{AB\}$ ($\varpi = 2, \alpha = 0.5$)

ECG	EC	ID	A	B
ECG_1	C_1	$\{r_2, r_3\}$	$\hat{a}_1^1$	$\hat{b}_1^1$
	C_4	$\{r_5, r_7\}$	$\hat{a}_4^1$	$\hat{b}_2^1$
ECG_2	C_2	$\{r_1\}$	$\hat{a}_2^1$	$\hat{b}_1^2$
	C_5	$\{r_6\}$	$\hat{a}_5^1$	$\hat{b}_2^2$
ECG_3	C_3	$\{r_4\}$	$\hat{a}_3^1$	$\hat{b}_1^3$
	C_6	$\{r_8\}$	$\hat{a}_6^1$	$\hat{b}_3^1$

(b) ECGs after $S\&S$

Figure 7.2: An example of splitting-and-scaling

Example 7.2.3. *The dataset D in Figure 7.1 (a) has two MASs. Consider the MAS $M = \{A, B\}$ and its five ECs shown in Figure 7.2 (a). Assume it is required to meet $\frac{1}{2}$-security (i.e., $\alpha = \frac{1}{2}$). The three ECGs are $ECG_1 = \{C_1, C_4\}$, $ECG_2 = \{C_2, C_5\}$, and $ECG_3 = \{C_2, C_6\}$, where C_6 is an artificial EC. All the ECGs only have collision-free ECs, and each ECG contains at least two ECs. Note that C_1 and C_2 cannot be put into the same ECG as they share the same value b_1, similarly for C_1 and C_3.*

Step 2.2. Splitting-and-Scaling ($S\&S$)

This step consists of two phases, *splitting* and *scaling*. In particular, consider an ECG $\overline{C} = \{C_1, \ldots, C_x\}$, in which each EC C_i is of size f_i $(1 \leq i \leq x)$. By *splitting*, for each C_i, its f_i (identical) plaintext values are encrypted to ϖ unique ciphertext values, each of frequency $[\frac{f_i}{\varpi}]$, where ϖ is the split factor whose value is specified by the user. The *scaling* phase is applied after splitting. By scaling, all the ciphertext values reach the same frequency by adding additional copies up to $[\frac{f_{max}}{\varpi}]$,

where f_{max} is the maximum size of all ECs in $\overline{\mathcal{C}}$. After applying $S\&S$, all ciphertext values in the same ECG are of the same frequency.

We illustrate how $S\&S$ works by using Figure 7.2 (b). Before $S\&S$, ECG_1 contains C_1 of frequency 2, and C_4 of frequency 1. By the $S\&S$ step, the record r_5 of C_4 is duplicated (as record r_7), ensuring that C_1 and C_4 have the same frequency. Note that before $S\&S$, a_1 is the only plaintext value on attribute A whose frequency is greater than 1. But after $S\&S$, there exist two ciphertext values ($\hat{a_1^1}$ and $\hat{a_4^1}$) whose frequency is greater than 1. Thus, the adversary's probability of breaking the encryption of a_1 based on the frequency distribution is $\frac{1}{2}$.

The splitting procedure can be implemented by the probabilistic encryption. For any plaintext value p, it is encrypted as $e =< r, F_k(r) \oplus p >$, where r is a random string of length λ, F is a pseudo random function, k is the key, and $\oplus$ is the *XOR* operation. The splits of the same EC can easily be generated by using different random values r. In order to decrypt a ciphertext $e =< r, s >$, we can recover p by calculating $p = F_k(r) \oplus s$. For security concerns, we require that the plaintext values that appear in different $ECGs$ are never encrypted as the same ciphertext value (more details in Section 7.3). Note that for all the records in the same EC, we can obtain its encrypted values by only encrypting the representative value of the equivalence class once.

It is not necessary that every EC must be split. Our aim is to find a subset of given ECs whose split will result in the minimal amounts of additional copies added by the scaling phase. In particular, given an ECG $\overline{\mathcal{C}}$ = $\{C_1, \ldots, C_k\}$ in which ECs are sorted by their sizes in ascending order (i.e., C_k has the largest size), we aim to find the *split point* j of $\overline{\mathcal{C}}$ such that each EC in $\{C_1, \ldots, C_{j-1}\}$ is not split but each EC in $\{C_j, \ldots, C_k\}$ is split. We call j the *split point*. Next, we discuss how to find the *optimal* split point that delivers the minimal amounts of additional copies

by the scaling phase. There are two cases: (1) the size of C_k is still the largest after the split; and (2) the size of C_k is not the largest anymore, while the size of C_{j-1} (i.e., the EC of the largest size among all ECs that are not split) becomes the largest. We discuss these two cases below.

Require: $\overline{C} = \{C_1, \ldots, C_k\}$ and their frequency $f_1, \ldots, f_k$, ϖ
Ensure: The best split point j that incur minimal number of inserted duplicates
1: Let $j_{max} = max\{j | f_{j-1} \leq [\frac{f_k}{\varpi}]\}$
2: Define $\Delta_{j_{max}} = R - R_1 = \sum_{i=1}^{j_{max}-1} (f_k - \frac{f_k}{\varpi}) = (j_{max} - 1)(1 - \frac{f_k}{\varpi})$
3: **for** $j = j_{max} + 1 \rightarrow k$ **do**
4: $\Delta_j = R - R_2 = \sum_{i=1}^{k}(f_k - f_i) - (\sum_{i=1}^{j-1}(f_{j-1} - f_i) + \varpi \sum_{i=j}^{k}(f_{j-1} - \frac{1}{\varpi} f_i)) =$
$kf_k - (\varpi k - (\varpi - 1)(j-1))f_{j-1}$
5: **end for**
6: $j' = \{j | \Delta_j = max\{\Delta_i\}, j_{max} \leq i \leq k\}$
7: **return** j'

Algorithm 7.2: $FindSplitPoint(\overline{C}, \varpi)$

Case 1: $[\frac{f_k}{\varpi}] \geq f_{j-1}$, i.e., the split of C_k still has the largest frequency within the group. In this case, the total number of copies added by the scaling step is

$$R_1 = \sum_{i=1}^{j-1}([\frac{f_k}{\varpi}] - f_i) + \sum_{i=j}^{k}(f_k - f_i).$$

It can be easily inferred that when $j = max\{j | f_{j-1} \leq [\frac{f_k}{\varpi}]\}$, R_1 is minimized.

Case 2: $[\frac{f_k}{\varpi}] < f_{j-1}$, which means that C_{j-1} enjoys the largest frequency after splitting. For this case, the number of duplicates that is to be added is:

$$R_2 = \sum_{i=1}^{j-1}(f_{j-1} - f_i) + \varpi \sum_{i=j}^{k}(f_{j-1} - [\frac{f_i}{\varpi}]).$$

R_2 is not a linear function of j. Thus we define $j_{max} = max\{j | [\frac{f_k}{\varpi}] > f_{j-1}\}$. We try all $j \in [j_{max}, k]$ and return j that delivers the minimal R_2.

Based on the analysis of the two cases, we design an algorithm to find the best split point in each equivalence class. The pseudo code is shown in Algorithm 7.2. Let j_{max} be the largest index j such that $[\frac{f_k}{\varpi}] \geq f_{j-1}$ (Line 1). According to the discussion in Case 1, j_{max} minimizes R_1. Let R denote the total amount of additional copies that are inserted when there is no split, then $\Delta_{j_{max}}$ represents the number of copies that are saved by setting the split point to be j_{max}. From Line 3 to 6 in Algorithm 7.2, we enumerate all indices j in Case 2, and calculate Δ_j. From all the indices ranging between j_{max} and k, we find the one with the largest Δ_j, i.e., the smallest amount of additional copies.

For any given ECG, the complexity of finding its optimal split point is $O(|ECG|)$. In practice, the optimal split point j is close to the ECs of the largest frequency (i.e., few split is needed). In general, the complexity of the $S\&S$ step is $O(t^2)$, where t is the number of equivalence classes of D. As $t = O(nq)$, the complexity is $O(n^2q^2)$, where n is the number of tuples in D, and q is the number of $MASs$.

7.2.3 Step 3: Conflict Resolution

ID	A	B
r_1	$\hat{a}_2^1$	$\hat{b}_1^2$
r_2	$\hat{a}_1^1$	$\hat{b}_1^1$
r_3	$\hat{a}_1^1$	$\hat{b}_1^1$
r_4	$\hat{a}_3^1$	$\hat{b}_1^3$
r_5	$\hat{a}_4^1$	$\hat{b}_2^1$
r_6	$\hat{a}_5^1$	$\hat{b}_2^2$

(a) $Enc_{\{A,B\}}(D)$

ID	B	C
r_1	$\hat{b}_1^2$	$\hat{c}_1^1$
r_2	$\hat{b}_1^2$	$\hat{c}_1^1$
r_3	$\hat{b}_1^3$	$\hat{c}_2^1$
r_4	$\hat{b}_1^3$	$\hat{c}_2^1$
r_5	$\hat{b}_2^1$	$\hat{c}_2^2$
r_6	$\hat{b}_2^2$	$\hat{c}_3^1$

(b) $Enc_{\{B,C\}}(D)$

ID	A	B	C
r_1	$\hat{a}_2^1$	$\hat{b}_1^2$	$\hat{c}_1^1$
r_2	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_1^2$
r_3	$\hat{a}_1^1$	$\hat{b}_1^1$	$\hat{c}_2^3$
r_4	$\hat{a}_3^1$	$\hat{b}_1^3$	$\hat{c}_2^1$
r_5	$\hat{a}_4^1$	$\hat{b}_2^1$	$\hat{c}_2^2$
r_6	$\hat{a}_5^1$	$\hat{b}_2^2$	$\hat{c}_3^1$
r_7	$\hat{a}_1^2$	$\hat{b}_1^2$	$\hat{c}_1^1$
r_8	$\hat{a}_1^3$	$\hat{b}_1^3$	$\hat{c}_2^1$

(c) $\hat{D}$: Encryption by conflict resolution

Figure 7.3: An example of conflict resolution of two overlapping $MASs$

So far the grouping and splitting & scaling steps are applied on one single MAS. In practice, it is possible that there exist multiple $MASs$. The problem is that, applying grouping and splitting & scaling separately on each MAS may lead to conflicts. For instance, consider the example in Figure 7.3, in which Figure 7.3 (a) and (b) show the encryption on two MASs $\{A, B\}$ and $\{B, C\}$ individually. The conflict appears at tuples r_2 and r_3 on attribute B. For example, $r_2[B]$ is encrypted as $\hat{b}_1^1$, while it is encrypted as $\hat{b}_1^2$.

There are two possible conflicts:

(1) *Type-1. Conflicts due to scaling*: there exist tuples that are scaled by one MAS but not so by another MAS; and

(2) *Type-2. Conflicts due to shared attributes*: there exist tuples whose value on attribute(s) Z are encrypted differently by multiple $MASs$, where Z is the overlap of these $MASs$.

It initially seems true that there may exist the conflicts due to splitting too; some tuples are required to be split according to one MAS but not by another. But our further analysis shows that such conflicts only exist for overlapping $MASs$, in particular, the type-2 conflicts. Therefore, dealing with type-2 conflicts covers the conflicts due to splitting.

The aim of the conflict resolution step is to synchronize the encryption of all $MASs$. Given two $MASs$ of the attribute sets X and Y, we say these two $MASs$ *overlap* if X and Y overlap at at least one attribute. Otherwise, we say the two $MASs$ are non-overlapping. Next, we discuss the details of the conflict resolution for non-overlapping $MASs$ (Section 7.2.3) and overlapping $MASs$ (Section 7.2.3).

Non-overlapping $MASs$

Given two $MASs$ of attribute sets X and Y such that X and Y do not overlap, only the type-1 conflicts (i.e., conflicts due to scaling) are possible. WLOG we assume a tuple r is required to be scaled by applying scaling on the ECG of r_X but not on any ECG of r_Y. The challenge is that simply scaling of r makes the ECG of r_Y fails to have homogenized frequency anymore. To handle this type of conflicts, first, we execute the grouping and splitting & scaling over π_X and π_Y independently. Next, we look for any tuple r such that r is required to have ℓ ($\ell > 1$) copies to be inserted by splitting & scaling over π_X but free of split and scaling over π_Y. For such r, we modify the values of the ℓ copies of tuple r, ensuring that for each copy r', $r'[X] = r[X]$ while $r'[Y] \neq r[Y]$. To avoid collision between $ECGs$, we require that no $r'[Y]$ value exists in the original dataset D. By doing this, we ensure that the $ECGs$ of both r_X and r_Y still achieve the homogenized frequency distribution. Furthermore, by assigning new and unique values to each copy on the attribute set Y, we ensure that $MASs$ are well preserved (i.e., both $MASs$ X and Y do not change to be $X \cup Y$). The complexity of dealing with type-1 conflicts is $O(n'q)$, where n' is the number of the tuples that have conflicts due to scaling, and q is the number of $MASs$.

Overlapping $MASs$

When $MASs$ overlap, both types of conflicts are possible. The type-1 conflicts can be handled in the same way as for non-overlapping $MASs$ (Section 7.2.3). In the following discussion, we mainly focus on how to deal with the type-2 conflicts (i.e., the conflicts due to shared attributes). We start our discussion from two overlapping $MASs$. Then we extend to the case of more than two overlapping $MASs$.

Two overlapping MASs. We say two ECs $C_i \in \pi_X$ and $C_j \in \pi_Y$ are *conflicting* if C_i and C_j share at least one tuple. We have the following theorem to show that conflicting ECs never share more than one tuple.

Theorem 17. *Given two overlapping $MASs$ X and Y, for any pair of ECs $C_i \in \pi_X$ and $C_j \in \pi_Y$, $|C_i \cap C_j| \leq 1$.*

Proof. The correctness of Theorem 17 is straightforward: if $|C_i \cap C_j| > 1$, there must exist at least one equivalence class of the partition $\pi_{X \cup Y}$ whose size is greater than 1. Then $X \cup Y$ should be a MAS instead of X and Y. □

Theorem 17 ensures the efficiency of the conflict resolution, as it does not need to handle a large number of tuples.

A naive method to fix type-2 conflicts is to assign the same ciphertext value to the shared attributes of the two conflicting ECs. By re-visiting the example in Figure 7.1 (a), if we follow the naive solution to resolve the conflict appears at tuples r_2 and r_3 on attribute B, only one value is picked for tuples r_2 and r_3 on attribute B. This scheme is incorrect as the FD $F : A \rightarrow B$ does not hold anymore.

We design a robust method to resolve the type-2 conflicts for two overlapping $MASs$. Given two overlapping $MASs$ X and Y, let $Z = X \cap Y$, for any tuple r, let $r^X[Z]$ and $r^Y[Z]$ ($r^X[Z] \neq r^Y[Z]$) be the value constructed by encryption over X and Y independently. We use $X - Z$ ($Y - Z$, resp.) to denote the attributes that appear in X (Y, resp.) but not Z. Then we construct two tuples r_1 and r_2:

- r_1: $r_1[X - Z] = r[X - Z]$, $r_1[Y - Z] = v_X$, and $r_1[Z] = r^X[Z]$;
- r_2: $r_2[X - Z] = v_Y$, $r_2[Y - Z] = r[Y - Z]$, and $r_2[Z] = r^Y[Z]$.

where v_X and v_Y are two values that do not exist in D. Note that both $X - Z$ and $Y - Z$ can be sets of attributes, thus v_X and v_Y can be set of values. Tuples

r_1 and r_2 replace tuple r in $\hat{D}$. As an example, consider the tables in Figure 7.3 (a) and (b), the encryption after conflict resolution is shown in Figure 7.3 (c). Tuple r_2 and r_7 in Figure 7.3 (c) are the two records constructed for the conflict resolution of r_2 in Figure 7.3 (a) and (b). Our conflict resolution method guarantees that the ciphertext values of each ECG of X and Y are of homogenized frequency.

However, it requires to add additional records. Next, we show that the number of records added by resolution of both types of conflicts is bounded.

Theorem 18. *Given a dataset D, let $\hat{D}_1$ be the dataset after applying grouping and splitting & scaling on D, and $\hat{D}_2$ be the dataset after conflict resolution on $\hat{D}_1$, then $|\hat{D}_2| - |\hat{D}_1| \leq hn$, where h is the number of overlapping MAS pairs, and n is the size of D.*

Proof. Due to the space limit, we only give the proof sketch here. Note that the resolution of type-1 conflicts does not add any fake record. Thus we only prove the bound of the new records for resolution of type-2 conflicts. This type of conflicts is resolved by replacing any conflicting tuple with two tuples for each pair of conflicting ECs. Since two overlapping $MASs$ have n conflicting equivalence class pairs at most, there will be at most hn new records inserted for this type of resolution for h overlapping MAS pairs. □

We must note that hn is a loose bound. In practice, the conflicting ECs share a small number of tuples. The number of such tuples is much smaller than n. We also note that when h = 0 (i.e., there is no overlapping MAS), no new record will be inserted.

More than Two Overlapping MASs. When there are more than two overlapping $MASs$, one way is to execute the encryption scheme that deals with two overlapping $MASs$ repeatedly for every two overlapping $MASs$. This raises the question

in which order the $MASs$ should be processed. We have the following theorem to show that indeed the conflict resolution is insensitive to the order of $MASs$.

Theorem 19. *Given a dataset D that contains a set of overlapping $MASs$ M, let $\hat{D}_1$ and $\hat{D}_2$ be the datasets after executing conflict resolution in two different orders of M, then $|\hat{D}_1| = |\hat{D}_2|$.*

Proof. It is easy to show that no new record is inserted by resolution of type-1 conflicts. To fix type-2 conflicts, for each overlapping MAS pair, every conflicting tuple is replaced by two tuples. The number of conflicting tuples equals the number of conflicting equivalence class pairs of the two overlapping MASs. Assume there are q such overlapping MAS pairs, each having o_i pairs of conflicting equivalence class pairs. The conflict resolution method adds $O = \sum_{i=1}^{q} o_i$ records in total. The number of records is independent from the orders of MASs in which conflict resolution is executed. □

Since the order of $MASs$ does not affect the encryption, we pick the overlapping MAS pairs randomly. For each overlapping MAS pair, we apply the encryption scheme for two overlapping $MASs$. We repeat this procedure until all $MASs$ are processed.

7.2.4 Step 4. Eliminating False Positive FDs

Before we discuss the details of this step, we first define false positive FDs. Given the dataset D and its encrypted version $\hat{D}$, we say the FD F is a *false positive* if F does not hold in D but holds in $\hat{D}$. We observe that the encryption scheme constructed by Step 1 - 3 may lead to false positive FDs. Next, we show an example of false positive FDs.

	EC	B	C	f
ECG_1	C_1	b_1	c_1	2
	C_2	b_2	c_2	1
ECG_2	C_3	b_1	c_2	2
	C_4	b_2	c_3	1

(a) Base table D ($B \rightarrow C$ does not hold)

	B	C	f
ECG_1	$\hat{b}_1^2$	$\hat{c}_1^1$	2
	$\hat{b}_2^1$	$\hat{c}_2^2$	2
ECG_2	$\hat{b}_1^3$	$\hat{c}_2^1$	2
	$\hat{b}_2^2$	$\hat{c}_3^1$	2

(b) $\hat{D}$: encryption by Step 1 - 3 ($B \rightarrow C$ becomes false positive)

A	B	C	f
$\hat{a}_6$	$\hat{b}_3$	$\hat{c}_4$	1
$\hat{a}_7$	$\hat{b}_3$	$\hat{c}_5$	1
$\hat{a}_8$	$\hat{b}_4$	$\hat{c}_6$	1
$\hat{a}_9$	$\hat{b}_4$	$\hat{c}_7$	1

(c) Constructed ΔD to remove false positive $FD\ B \rightarrow C$

Figure 7.4: An example of eliminating false positive FDs

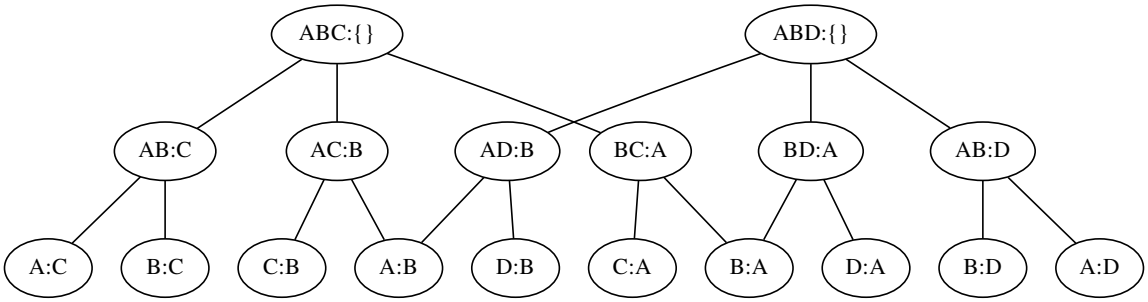


Figure 7.5: An example of FD lattice

Example 7.2.4. *Consider the table D in Figure 7.4 (a) generated from the dataset in Figure 7.1 (a). Apparently the $FD\ F: B \rightarrow C$ does not exist in D, as the two $ECs\ C_1$ and C_3 have collisions. However, in the encrypted dataset $\hat{D}$ constructed by Step 1 - 3 (Figure 7.4 (b)), since no split of any two ECs have collision anymore, $F: B \rightarrow C$ holds in $\hat{D}$.*

Indeed, we have the following theorem to show that free of collision among ECs is the necessary and sufficient conditions for the existence of FDs.

Theorem 20. *For any MAS X, there must exist a FD on X if and only if for any two $ECs\ C_i$ and C_j of π_X, C_i and C_j do not have collision.*

Proof. The proof of Theorem 20 is straightforward. As X is a MAS, there must exist at least an $EC\ eq \in \Pi_X$ such that $|eq| > 1$. As there does not any pair of ECs with collision, for any attribute $A \in X$, it must be true that $X \setminus \{A\} \rightarrow A$ is a FD. $\square$

Following Theorem 20, the fact that Step 1 - 3 construct only collision-free ECs may lead to a large number of false positive FDs, which hurts the accuracy of dependency discovering by the server.

The key idea to eliminate the false positive FDs is to restore the collision within the ECs. Note that eliminating the false positive FD $F : X \rightarrow Y$ naturally leads to the elimination of all false positive FDs $F' : X' \rightarrow Y$ such that $X' \subseteq X$. Therefore, we only consider eliminating the *maximum false positive FDs* whose LHS is not a subset of LHS of any other false positive FDs.

We use the *FD lattice* to help restore the ECs with collision and eliminate false positive FDs. The lattice is constructed in a top-down fashion. Each MAS corresponds to a level-1 node in the lattice. We denote it in the format $M : \{\}$, where M is the MAS that the node corresponds to. The level-2 nodes in the lattice are constructed as following. For each level-1 node N, it has a set of children nodes (i.e., level-2 nodes) of format $X : Y$, where Y corresponds to a single attribute of D, and $X = M - \{Y\}$, where M is the MAS that node N corresponds to. Starting from level 2, for each node $X : Y$ at level ℓ ($\ell \geq 2$), it has a set of children nodes of format $X' : Y'$, where $Y' = Y$, and $X' \subset X$, with $|X|' = |X| - 1$ (i.e., X' is the largest subset of X). Each node at the bottom of the lattice is of format $X : Y$, where both X and Y are 1-attribute sets. An example of the FD lattice is shown in Figure 7.5.

Based on the FD lattice, the data owner eliminates the false positive FDs by the following procedure. Initially all nodes in the lattice are marked as "un-checked". Starting from the second level of lattice, for each node N (in the format $X : Y$), the data owner checks whether there exists at least two ECs $C_i, C_j \in \pi_M$ such that $C_i[X] = C_j[X]$ but $C_i[Y] \neq C_j[Y]$, where M is the MAS that N's parent node corresponds to. If it does, then the data owner does the following two steps.

First, the data owner inserts $k = \lceil \frac{1}{\alpha} \rceil$ artificial record pairs $\{P_1, \ldots, P_k\}$, where α is the given threshold for α-security. The reason why we insert k such pairs will be explained in Section 7.3. Each P_i consists of two records r_i^1 and r_i^2:

- $r_i^1 : r_i^1[X] = x_i$, $r_i^1[Y] = a_i^1$, and $r_i^1[MAS - X - \{Y\}] = v_i^1$;
- $r_i^2 : r_i^2[X] = x_i$, $r_i^2[Y] = a_i^2$, and $r_i^2[MAS - X - \{Y\}] = v_i^2$.

where x_i, a_i^1, a_i^2, v_i^1 and v_i^2 are artificial values that do not exist in $\hat{D}$ constructed by the previous steps. We require that $a_i^1 \neq a_i^2$, and $v_i^1 \neq v_i^2$. We also require that all artificial records are of frequency one. Second, the data owner marks the current node and all of its descendants in the lattice as "checked" (i.e., the current node is identified as a maximum false positive FD). Otherwise (i.e., the ECs do not have collisions), the data owner simply marks the current node as "checked". After checking all the nodes at the level ℓ, the data owner moves to the level $\ell + 1$ and applies the above operation on the "un-checked" lattice nodes. The data owner repeats the procedure until all nodes in the lattice are marked as "checked".

Let ΔD be the artificial records that are constructed by the above iterative procedure. It is easy to see that for any FD $X \rightarrow Y$ that *does not* hold in D, there always exist two ECs C_i' and C_j' in ΔD, where $C_i'[X] = C_j'[X]$ but $C_i'[Y] \neq C_j'[Y]$. This makes the FD $X \rightarrow Y$ that does not hold in D fails to hold in $\hat{D}$ too.

Example 7.2.5. *To continue our Example 7.2.4, we show how to construct ΔD. It is straightforward that the MAS $\{B, C\}$ contains the ECs that have collision (e.g. C_1 and C_3 in Figure 7.4 (a)). Assume $\alpha = 1/2$. Then ΔD (shown in Figure 7.4 (c)) consists of two pairs of artificial tuples, each pair consisting of two records of the same value on attribute B but not on C. The false positive FD $F : B \rightarrow C$ does not exist in $\hat{D} + \Delta D$ any more.*

Next, we show that the number of artificial records added by Step 4 is bounded.

Theorem 21. *Given a dataset D, let $M_1, \ldots, M_q$ be the $MASs$ of D. Let $\hat{D}_1$ and $\hat{D}_2$ be the dataset before and after eliminating false positive FDs respectively. Then*

$$2k \leq |\hat{D}_2| - |\hat{D}_1| \leq min(2km\binom{m-1}{[\frac{m-1}{2}]}, 2k\sum_{i=1}^{q}|M_i|\binom{|M_i|-1}{[\frac{|M_i|-1}{2}]}),$$

where $k = \lceil \frac{1}{\alpha} \rceil$ (α as the given threshold for α-security), m is the number of attributes of D, q is the number of $MASs$, and $|M_i|$ as the number of attributes in M_i.

Proof. Due to the limited space, we show the proof sketch here. We consider two extreme cases for the lower bound and upper bound of the number of artificial records. First, for the lower bound case, there is only one MAS whose ECs have collisions. It is straightforward that our construction procedure constructs $2k$ artificial records, where $k = \lceil \frac{1}{\alpha} \rceil$. Second, for the upper bound case, it is easy to infer that the maximum number of nodes in the FD lattice that needs to construct artificial records is $\sum_{i=1}^{q}|M_i|\binom{|M_i|-1}{[\frac{|M_i|-1}{2}]}$. For each such lattice node, there are $2k$ artificial records. So the total number of artificial records is no larger than $2k\sum_{i=1}^{q}|M_i|\binom{|M_i|-1}{[\frac{|M_i|-1}{2}]}$. On the other hand, as for each maximum false positive FD, its elimination needs $2k$ artificial records to be constructed. Therefore, the total number of artificial records added by Step 4 equals $2ku$, where u is the number of maximum false positive FDs. It can be inferred that $u \leq m\binom{m-1}{[\frac{m-1}{2}]}$. Therefore, the number of artificial records cannot exceed $min(2k\sum_{i=1}^{q}|M_i|\binom{|M_i|-1}{[\frac{|M_i|-1}{2}]}, 2km\binom{m-1}{[\frac{m-1}{2}]})$. Note that the number of artificial records is independent of the size of the original dataset D. It only relies on the number of attributes of D and α. □

The time complexity of eliminating false positive FDs for a single MAS M is $O(2^{|M|}t)$, where $|M|$ is the number of attributes in M, and t is the number of ECs of M. In our experiments, we observe $t << n$, where n is the number of records in D. For instance, on a benchmark dataset of 15 million records, the average value of t is $11,828$. Also, the experiment results on all three datasets show that at most $|M| = \frac{m}{2}$. With the existence of $q > 1$ MASs, the time complexity is $O(\sum_{i=1}^{q} 2^{|M_i|}t_i)$, where t_i is the number of equivalence classes in M_i, and $|M_i|$ is the number of attributes of MAS M_i. Considering that $t_i << n$, the total complexity is comparable to $O(nm^2)$.

We have the following theorem to show that the FDs in $\hat{D}$ and D are the same.

Theorem 22. *Given the dataset D, let $\hat{D}$ be the dataset after applying Step 1 - 4 of F^2 on D, then: (1) any FD of D also hold on $\hat{D}$; and (2) any FD F that does not hold in D does not hold in $\hat{D}$ either.*

Proof. First, we prove that the grouping, splitting & scaling and conflict resolution steps keep the original FDs. We prove that for any FD $X \rightarrow A$ that holds on D, it must also hold on $\hat{D}$. We say that a partition π is a *refinement* of another partition π' if every equivalence class in π is a subset of some equivalence class (EC) of π'. It has been proven that the functional dependency $X \rightarrow A$ holds if and only if π_X refines $\pi_{\{A\}}$[62]. For any functional dependency $X \rightarrow A$, we can find a MAS M such that $(X \cup \{A\}) \subset M$. Obviously, π_M refines π_X, which means for any EC $C \in \pi_M$, there is an EC $C_p \in \pi_X$ such that $C \subset C_p$. Similarly, π_M refines $\pi_{\{A\}}$. So for any equivalence class $C \in \pi_M$, we can find $C_p \in \pi_X$ and $C_q \in \pi_{\{A\}}$ such that $C \subset C_p \subset C_q$. First, we prove that our grouping over M keeps the FD: $X \rightarrow A$. It is straightforward that grouping ECs together does not

affect the FD. The interesting part is that we add fake ECs to increase the size of a ECG. Assume we add a fake equivalence class C_f into π_M. Because C_f is *non-collisional*, $\pi_X = \pi_X \cup \{C_f\}$ and $\pi_{\{A\}} = \pi_{\{A\}} \cup \{C_f\}$. Therefore, π_X still refines $\pi_{\{A\}}$. The FD is preserved. Second, we show that our splitting scheme does not break the FD: $X \rightarrow A$. Assume that we split the equivalence class $C \in \pi_M$ into ϖ unique equivalence classes $C^1, \ldots, C^\varpi$. After the split, $C_p = C_p - C$, $C_q = C_q - C$. This is because the split copies have unique ciphertext values. As a result, $\pi_X = \pi_X \cup \{C^1, \ldots, C^\varpi\}$ and $\pi_{\{A\}} = \pi_{\{A\}} \cup \{C^1, \ldots, C^\varpi\}$. It is easy to see that π_X is still a refinement of $\pi_{\{A\}}$. The scaling step after splitting still preserves the FD, as it only increases the size of the equivalence class $C \in \pi_M$ by adding additional copies. The same change applies to both C_p and C_q. So C_p is still a subset of C_q. As a result, the FD: $X \rightarrow A$ is preserved after splitting and scaling. Lastly, we prove that our conflict resolution step keeps the FDs. First, the way of handling non-overlapping $MASs$ is FD-preserving because we increase the size of an equivalence class in a partition while keeping the stripped partitions of the other $MASs$. Second, the conflict resolution for overlapping $MASs$ is also FD-preserving. Assume $C \in \pi_M$ and $C' \in \pi_N$ conflict over a tuple r. According to our scheme, we use r_1 and r_2 to replace r with $r_1[M] = r^M[M]$, $r_2[N] = r^N[N]$, $r_1[N - M]$ and $r_2[M - N]$ having new values. The effect is to replace $r \in C$ with r_1. This change does not affect the fact that $C_p \subset C_q$. Therefore the FDs are still preserved. Here we prove that by inserting artificial records, all original FDs are still kept while all the false positive FDs are removed.

Next, we prove that insertion of artificial records preserves all original FDs. The condition to insert fake records is that there exists equivalence classes C_i and C_j such that $C_i[X] = C_j[X]$ but $C_i[Y] \neq C_j[Y]$ on the attribute sets X and Y. For any FD that holds on D, this condition is never met. Hence the real FDs in D will

be kept.

Last, we prove that any FD $F : X \rightarrow Y$ that does not hold in D is also not valid in $\hat{D}$. First we show that if there does not exist a MAS M such that $X \cup \{Y\} \not\subset M$, $X \rightarrow Y$ can not hold in $\hat{D}$. Since our splitting & scaling procedure ensures that different plaintext values have different ciphertext values, and each value in any equivalence class of a MAS is encrypted to a unique ciphertext value, the set of MASs in D must be the same as the set of MASs in $\hat{D}$. As a consequence, in $\hat{D}$, there do not exist any two records r_i, r_j such that $r_i[X] = r_j[X]$ and $r_i[Y] = r_j[Y]$. Therefore, $X \rightarrow Y$ cannot be a FD in $\hat{D}$. Second, we prove that for any MAS M such that $X \cup \{Y\} \subset M$, if F does not hold on D, then F must not hold on $\hat{D}$. $\square$

7.3 Security Analysis

In this section, we analyze the security of F^2 against both the frequency analysis attack (FA) and the FD-preserving chosen plaintext attack (FCPA).

7.3.1 Frequency Analysis Attack

Frequency analysis (FA) attack is the most basic and well-known inference attack. It is formally defined in [90]. We refer the readers to [90] for more details. For any $e \in \mathcal{E}$ be a ciphertext value, let $G(e) = \{p | p \in \mathcal{P}, freq_{\mathcal{P}}(p) = freq_{\mathcal{E}}(e)\}$ be the set of distinct plaintext values having the same frequency as e. It has been shown in [106] that for any adversary A and any ciphertext value e, the advantage of the frequency analysis attack is $Adv^{FA}_{F^2}(A) = \frac{1}{|G(e)|}$, where $|G(e)|$ is the size of $G(e)$. In other words, the size of $G(e)$ determines the advantage $Adv^{FA}_{F^2}(A)$.

Apparently, the scaling step of F^2 ensures that all the equivalence classes in the same ECG have the same frequency. Hence, for any encrypted equivalence

class EC', there are at least $|ECG|$ plaintext ECs having the same frequency. Recall that we do not allow any two equivalence classes in the same ECG to have the same value on any attribute. Therefore, for any attribute A, a ECG contains k distinct plaintext values on A, where k is the size of ECG. Thus for any $e \in \mathcal{E}$, it is guaranteed that $|G(e)| = k$. As $k \geq [\frac{1}{\alpha}]$, it is guaranteed that $G(e) \geq [\frac{1}{\alpha}]$. In this way, we have $Adv_{F^2}^{FA}(A) \leq \alpha$. Thus F^2 provides α-security against the FA attack.

In our empirical study, we implemented the FA attack in [90], and compared the attack accuracy of F^2 with three different encryption schemes: (1) frequency-hiding order-preserving encryption [70]; (2) AES, a well-known symmetric encryption algorithm; and (3) Pallier, a probabilistic asymmetric algorithm. More details can be found in Section 7.4.

7.3.2 FD-preserving Chosen Plaintext Attack (FCPA)

Given a dataset D with a FD $X \rightarrow Y$, consider the following FCPA adversary A against F^2. Let $LR(.,.,b)$ denote the function that on inputs m_0, m_1 returns m_b $(b \in$

$\{0, 1\}$).

$$\textbf{Experiment } Exp_{F^2}^{IND-FCPA}(k, LR(.,.,b))$$
$$r_1, r_2, r_3, r_4 \in D$$
$$r_1[X, Y] = r_3[X, Y] = (x_1, y_1)$$
$$r_2[X, Y] = (x_2, y_2)$$
$$r_4[X, Y] = (x_3, y_3)$$
$$c_1 \leftarrow Encrypt(k, LR(r_1, r_2, b))[X, Y]$$
$$c_2 \leftarrow Encrypt(k, LR(r_3, r_4, b))[X, Y]$$
$$Return\ 0\ if\ c_1 = c_2$$
$$Return\ 1\ otherwise$$

Intuitively, if both r_1 and r_3 (i.e., the left side of (r_1, r_2) and (r_3, r_4)) are encrypted, the adversary can infer that $b = 0$ as F^2 is FD-preserving. Otherwise $b = 1$. Next, we analyze the security guarantee of F^2 against the FCPA adversary A. Let $Exp(A)$ denote the output of experiment.

When $b = 1$, $c_1 = Encrypt(k, r_2)[X, Y] = Encrypt(k, (x_2, y_2))$, and $c_2 = Encrypt(k, (x_3, y_3))$. Thus $Prob(Exp(A) = 1|b = 1) = Prob(Encrypt(k, (x_2, y_2)) \neq Encrypt(k, (x_3, y_3))) = 1 - negl(\lambda)$. This is because the probability that two different plaintext values x_2, y_2 and (x_3, y_3) are encrypted to the same ciphertext value is negligible.

When $b = 0$, $c_1 = Encrypt(k, r_1)[X, Y]$, and $c_2 = Encrypt(k, r_2)[X, Y]$. According to F^2, it is possible that r_1 and r_2 have different ciphertext values on $[X, Y]$, even though they have the same plaintext values. Next, we infer the probability that $c_1 = c_2$. Let M be the MAS that include both X and Y. We define a set of

equivalence classes $EQ = \{eq \in \pi_M | eq[X,Y] = (x_1, y_1)\}$, and g as the number of equivalence classes in the partition of M whose attribute values on $[X,Y]$ is (x_1, y_1). It must be true that $r_1 \in eq_i \in EQ$ and $r_2 \in eq_j \in EQ$, where $1 \leq i, j \leq g$. Based on the relationship between i and j, we have the following cases for the adversary to output 1 (i.e., $c_1 = c_2$).

Case 1: $i \neq j$**.** When r_1 and r_2 belong to different equivalence classes of π_M, their ciphertext values must be different. This is because according to Step 2.1, eq_i and eq_j must fall into different ECGs, as there exist collision between them. We also require that the same plaintext values that appear in different ECGs are never encrypted as the same ciphertext value. Therefore, $Prob(c_1 \neq c_2 | i \neq j) = 1$. As $Prob(i \neq j) = 1 - Prob(i = j) = 1 - \frac{1}{g}$, $Prob(Exp(A) = 1, i \neq j | b = 0) = 1 - \frac{1}{g}$.

Case 2: $i = j$**.** When r_1 and r_2 belong to the same equivalence class of π_M, it is still possible for them to have different ciphertext values. Let the equivalence class be eq. Based on our *Splitting-and-scaling* scheme, when eq is split and when r_1 and r_2 belong to different split copies, r_1 and r_2 are encrypted to be different ciphertext values. In other words, $Prob(Exp(A) = 1, i = j | b = 0) = Prob(E_1, E_2) * \frac{1}{g}$, where E_1 is the event that eq is split, and E_2 is the event that r_1 and r_2 belong to different split copies. Let y be the split point in the ECG, we have $Prob(E_1) = \frac{y}{k}$, where $k = [\frac{1}{\alpha}]$. This is because in the ECG, we pick y out of k equivalence classes to split. Let ϖ be the split factor. Then $Prob(E2|E_1) = 1 - \frac{1}{\varpi}$, since the probability that r_1 and r_2 are encrypted as the same split copy is $\frac{1}{\varpi}$. Therefore, we have $Prob(Exp(A) = 1, i = j | b = 0) = Prob(E_1, E_2) = Prob(E2|E1) * Prob(E1) = (1 - \frac{1}{\varpi}) * \frac{y}{gk}$.

Based on the two cases, $Prob(Exp(A) = 1 | b = 0) = 1 - \frac{1}{g} + \frac{(\varpi - 1)y}{gk\varpi}$. Thus, the adversary's advantage on F^2 is

$$Adv_{F^2}^{FCPA}(A) = \frac{\varpi k - (\varpi - 1)y}{g\varpi k} - negl(\lambda).$$

In practice, we find that in most cases, y is close to 0, so

$$Adv_{F^2}^{FCPA}(A) \approx \frac{1}{g}.$$

In general, when g is large (i.e., the equivalent classes are of large size), $Adv_{F^2}^{FCPA}(A)$ is negligible, and thus F^2 can provide IND-FCPA. In our experiments on real-world datasets, g can be 5,000,000 for a dataset that contains 15 million records. When g is small, we can always reach IND-FCPA by inserting artificial records to make g sufficiently large. It is possible that the number of artificial records may be significant; but that is the price we pay for the security.

7.4 Experiments

In this section, we discuss our experiment results and provide the analysis of our observations.

7.4.1 Setup

Computer environment. We implement our algorithm in Java. All the experiments are executed on a PC with 2.5GHz i7 CPU and 60GB memory running Linux.

Datasets. We execute our algorithm on the *Customer* dataset from TPC-C benchmark, the *Orders* dataset from TPC-H benchmark, and one synthetic dataset. The *Orders* dataset contains 9 attributes and 1.5 million records (size 1.67GB). *Orders* dataset contains nine maximal attribute sets $MASs$. All $MASs$ overlap pairwise. Each MAS contains either four or five attributes. The *Customer* dataset includes 21 attributes and 960K records (size 282MB). There are fifteen MASs in *Customer* dataset. The size of these $MASs$ (i.e., the number of attributes) ranges from nine

to twelve. All MASs overlap pairwise. The synthetic dataset contains 7 attributes and 4 million records (size 224MB). The synthetic dataset has two MASs, one of three attributes, while the other of six attributes. The two $MASs$ overlap at one attribute.

Evaluation. We evaluate the efficiency and practicality of our encryption scheme according to the following criteria:

- Encryption time: the time for the data owner to encrypt the dataset (Sec. 7.4.2);
- Space overhead: the amounts of artificial records added by F^2 (In our experiments, the number of artificial records never exceeds 12% of the original data size. Due to the space limitation, we omit the details of this set of results);
- FD-preserving accuracy: the fraction of original FDs that are preserved in the encrypted dataset (Sec. 7.4.4);
- Security against the FA attack: the fraction of the encrypted values that can be mapped to the original plaintext by the FA attack (Sec. 7.4.5);
- Outsourcing versus local computations: (1) the time of discovering FDs versus encryption by F^2, and (2) the FD discovery time on the original data versus that on the encrypted data (Sec. 7.4.6).

Baseline approaches. We implement three baseline encryption methods, including *AES*, *Paillier*, and frequency-hiding order-preserving encryption (FHOP) [70], to encode the data at cell level. The *AES* baseline approach uses the well-known AES algorithm with key length of 256 bits for the deterministic encryption. We use the implementation of AES in the *javax.crypto* package[1]. The *Paillier* baseline approach uses the asymmetric Paillier encryption with key length of 256 bits for the probabilistic encryption. We use the *UTD Paillier Threshold Encryption Toolbox*[2].

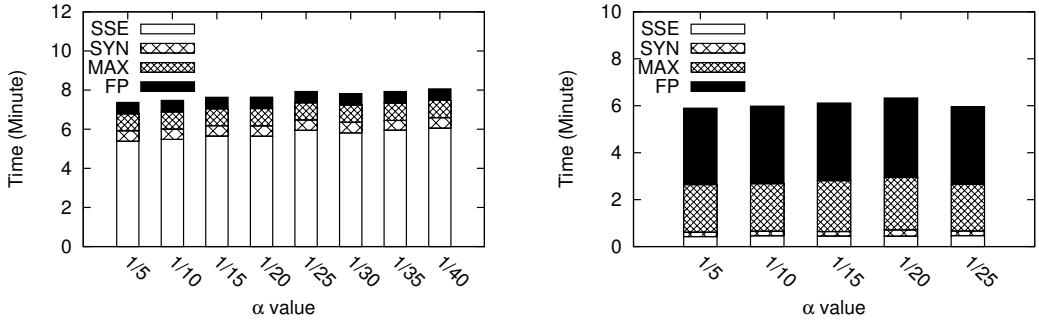
[1] `https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html`
[2] `http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/`.

FHOP is based on a binary search tree structure. For any plaintext value (not necessarily distinct), its ciphertext value is generated by inserting a new node to the tree. We compare F^2 with *AES*, *Paillier*, *FHOP* in terms of FD-preserving accuracy, security, and time performance.

7.4.2 Encryption Time

In this section, we measure the time performance of F^2 to encrypt the dataset. First, we evaluate the impact of security threshold α on the running time. We measure the time of our four steps of the algorithm: (1) finding maximal attribute sets (MAX), (2) splitting-and-scaling encryption (SSE), (3) conflict resolution (SYN), and (4) eliminating false positive FDs (FP), individually. We show our results on both *Orders* and the synthetic datasets in Figure 7.6. First, we observe that for both datasets, the time performance does not change much with the decrease of α value. This is because the running time of the MAX, SYN and FP steps is independent on α. In particular, the time of finding $MASs$ stays stable with the change of α values, as its complexity relies on the data size, not α. Similarly, the time performance of FP step is stable with various α values, as its complexity is only dependent on the data size and the data schema. The time performance of SYN step does not vary with α value because we only need to synchronize the encryption on the original records, without the worry about the injected records. It is worth noting that on the *Orders* dataset, the time took by the SYN step is negligible. This is because the SYN step leads to only 24 artificial records on the *Orders* dataset (of size 0.325GB). Second, the time of SSE step grows for both datasets when α decreases (i.e., tighter security guarantee). This is because smaller α value requires larger number of artificial equivalence classes to form ECGs of the desired size.

This addresses the trade-off between security and time performance. We also observe that the increase in time performance is insignificant. For example, even for *Orders* dataset of size 0.325GB, when α is decreased from 0.2 to 0.04, the execution time of the SSE step only increases by 2.5 seconds. This shows that F^2 enables to achieve higher security with small additional time cost on large datasets. We also observe that different steps dominate the time performance on different datasets: the SSE step takes most of the time on the synthetic dataset, while the MAX and FP steps take the most time on the *Orders* dataset. This is because the average number of ECs of all the MASs in the synthetic dataset is much larger than that of the *Orders* dataset (128,512 v.s. 1003). Due to the quadratic complexity of the SSE step with regard to the number of ECs, the SSE step consumes most of the time on the synthetic dataset.



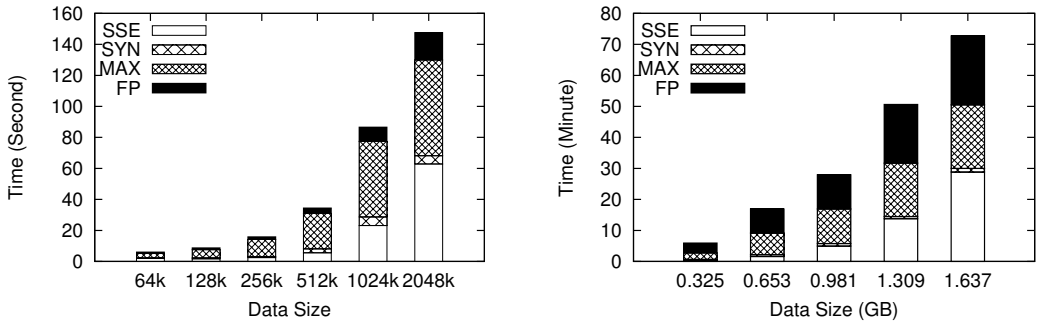
(a) Synthetic dataset (53MB) (b) Orders dataset (0.325GB)

Figure 7.6: Time performance for various α

Second, to analyze the scalability of our approach, we measure the time performance for various data sizes. The results are shown in Figure 7.7. It is not surprising that the time performance of all the four steps increases with the data size. We also notice that, on both datasets, the time performance of the SSE step is not linear to the data size. This is due to the fact that the time complexity of

the SSE step is quadratic to the number of ECs. With the increase of the data size, the average number of ECs increases linearly on the synthetic dataset and super-linearly on the *Orders* dataset. Thus we observe the non-linear relationship between time performance of SSE step and data size. On the synthetic dataset, the dominant time factor is always the time of the SSE step. This is because of the large average number of ECs (can be as large as 1 million) and the quadratic complexity of the SSE step. In contrast, the average number of ECs is at most $11,828$ on the *Orders* dataset. So even though the synthetic dataset has fewer and smaller MASs than the *Orders* dataset, the vast difference in the number of ECs makes the SSE step takes the majority of the time performance on the synthetic dataset.

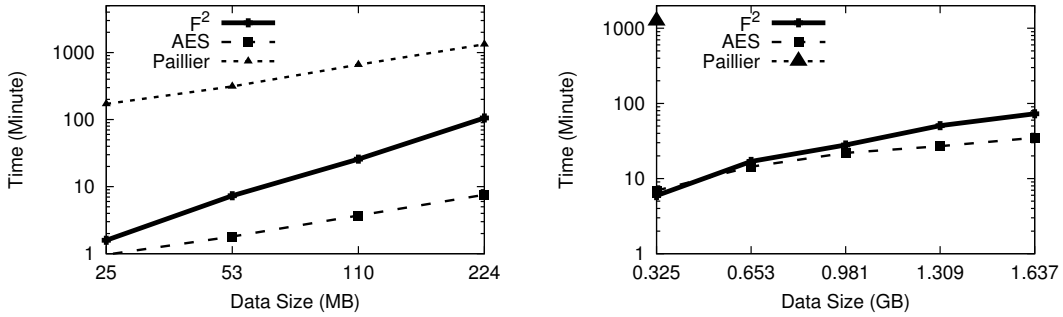
To sum up the observations above, our F^2 approach can be applied on the large datasets with high security guarantee, especially for the datasets that have few number of ECs on the MASs. For instance, it takes around 30 minutes for F^2 to encrypt the *Orders* dataset of size 1GB with security guarantee of $\alpha = 0.2$.



(a) Synthetic dataset (α=0.25) (b) Orders dataset (α=0.2)

Figure 7.7: Time performance for various data sizes

We also compare the time performance of F^2 with *AES* and *Paillier*. The result is shown in Figure 7.8. It is not surprising that F^2 is slower than *AES*, as it



(a) Synthetic dataset (α=0.25) (b) Orders dataset (α=0.2)

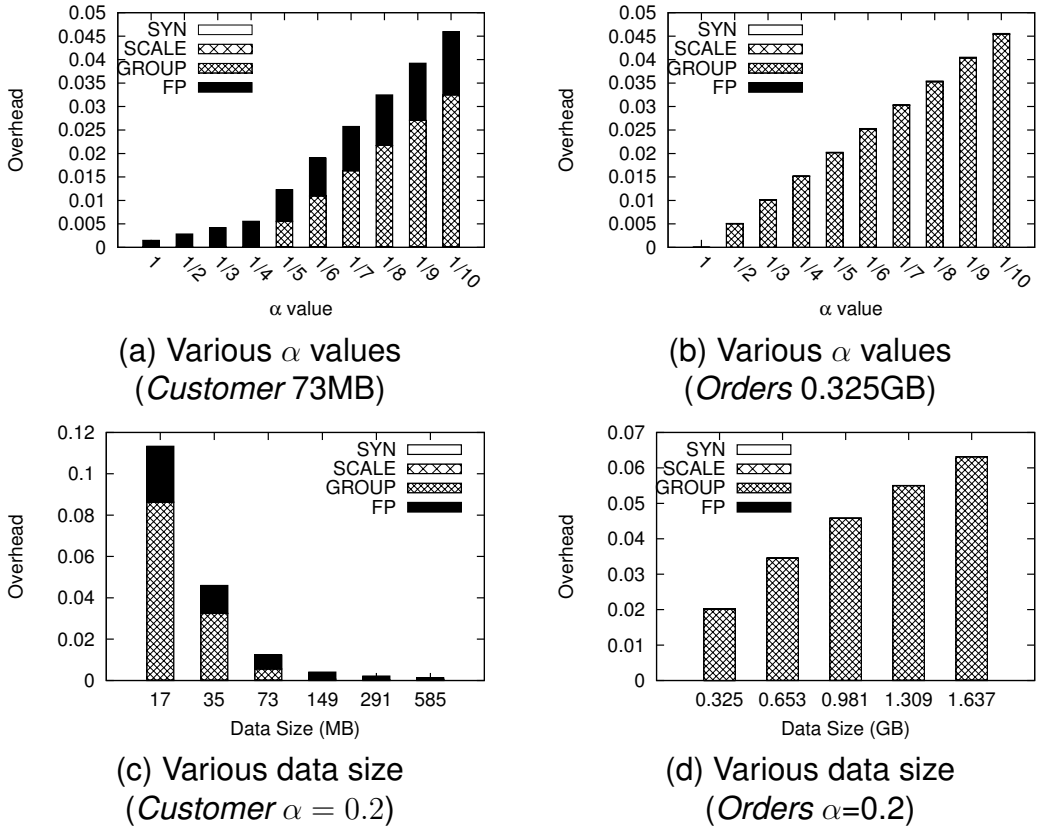
Figure 7.8: Time performance Comparison

has to handle with the FD-preserving requirement. On the other hand, even though F^2 has to take additional efforts to be FD-preserving (e.g., finding $MASs$, splitting and scaling, etc.), its time performance is much better than *Paillier*. This shows the efficiency of our probabilistic encryption scheme. It is worth noting that on the *Orders* dataset, *Paillier* takes 1247.27 minutes for the data of size 0.325GB, and cannot finish within one day when the data size reaches 0.653GB. Thus we only show the time performance of *Paillier* for the data size that is below 0.653GB.

7.4.3 Amounts of Artificial Records

In this section, we measure the amounts of the artificial records added by F^2 on both the *Orders* and *Customer* datasets. We measure the amounts of the artificial records added by Step 2.1 grouping (GROUP), Step 2.2 splitting-and-scaling encryption (SCALE), Step 3 conflict resolution (SYN), and Step 4 eliminating false positive FDs (FP) individually. For each step, we measure the data size before and after the step, let them be s and s', and calculate the *space overhead* $r = \frac{s'-s}{s}$.

On the *Customer* dataset, we measure the overhead under various α values in Figure 7.9 (a). We observe that the GROUP and FP steps introduce most of the



(a) Various α values (*Customer* 73MB)

(b) Various α values (*Orders* 0.325GB)

(c) Various data size (*Customer* $\alpha = 0.2$)

(d) Various data size (*Orders* α=0.2)

Figure 7.9: Amounts of Artificial Records Added by F^2

overhead. The overhead increases when α decreases, since smaller α value requires larger $ECGs$, and thus more collision-free equivalence classes. But in general, the overhead is very small, always within in 5%. This is because the domain size of the attributes in MASs of the *Customer* dataset is large. For instance, both the *C_Last* and *C_Balance* attribute have more than 4,000 unique values across 120,000 records. Under such setting, different $MASs$ are unlikely collide with each other. As a consequence, the number of artificial records added by the GROUP step is very small. When $\alpha < 0.2$, the GROUP step does not even introduce any overhead. The overhead brought by the FP step increases with the decrease of α value. This is because for each maximum false positive FD, F^2 inserts $2k$ ar-

tificial records, where $k = \lceil \frac{1}{\alpha} \rceil$. As k increases, the number of inserted records decreases. But even for small α value such as $\frac{1}{10}$, the overhead brought by the FP step is still very small (around 1.5%). In any case, the space overhead of F^2 on the *Customers* dataset never exceeds 5%. We also measure the space overhead with various α values on the *Orders* dataset. The result is shown in Figure 7.9 (b). First, the GROUP step adds dominant amounts of new records. The reason is that the domain size of attributes in MASs on the *Orders* dataset is very small. For example, among the 1.5 million records, the *OrderStatus* and *OrderPriority* attributes only have 3 and 5 unique values respectively. Therefore the ECs of these attributes have significant amounts of collision. This requires F^2 to insert quite a few artificial records to construct the $ECGs$ in which ECs are collision-free. Nevertheless, the amounts of artificial records is negligible compared with the number of original records. The space overhead is 4.5% at most.

In Figure 7.9 (c) and (d), we show the space overhead of both datasets of various sizes. On the *Customer* dataset, the overhead reduces with the increase of data size. Such overhead decreases for both GROUP and FP steps. Next we give the reasons. Regarding the GROUP step, this is because in the *Customer* dataset, the collision between ECs is small. It is more likely that the GROUP step can find collision-free ECs to form the ECGs, and thus smaller number of the injected artificial records. Regarding the FP step, its space overhead decreases when the data size grows is due to the fact that the number of artificial records inserted by the FP step is independent of the data size. This makes the number of artificial records constant when the data size grows. Therefore, the overhead ratio decreases for larger datasets. On the *Orders* dataset, again, the GROUP step contributes most of the injected artificial records. However, contrary to the *Orders* dataset, the overhead increases with the data size. This is because the ECs of

Approach	Precision	Recall
F^2 (before Step 4)	0.5385	0.7
F^2 (after Step 4)	1	1
FHOP	0	0
Pallier	0	0
AES	1	1

Table 7.1: FD discovery accuracy

Orders have significant amounts of collision. Thus, the number of ECs increases quadratically with the dataset size. Therefore, the space overhead of the GROUP step increases for larger datasets.

Combining the observations above, our F^2 method introduces insignificant amounts of artificial records. For instance, the amounts of artificial records takes at most 6% for the *Orders* dataset, and 12% for the *Customer* dataset.

7.4.4 FD Discovery Accuracy

We evaluate the FD discovery accuracy of F^2 and the three baseline approaches. We launch this set of experiments on the first $30,000$ records from the *Orders* dataset. In particular, let $\mathcal{F}$ be the set of FDs discovered from the original dataset D, and $\mathcal{F}'$ be the FDs discovered from the encrypted dataset $\hat{D}$. We use *precision* and *recall* to evaluate the accuracy of FD discovery. Formally, the precision is defined as $\frac{|\mathcal{F} \cap \mathcal{F}'|}{|\mathcal{F}'|}$ (i.e., the fraction of the FDs in $\hat{D}$ that also exist in D), while recall is $\frac{|\mathcal{F} \cap \mathcal{F}'|}{|\mathcal{F}|}$ (i.e., the fraction of original FDs in D that can be discovered in $\hat{D}$). Intuitively, an encryption approach is FD-preserving only if both precision and recall are 1. To measure the impact of Step 4 of F^2 on false positive FDs, we also measure the precision and recall before and after Step 4 of F^2.

We report the precision and recall results in Table 7.1. First, we observe that the precision and recall of F^2 (after Step 4) are 1, which demonstrates that

F^2 is FD-preserving. In particular, it shows that Step 4 is effective to remove false positive FDs and recover the real FDs that were eliminiated by the false positive FDs. The other two probabilistic encryption schemes, Pallier and FHOP, cannot preserve any FD at all. AES, a deterministic encryption scheme, is able to retain all the original FDs.

7.4.5 Security Against FA Attack

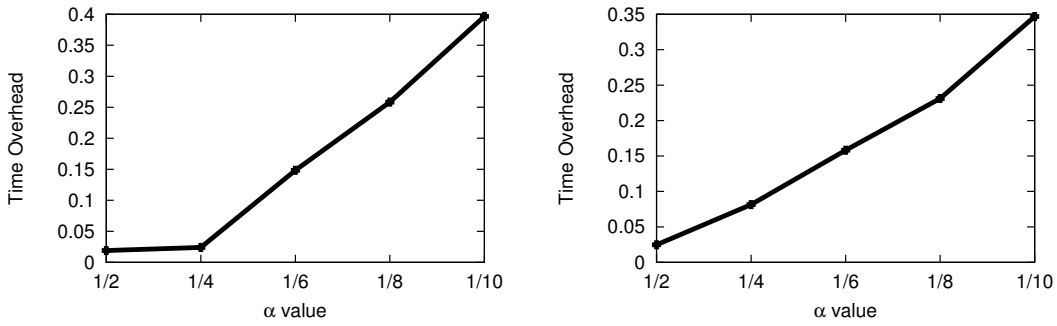
We implement the frequency analysis (FA) attack in [90], and launch the FA attack on the attribute values. Given any attribute, let h be the number of ciphertext values that are successfully mapped to their plaintext values by the FA attack. The accuracy of the FA attack is defined as $acc = \frac{h}{n}$, where n is the amount of column values. Intuitively, the higher acc is, the weaker the encryption scheme is against the FA attack. We execute the FA attack on all the 9 attributes of the first $30,000$ records from the *Orders* dataset that has been encrypted by F^2 and three baseline schemes respectively, and report the average accuracy in Table 7.2. We notice that F^2 can provide comparable security against the FA attack as FHOP and Pallier, even for the weaker security guarantee as $\alpha = 0.25$. When we choose smaller values for α, the security provided by F^2 can be magnificently better than FHOP and Pallier. It proves the elegance of F^2 in defending against FA attack. In contrast, FA attack can recover much more information from the ciphertext value encrypted by AES. In the worst case, in our experiment, we find that the FA attack can successfully recover all the plaintext values from the ciphertext by AES.

Approach	Attack accuracy
$F^2(\alpha = 0.02)$	0.01417
$F^2(\alpha = 0.05)$	0.03192
$F^2(\alpha = 0.1)$	0.0719
$F^2(\alpha = 0.25)$	0.1056
FHOP	0.1214
Pallier	0.1002
AES	0.3395

Table 7.2: Security against FA attack

7.4.6 Outsourcing VS. Local Computations

First, we compare the data owner's performance of finding FDs locally and encryption for outsourcing. We implemented the TANE algorithm [62] and applied it on our datasets. First, we compare the time performance of finding FDs locally (i.e. applying TANE on the original dataset D) and outsourcing preparation (i.e., encrypting D by F^2). It turns out that finding FDs locally is significantly slower than applying F^2 on the synthetic dataset. For example, TANE takes 1,736 seconds on the synthetic dataset whose size is 25MB to discover FDs, while F^2 only takes 2 seconds.



(a) Customer dataset (73MB) (b) Orders dataset (0.325GB)

Figure 7.10: FD Discovery Time Overhead

Second, we compare the performance of discovering FDs from the original and encrypted data, for both *Customer* and *Orders* datasets. We define the

dependency discovery time overhead $o = \frac{T'-T}{T}$, where T and T' are the time of discovering FDs from D and $\hat{D}$ respectively. The result is shown in Figure 7.10. For both datasets, the time overhead is small. It is at most 0.4 for the *Customers* dataset and 0.35 for the *Orders* dataset. Furthermore, the discovery time overhead increases with the decrease of α value. This is because with smaller α, the GROUP and FP steps insert more artificial records to form ECGs for higher security guarantee. Consequently the FD discovery time increases. This is the price to pay for higher security guarantee.

Chapter 8
Future Work

In this section, I discuss my future work.

8.1 Privacy-preserving Outsourced Data Inconsistency Repair based on Approximate Functional Dependencies

Integrity constraints such as *functional dependencies* (FD) play an important role in database design. So far, we only consider the scenarios where the FDs are either available to the client, or hold exactly in an initial version of the dataset (Chapter 6 and 7). However, in an operational database, these constraints are often violated due to various reasons, e.g., by data integration from other resources or careless maintenance. Under this circumstance, the original functional dependencies do not exist any more. Instead, they hold *approximately* on the dirty database (i.e. they hold exactly on a large portion of the dirty data). We call them *approximate functional dependencies* (AFDs) [62]. There are a couple of metrics to measure if a AFD holds in a database. Among them, we use the g_3 error:

$$g_3(X \to Y) = 1 - \sum_{EC \in \Pi_X} max\{|EC'| \mid EC' \in \Pi_{X \cup Y} \text{ and } EC' \subset EC\}/n, \qquad (8.1)$$

where n is the number of records in the dataset, EC is an equivalence class denoting the set of tuples with the same attribute values on X, and Π_X is the partition of attribute set X that consists of all the equivalence classes. Equation 8.1 measures the minimum fraction of records to be deleted from database for the $AFD: X \to Y$ to hold exactly. Given a dirty database D, an $AFD: X \to Y$ holds if

and only if $g_3(X \rightarrow Y) \leq \theta$, where $\theta \in (0, \_1)$ is a user-specified value. The data inconsistency repair algorithm [31, 12, 24] attempts to find low cost changes by which these AFDs hold exactly on the repaired database. We assume that the client has no knowledge about the AFDs. Thus, the server is expected to discover the AFDs and repair the data so that the AFDs hold exactly. It has been shown [31] that the inconsistency repair process is computationally expensive even when the AFDs are known. Therefore, the DCaS paradigm fits the client's need to repair the inconsistencies in her dirty data.

The data privacy attracts our attention in the outsourced data inconsistency repair because there may exist sensitive personal information in the data. How to encrypt the data so that the server is able to perform inconsistency repair becomes a problem. There have been very few efforts on finding data integrity constraints and cleaning of inconsistent data in private settings. We consider the server has prior knowledge about the frequency distribution of data to be outsourced and aim at designing an encryption scheme in this scenario. From our perspective, the problem is challenging in the following ways. First, the encryption scheme must preserve the original AFDs without the awareness of them. Second, we need to change the frequency distribution in the outsourced data. However, the alternations in the frequency could lead to the arbitrary variation of g_3 error of a AFD, which may eliminate some AFDs that originally exist. Third, we do not want to introduce any additional AFDs as the server can produce a different inconsistency repair result from new AFDs.

In order to resolve these challeges, we plan to apply the encryption on the superset of AFDs, like the F^2 approach in Chapter 7. The g_3 error of an attribute set X is defined as $g_3(X) = 1 - \frac{|\Pi_X|}{n}$, where $|\Pi_X|$ is the number of equivalence classes in the partition of X, and n is the number of tuples. In other words, $|\Pi_X|$ measures

the number of distinct attribute values on X. A nice property about the g_3 error between an attribute set and an AFD is $g_3(X \rightarrow Y) \leq g_3(X)$ [62]. We can apply this property to find the appropriate granularity to apply encoding. To be specific, we find the maximal attribute set X s.t. $g_3(X) \leq \theta$, where θ is the user-specified threshold value to determine if an AFD holds or not. Intuitively, for any attribute $A \notin X$, $X \rightarrow A$ must be an AFD. This is because $g_3(X \rightarrow A) \leq g_3(X) \leq \theta$. We apply frequency-hiding encryption on the attribute set X to defend against the frequency analysis attack. To preserve the AFDs, we make sure that after encryption, $g_3(X)$ remains smaller than θ. This can be accomplished by intensionally inserting a certain amount of artificial equivalence classes with small frequency (can be equal to 1).

8.2 Authenticated Outsourced Data Inconsistency Repair

Data inconsistency repair mainly consists of two steps. The first step involves discovering (approximate) functional dependencies from the dirty dataset. In the second step, a minimal-repair strategy is proposed to make the data and the functional dependencies consistent. It has been well known that both steps demand extensive computational efforts. Therefore, it becomes a cost-effective solution for the data owner (client) to outsource the inconsistency repair service to a server. The server conducts the essential computation to return a consistent version of the data which is minimally different from the original dataset. Bohannon et al. [12] design a cost-based model to fix the inconsistency problem by value modification with small repair cost. The key idea of [12] is to eliminate the inconsistency by resolving the conflicting equivalence classes, where an equivalence class is a set of tuples with the same values on a certain attribute set. To deliver small cost, the

algorithm iteratively finds the equivalence classes with the smallest repair cost, and assigns a new value which yields the smallest cost. As the server is not trusted, it is essential to provide the client an efficient authentication framework to verify the correctness of outsourced data inconsistency repair result.

However, the authentication of outsourced inconsistency repair is a complex procedure. As the first step, we need to authenticate the discovered approximate functional dependencies (AFDs). As the complexity of AFD discovery is exponential to the number of attributes, it is very difficult to design an efficient verification method to check the correctness of the AFDs discovered by the server. Moreover, given a set of conflicting equivalence classes, it is possible that there exist multiple values with the smallest repair cost. In other words, it is a non-deterministic procedure to calculate the fixing value. Therefore, we need to design a deterministic authentication algorithm to verify the non-deterministic procedure. The study in [12] proved that given a single AFD, it is NP-complete to find a repair with the smallest cost. In our setting, with high probability, there exist multiple AFDs. Thus, the inconsistency repair is actually NP-complete. It is extremely challenging to design an efficient authentication scheme to verify a NP-complete process.

The authentication of inconsistency repair can be split into two steps. In the first step, we verify the correctness of the discovered AFDs. $X \rightarrow A$ is taken as an AFD if $g_3(X \rightarrow A) \leq \theta$, and for any attribute B included in X, $g_3(X \setminus \{B\} \rightarrow A) > \theta$. In other words, X is the smallest attribute set such that $g_3(X \rightarrow A) \leq \theta$. Upon receiving the set of AFDs from the server, the client can use a probabilistic way to randomly select a small number of AFDs for verification. Basically the client calculates the exact g_3 errors for $X \rightarrow A$ and $g_3(X \setminus \{B\} \rightarrow A)$, and compare them with θ to check if $X \rightarrow A$ is a valid AFD. The probabilistic approach can catch the misbehavior in AFD discovery with high probability. The second step

checks that for any group of conflicting equivalence classes, the server finds a value with the smallest repair cost to resolve the conflict. The procedure becomes lightweight if we move the verification to Euclidean space. The key idea is that the client maps the string values into the Euclidean space in a similarity preserving way [47, 68, 59, 79, 60]. The group of conflicting equivalence classes are mapped into a set of Euclidean points, each associated with a frequency value that denotes the number of tuples that have the corresponding attribute values. It is *the center of mass* that leads to the minimum sum of the squared Euclidean distances to these embedded points. Therefore, the client can check if the cost of the modified value is minimal by mapping the value to a Euclidean point, and measuring its closeness to the center of mass. It is true that this efficient approach can not verify the correctness of the modified value exactly. However, we can derive a probabilistic guarantee by adjusting an appropriate threshold value of the Euclidean distance.

8.3 Authenticated Outsourced Data Imputation

Real-word datasets are usually incomplete due to missing values. Missing values can occur because of the non-response (no information is provided for one or more items or for a whole unit), or the packet loss in data transmission. Missing data values can introduce a substantial inaccuracy in data analysis. An easy fix to this problem is to delete the records with missing fields. In this way, the bias introduced by missing values is removed. However, it can significantly degrade the analysis accuracy because of the serious information loss.

In order to fill in the missing fields with plausible values, data imputation is the process of replacing the missing data with substituted values. Data imputation has been studied for more than two decades [40, 43, 18]. In the earlier stage, the

values are assumed to be missing at random, so as to decrease the amount of bias in the data. To impute the missing values, random value or the mean value is inserted to the missing fields. But this approach may accidentally remove the outliers from the dataset by ignoring their specific values in other attributes.

Recently, a novel missing value imputation technique [102] using existing patterns of the dataset including co-appearences of attribute values, correlations among the attributes and similarity of values belonging to an attribute is developped. This process consists of three essential steps. In the first step, a co-appearance square-matrix C is generated from the incomplete dataset. The dimension of the matrix is the sum of domain sizes of all attributes. $C[i, j]$ denotes the number of occurrences that the $i-$th and $j-$th value co-exist in the records. After that, for each attribute, a similarity matrix is computed by calculating the pairwise similarity between all attribute values. In the last step, for each missing value, we generated an imputed value by considering all the available attribute values of the specific record, the co-appearence matrix, and the similarity matrix for all the attributes. The last step is repeated until the there is no change in any imputated value. This complicated approach [102] achieves significantly better imputation accuracy than the existing techniques.

Considering that the replacement of missing values has an vital influence on the data quality, we aim at designing a lightweight authentication framework to verify the data imputation result returned by the untrusted server. In order to authenticate the outsourced data imputation based on [102] that achieves significantly better imputation accuracy than the existing techniques, we face three unique challenges. First, the imputation of the missing values is not independent. The value imputated for x has an impact on the value of y. This makes it impossible to create verification objects (VOs) for different missing values separately.

Second, it is extremely difficult to design an efficient authenticated data structure to verify the similarity matrix computed in the second step. This structure must enable the client to check the correctness of the similarity calculation between any pair of attribute values with cheap computational cost. Third, the imputation may iterate for a large amount of loops. To alleviate the computational cost at the client side, we must design a solution to authenticate the final result without checking the correctness of each imputation loop.

From the above reasoning, we believe that there is no practical deterministic solution to authenticated outsourced data imputation. Thus we concentrate on a probabilistic method to catch the misbehavior in data imputation with high probability. We notice that the co-appearence matrix plays a vital role in data imputation. To be specific, if there exist two attributes, namely A and B, and if the attribute value a is always associated with the value b, then in case of the missing value on the A attribute in a record which has b on the B attribute, then the original value on attribute A should be a. Based on this observation, to authenticate the data imputaiton on the dataset D, we can append two attributes to D. These two attributes must share the same partition, i.e., there is a tight binding between the values in these two attributes. We intensionally remove a small amount of attribute values in these two attributes, and let the server imputes the missing values. After the server finishes the imputation, the client compares the imputated values on the additional attributes against the original values. If they do not match, then the client concludes that the server does not execute data imputation faithfully.

Chapter 9
Conclusion

In this thesis, we present our research on privacy-preserving and authenticated outsourced data cleaning. When sending the data to an untrusted third party service provider for data cleaning, it is necessary to prevent the sensitive information in the data from leakage. What makes the problem more challenging is that the server may posses auxiliary information about the private dataset, and exploit it to infer the sensitive data values. On the other hand, it is paramount to provide the client with an efficient way to check if the server is executes the data cleaning computation faithfully. Among various data cleaning tasks, we concentrate on data deduplication and data inconsistency repair. Data deduplication is becoming increasingly important as the volume and the need to share data explodes. In these scenarios, data deduplication is the key procedure in combine data residing in different sources. Data inconsistency repair modifies the data to make it consistent with the integrity constraints.

In the outsourced data deduplication setting, we aim at a privacy-preserving encoding framework to assure the data privacy, and an efficient authentication framework for the client to verify the correctness of the results returned by the server. To reach the privacy-preserving goal, we design two encoding schemes to transform the original data into another format. Our locality-sensitive hashing based (LSHB) scheme utilizes locality-sensitive hashing (LSH) to map strings to hash values such that similar strings share similar hash values with high probability. The embedding and homophonic substitution (EHS) scheme maps string values to a Euclidean space and achieve flattened frequency distribution in the Euclidean

space. We prove the robustness of our two methods against the attacks that exploit the auxiliary frequency distribution and the knowledge of the encoding algorithms. At the same time, both approaches guarantee high data deduplication accuracy. In an effort to authenticate the outsourced data deduplication, we require the server to return the verification object (VO) to prove the soundness and completeness of the result. In specific, we design an authentication tree structure named MB-tree. The server uses MB-tree as an indexing structure to search for matching pairs. During the tree traversal process, the server can construct the VO that consists of elements from the tree. By verifying the VO, the client is able to catch the servers cheating behaviors such as tampered records, as well as soundness and completeness violations. Theoretical analysis demonstrates that the verification cost at the client side is much cheaper than a natural execution of deduplication.

In the outsourced data inconsistency repair, we consider that two cases of adversary and security model. In the first case, only a part of the data values are sensitive. The adversary has knowledge about the FDs in the dataset, and thus expliots such information to infer the sensitive values. We prove that it is a NP-complete problem to find the minimum amount of insensitive data values for encryption to remove the inference channel based on adversarial FDs. Following that, we design an efficient heuristic algorithm that only encrypts a small number of additional values and is safe against the FD-attack. In the second setting, all the data cells demands protection. The server has the frequency distribution knowledge about the dataset. We design an efficient probabilistic encryption encheme that provides provable security guarantee against the frequency analysis attack. At the same time, even without the awareness of the FDs, the encryption scheme is guaranteed to preserves all the original FDs. Given the weak computational power at the client side, the encryption scheme is much more efficient than conducting

inconsistency repair locally.

The proposed encryption and authentication schemes consist of a robust system for privacy-preserving and authenticated outsourced data cleaning. We believe that this suit of technology can address the security and privacy concerns in outsourced computing. By applying the proposed techniques, small- and medium-sized organizations can enjoy the benefits of cloud computing in a trustful way.

Bibliography

[1] `http://fortune.com`.

[2] `http://openrefine.org/`.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[5] Tristan Allard, Georges Hébrail, Florent Masseglia, and Esther Pacitti. Chiaroscuro: Transparency and privacy for massive personal time-series clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 779–794, 2015.

[6] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of the VLDB Endowment*, 2006.

[7] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, and Franck Youssef. Transparent data deduplication in the cloud. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 886–900, 2015.

[8] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, volume 74, pages 580–583, 1974.

[9] Mikhail J Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, pages 39–44, 2003.

[10] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Symposium on Foundations of Computer Science*, pages 394–403, 1997.

[11] George Beskales, Ihab F Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proceedings of the VLDB Endowment*, 3(1-2):197–207, 2010.

[12] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 143–154, 2005.

[13] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *IEEE International Conference on Data Engineering*, pages 746–755, 2007.

[14] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam Oneill. Order-preserving symmetric encryption. In *Advances in Cryptology-EUROCRYPT*, pages 224–241. 2009.

[15] Luca Bonomi, Li Xiong, Rui Chen, and Benjamin Fung. Frequent grams based embedding for privacy preserving record linkage. In *Proceedings of*

the ACM International Conference on Information and Knowledge Management, pages 1597–1601, 2012.

[16] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 341–357, 2013.

[17] Alexander Brodsky, Csilla Farkas, and Sushil Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):900–919, 2000.

[18] Sharon R Browning. Missing data imputation and haplotype phase inference for genome-wide association studies. *Human Genetics*, 124(5):439–450, 2008.

[19] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the International Conference on Management of Data*, 2003.

[20] Qian Chen, Haibo Hu, and Jianliang Xu. Authenticated online data integration services. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 167–181, 2015.

[21] Rui Chen, Noman Mohammed, Benjamin CM Fung, Bipin C Desai, and Li Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[22] Rui Chen, Qian Xiao, Yu Zhang, and Jianliang Xu. Differentially private high-dimensional data publication via sampling-based inference. In *Proceedings*

of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 129–138, 2015.

[23] James Cheng, Ada Wai-chee Fu, and Jia Liu. K-isomorphism: privacy preserving network publication against structural attacks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 459–470, 2010.

[24] Fei Chiang and Renée J Miller. A unified model for data and constraint repair. In *International Conference on Data Engineering (ICDE)*, pages 446–457, 2011.

[25] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1):90–121, 2005.

[26] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the ACM Workshop on Cloud Computing Security*, pages 85–90, 2009.

[27] Tim Churches and Peter Christen. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making*, 4(1):9, 2004.

[28] Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):22, 2010.

[29] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64–78, 2001.

[30] Douglas Comer. Ubiquitous b-tree. *Computing Surveys*, 1979.

[31] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the International Conference on Very Large Data Bases*, pages 315–326, 2007.

[32] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM Journal of Research and Development*, 1994.

[33] Carlos Coronel, Steven Morris, and Peter Rob. *Database systems: design, implementation, and management*. Cengage Learning, 2009.

[34] Carlo Curino, Evan PC Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational cloud: A database-as-a-service for the cloud. 2011.

[35] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[36] Boxiang Dong, Ruilin Liu, and Hui Wendy Wang. Result integrity verification of outsourced frequent itemset mining. In *Data and Applications Security and Privacy XXVII*, pages 258–265. 2013.

[37] Boxiang Dong, Ruilin Liu, and Hui Wendy Wang. Trust-but-verify: Verifying result correctness of outsourced frequent itemset mining in data-mining-as-a-service paradigm. *IEEE Transactions on Services Computing*, 9(1):18–32, 2016.

[38] Boxiang Dong, Ruilin Liu, and Wendy Hui Wang. Integrity verification of outsourced frequent itemset mining with deterministic guarantee. In *IEEE International Conference on Data Mining (ICDM)*, pages 1025–1030, 2013.

[39] Boxiang Dong, Ruilin Liu, and Wendy Hui Wang. Prada: Privacy-preserving data-deduplication-as-a-service. In *Proceedings of the ACM International Conference on Conference on Information and Knowledge Management*, pages 1559–1568, 2014.

[40] LL Doove, Stef Van Buuren, and Elise Dusseldorp. Recursive partitioning for missing data imputation in the presence of interaction effects. *Computational Statistics & Data Analysis*, 72:92–104, 2014.

[41] Durham E. Ashley, Kantarcioglu M., Xue Y., Kuzu M., and Malin Bradley. Composite bloom filters for secure record linkage. In *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[42] Wayne W Eckerson. Data quality and the bottom line. *The Data Warehouse Institute Report*, 2002.

[43] Bradley Efron. Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association*, 89(426):463–475, 1994.

[44] Stephan Eidenbenz and Christoph Stamm. Maximum clique and minimum clique partition in visibility graphs. In *Theoretical Computer Science*, pages 200–212. 2000.

[45] Fatih Emekçi, Ozgur D Sahin, Divyakant Agrawal, and Amr El Abbadi. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering*, 63(2):348–361, 2007.

[46] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[47] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.

[48] Wenfei Fan, Floris Geerts, Shuai Ma, and Heiko Muller. Detecting inconsistencies in distributed data. In *IEEE International Conference on Data Engineering (ICDE)*, pages 64–75, 2010.

[49] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO*, pages 465–482. 2010.

[50] Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Divesh Srivastava, et al. Approximate string joins in a database (almost) for free. In *Proceedings of the International Conference on Very Large Data Bases*, volume 1, pages 491–500, 2001.

[51] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340, 2010.

[52] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceed-*

ings of the ACM SIGMOD International Conference on Management of Data, pages 216–227, 2002.

[53] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *International Conference on Data Engineering*, pages 29–38, 2002.

[54] Hakan Hacigms, Balakrishna R. Iyer, and Sharad Mehrotra. Secure computation on outsourced data: A 10-year retrospective. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 16–27, 2014.

[55] Isabelle Hang, Florian Kerschbaum, and Ernesto Damiani. Enki: access control for encrypted query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 183–196, 2015.

[56] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021–1032, 2010.

[57] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. Scalable discovery of unique column combinations. *Proceedings of the VLDB Endowment*, 7(4):301–312, 2013.

[58] G Hjaltason and H Samet. Contractive embedding methods for similarity searching in metric spaces. Technical report, Computer Science Department, University of Maryland.

[59] Gisli R. Hjaltason and Hanan Samet. Properties of embedding methods for

similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[60] Gabriela Hristescu and Martin Farach-Colton. Cluster-preserving embedding of proteins. Technical report, Computer Science Department, Rutgers University, 1999.

[61] Haibo Hu, Jianliang Xu, Qian Chen, and Ziwei Yang. Authenticating location-based services without compromising location privacy. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 301–312, 2012.

[62] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[63] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Karl Aberer, et al. Privacy-preserving schema reuse. In *Database Systems for Advanced Applications*, pages 234–250, 2014.

[64] Ali Inan, Murat Kantarcioglu, Elisa Bertino, and Monica Scannapieco. A hybrid approach to private record linkage. In *International Conference on Data Engineering*, pages 496–505, 2008.

[65] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita, and Elisa Bertino. Private record matching using differential privacy. In *Proceedings of the International Conference on Extending Database Technology*, pages 123–134, 2010.

[66] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita, and Elisa Bertino. A hybrid

approach to private record matching. *IEEE Transactions on Dependable and Secure Computing*, 9(5):684–698, 2012.

[67] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[68] Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 137–146, 2003.

[69] Murat Kantarcioglu, Chris Clifton, et al. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

[70] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667, 2015.

[71] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

[72] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the International Conference on Management of Data*, 2006.

[73] Mehmet Kuzu, Murat Kantarcioglu, Elizabeth Durham, and Bradley Malin. A constraint satisfaction cryptanalysis of bloom filters in private record link-

age. In *International Symposium on Privacy Enhancing Technologies*, pages 226–245, 2011.

[74] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[75] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. In *International Conference on Data Engineering*, 2008.

[76] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 121–132. ACM, 2006.

[77] Rui Li, Alex X Liu, Ann L Wang, and Bezawada Bruhadeshwar. Fast range query processing with strong privacy protection for cloud computing. *Proceedings of the VLDB Endowment*, 7(14):1953–1964, 2014.

[78] Yehuda Lindell and Benny Pinkas. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[79] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[80] An Liu, Kai Zhengy, Lu Liz, Guanfeng Liu, Lei Zhao, and Xiaofang Zhou. Efficient secure similarity computation on encrypted trajectory data. In *IEEE International Conference on Data Engineering*, pages 66–77, 2015.

[81] Jian Liu, N Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 874–885, 2015.

[82] Ruilin Liu, Hui Wendy Wang, Anna Monreale, Dino Pedreschi, Fosca Giannotti, and Wenge Guo. Audio: An integrity auditing framework of outlier-mining-as-a-service systems. In *Machine Learning and Knowledge Discovery in Databases*, pages 1–18. 2012.

[83] Ruilin Liu, Hui Wendy Wang, Philippos Mordohai, and Hui Xiong. Integrity verification of k-means clustering outsourced to infrastructure as a service (iaas) providers. 2013.

[84] Ruilin Liu, Hui Wendy Wang, and Changhe Yuan. Result integrity verification of outsourced bayesian network structure learning. 2014.

[85] Richard Marsh. Drowning in dirty data? it's time to sink or swim: A four-stage methodology for total data quality management. *Journal of Database Marketing & Customer Strategy Management*, 12(2):105–112, 2005.

[86] Mirjana Mazuran, Elisa Quintarelli, Letizia Tanca, and Stefania Ugolini. Semi-automatic support for evolving functional dependencies. In *International Conference on Extending Database Technology (EDBT)*, 2016.

[87] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[88] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 19–30, 2009.

[89] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *ACM Transactions on Storage (TOS)*, 2(2):107–138, 2006.

[90] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655, 2015.

[91] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2005.

[92] HweeHwa Pang and Kyriakos Mouratidis. Authenticating the query results of text search engines. *Proceedings of the VLDB Endowment*, 2008.

[93] Dimitrios Papadopoulos, Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Practical authenticated pattern matching with optimal proof size. *Proceedings of the VLDB Endowment*, 8(7):750–761, 2015.

[94] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. Nearest neighbor search with strong location privacy. *Proceedings of the VLDB Endowment*, 3(1-2):619–629, 2010.

[95] Stavros Papadopoulos, Graham Cormode, Antonios Deligiannakis, and Minos Garofalakis. Lightweight authentication of linear algebraic queries on data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 881–892, 2013.

[96] Stavros Papadopoulos, Lixing Wang, Yin Yang, Dimitris Papadias, and Panagiotis Karras. Authenticated multistep nearest neighbor search. *Transactions on Knowledge and Data Engineering*, 2011.

[97] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[98] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy (SP)*, pages 238–252, 2013.

[99] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*, pages 422–439, 2012.

[100] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Processing queries on an encrypted database. *Communications of the ACM*, 55(9):103–111, 2012.

[101] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engieering Bulletin*, 23(4):3–13, 2000.

[102] Md Geaur Rahman and Md Zahidul Islam. Fimus: A framework for imputing missing values using co-appearance, correlation and similarity analysis. *Knowledge-Based Systems*, 56:311–327, 2014.

[103] Pradeep Ravikumar, William W Cohen, and Stephen E Fienberg. A secure protocol for computing string distance metrics. In *the Workshop on Privacy and Security Aspects of Data Mining at the International Conference on Data Mining*, pages 40–46, 2004.

[104] R.C.Merkle. Protocols for public key cryptosystems. In *Symposium on Security and Privacy*, 1980.

[105] Ronald L Rivest et al. Chaffing and winnowing: Confidentiality without encryption. *CryptoBytes (RSA Laboratories)*, 4(1):12–17, 1998.

[106] Tahmineh Sanamrad, Lucas Braun, Donald Kossmann, and Ramarathnam Venkatesan. Randomly partitioned encryption for cloud databases. In *Data and Applications Security and Privacy XXVIII*, pages 307–323. 2014.

[107] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 653–664, 2007.

[108] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41, 2009.

[109] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking proof-based verified computation a few

steps closer to practicality. In *The USENIX Security Symposium*, pages 253–268, 2012.

[110] Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *The Network and Distributed System Security Symposium*, volume 1, page 17, 2012.

[111] Yasin N Silva, Walid G Aref, and Mohamed H Ali. The similarity join database operator. In *IEEE International Conference on Data Engineering*, volume 10, pages 892–903, 2010.

[112] Radu Sion. Query execution assurance for outsourced databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 601–612, 2005.

[113] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography–PKC*, pages 420–443. 2010.

[114] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Symposium on Security and Privacy*, pages 44–55, 2000.

[115] Mark W Storer, Kevin Greenan, Darrell DE Long, and Ethan L Miller. Secure data deduplication. In *Proceedings of the International Workshop on Storage Security and Survivability*, pages 1–10, 2008.

[116] T-A Su and Gultekin Ozsoyoglu. Controlling fd and mvd inferences in mul-

tilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):474–485, 1991.

[117] Nilothpal Talukder, Mourad Ouzzani, Ahmed K Elmagarmid, and Mohamed Yakout. Detecting inconsistencies in private data with secure function evaluation. Technical report, Computer Science Department, Purdue University, 2011.

[118] Florian Tramèr, Zhicong Huang, Jean-Pierre Hubaux, and Erman Ayday. Differential privacy with bounded priors: reconciling utility and privacy in genome-wide association studies. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1286–1297, 2015.

[119] Stanley Trepetin. Privacy-preserving string comparisons in record linkage systems: a review. *Information Security Journal: A Global Perspective*, 17(5-6):253–266, 2008.

[120] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, 2002.

[121] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 457–468, 2014.

[122] Xiong Wang, Jason TL Wang, King-Ip Lin, Dennis Shasha, Bruce A Shapiro, and Kaizhong Zhang. An index structure for data mining and clustering. *Knowledge and Information Systems*, 2(2):161–184, 2000.

[123] Wai Kit Wong, David W Cheung, Edward Hung, Ben Kao, and Nikos Mamoulis. An audit environment for outsourcing of frequent itemset mining. *Proceedings of the VLDB Endowment*, 2(1):1162–1173, 2009.

[124] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. Secure query processing with data interoperability in a cloud database environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1395–1406, 2014.

[125] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 101–110, 2001.

[126] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the VLDB Endowment*, 1(1):933–944, 2008.

[127] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *Proceedings of the International Conference on Very Large Data Bases*, pages 782–793, 2007.

[128] Xiaofeng Xu, Li Xiong, and Jinfei Liu. Database fragmentation with confidentiality constraints: A graph search approach. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*, pages 263–270, 2015.

[129] Su Yan, Dongwon Lee, Min-Yen Kan, and Lee C Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 185–194, 2007.

[130] Yanling Yang, Kaizhong Zhang, Xiong Wang, Jason TL Wang, and Dennis Shasha. An approximate oracle for distance in metric spaces. In *Annual Symposium on Combinatorial Pattern Matching*, pages 104–117, 1998.

[131] Yin Yang, Dimitris Papadias, Stavros Papadopoulos, and Panos Kalnis. Authenticated join processing in outsourced databases. In *Proceedings of the International Conference on Management of Data*, 2009.

[132] Andrew C Yao. Protocols for secure computations. In *The IEEE Symposium on Foundations of Computer Science (SFCS)*, pages 160–164, 1982.

[133] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *The IEEE Symposium on Foundations of Computer Science (SFCS)*, pages 162–167, 1986.

[134] Zhenjie Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, and Divesh Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *Proceedings of the International Conference on Management of Data*, 2010.

Boxiang Dong

3 Del Ln, Commack, NY 11725
(201) 707 - 1480, bdong@stevens.edu
http://www.cs.stevens.edu/~bdong/

Date and Place of Birth

April 17, 1989, Shanxi, China

Education

Ph.D. Computer Science, Stevens Institute of Technology, NJ, USA 08/2011 - 12/2016

B.E. Software Engineering, Dalian University of Technology, Dalian, P.R. China 09/2007 - 06/2011

Industrial Experience

Research Intern 09/2014 - 08/2015
Mentor: Dr. Zhengzhang Chen, Dr. Tan Yan NEC Labs America, NJ

Publications

Boxiang Dong, Hui (Wendy) Wang. ARM: Authenticated Approximate Record Matching for Outsourced Databases. *The IEEE International Conference on Information Reuse and Integration (IRI 2016).* Pittsburgh, PA, 2016.

Boxiang Dong, Hui (Wendy) Wang, Jie Yang. Secure Data Outsourcing with Adversarial Data Dependency Constraints. *The IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2016).* New York, NY, 2016.

Boxiang Dong, Ruilin Liu, Hui (Wendy) Wang. Trust-but-Verify: Verifying Result Correctness of Outsourced Frequent Itemset Mining. *The IEEE Transactions on Services Computing, 2015.*

Boxiang Dong, Ruilin Liu, Hui (Wendy) Wang. PraDa: Privacy-preserving Data-Deduplication-as-a-Service. *The ACM International Conference on Information and Knowledge Management (CIKM 2014).* Shanghai, P.R.China, 2014.

Boxiang Dong, Ruilin Liu, Hui (Wendy) Wang. Integrity Verification of Outsourced Frequent Itemset Mining with Deterministic Guarantee. *The IEEE International Conference on Data Mining (ICDM 2013).* Dallas, TX, 2013.

Boxiang Dong, Ruilin Liu, Hui (Wendy) Wang. Result Integrity Verification of Outsourced Frequent Itemset Mining. *The IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2013).* Newark, NJ, 2013.

Weifeng Sun, Juanyun Wang, **Boxiang Dong**, Mingchu Li, Zhenquan Qin. A Mediated RSA-based End Entity Certificates Revocation Mechanism with Secure Concern in Grid. *The International Journal of Information Processing and Management, Volume 1*, 2010.

Weifeng Sun, **Boxiang Dong**, Zhenquan Qin, Juanyun Wang, Mingchu Li. A Low-Level Security Solving Method in Grid. *The International Conference on Intelligent Information Hiding and Multimedia Signal*

Processing. Darmstadt, Germany, 2010

Presentations

ARM: Authenticated Approximate Record Matching for Outsourced Databases. *The IEEE International Conference on Information Reuse and Integration (IRI 2016)*. Pittsburgh, PA, 2016.

Secure Data Outsourcing with Adversarial Data Dependency Constraints. *The IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2016)*. Columbia University, New York, 2016.

Privacy-preserving Data Deduplication-as-a-Service. *Stevens Graduate Research Conference*. Hoboken, NJ. 2016.

Integrity Verification of Outsourced Frequent Itemset Mining with Deterministic Guarantee. *The IEEE International Conference on Data Mining (ICDM 2013)*. Dallas, TX, 2013.

Result Integrity Verification of Outsourced Frequent Itemset Mining. *The IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2013)*. Rutgers University, NJ, 2013.

A Novel Grid Resource Scheduling Model Based on Extended Second Price Sealed Auction. *The International Symposium on Parallel Architectures, Algorithms and Programming*. Dalian, Liaoning, P.R. China. 2010.

Honors

Stevens Graduate Conference Fund, 2013.
IEEE ICDM Student Travel Award, 2013.
ACM SIGMOD Student Travel Award, 2012.

Research Interests

Cybersecurity

- Verifiable computing
- Privacy-preserving data mining
- Intrusion detection

Big data analytics

- Data cleaning, including data integration, data inconsistency repair
- Distributed computing
- Graph mining