CERIAS Tech Report 2017-01
Privacy-Preserving Analysis with Applications to Textual Data
by Balamurugan Anandan
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

PRIVACY-PRESERVING ANALYSIS WITH APPLICATIONS TO TEXTUAL DATA

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Balamurugan Anandan

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2017

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Christopher W. Clifton, Chair

Department of Computer Science

Dr. Jennifer Neville

Department of Computer Science

Dr. Luo Si

Department of Computer Science

Dr. Samuel S. Wagstaff, Jr.

Department of Computer Science

Approved by:

Dr. Sunil Prabhakar

Head of the Department Graduate Program

To

Amma, Appa and *Akka*

ACKNOWLEDGMENTS

This dissertation would not have been possible without the guidance and motivation of my advisor Prof. Chris Clifton. Working with him has helped me grow as an individual, both professionally and personally. I am forever indebted to him.

My sincere gratitude to my committee members Prof. Jennifer Neville, Prof. Luo Si and Prof. Samuel Wagstaff, who have helped me greatly in improving my dissertation.

I would like to acknowledge my friends and colleagues especially Nesreen Ahmed, Srikanth GV, Akash Kumar, Jaewoo Lee, Mummoorthy Murugesan, Ahmet Erhan Nergiz, Keehwan Park, Pedro Pastrano, Ryan Rossi, Siddharth Singh, Christine Task and John Ross Wallrabenstein for their useful personal and technical discussions, which helped me succeed in my PhD.

I express my gratitude to my best friend Vikram Ilavarasan, who has always been there for me. I would like to appreciate my friends Sureshkumar Govindaraj, Raj Prabhu, Ashokvarda Rajagopalan, Vijay Ranganathan, Sakthiyuvraja Sakthivelmurugan and Thillaivasan Veeranathan for helping me get through difficult times.

I am grateful to Dr. Jon Coker and my employer Omnitier for being very flexible and supportive during my final year of PhD. I would also like to thank Micela Shivva for her support during my stay in Rochester.

I sincerely thank Mrs. Patrica Clifton for her hospitality and kindness. I would like to thank my brother-in-law for his support. I would also like to acknowledge my niece Ragavi Rajkumar, who has brought me laughter and joy over the years.

Special thanks to Madhura R Choudhary for her friendship, support and encouragement.

Finally, I want to thank my parents and sister for their unconditional support and sacrifice. To them, I dedicate this dissertation.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Anandan, Balamurugan PhD, Purdue University, May 2017. Privacy-Preserving Analysis with Applications to Textual Data. Major Professor: Christopher W. Clifto.

Textual data plays a very important role in decision making and scientific research, but cannot be shared freely if they contain personally identifiable information. In this dissertation, we consider the problem of privacy-preserving text analysis, while satisfying a strong privacy definition of differential privacy.

We first show how to build a two-party differentially private secure protocol for computing similarity of text in the presence of malicious adversaries. We then relax the utility requirement of computational differential privacy to reduce computational cost, while still giving security with rational adversaries.

Next, we consider the problem of building a data-oblivious algorithm for minimum weighted matching in bipartite graphs, which has applications to computing secure semantic similarity of documents. We also propose a secure protocol for detecting articulation points in graphs. We then relax the strong data-obliviousness definition to give ϵ-data-obliviousness based on the notion of indistinguishability, which helps us to develop efficient protocols for data-dependent algorithms like frequent itemset mining.

Finally, we consider the problem of privacy-preserving classification of text. A main problem in developing private protocols for unstructured data is high dimensionality. This dissertation tackles high dimensionality by means of differentially private feature selection. We show that some of the well known feature selection techniques perform poorly due to high sensitivity and we propose techniques that perform well in a differential private setting. The feature selection techniques are em-

pirically evaluated using differentially private classifiers: naïve Bayes, support vector machine and decision trees.

1 INTRODUCTION

In many real world applications, there is a necessity to share data (e.g., text documents and network data) with others. Sensitive information (e.g., electronic health records) that contain personally identifiable information when disclosed intentionally or inadvertently without proper measures can cause serious privacy concerns. k-safety [1], t-plausibility [2,3] and information theory based sanitizers [4,5] are some of the syntactic and semantic text publishing techniques used for redacting sensitive information before publishing them for data mining purposes. These data publishing techniques are also prone to correlation based inference [6] and may be inadequate for settings where data is shared among multiple parties, who want to learn useful information from their combined data. This dissertation considers various scenarios where privacy is an issue when computing with sensitive textual data and proposes novel algorithms for solving them.

Let us consider a simple example of two mutually distrustful parties, who want to compute the similarity of their input documents (represented by a binary vector) without revealing their input documents. Secure multi-party computation (MPC) deals with the problem of how to securely compute a function among mutually distrustful parties, but a straightforward secure function evaluation approach (blindly computing the result, with neither party learning anything but the final result) may not be sufficient as computing a similarity function like hamming distance or dot product could leak information from the final result. A malicious party whose only intention is to learn if the other party has a particular feature/word in their document can construct their input document with all zeros except for a one as the value for the targeted word/feature in the document vector that he/she wants to learn. The final result is then the other party's value for that targeted word/feature, resulting in a information leak.

Differential privacy, on the other hand, asks the question of what aggregate functions can be computed on private data such that the output does not leak information about an individual in the database. A differentially private function computed using MPC techniques would solve the above problem, as the result would have sufficient noise to mask that single individual's value. Unfortunately, a simple solution like each party contributing noise to the other party does not work in the two-party malicious setting as we will see this in Chapter 3.

Computing a differentially private function securely using multi-party computation techniques prevents private information leakage both in the process, and from information present in the function output, but poses new challenges if any of the parties are malicious. A key challenge in developing a distributed differentially private protocol is

> How to securely sample a pseudo-random number from a Laplace distribution in the two-party malicious setting?

In Chapter 3, we show how to build a two-party differentially private secure protocol in the presence of malicious adversaries. We then relax the utility requirement of computational differential privacy to reduce computational cost, which leads to the notion of security with rational adversaries. Finally, we provide a modified two-party computational differential privacy definition and show correctness and security guarantees in the rational setting.

Applying secure multi-party computation techniques on an algorithm alone does not guarantee privacy, if the underlying algorithm is data-dependent or if the output of the algorithm can leak information. The first issue can be addressed through data-oblivious computation, the second through differential privacy. However, both can be difficult to achieve with graph algorithms. Chapter 4 addresses both problems, demonstrating a differentially private data-oblivious protocol for minimum weighted bipartite matching, minimum vertex cover for bipartite graphs and privately detecting articulation points in an undirected graph.

There are situations where strict data-obliviousness is inefficient. For example, consider an algorithm like frequent itemset mining, whose running time is dependent on the input query (e.g., retrieve the itemsets that satisfy a threshold ϕ.) Frequent itemset mining has been used on text [7, 8] for identifying topics and knowledge discovery. A true data-oblivious algorithm would then have to access every itemset to prevent information leaks due to the sequence and number of memory accesses. But this is infeasible because the run time of the data-oblivious algorithm would be exponential. This raises our next question.

> Is it possible to develop efficient data-oblivious algorithms under a weaker security guarantee?

In Section 4.5, we propose a relaxed data-oblivious definition ϵ-data-obliviousness that provides a weaker notion of data-obliviousness. We also develop an efficient algorithm for frequent itemset mining and prove that it satisfies ϵ-data-obliviousness.

Finally, we consider the problem of differentially private classification on unstructured data. A key challenge in applying differential privacy to text analysis is that the noise added to the feature parameters is directly proportional to the number of parameters learned. While careful feature selection would alleviate this problem, the process of feature selection itself can reveal private information, requiring the application of differential privacy to the feature selection process, which leads us to the question.

> Is it possible to build efficient private classifiers for text that satisfy differential privacy? Given the high dimensionality of text, which feature selection techniques are suitable for differentially private analysis?

In Chapter 5, we analyze the sensitivity of various feature selection techniques used in text classification and show that some of them are not suitable for differentially private analysis due to high sensitivity. We also perform empirical evaluation on differentially private naïve Bayes classifier, support vector machine and decision trees to evaluate the efficiency of the private feature selection methods.

2 BACKGROUND AND RELATED WORK

This chapter provides the standard techniques and definitions from secure multi-party computation and differential privacy for completeness of the solutions proposed in the following chapters.

2.1 Secure Computation

Consider two parties P_1 and P_2 having private inputs, who wish to collaboratively compute a function of their inputs without divulging their inputs because of confidentiality issues. An example scenario occurs in knowledge discovery from sensitive inputs like medical databases. The above problem can be solved by secure multi-party computation (MPC) using well known generic protocols. A brief description of the tools used for building secure protocols are described below.

2.1.1 Garbled Circuits

Garbled circuit technique introduced by Yao in [9] is a generic method for secure two-party computation in the semi-honest setting. It allows two parties having inputs x, y respectively to evaluate an arbitrary function $f(x, y)$ without leaking any information other than what can be inferred from the output and their own input. To summarize, one party generates a Boolean circuit and associates with each input wire i, two random keys w_i^0, w_i^1 corresponding to $0, 1$ bit respectively. Then for each gate, the generator computes $E_{w_i^{b_i}, w_j^{b_j}}(w_k^{o(b_i,b_j)})$ for all inputs $b_i, b_j \in \{0, 1\}$. The four cipher texts corresponding to each gate are then permuted and sent to the evaluator along with the keys corresponding to its own input wires. The evaluator obtains the keys associated with its input using oblivious transfer (OT) and then begins evaluating

the circuit. At the end, the generator reveals the mapping between the output keys to bits.

2.1.2 Malicious Setting

The following definition and description is based on [10] Chapter 7. In a malicious model, the adversaries may arbitrarily deviate from a specified protocol. The security of a protocol in the malicious model is defined by comparing an execution of a protocol in the real model to an execution in the ideal model. In an ideal scenario, there exists an incorruptible trusted third party $\mathcal{T}$ to whom the parties send their inputs. The trusted third party $\mathcal{T}$ computes the function on the inputs and returns back their respective output.

Execution in the ideal model: Let P_0, P_1 be the parties computing the functionality $f = (f_0, f_1)$, $\mathcal{A}$ be an adversary controlling P_i, where $i \in \{0, 1\}$ and $\mathcal{T}$ be the incorruptible trusted third party. Then, an execution in the ideal model proceeds as follows.

Inputs: Each party P_j obtains its input x_j of the same length n and let z be the auxiliary input of the adversary $\mathcal{A}$.

Send inputs to trusted party: The honest party P_{1-i} always sends its received input x_{1-i} to the $\mathcal{T}$. An adversary $\mathcal{A}$ controlling the party P_i may send its received input x_i or send some other input depending upon the auxiliary input z of the same length to $\mathcal{T}$ on behalf of P_i. Let $\bar{x}$ be the input of both the parties.

$\mathcal{T}$ sends output to adversary: $\mathcal{T}$ computes $(f_0(\bar{x}), f_1(\bar{x}))$ and sends $f_i(\bar{x})$ to $\mathcal{A}$ controlling P_i.

Adversary instructs $\mathcal{T}$ to continue or halt: The adversary $\mathcal{A}$ upon receiving its output could either send continue or abort_i to $\mathcal{T}$. If $\mathcal{A}$ sends continue to $\mathcal{T}$, then $\mathcal{T}$ sends $f_{1-i}(\bar{x})$ to the honest party P_{1-i}. Otherwise, if $\mathcal{A}$ sends $abort_i$ to $\mathcal{T}$, then the $\mathcal{T}$ sends $abort_i$ to P_j.

Outputs: An honest party P_{1-i} always outputs whatever it has received from the $\mathcal{T}$. The corrupted party outputs nothing. $\mathcal{A}$ can output a function (efficiently computable) of its input, the auxiliary input z and messages it received from the $\mathcal{T}$.

Let $\text{IDEAL}_{f,\mathcal{A}(z)}(x, y, n)$ be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Execution in the real model: Let π be a two-party protocol for computing f in the real model (in the absence of $\mathcal{T}$), $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary that sends all messages in place of the corrupted party. The honest party follows the protocol. Then, the joint execution of π with inputs (x, y), and auxiliary input z to $\mathcal{A}$ in the real model is denoted as $\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, n)$, is defined as the output of the honest party and the adversary $\mathcal{A}$ resulting from the protocol execution. The secure two-party computation is defined as follows:

Definition 2.1.1 (Secure Two-Party Computation) *Protocol π is said to securely compute f with abort in the presence of malicious adversaries if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $i \in \{0, 1\}$*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(z),i}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$$

2.1.3 Homomorphic Encryption

A public-key encryption scheme is homomorphic if it allows computations on the ciphertexts without decrypting them first. Let E_{pk} and D_{sk} denote the encryption and decryption functions with pk and sk as the public and private keys respectively. Then, an additive homomorphic encryption system has the following property. Given an encryption of m_1 and m_2, $E_{pk}(m_1)$ and $E_{pk}(m_2)$, there exists an efficient algorithm to compute the encryption of m_1+m_2, denoted by $E_{pk}(m_1+m_2) = E_{pk}(m_1) \oplus E_{pk}(m_2)$.

The homomorphic addition is denoted by the operator $\oplus$. A formal definition is given below.

Definition 2.1.2 *[11] A public key encryption scheme (G, E, D) is additively homomorphic if for all n and all (pk, sk) output by $G(1^n)$, it is possible to define groups $\mathbb{M}, \mathbb{C}$ such that:*

- *The plaintext space is $\mathbb{M}$, and all ciphertexts output by E_{pk} are elements of $\mathbb{C}$.*
- *For every $m_1, m_2 \in \mathbb{M}$, it holds that*

$$pk, c_1 = E_{pk}(m_1), c_1 \otimes E_{pk}(m_2) \quad \equiv \quad pk, E_{pk}(m_1), E_{pk}(m1 + m2)$$

where the group operations are carried out in $\mathbb{C}$ and $\mathbb{M}$, respectively.

Such an additive homomorphic scheme also supports the multiplication of a ciphertext and a scalar constant by repeated addition. i.e., Given a constant c and the encryption of m_1, $E_{pk}(m_1)$, there exists an efficient algorithm to compute the encryption of $c \times m_1$, denoted by $E_{pk}(c \times m_1) = c \otimes E_{pk}(m_1)$. The operator $\otimes$ is used to denote the homomorphic multiplication of a constant with a ciphertext. An encryption scheme that meets the above definition is Paillier [12]. The threshold variation of Paillier is shown in [13], which is used for building protocols in Chapter 3. A few other protocols proposed in [13] that are useful for building protocols in the malicious setting are as follows.

Proving that you know a plain text (POK): A prover P_i than created the encryption $E_{pk}(x)$ can give the zero-knowledge proof of knowledge $POK(E_{pk}(x))$ that it knows an element x in the domain of valid plaintexts such that $D_{sk}(E_{pk}(x)) = x$

Proving that multiplication is correct (POMC): Assume prover P_i is given an encryption $E_{pk}(x)$ and chooses a constant c and calculates a random encryption

$E_{pk}(x \times c)$ and sends $E_{pk}(x \times c)$, $E_{pk}(c)$ to verifier. Then, P_i can give a zero-knowledge proof $POMC(E_{pk}(x), E_{pk}(c), E_{pk}(x \times c))$ to prove that $E_{pk}(x \times c)$ is indeed the product of the values contained in $E_{pk}(x)$ and $E_{pk}(c)$

Threshold decryption: Given the common public key pk, and an encryption $E_{pk}(x)$. There exists an efficient secure protocol in which each party uses their share of the private key sk to output x for everyone.

2.1.4 Secret Sharing

The goal of a (t, n) secret sharing scheme is to divide a secret S in to n shares $S_1, S_2, \ldots, S_n$ such that given any t shares, it is possible to reconstruct the secret S and knowledge of $k-1$ or few shares does not reveal any information about the secret S.

Shamir secret sharing [14] is an example (t, n) secret sharing scheme based on polynomial interpolation. In Shamir's secret scheme, the domain of secret and the shares are elements of a finite field F_p, where p is a prime and $p > n$. To share a secret $S \in F_p$, the dealer first chooses $t-1$ elements $a_1, \ldots, a_{t-1}$ uniformly at random from F_p. Then, builds a polynomial over the field F_p as follows $f(x) = a_0 + \sum_{i=1}^{t-1} a_i x^i$, where $a_0 = S$. To share a secret among n parties, the dealer constructs n points on the polynomial. For example, let $z = 1, \ldots, n$ and evaluates the polynomial f at each z to get a point $(z, f(z))$, which is given to party P_z. Note that, we can recover the $t-1$ degree polynomial (along with the secret S) with t or more unique points on the polynomial using Lagrange interpolation, but no information about the polynomial or the secret is leaked with less than t points.

2.2 Differential Privacy

Differential privacy introduced in [15, 16] provides a strong guarantee of privacy against an adversary with background knowledge, while learning some statistic over

a statistical database. It protects individual privacy by guaranteeing that the output of a mechanism is approximately the same (or more precisely, from nearly indistinguishable distributions), regardless if any single individual is present or absent in a dataset. Since, any information that can be learned with having an individual's data on the dataset can also be learned without it, there is no significant advantage for an individual to opt-out of the dataset.

We will review differential privacy and then discuss the techniques used to achieve differential privacy. Let D denote a sensitive database (collection of data elements) with each tuple corresponding to an individual. Let $\mathcal{M} : D \to \mathbb{R}^d$ be a randomized algorithm. Then, $\mathcal{M}$ satisfies ϵ-Differential Privacy if and only if for any two neighboring datasets D_1 and D_2, the distributions $\mathcal{M}(D_1)$ and $\mathcal{M}(D_2)$ differ at most by a multiplicative factor of e^ϵ. A formally definition of differential privacy is as follows.

Definition 2.2.1 (ϵ-Differential Privacy [15, 16]) *A randomized mechanism $\mathcal{M}$ is ϵ-differentially private if for all datasets D_1 and D_2 differing by at most one element, and for all $S \subseteq Range(\mathcal{M})$, the following holds*

$$\frac{P(\mathcal{M}(D_1) \in S)}{P(\mathcal{M}(D_2) \in S)} \leq e^\epsilon$$

The key idea behind differential privacy is that the contribution of a single individual to the publicly released result is small relative to the noise. This is done by calibrating the noise based on the potential difference in results between *any* two neighboring databases (databases that differ by one individual.). The difference between the results from the true world D and its neighbor D' is the difference the privatization noise will need to obfuscate in order for the privatized results to not give evidence about whether D or D' is the true world. The upper bound of this difference over $D_I \in \mathcal{D}$ is the *sensitivity* of query f. For example, if we assume a binary dot product (the count of individuals for whom both parties have value 1), the sensitivity is 1. Removing/Adding an individual (modifying a value from 1 to 0 or vice versa) will change the outcome by at most one, *regardless of the initial vectors.*

In general, the sensitivity of dot product is the multiple of the maximum possible values in the domain.

As with secure multi-party computation, differential privacy has a seminal result giving a method for any query. The technique proposed by [16] to achieve ϵ-differential privacy is by adding a suitable noise generated from the Laplace distribution to the output.

2.2.1 Sensitivity

One of the key parameters that determines the accuracy with which a query f can answered with differential privacy is the ℓ_1 sensitivity of f. It captures the largest change in f due to a change in single individual's data item. Now, we define the two sensitivities that have been used to achieve differential privacy.

Definition 2.2.2 (Global Sensitivity [16]) *For a given function $f : \mathcal{D} \to \mathbb{R}^d$, the global sensitivity of f (with respect to the ℓ_1 metric) is*

$$GS_f = \max_{D_1, D_2} \Big|\Big| f(D_1) - f(D_2) \Big|\Big|_1$$

where D_1 and D_2 differ in at most one element.

2.2.2 Laplace Mechanism

Let $Lap(\mu, \lambda)$ be a Laplace distribution with mean μ and scale factor $\lambda (> 0)$, whose density function is given by

$$h(x) = \frac{1}{2\lambda} \exp\left(-\frac{|x - \mu|}{\lambda}\right)$$

In [16] Dwork et al. proved that for a given query function f and a database D, a randomized mechanism $\mathcal{M}$, which returns $f(D) + Y$, where Y is drawn i.i.d from $Lap\left(\frac{GS_f}{\epsilon}\right)$ satisfies ϵ-differential privacy.

2.2.3 Exponential Mechanism

The Laplace mechanism achieves differential privacy by adding a real-valued noise to the true answer. However, it is not suitable for queries that return non-numeric values, or in situations where noise is irrelevant. Exponential mechanism $\mathcal{E}$, proposed in [17] is applicable for non-numeric queries. Let $\mathcal{D}$ be the domain of input datasets, $\mathcal{R}$ be the range of noisy outputs and $\mathbb{R}$ be the real numbers. The exponential mechanism $\mathcal{E}$ defines a scoring function $q : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ that assigns a score to each pair (D, r) where $D \in \mathcal{D}$ and $r \in \mathcal{R}$. Given a database D and privacy parameter ϵ, $\mathcal{E}$ outputs r with probability proportional to $e^{\frac{\epsilon \times q(D,r)}{2S(q)}}$.

Theorem 2.2.1 *[17] Let $q : (\mathcal{D}^n \times \mathcal{R}) \to \mathbb{R}$ be a scoring function that, given a database $D \in \mathcal{D}^n$, assigns a score to each outcome $r \in \mathcal{R}$. The sensitivity of the scoring function q is $S(q) = \max_{r, A\Delta B=1} |q(A, r) - q(B, r)|$. Let $\mathcal{E}$ be a mechanism for choosing an outcome $r \in \mathcal{R}^n$ given a database instance $D \in \mathcal{D}^n$ then the mechanism*

$$\mathcal{E}(D, q) = \left\{ \textit{return } r \textit{ with probability } \propto \exp \frac{\epsilon q(D, r)}{2S(q)} \right\}$$

satisfies ϵ-differential privacy.

In the global sensitivity framework, the noise magnitude depends upon the function f and the privacy parameter ϵ. The global sensitivity measures the noise needed to protect the privacy of an individual in the worst case scenario. But, it may not be suitable for all functions because the noise magnitude may be very high and this could lead to poor utility for highly sensitive functions. In the local sensitivity framework [18], the noise will also be based upon the database D. The local sensitivity $LS_f(D)$ of the function f with database D is defined as follows

Definition 2.2.3 (Local Sensitivity [18]) *For a given function $f : \mathcal{D} \to \mathbb{R}^d$ and $D \in \mathcal{D}$, the local sensitivity of f at D (with respect to the ℓ_1 metric) is*

$$LS_f(D) = \max_{D_1} \quad f(D) - f(D_1) \quad _1$$

where D_1 differs from D by a single element.

It is also easy to see that $GS_f = \max_x LS_f(x)$. However, releasing a function calibrated with noise magnitude proportional to $LS_f(x)$ will not satisfy differential privacy because the local sensitivity is itself sensitive. Therefore, a smooth upper bound S_f to LS_f is used such that adding noise proportional to S_f is safe.

Definition 2.2.4 (A Smooth Bound [18]) *For $\beta > 0$, a function $S : \mathcal{D}^n \to \mathbb{R}^+$ is a β-smooth upper bound on the local sensitivity of f if it satisfies the following requirements:*

$$\forall x \in \mathcal{D}^n : S(x) \geq LS_f(x)$$

$$\forall x, y \in \mathcal{D}^n, d(x, y) = 1 : S(x) \leq e^{\beta} S(y)$$

An example of a function that satisfies Definition 2.2.4 is the smooth sensitivity of f.

Definition 2.2.5 (Smooth sensitivity [18]) *For $\beta > 0$, the β-smooth sensitivity of f is*

$$S^*_{f,\beta}(x) = \max_{y \in \mathcal{D}^n} e^{-\beta d(x,y)} LS_f(y)$$

Definition 2.2.1 also called information theoretic privacy is the strongest definition of differential privacy as it holds against unbounded adversaries. A relaxed indistinguishability-based computational differential privacy (*IND-CPD*) definition proposed in [19] protects privacy against computationally bounded adversaries. IND-CDP is the computational analogue of (ϵ, δ)-DP where $\delta = negl(\kappa)$, where k is the security parameter. In CDP, the adversary is modeled as polynomial sized circuits (or non uniform probabilistic polynomial time TMs) and is denoted by $\{A_\kappa\}_{\kappa \in \mathbb{N}}$.

Definition 2.2.6 (IND-CDP [19]) *An ensemble* $\{f_\kappa\}_{\kappa\in\mathbb{N}}$ *of randomized functions* $f_\kappa : \mathcal{D} \to \mathbb{R}^\kappa$ *provides* ϵ_κ*-IND-CDP if there exists a negligible function* $negl(.)$ *such that for every non-uniform PPT TM* A*, every polynomial* $p(.)$*, every sufficiently large* $\kappa \in \mathbb{N}$*, all data sets* $D, D' \in \mathcal{D}$ *of size at most* $p(\kappa)$ *such that* $|D\Delta D'| \leq 1$*, and every advice string* z_κ *of size at most* $p(\kappa)$*, it holds that*

$$Pr[A_\kappa(f_\kappa(D)) = 1] \leq e^\epsilon \times Pr[A_\kappa(F_\kappa(D')) = 1] + negl(\kappa)$$

where we write $A_\kappa(x)$ *for* $A(1^\kappa, z_\kappa, x)$ *and the probability is taken over the randomness of mechanism* f_κ *and adversary* A_κ*.*

2.3 Two-Party Computational Differential Privacy

We briefly review the two-party CDP definition in the malicious setting proposed in [19]. Let $\{g_\kappa\}_{\kappa\in\mathbb{N}}, \{h_\kappa\}_{\kappa\in\mathbb{N}}$ denote the ensembles of randomized interactive Turing machines of g_κ, h_κ respectively and $\{\langle g_\kappa, h_\kappa\rangle\}_{\kappa\in\mathbb{N}}$ denote the ensemble of interactive protocols of $\{g_\kappa\}_{\kappa\in\mathbb{N}}, \{h_\kappa\}_{\kappa\in\mathbb{N}}$. Then the definition of two-party differentially private computation using the ideal/real paradigm is

Definition 2.3.1 (Two-Party CDP [19]) *A two-party interactive protocol ensemble* $\{\langle g_\kappa, h_\kappa\rangle\}_{\kappa\in\mathbb{N}}$ *for computing a function* $f(x, y)$ *satisfies* (γ, ξ) ϵ_κ*-SIM*$^+-$*CDP if there exists an* ϵ_κ*-DP randomized mechanism* $\hat{f} = (\hat{f}_g, \hat{f}_h)$ *such that*

- *Mechanism* $\hat{f}$ *provides* (γ, ξ) *additive usefulness with respect to* f*.*
- *The protocol ensemble* $\{\langle g_\kappa, h_\kappa\rangle\}_{\kappa\in\mathbb{N}}$ *securely realizes the randomized functionality* $\hat{f}$ *as per the ideal/real simulation paradigm.*

Informally, the above definition states that for every A_κ in the real world, there exists a simulator S_κ in ideal world when given a differentially private output $\hat{f}$ (computed by trusted third party), S_κ should be able to simulate the protocol with A_κ such that for every $x, y \in \{0, 1\}^\kappa$ the joint output in the ideal world is computationally indistinguishable with the joint output in the real world.

The usefulness property is used to describe the utility/correctness of a differentially private mechanism.

Definition 2.3.2 ((γ, ξ)-usefulness [19]) *A differentially private output $\hat{f}(x, y)$ is an additive (γ, ξ)-useful for a deterministic function $f(x, y)$ if for all $x, y \in \mathcal{D}$*

$$Pr[|\hat{f}(x, y) - f(x, y)| > \gamma(\kappa)] \leq \xi(\kappa)$$

3 PRIVACY-PRESERVING ANALYSIS IN RATIONAL SETTING

Secure multi-party computation (MPC) and differential privacy are two notions of privacy that deal respectively with how and what functions can be privately computed. Computing a differentially private function using MPC techniques was first considered in [20]. The idea is to design $\hat{f}$, an ϵ-differentially private approximation of the function f, and evaluate it using MPC.

As an example application, suppose two companies wish to compare customer lists. If they share enough customers, they may wish to establish a collaboration. Using secure function evaluation they can compute the distance without revealing their inputs. Suppose one company simply wishes to know if the other has a particular customer, it can construct a document containing only that name. The output/value of the distance protocol reveals the presence of that individual in the other party's list. Differential privacy protects against this, adding sufficient noise to the outcome to only give highly uncertain information about any individual, while still providing reasonably accurate aggregate information.

3.1 Related Work

The closest work related to ours is [21]. It gives a distributed protocol to generate a Laplace sample from two exponential samples that involves computing secure logarithm twice. We show how the composition method can be used to generate a Laplace sample from a single uniform sample. While [21] does give a malicious secure protocol, the malicious model security holds only when the number of parties is greater than two.

Distributed pseudo-random number generation has been used in privacy-preserving aggregation of time series data. It considers the problem of finding statistics from

a user's data in the presence of an untrusted aggregator. A distributed protocol for Laplace sample generation from multiple Gaussian samples (each party generates a single Gaussian sample) is given in [22]. In [23], a method is given for distributed sample generation from a geometric distribution. Our work faces a different challenge, as rather than an untrusted aggregator, it is the participating parties that may not be trusted.

Some of the papers that deal with distributed computational differential privacy (CDP) are described below. The problem of an adversary gaining exclusive access to output without getting caught in a two-party CDP malicious setting was discussed in [24], but the paper does not provide solutions. [19] gives fundamental definitions for two-party computational differential privacy and various relationships among them. They also provide a two-party Hamming distance protocol for the honest but curious (semi-honest) model and the malicious model, but they do not deal with verifiable sample generation. In [19], at the end of the protocol one party gets a differentially private output and the other party gets nothing. The party not receiving the output generates the noise; the assumption is that a malicious party generating large noise is essentially equivalent to a malicious party aborting. None of the above protocols have a mechanism for verifiable noise generation. Even though one might argue that the individual privacy is not compromised, a malicious adversary who would like to have exclusive access to the output could add more noise than what is needed to render the honest party's output unusable.

The notion of allowing an adversary to cheat with non-negligible probability as long as it caught with some high probability ω was formalized in [25], which introduces secure multi-party computation in the presence of covert adversaries, a slightly weakened view that allows malicious behavior to benefit the malicious party as long as it is eventually detected. In our work, a rational adversary does not learn the honest party's input; the only consequence of successful cheating by an adversary is a low quality output for the honest party.

Encrypting Reals: A p precision real value is converted into an integer by first multiplying it with a constant 10^p before encryption. To recover the real value after decryption, the integer is multiplied with the scaling factor $\frac{1}{10^p}$. Since N $-$ 1 $\equiv$ -1(mod N), we can represent $-i$ by $N - i$ in $\mathbb{Z}_N$. The lower half $[1, \lfloor \frac{N}{2} \rfloor]$ and upper half $[\lceil \frac{N}{2} \rceil, N - 1]$ of the range $[1, N - 1]$ is used to represent positive and negative numbers respectively.

3.2 Two-Party CDP: Semi-Honest Model

We illustrate the need for a secure protocol in the malicious setting using a Hamming distance protocol, although the approach can be extended to any scalar-valued function. A simple semi-honest solution is for each party to generate noise satisfying differential privacy, and incorporate it in the result. An example two-party differentially private Hamming distance protocol secure in the semi-honest setting using a semantically secure additive homomorphic encryption scheme is given in Algorithm 1. Some of the notations used in this chapter are given in Table 3.1.

Table 3.1.: Notations

P_i	Party i
pk	Public Key
sk_i	Private key share of P_i
x_i	P_i's input vector x
x_{ij}	j^{th} element of x_i
$\tilde{x}$	Encryption of x with public key pk ($E_{pk}[x]$)
$\oplus$	Xor
$\boxplus$	Homomorphic addition
$\boxtimes$	Multiplication of constant with encrypted value

This gives a "doubly noisy" result for P_1, but since P_1 knows the noise it contributed, it can factor it's own noise out to obtain a less noisy, but still differentially private, output. P_1 subtracts r_1 from f_1 to obtain its output. We can see that each

Algorithm 1 Two-party secure Hamming distance in HbC

Input: Party i's input vector is x_i.
Output: $f_i = (\quad \overset{n}{j} x_{0j} \oplus x_{1j}) + r_{1-i}$

1: Party P_0 creates a key pair (pk, sk) and sends its encrypted input vector $\tilde{x}_0$ and public key pk to party P_1.
2: $\forall\ i$, P_1 computes $\tilde{t_i} = \tilde{x}_{0i}$, if $x_{1i} = 0$ and $\tilde{t_i} = \tilde{1} \boxplus (-1 \quad \tilde{x}_{0i})$ if $x_{1i} = 1$.
3: P_1 computes the Hamming distance by homomorphically summing $\tilde{t_i}$ to get $\tilde{s} = \boxplus_{i=1}^{n} \tilde{t_i}$.
4: P_1 can homomorphically add a suitable noise $r_1 \sim Lap(0, \frac{1}{\epsilon})$ to $\tilde{s}$ to obtain $\tilde{f_0} = \tilde{s} \boxplus \tilde{r}_1$ and send the differentially private value to P_0.
5: P_0 decrypts the value $f_0 = D_{pk}(\tilde{f_0})$ and sends $f_1 = f_0 + r_0$ to P_1 where r_0 is noise selected by P_0.

party learns $f_i = (\quad \overset{n}{j} x_{0j} \oplus x_{1j}) + r_{1-i}$, where r_{1-i} is randomly selected by P_{1-i}, so each party is left with a result containing sufficient (unknown) noise to provide differential privacy. A brief argument that the protocol provides ϵ_κ-SIM-DP for P_0 and ϵ_κ-DP for P_1 in the semi-honest setting is given in [26].

The above protocol works fine as long as the parties do not deviate from the protocol, but fails if a party deviates from the protocol. There exist standard techniques, like zero knowledge proofs as shown in [13] for Paillier encryption, to prove the veracity of the statement at each step. But, the fundamental problem still persists because a party who wishes to have exclusive access to the result can add a predetermined large noise to make the output unusable for the honest party. The problem in the above protocol is that each party's output contains a noise sample randomly selected by the other party. In Section 3.3, we show how two parties can engage in a protocol to draw a sample from the required distribution, preventing this problem.

This chapter makes the following contributions

- A two-party protocol is given in Section 3.3 to generate a pseudo-random sample from Laplace distribution in the presence of a malicious adversaries. As long as one of the parties follow the protocol, the sample generated is a pseudo-

random sample from a Laplace distribution. Unfortunately, this protocol is computationally quite expensive.

- We then introduce the notion of rational adversaries, which models the behavior of an adversary in two-party CDP. Rational adversaries in two-party CDP have the property that they cheat with the intention of getting exclusive access to the output without being caught. Section 3.4 presents a definition of two-party CDP in the rational setting with relaxed utility guarantees to develop more practical protocols. We do this by defining a deterrence factor $1 - \omega$ where $0 \leq \omega \leq 1$. I.e., any attempt to gain exclusive access to the output by an adversary in the execution of the protocol is caught with probability at least $1 - \omega$. If ω is equal to 0, then the model is equivalent to two-party CDP in the malicious model.

3.3 Two-Party CDP: Malicious Case

We show a generic method to compute two-party differentially private analysis (using a Laplace mechanism to achieve DP) using garbled circuits, if there exists one in the ideal environment. In a semi-honest model, the parties are assumed to follow the protocol, which implies that the parties send their true inputs during an execution. Given a protocol that is secure in the semi-honest setting, we can apply zero knowledge proofs at each step to make it secure in the malicious model. However, this does not impose any restriction on the choice of inputs. An adversary sending incorrect inputs during an execution will go undetected. Hence, the idea of having one party generate random noise that impacts the output of the other party does not work; a party desiring exclusive access to the result can generate arbitrarily large "noise" to corrupt the other party's output; as this is a legitimate input, it is allowed even in a malicious-secure protocol. In an ideal model, the pseudo-random sample is generated by an incorruptible trusted third party. The key step in emulating the

ideal model is to generate a random sample from the required distribution even if only one party behaves correctly.

This is easy if we desire a sample from a uniform distribution; the modulo sum of the numbers generated by each party is a random sample as long as one party behaves honestly. But we need a sample from a Laplace distribution; this can be done using the composition method. Algorithm 2 gives the steps to generate a Laplace sample with a specified mean 0 and scale parameter λ. The protocol given in [27] can be used to securely compute an approximation of $c[\ln(x)]$, where c is a publicly known multiplicative factor. Algorithm 2 outputs $c \times \ell$ where $\ell \sim Lap(0, \lambda)$. Since c is public, each party can remove it from the differentially private result.

Algorithm 2 Two-party Laplace noise generation protocol

Input: Each party P_i has two random inputs, $X_i \in \{0, 1\}$ and Y_i, λ are p precision numbers.
Output: $c * l$, where $l \sim Lap(0, \lambda)$ and c is a publicly known multiplicative factor.

1: $U_1 = X_1 \oplus X_2$ (compute a random bit).
2: $U_2 = Y_1 + Y_2 \bmod (10^p + 1)$
3: $Z = \Pi_{log}(U_2) - c[\ln(10^p)] = c[\ln(U * 10^p)] - c[\ln(10^p)) = c[\ln(U)]$, where $U \sim (0, 1)$.
4: If $U_1 == 0$, then $Z' = Z$. Else, $Z' = -Z$.
5: return $\lambda * Z'$

The composition method is a generic method that can be used when the target Cumulative Distribution Function (CDF) can be expressed as the convex sum of other CDFs.

$$F(x) = \sum_{j=1}^{\infty} p_j F_j(x)$$

where $p_j > 0$ and $\sum_{j=1}^{\infty} p_j = 1$

Laplace Distribution: A standard Laplace distribution is a symmetric exponential distribution with pdf and cdf as

$$f(x) = \begin{cases} \frac{1}{2}e^{x} & \text{if } x < 0 \\ \frac{1}{2}e^{-x} & \text{if } x \geq 0 \end{cases} \text{ and } F(x) = \begin{cases} \frac{1}{2}e^{x} & \text{if } x < 0 \\ 1 - \frac{1}{2}e^{-x} & \text{if } x \geq 0 \end{cases}$$

$$F_1(x) = \begin{cases} e^{x} & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \text{ and } F_2(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{-x} & \text{if } x \geq 0 \end{cases}$$

Then,

$$F(x) = \frac{1}{2}F_1(x) + \frac{1}{2}F_2(x)$$

Computing the inverse we get

$$F^{-1}(u) = \begin{cases} \log(u) & \text{with prob. } 0.5 \\ -\log(u) & \text{with prob. } 0.5 \end{cases}$$

3.3.1 Distributed Uniform Pseudo-Random Number Generation

A fixed precision uniform random sample from the interval [0,1] is generated by each party. A p precision floating point sample can be scaled to the interval [0,10^p] by multiplying it with 10^p. The sum of the individual samples modulo 10^p gives a uniformly random sample from the interval [0,10^p]. The scaling factor used here is $\frac{1}{10^p}$. Proposition 3.3.1 states this formally.

Proposition 3.3.1 *Let U_1, U_2 be integers in the interval $[0, 10^p]$. Then, $U = U_1 + U_2 (\mathrm{mod} 10^p + 1)$ is a uniform sample from $\mathcal{U}(0, 10^p)$ if at least one of the sample $U_i \sim \mathcal{U}(0, 10^p)$. If $U_1 \sim \mathcal{U}(0, 10^p)$, then*

$$Pr[U = u] = Pr[U = U_1 + U_2 (\mathrm{mod} 10^p + 1)]$$
$$= Pr[U_1 = U + (10^p - U_2) (\mathrm{mod} 10^p + 1)] = \frac{1}{10^p + 1}$$

3.3.2 Security

The security of the sample generation protocol against malicious adversaries holds due to the generic techniques available for converting a semi-honest Yao's garbled circuit to be secure in the malicious model. There are two issues to consider when considering malicious parties.

1. The circuit evaluator could deviate from the OT protocol and obtain keys for both 0 and 1, thereby evaluating the function for different inputs.

2. Correctness of the protocol (the circuit generator could construct a different circuit), which can in turn leak the input of the evaluator.

Security against a malicious evaluator can be prevented by using an oblivious transfer secure against malicious adversaries [28]. Informally, the circuit evaluator can only obtain keys corresponding to its own input and can only evaluate the function on its input. In order to protect against a malicious circuit generator, techniques like cut-and-choose have been widely used in which the generator constructs multiple circuits and sends them to the evaluator. The evaluator then randomly ask the generator to open half of the circuits to check the validity of the construction. The evaluator finally evaluates the remaining circuits and uses the majority output as the true output.

3.3.3 Efficiency

The expensive operation in distributed Laplace sample generation is the secure logarithm function. We used the secure logarithm proposed in [27], which approximates the logarithm function by the Taylor series to q places. The latest work on cut-and-choose for garbled circuits [29] shows that to achieve a negligible cheating probability of 2^{-s} requires constructing s circuits. Hence, the cut-and-choose technique to get a cheating probability of approximately 1%, or 2^{-7}, would thus at least be 7 times the cost of the semi-honest circuit. To give an idea of efficiency of the method, we implemented a semi-honest version of the differentially private hamming

distance using FairplayMP [30]. We used $q = 4$ for our experimentation (i.e., the first four terms of the Taylor series was used to approximate log). It took around 25 minutes on a 2.4GHz processor with 8GB of RAM to evaluate the circuit in a semi-honest setting. A number of improvements in building efficient garbled circuits have proposed in [31,32], but they are not practical at the moment against malicious parties. This gives a protocol running time of several hours – feasible for some uses, but in many cases impractical.

3.4 Two-Party CDP with Rational Adversaries

Secure multi-party computation in the semi-honest model offers no guarantees on the quality of output for honest parties in the presence of dishonest parties. Although MPC in the malicious model offers strict guarantees on output, it does not easily produce efficient protocols for practical implementation and data analysis. We now give a middle ground by relaxing the utility guarantee of the malicious model, which leads to MPC in the presence of rational adversaries. This is done by introducing a parameter ω that captures the probability of undetected cheating by an adversary in the rational setting. A more formal definition follows.

3.4.1 Rational Adversaries

We define rational adversaries in MPC as parties who wish to gain exclusive access to the correct output without getting caught. This is slightly different from the fairness property requirement in MPC because an unfair party is always caught at the end of the protocol. The scenario happens in differentially private data analysis, where a randomized input of the parties directly contributes to the output of the function. A rational adversary could generate arbitrarily large noise, distorting the outcome for the other party, and argue that the large noise was generated as a random sample. Hence, we introduce a deterrence factor $1 - \omega$ such that $0 \leq \omega \leq 1$, which denotes the probability with which an honest party can detect cheating, if a rational

adversary attempts to do so. For $\omega = 0$, any attempt to cheat by a rational adversary is always caught, equivalent to the malicious model.

Note that the protocol must still *allow* arbitrarily large noise, in order to satisfy differential privacy. Thus detecting a high noise level does not imply cheating. The key is that high noise levels must be an unlikely event, as opposed to an event a dishonest party could cause on a regular basis.

3.4.2 Definition: Ideal/Real Style

We define two-party computational differential privacy in the rational model using a redefined ideal/real style paradigm to capture the probability of an adversary gaining exclusive access to the output. Let P_1, P_2 be the parties, A be an adversary controlling $j \in \{1, 2\}$ and $\mathcal{T}$ be the incorruptible trusted third party. Then, an execution in the modified ideal model with parameter ω proceeds as follows.

Inputs: Each party P_i obtains its input x_i of length n; let z be the auxiliary input of the adversary $\mathcal{A}$.

Send inputs to trusted party: An honest party P_i always sends its received input x_i to $\mathcal{T}$. An adversary $\mathcal{A}$ controlling the party P_j may send its received input x_j or send some other input of length n or $abort_j$ (may depend upon the auxiliary input z) to $\mathcal{T}$ on behalf of P_j. Let $\bar{x}$ be the received input of both the parties.

$\mathcal{T}$ sends output to adversary: $\mathcal{T}$ computes $f(x, y) = (f_1(\bar{x}), f_2(\bar{x}))$ and sends f_j to $\mathcal{A}$ controlling P_j.

Adversary instructs $\mathcal{T}$ to continue or halt: The adversary $\mathcal{A}$ upon receiving the outputs could either send continue or abort to $\mathcal{T}$.

Cheat Option: If $\mathcal{A}$ controlling the corrupted party P_j sends $w_j = \text{cheat}_j$ to $\mathcal{T}$, then:

1. With probability $1-\omega$, the $\mathcal{T}$ sends corrupted_j to the adversary and the honest party.

2. With probability ω, the $\mathcal{T}$ sends undetected to the adversary and further asks the adversary for the output $f_i(\bar{x})$ that needs to be sent to the honest party.

$\mathcal{T}$ sends output to Honest Party: $\mathcal{T}$ sends $f_i(\bar{x})$ to the honest party P_i.

Outputs: An honest party always outputs whatever it has received from $\mathcal{T}$. The corrupted party outputs nothing. $\mathcal{A}$ can output anything (efficiently computable) from its input x_j , the advice string z and messages it received from $\mathcal{T}$.

The output of the honest parties and adversary in the above ideal model execution is defined as $\text{IDEALR}^{\omega}_{f,\mathcal{A}(z)}(\bar{x})$. There are two types of unfairness in the model. One is the abort call that is present in the standard ideal model in which the honest party receives $\perp$ as output. The second is when the adversary can with certain probability ($< \omega$) cause the honest party to obtain an inaccurate result. In our case, this is noisy/inaccurate result with significantly higher probability than would be expected from selecting a value from a Laplace distribution.

Security as emulation of real execution in the ideal model

Protocol Π is said to securely compute f (in the rational model with $1-\omega$ deterrent) if for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a nonuniform probabilistic polynomial-time adversary B for the ideal model, such that

$$\left\{\text{IDEALR}^{\omega}_{f,\mathcal{B}}(\bar{x},n) \stackrel{\text{c}}{\equiv} \text{REAL}_{\Pi,\mathcal{A}}(\bar{x},n)\right\}$$

3.4.3 Differentially Private Function

If two parties P_1, P_2 want to securely compute a differentially private function $\hat{f}(x,y)$ on their private inputs x, y respectively, then in an ideal environment, they would send their inputs to $\mathcal{T}$. $\mathcal{T}$ then computes $f(x,y)$ and adds to it a random noise sample (e.g., selected from Laplace distribution with the appropriate scale parameter) and sends the approximated output to the parties. The ideal environment provides ϵ-DP to both the parties. We say a real protocol is secure when it emulates the

ideal world. Since the real world is only guaranteeing computational differential privacy, the security is maintained even when the simulator is not efficient as pointed out in [19]. Another way of looking at this is for any adversary in the real model A, if there exists an adversary S in the ideal model, then the protocol in the real model securely realizes the ideal functionality. In this case, the ideal model provides information theoretic differential privacy, hence even an inefficient S should not be able to simulate the attack in the ideal model. For interactive protocols, this leads to the relaxed definition of ϵ^{κ}-SIM$^{+}-$CDP.

Definition 3.4.1 ($(\gamma, \xi, \omega)\epsilon_{\kappa}$-SIM$^{+}-$CDP) *An ensemble of interactive protocols* $\{\langle g_{\kappa}, h_{\kappa}\rangle\}_{k\in\mathbb{N}}$ *is a* (γ, ξ, ω) ϵ_{κ}*-SIM*$^{+}-$*CDP two-party computation protocol for* $f = (f_g, f_h)$ *in the presence of rational adversaries with* $(1-\omega)$*-deterrence if there exists an* ϵ_{κ}*-DP mechanism* $\hat{f}$ *such that*

- $\hat{f}$ *provides* (γ, ξ) *additive usefulness with respect to* f*.*
- *The protocol ensemble* $\{\langle g_{\kappa}, h_{\kappa}\rangle\}_{k\in\mathbb{N}}$ *securely realizes* $\hat{f}$ *as per the modified ideal/real style definition with parameter* ω

(γ, ξ, ω) ϵ_{κ}-SIM$^{+}-$CDP is very similar to the definition of (γ, ξ) ϵ_{κ}-SIM$^{+}-$CDP except that protocol needs to realize $\hat{f}$ with respect to the relaxed ideal/real paradigm that guarantees correctness/usefulness of output for the honest party with probability $1-\omega$.

3.4.4 Hamming Distance Protocol

In this section, we show how to build an efficient protocol for finding Hamming distance using a (semantically secure) threshold Paillier encryption between two-parties with $\omega = \frac{1}{m} + \beta$, where m, β are parameters in the noise selection protocol.

The two-party computationally differentially private hamming distance protocol (Algorithm 3) works as follows. Initially, each party P_i encrypts its input vector x_i and sends it along with its proof of knowledge of plain text (POK) to the other party

Algorithm 3 Secure Hamming distance protocol

Input: Two Parties holds their share sk_0 and sk_1 of the private key and a common public key pk. Party i's input vector is x_i. Output: $\sum_{i}^{n} x_{0i} \oplus x_{1i} + r'$

1: **for** i **do**
2: $\forall j\ \tilde{x}_{ij} = E_{pk}(x_{ij})$ and create $POK(\tilde{x}_{ij})$
3: Send encryptions $\tilde{x}_{ij}$ and $POK(\tilde{x}_{ij})$ to P_{1-i}
4: **end for**
5: **for** i **do**
6: $\forall j$ check whether $POK(\tilde{x}_{(1-i)j})$ is correct, Else ABORT
7: **end for**
8: Run Noise Selection protocol to select $\tilde{r}_0, \tilde{r}_1$
9: **for** P_1 **do**
10: $\forall j$ calculate $\tilde{z}_{1j} = E_{pk}(x_{0j} \oplus x_{1j})$ using $\tilde{x}_{0j} \boxplus \tilde{x}_{1j} \boxplus (-2x_{1j} \quad \tilde{x}_{0j})$
11: $\tilde{s} = \tilde{z}_{11} \boxplus \tilde{z}_{12} \ldots \tilde{z}_{1n} \boxplus \tilde{r}_0 \boxplus \tilde{r}_1 = E_{pk} \quad \sum_{j}^{n}(x_{0j} \oplus x_{1j} + \tilde{r}_0 + \tilde{r}_1)$
12: Send $\tilde{s}$, $\forall j\ POMC(\tilde{x}_{0j}, \tilde{x}_{1j}, E_{pk}(-2x_{0j}x_{1j}))$
13: **end for**
14: **for** P_0 **do**
15: Check $\forall j$ if $POMC(\tilde{x}_{0j}, \tilde{x}_{1j}, E_{pk}(-2x_{0j}x_{1j}))$ is correct, Else ABORT
16: Calculate $\tilde{z}_{11} \boxplus \tilde{z}_{12} \ldots \tilde{z}_{1n} \boxplus \tilde{r}_0 \boxplus \tilde{r}_1$ and verify if it matches with $\tilde{s}$
17: **end for**
18: Jointly decrypt $\tilde{s}$.
19: P_i gets the ϵ-DP hamming distance by subtracting r_i from s

P_{1-i}. Since the secret key is split between the two parties, it is not possible for P_{1-i} to decrypt the encrypted values. P_{1-i} checks for consistency of $\tilde{x}_i$ using the zero knowledge proof. Then, the parties engage in the secure noise selection protocol to select the random noise sample $\tilde{r}_i$ from a carefully selected Laplace distribution. To compute the Hamming distance P_1 does the following. For each j, P_1 computes $\tilde{z}_{1j}$ by homomorphically adding the values of $\tilde{x}_{0j}$, $\tilde{x}_{1j}$ and $-2 \quad \tilde{z}_{1j}$. P_1 computes the Hamming distance by homomorphically summing $\tilde{z}_{1j}$. Adding the left-over sample from noise selection protocol $\tilde{r}_{ik}$ to the encrypted Hamming distance gives the differentially private value. In order to the confirm that P_1 does not deviate from the protocol, it sends $\tilde{s}$ and proof of multiplication by constant (POMC) for each $\tilde{z}_{1j}$. P_0 verifies if the multiplications were done correctly using POMC and checks if $\tilde{s}$ is correct by calculating the homomorphic additions of z_{1j} and $\tilde{r}_{ik}$. Finally, they jointly decrypt the value $\tilde{s}$. Since each party knows exactly one random noise added, they can subtract it

from the decrypted value to get the final answer (still containing the unknown noise, thus guaranteeing each party's result independently satisfies differential privacy.)[1]

3.4.5 Noise Selection

In the noise selection protocol, each party P_i generates a random set of samples m from the Laplace distribution and sends it to the other party P_{1-i}. P_{1-i} randomly selects $m-1$ values to be decrypted by P_i and runs a goodness-of-fit test to verify that they come from the appropriate distribution. The leftover encrypted $\tilde{r}_{ik}$ is used for perturbing the output of P_{1-i}. If a party tries to add more noise than needed by generating samples with more noise than would be expected of a Laplace distribution (to ensure a noisy sample is selected as the leftover), then it is caught with high probability.

Algorithm 4 Secure noise selection protocol

Two Parties holds their share sk_0 and sk_1 of the private key and a common public key pk and know the parameters μ and λ of the Laplace distribution.

1: **for** i **do**
2: P_i selects m random samples r_{ij} from the Laplace distribution and sends $\tilde{r}_{(i)j}$ and $POK(\tilde{r}_{ij})$ to P_{1-i}.
3: P_{1-i} verifies $POK(\tilde{r}_{ij})$. If any of them fail then ABORT.
4: P_{1-i} randomly selects $m-1$ samples sent by P_i to be decrypted. Let the left out sample be $\tilde{r}_{il}$.
5: P_{1-i} runs Anderson-Darling goodness of fit test on the decrypted samples to check if they are sampled from $Lap(\mu, \lambda)$. If the test fails then goto step 2.
6: **end for**
7: The left out sample $\tilde{r}_{(1-i)l}, \tilde{r}_{(i)l}$ obtained will be noise added for party P_i, P_{1-i} respectively.

[1]Note that we assume the parties do not share output, which would only give 2ϵ-differential privacy. If the parties chose to collude, they could simply share the original data to defeat the protections of any protocol.

Goodness of Fit Test

We use the Anderson-Darling test [33] to determine if the samples fit the required Laplace distribution. The Anderson-Darling test is defined as follows

H_0: The data follow a specified distribution, H_a: The data do not follow a specified distribution , α: significance value and the test statistic

$$A^2 = -m - S, \text{where}$$

$$S = \sum_{i=1}^{m} \frac{2i-1}{m} [\ln(F(Y_i)) + \ln(1 - F(Y_{m+1-i}))]$$

where F is CDF of the specified distribution.

Given a set of samples, the test statistic is calculated based on distribution assumed in null hypothesis(H_0). Based on the significance value, the critical value is also found. If the test statistic value A^2 is greater than the critical value then H_0 is rejected.

Each party P_i uses the goodness of fit test to determine if the set of the samples sent by P_{1-i} is indeed generated from a Laplace distribution. The null hypothesis is H_0 : the samples $r_1, r_2 \ldots r_m$ come from the Laplace distribution with parameters μ, λ. Two types of errors are associated with the above protocol. The first type of error is that an adversary may succeed in generating consistently noisier results than would be expected of differential privacy. This could happen if the adversary slips in a large fixed value hoping that it will not be selected for decryption, while picking the rest of the $m-1$ samples from the correct distribution. Or, the Goodness of fit could fail to detect that the random samples are not generated from a Laplace distribution.

$$\begin{aligned} \Pr(\text{Cheating}) &= \Pr(\text{Not Rejecting } r_i | r_i \nsim \text{Lap}(\mu, \lambda)) + \frac{1}{m} \\ &= \Pr(\text{Type II}) + \frac{1}{m} = \beta + \frac{1}{m} \end{aligned}$$

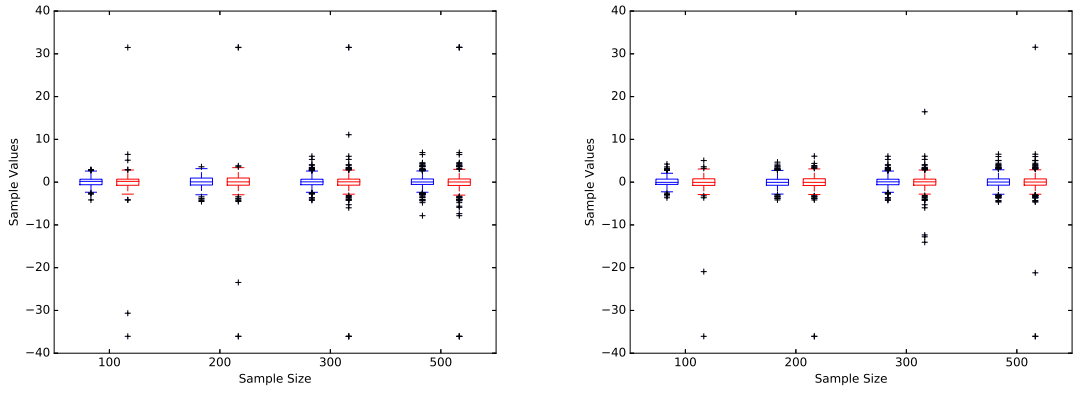
where β is related to the power of the test.

The second type of error is a false Negative: when a party incorrectly rejects the samples, when in fact the samples were generated from the correct distribution (but fail to satisfy the goodness of fit test). The probability of this occurring is:

$$\Pr(\text{ Rejecting } r_i | r_i \sim \text{Lap}(0, \lambda)) = \Pr(\text{Type I}) = \alpha$$

where α is the level of significance of the test of hypothesis.

One could argue that an adversary can send a worst predetermined sample that barely passes the goodness of fit tests during the noise selection protocol. One strategy would be for an adversary to draw a sample from the correct distribution and gradually increase the values until it fails the test. We box plotted the original sample values against the maliciously modified sample values in Figure 3.1 for different values of significance and sample sizes. We can see that as the sample size increases the modified sample is pretty close to the actual distribution; while such malicious behavior is possible, it has little impact on the utility of the result.



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.3$

Figure 3.1.: Honest draw (blue/left column) vs malicious draw (red/right column)

3.4.6 Complexity Analysis

We show the complexity of the protocol in terms of the number of modular exponentiations. In steps 1-3, each party P_i creates an encryption of its input and sends it to the other party P_{1-i}. It also verifies a constant round zero knowledge proof of knowledge with P_{1-i}. Hence, the total number of exponentiation is bounded by $O(n)$, where n is the size of the input vector. In secure noise selection, the number of exponentiations is bounded by $O(m)$, where m is the number of samples selected by each party. In steps 10-12, party P_1 does n homomorphic multiplications and n zero knowledge proof of correct multiplications with P_0. Hence, the total number of exponentiations done in the protocol is bounded by $O(n + m)$. Calculating the Anderson-darling test statistic requires $O(m \log m)$ steps as the samples need to be sorted. Hence, the total running time of the protocol is $O(n + m \log m)$.

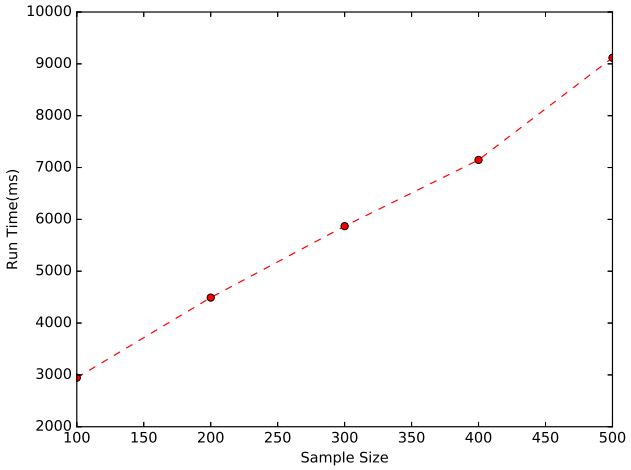


Figure 3.2.: Run time in ms

To estimate the practical time cost, we implemented the noise selection protocol using the Paillier encryption scheme in Java. Figure 3.2 shows the runtime of the protocol for various values of samples sizes (m=100 to 500). By contrast,the garbled circuit(GC) implementation of the protocol in the semi-honest setting took 28 minutes to generate a pseudo-random sample using FairPlay. Adapting the garbled

circuit to the malicious setting will only increase the computational cost of the protocol. Hence, building a CDP protocol in the presence of rational adversaries with small error probability is significantly faster.

3.4.7 Security

Theorem 3.4.1 *Assuming that the additively homomorphic threshold encryption scheme is semantically secure, and zero knowledge proofs specified are secure, protocol 5 is* $(\gamma, \xi, \omega)\epsilon_\kappa$*-SIM*$^+-$*CDP secure in the presence of rational adversaries.*

In order to show that the protocol is $(\gamma, \xi, \omega)\epsilon_\kappa$-SIM$^+-$CDP we need to show that security holds for both P_0 and P_1 (represented by the function ensembles $\{g_\kappa\}_{k\in\mathbb{N}}$, $\{h_\kappa\}_{k\in\mathbb{N}}$ respectively). We will consider the cases separately. We first show that the protocol ensures ϵ_κ-SIM$^+-$CDP for P_0 (i.e., when P_1 is rational). In order to prove that we need to show that for every adversary P_0^* (represented as a function ensemble $\{h_\kappa^*\}_{k\in\mathbb{N}}$) in the real model, there exists an adversary H_κ in the ideal model such that the views of $H_\kappa(x)$ and $h_\kappa^*(x)$ are indistinguishable. The simulator H_κ is given the black box of h_κ^*, works as follows.

1. Simulates h_κ^* to get the encrypted input $\tilde{x}_{1j}$ and $POK(\tilde{x}_{1j})$ for all j. H_κ acts as the verifier and h_κ^* as the prover. H_κ can extract the values of x_{1j} with overwhelming probability.
2. Sends x_1 as the input to $\mathcal{T}$ and obtains the result $f_h = \sum_j^n (x_{0j} \oplus x_{1j}) + r$.
3. H_κ on f_h comes up with x'_{0j} for $j = 1$ to n and r' such that $f_h = \sum_j^n (x'_{0j} \oplus x_{1j}) + r'$.
4. $\forall j$ H_κ encrypts x'_{0j} and sends $\tilde{x}'_{0j}, POK(x'_{0j})$ to h_κ^*.
5. H_κ then runs the noise selection protocol with h_κ^* to select r' as follows.
6. H_κ sends $m-1$ random samples from Laplace distribution and sends it along with r' to h_κ^*. It then makes h_κ^* to select r' as its random noise by rerunning

h^*_κ with the different random tape. The probability of h^*_κ not selecting r' in t iterations is $(\frac{m-1}{m})^t$, which goes to 0 as t increases. Hence, with multiple re-runs, the probability of picking r' in one of the iterations will be extremely close to 1.

7. H_κ receives a set of Laplace samples generated by h^*_κ. It reruns h^*_κ until it comes up with a noise r' that is consistent with the output it received from $\mathcal{T}$. Use goodness of fit test to determine if h^*_κ is trying to cheat in any of the runs. If yes, then sends cheat_2 to $\mathcal{T}$. If $\mathcal{T}$ returns undetected, then H_κ sends $\sum_j^n (x'_{0j} \oplus x_{1j}) + r''$ to $\mathcal{T}$ as the output for P_1. If $\mathcal{T}$ returns detected, then H_κ sends corrupted_2 to h^*_κ and outputs whatever h^*_κ outputs. This step is inefficient, but an inefficient simulator is sufficient for *Computational* Differential Privacy.

8. H_κ continues to run the protocol as the honest party g_κ by computing $\tilde{z}_{1j}$ for all j and homomorphically summing them along with r' to get a differentially private hamming distance value $\tilde{s}$. H_κ then sends $\tilde{s}$ and $POMC$ to h^*_κ.

9. H_κ outputs whatever h^*_κ outputs.

Now we show that the views of H_κ and h^*_κ are indistinguishable. In steps 1-3, H_κ behaves similar to g_κ except that instead of acting as verifier, it extracts the inputs using the knowledge extractor and hence the views of H_κ and h^*_κ are indistinguishable. In step 4, H_κ instead of sending $\tilde{x}_{0j}$ for all j, it sends $\tilde{x'}_{0j}$ that satisfies the constraints mentioned. Since, the underlying encryption scheme is semantically secure, the views are indistinguishable. In Steps 5-7, H_κ works similarly to g_κ except that it sends r' along with $m-1$ random samples along with its required zero knowledge proofs and reruns h^* until it generates r' as its sample. In step 8, H_κ runs exactly like the honest party g_κ computing $\tilde{z}_{1j}, \tilde{s}$ and its corresponding zero knowledge proofs and acts as a prover to h^*_κ. In the last step, H_κ jointly decrypts $\tilde{s}$ and outputs whatever h^*_κ, hence the views are identical. At each step, the views are either computationally indistinguishable or identical.

We prove the usefulness property of the protocol from the output in the ideal model. It is easy to see that the output in the ideal environment satisfies (γ, ξ)-usefulness because the noise sample selected from Laplace distribution is generated by $\mathcal{T}$ and when the malicious adversary sends 'cheat' it is detected with $1-\omega$ probability. Since the simulation in the real environment is indistinguishable from the ideal environment, the usefulness property also holds for protocol 5.

We wish to reiterate that the simulation works because we are restricting to a differentially private function that holds even against inefficient adversaries. Similarly, we can argue the security when P_0 is rational.

3.4.8 Impact of Differential Privacy

One concern that arises with differential privacy is the usefulness of the results. Are they too noisy for practical utility? To do this, we look at a practical use of the protocol: document comparison using the cosine similarity metric. There are situations where two parties might want to calculate the similarity of their documents without revealing the input documents. Cosine similarity is a widely used metric to measure the similarity of two documents. Cosine similarity can be viewed as the dot product of the normalized input vectors vectors.

$$\frac{{}_j\, x_{0j}.x_{1j}}{\|x_0\|_2\|x_1\|_2} = \sum_j \frac{x_{0j}}{\|x_0\|_2}.\frac{x_{1j}}{\|x_1\|_2} = <x'_0.x'_1>$$

If we assume that every term in the document has equal weight, i.e., 1 or 0 depending upon the presence or absence of a term, then the global sensitivity of the cosine similarity function is upper bounded by $\frac{1}{\sqrt{nm}}$, where n and m are the total number of terms present in the P_0 and P_1 document respectively. The notion of differential privacy allows us to assume that each party knows the size of the other party's document.

Similarity measures usually weight the terms in order to efficiently compute the metric. If tf-idf weighting mechanism is used to measure the importance of words.

Let β be the highest weight of a term in the domain, then the global sensitivity of squared cosine similarity for Party P_0 is always $\leq \frac{2\beta^2}{||x_0||_2^2}$, where $||x_0||_2$ is the norm of the party P_0 input vector. Let x_1' contain one term less than x_1, the sensitivity is given as

$$\Delta s = \max_{x_1, x_1'} \quad \frac{<x_0.x_1>^2}{x_0 \ _2^2 \ x_1 \ _2^2} - \frac{<x_0.x_1'>^2}{x_0 \ _2^2 \ x_1' \ _2^2}$$

$$\leq \frac{<x_0.x_1>^2}{x_0 \ _2^2 \ x_1 \ _2^2} - \frac{<x_0.x_1'>^2}{x_0 \ _2^2 \ x_1 \ _2^2} = \frac{(\quad _j \, x_{0j}.x_{1j})^2 - (\quad _j \, x_{0j}.x_{1j}')^2}{x_0 \ _2^2 \ x_1 \ _2^2}$$

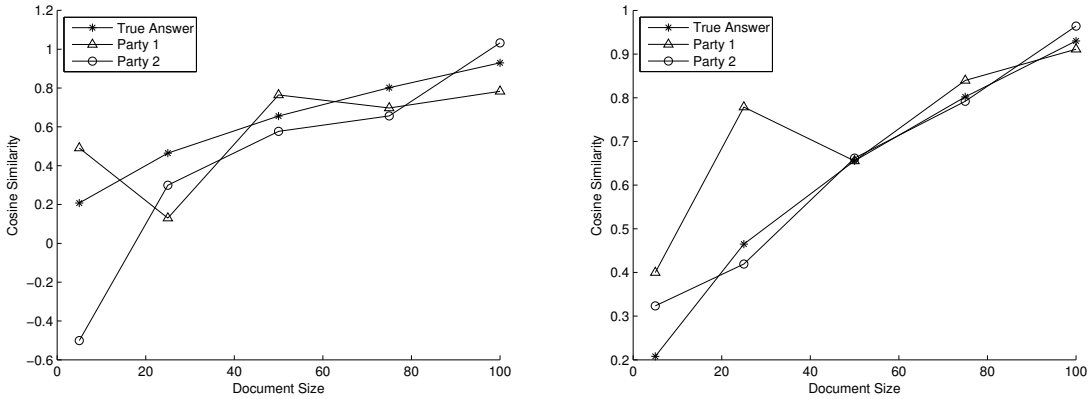
$$= \frac{(\quad _j \, x_{0j}.x_{1j} - \quad _j \, x_{0j}.x_{1j}')(\quad _j \, x_{0j}.x_{1j} + \quad _j \, x_{0j}.x_{1j}')}{x_0 \ _2^2 \ x_1 \ _2^2}$$

$$\leq \frac{(x_{0s}.x_{1s})(2 \quad _j \, x_{0j}.x_{1j})}{x_0 \ _2^2 \ x_1 \ _2^2} \leq \frac{2\beta^2}{||x_0||_2^2}$$

In step 2 and 5, we are using the fact that $|x_1| > |x_1'|$. Since, $x_{0s}.x_{1s} \leq \beta^2$ and $_j \, x_{0j}.x_{1j} \leq \ x_1 \ _2^2$, we can upper bound $\Delta s \leq 2\beta^2/||x_0||_2^2$. Similarly, we can estimate the global sensitivity for party P_1 as $2\beta^2/||x_1||_2^2$. Note that the noise distribution of P_i only depends on his input vector x_i and β (the highest term weight in the domain).

We now evaluate the utility of the protocol by computing differentially private values for different levels of security. An ϵ value of 0 in differential privacy denotes perfect privacy as the probability of seeing the output in D and D' are equal but on the downside the utility will be low. In our experiments (Figure 3.3) we have used ϵ values of 0.1 and 0.3. We implemented the two-party secure differentially private cosine similarity measure without term weighting using the secure dot product protocol and ran the tests for different values of ϵ. The global sensitivity of cosine similarity is $\frac{1}{\sqrt{nm}}$, so the random noise r' is generated from Laplace distribution with mean 0 and scale $\frac{1}{\epsilon\sqrt{nm}}$, where n, m are the number of terms in P_0 and P_1 respectively.

In order to show the deviation of the differentially private similarity score from the true score (cosine similarity without privacy), we plotted the scores of each party obtained on running the protocol along with the true scores. For each ϵ value, we fixed the input (i.e., document) of one party and varied the size of the document of other party. We can see that if the document sizes are small, then the differentially private



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.3$

Figure 3.3.: True output vs differentially private output

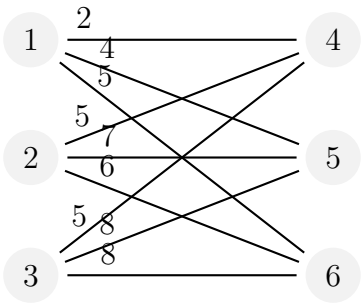
similarity scores are far away from the true but as the size increases, the differentially private similarity score are better approximations of the true score. Hence a party without malicious intent on running the protocol will be able to obtain similarity score closer to the true value.

4 PRIVACY-PRESERVING DATA-OBLIVIOUS ALGORITHMS

Secure multi-party computation (MPC) enables multiple parties with private inputs to jointly engage in a protocol to securely compute a function of their private data. It guarantees different properties of security (i.e., privacy of inputs, correctness, fairness, etc.) depending upon the model. At the end of the protocol, each party gets an output and MPC guarantees that any information that can be learned from the protocol execution can also be inferred from the output and one's own input. There exists various cryptographic primitives [9, 12, 14] that have been used for developing secure protocols.

One limitation of this model is that to really achieve this goal, the protocol must be *data-oblivious*: the visible execution path (time, memory accessed, etc.) must be independent of the other party's data. More precisely, the distribution of executions over a given input must be indistinguishable from the distribution of executions over other inputs of the same size.

This problem is particularly apparent in graph algorithms - where presence or absence of an edge may be significant from a privacy perspective. A naïve method to satisfy data-obliviousness is to represent the data as an (encrypted) adjacency matrix, and access every element when any one is accessed. To avoid this huge inefficiency, [34] proposed to obliviously permute the matrix before accessing an element such that the accesses look uniformly random. For example, let us assume we are running an algorithm (say shortest path) on Figure 4.1a and we want to explore the edges of Node 2. Instead of accessing the entire matrix (4.1b), the nodes are permuted (Figure 4.1c), and only row 1 is accessed. Since the permutation hides that row 1 corresponds to node 2 (we assume the labels are encrypted), this reveals no information. Care must be taken to avoid accessing the same row again, as such frequency of access does reveal information.



(a) Bipartite graph

	1	2	3	4	5	6
1	0	0	0	2	4	5
2	0	0	0	5	7	6
3	0	0	0	5	8	8
4	4	5	5	0	0	0
5	2	7	8	0	0	0
6	5	6	8	0	0	0

(b) Adjacency matrix

	2	4	1	6	3	5
2	0	5	0	6	0	7
4	5	0	2	0	5	0
1	0	2	0	5	0	4
6	6	0	5	0	8	0
3	0	5	0	8	0	8
5	7	0	4	0	8	0

(c) Permuted adjacency matrix

Figure 4.1.: An example bipartite graph and its adjacency matrices

There is a second problem that is not directly addressed by MPC: inference from the output. While clever techniques such as the above can get us to the point where we learn nothing that we cannot infer from our own input and the output, it is possible that the output discloses sensitive information. Differential Privacy [35] deals with what functions can be safely computed. It protects the privacy of individuals by adding noise with magnitude proportional to the *sensitivity* of the queries posed to the database: How much one individual can potentially impact the result of the function. A relaxed definition of computation differential privacy was proposed in [19] that holds against computationally bounded adversaries. Several works have dealt with distributed computationally differentially private protocols [22–24]. A key component of developing differentially private secure protocols is distributed noise generation, this has been addressed for both two-party [36] and three or more party [21] protocols. Combining MPC with differential privacy on the output gives a differentially private

secure multi-party computation, addressing the potential privacy risk inherent in the output of the computation.

Achieving data-obliviousness and differential privacy can be difficult - graphs exist that cause high execution times or require significant noise to be added. However, if our problem allows us to restrict the space of graphs, these problems can be overcome. We demonstrate this idea on bipartite graphs (Figure 4.1); specifically with weighted matching and minimum vertex cover. The idea is that each party holds one "side" of the bipartite graph, and the information on edge weights is split between the parties.

A simple example would be in computing the semantic similarity/distance between two documents (held by two parties). Each party has a sensitive document (represented by a set of nodes/features), which they don't want to reveal to the other party. The edge weights represent the semantic similarity/distance between the two nodes/features. One way to capture the semantic distance between two features/words is shown in [37]. Then, computing the semantic distance between the two documents can be formulated as a minimum weighted bipartite matching problem.

We now introduce relevant definitions from data-obliviousness and differential privacy, introducing notation along the way. We will also discuss related work as we go through this background material. Sections 4.2 and 4.3 give algorithms for data oblivious weighted bipartite matching and minimum vertex cover, respectively. For each, we give a secure multi-party computation to solve the problem, then show how to incorporate differential privacy.

We use square brackets around a variable (e.g., $[x]$) to say that the value of x is encrypted and it is not known to any of the parties involved in the protocol. Since techniques such as linear secret sharing, threshold homomorphic encryption, or Boolean garbled circuits can be used for realizing this, we use these terms interchangeably. For example, a linear secret sharing scheme (like [14]) can be used to securely implement basic operations like addition and multiplication. Efficient techniques for secure equality testing($[x] \stackrel{?}{=} [y]$) and less than ($[x] \stackrel{?}{<} [y]$) are given in [38]. We use

the notations $\oplus, \ominus$, $\otimes$ and $\vee$ to denote secure addition, subtraction, multiplication and inclusive-or operations respectively.

4.1 Related Work

4.1.1 Data-Obliviousness

This work is based on the key idea introduced in [34], which presents nearly optimal secure graph algorithms for breadth-first search (BFS) and single-source single destination (SSSD) shortest path.

Definition 4.1.1 (Data-Obliviousness [34]) *Let d denote the input to the graph algorithm. Also, let $A(d)$ denote the sequence of memory access that the algorithm makes. The algorithm is considered data-oblivious if for two inputs d and d' of equal length, the algorithm executes the same sequence of instructions and access patters $A(d)$ and $A(d')$ are indistinguishable to each party carrying out the computation.*

An if/else condition (IFC) can be obliviously evaluated by executing both the if and else blocks and updating the values depending upon the value of the condition. For example,

if($[cond]$) return $[value_1]$; else return $[value_2]$;

can be obliviously evaluated as

return $([cond] \otimes [value_1]) \oplus ((1 \ominus [cond]) \otimes [value_2])$

The above code snippet is used in the rest of the chapter by calling IFC$([cond], [value_1], [value_2])$.

Oblivious Vector Permutation: One of the basic building blocks used in this chapter is random permutation of a vector V. This can be accomplished by generating a random number for each element of the vector and then obliviously sorting the vector V according to the random values assigned. [39, 40] proposes an $O(n \log n)$ data-oblivious algorithm for sorting a vector of size n. We use this algorithm to obliviously permute the rows and columns of a matrix $n \times n$ matrix $[M]$ consistently

in $O(n^2 \log n)$. Figure 4.1 shows a bipartite graph and its corresponding permuted matrix. Note that the values are re-encrypted as part of the permutation, so that it is not possible to link old/new entries to determine the permutation.

Single Source Single Destination (SSSD) Algorithm: Given a secret shared weighted matrix $[W]$ of a graph $G = (V, E)$, a source $[s]$ and destination $[t]$, [34] presents a secure data-oblivious SSSD path algorithm to get a shortest path $[PT]$ from the source $[s]$ to a given destination $[t]$ in $O(|V|^2)$ time. $[PT]$ is a set of tuples of the form $\langle [v_1], [v_2], [c] \rangle$, where $[v_1], [v_2]$ and $[c]$ correspond to the head, tail and capacity of an edge in the path and it always returns a path of length $|V| - 1$ to conceal the distance between $[s]$ and $[t]$.

[41,42] have proposed data-oblivious algorithms for maximum matching in bipartite graphs. [41] proposes to securely solve fingerprint identification matching using a data-oblivious maximum bipartite matching algorithm. Maximum flow algorithms can also be used for solving maximum matching in bipartite graphs. [42] proposes data-oblivious algorithms for maximum flow based on Edmonds-Karp algorithm and Push-Relabel algorithm, with a runtime complexity of $O(|V|^5)$ and $O(|V|^4)$ respectively. In this section, we propose data-oblivious protocols for weighted bipartite matching as the above protocols are not suitable for our task. [43] proposes an algorithm to privately release the maximum weighted matching in a bipartite graph under a relaxed notion of differential privacy. However, when we extend this to a distributed setting we must take into account that the access patterns can reveal information; the setting in [43] does not face this issue.

Another technique that can be used to develop data-oblivious protocols is by oblivious RAM (ORAM) [44, 45]. It was originally developed for the client-server setting in which client stores its sensitive data in the server. ORAM enables the client with a small trusted memory to access the data without revealing its access patterns to the server. Each memory access has a polylogarithmic overhead $O((\log n)^2)$, where n is the size of the memory. There are ORAM extensions to the multi-party setting [46, 47] which has an overhead of $O((\log n)^3)$ for hiding access patterns. Similar to other

techniques, additional overhead may have to be introduced to prevent information leaks due to sequence and total number of memory access, when using ORAM to develop data-oblivious algorithms.

Distributed Computational Differential Privacy

One challenge in generating a differentially private output in a multi-party setting is the generation of random noise from a specific distribution. Chapter 3 proposed a two-party protocol for generating a random sample from a Laplace distribution with a specific scale parameter in the presence of malicious adversaries. [21] presents protocols for generating random variates with more than two parties. We use these protocols for generating a random sample from Laplace distribution.

4.2 Privacy-Preserving Weighted Bipartite Matching

Given a bipartite graph $G = (V, E)$ where $Q \cup R = V$ and $Q \cap R = \varphi$; and W, a cost matrix that assigns integer edge weights to $e \in E$; the assignment problem is to find a perfect matching ($E' \subseteq E$) in G such that the total cost of the matching is minimized. In a perfect matching, each node in Q is connected to a node in R and vice-versa. A solution to this problem is the Hungarian algorithm [48], which finds the minimum weighted bipartite matching by the following algorithm. Let C be a temporary matrix initialized to W.

1. Construct G', a subgraph of G with only the 0-weight edges (i.e., there is an edge (i, j) in G' iff $C_{i,j} = 0$).

2. Find the maximum matching E' in G'.

3. If there is a perfect matching in G', then $\sum_{(i,j)\in E} W_{i,j}$ is the minimum weighted matching solution, where (i, j) is an edge in E'.

4. Otherwise, the algorithm finds the vertex cover of G'. Let $X \subseteq Q$ and $Y \subseteq R$ be the vertex cover, then the weight matrix C is updated by equation 1, with Δ set such that at least one new 0-weight edge is introduced in G' (i.e., one of the non-zero edge weight in C has become zero.)

$$C_{i,j} = \begin{cases} C_{i,j} - \Delta & \text{if } i \notin X, j \notin Y \\ C_{i,j} & \text{if } i \in X, j \notin Y \\ C_{i,j} + \Delta & \text{if } i \in X, j \in Y \end{cases} \tag{4.1}$$

where $\Delta = \min_{i \notin X, j \notin Y}(C_{i,j})$. Δ is minimum weight of the uncovered edges (i, j), where $i \in QnX$ and $j \in RnY$.

5. Goto step 2.

Given $n = |G|$ (i.e., $|Q| = |R| = \frac{n}{2}$), the algorithm requires a maximum of $\frac{n^2}{4}$ iterations. Each iteration requires a maximum of $O(n^2)$. Hence, the algorithm finishes in $O(n^4)$.

In a two-party scenario, let Q and R belong to two different parties P_0 and P_1. There exist situations where W may be private and not known to both the parties, but they still want to find the minimum weighted bipartite matching. For example, P_0 may have several customizable production plants; P_1 has several components it needs. The weights from P_0 are the cost to produce each component on each line (e.g., in Figure 4.2, P_0 can produce component 4 with cost 1, component 5 with cost 3, and component 6 with cost 4.) P_1 has costs to move components from P_0's plants to where the component is needed (e.g., 1, 2, and 4 for component 4). The sum of these weights is the total cost (Figure 4.1a). While both parties want to achieve the minimum cost, they do not want to reveal their costs and thus compromise pricing negotiations. Instead, $[W]$ (the encrypted weight matrix) is constructed to sum these without revealing weights to either party.

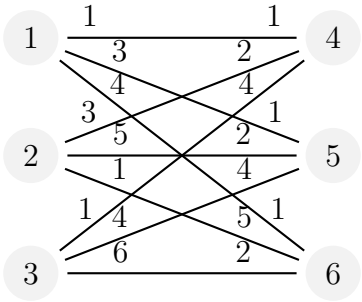
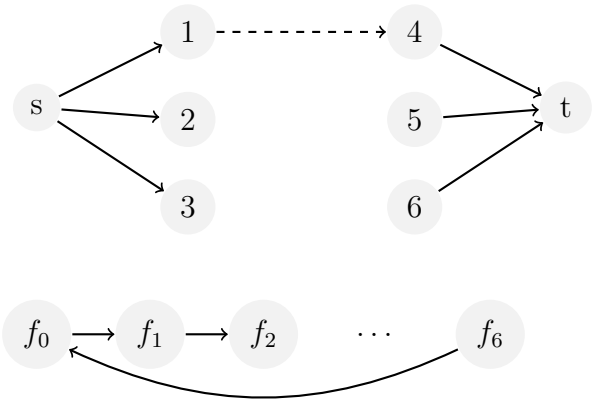


Figure 4.2.: Bipartite graph with shared edge weights

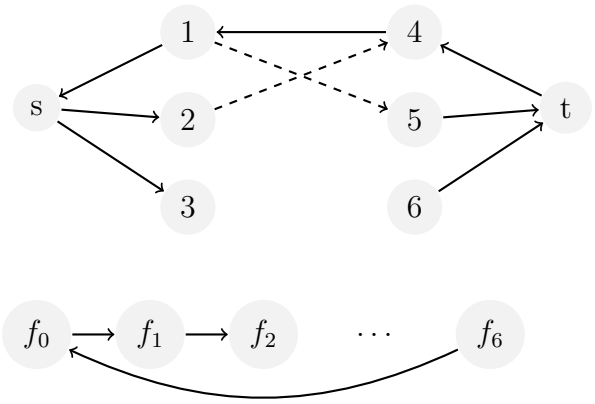
4.2.1 Secure Two-Party Matching Algorithm

In the secure version, the two parties P_0, P_1 hold Q, R respectively. They want to compute the minimum weighted bipartite matching without revealing any information about their individual edge weights. If a linear secret sharing scheme like [14] is used, then the encrypted weight matrix $[W]$ (for the above example) is constructed by each party secret sharing its input weight with the other party and locally summing their shares to obtain $[W]$. If [14] is used, then we assume that there exists a third party to aid in secure multiplication. A third party may not be needed if other techniques such as Boolean circuits or homomorphic encryption is used. We present a data-oblivious algorithm for minimum weighted bipartite matching. In Section 4.2.4 we show how to extend this algorithm to provide differential privacy in the output.

1. Without loss of generality, let the input nodes (Q and R) of the parties P_0 and P_1 be of size $\frac{n}{2}$. Let $[M]$ be the residual graph represented by an adjacency matrix of $2n + 3 \times 2n + 3$ nodes. The edge weights in $[M]$ are either 0 or 1, capturing the presence or absence of an edge. An edge (i, j), where $i \in Q$ and $j \in R$ exists in $[M]$, if and only if $[C_{i,j}] = 0$ ($[C]$ is a copy of $[W]$). Note that $[M_{i,j}]$ is always 0 if $i, j \in R$ or $i, j \in Q$. We need a copy of $[W]$, so that the edge weights can be modified in $[C]$ without destroying the original weight matrix. The locations from 1 to $\frac{n}{2}$ and $\frac{n}{2} + 1$ to n correspond to the nodes in Q and R, $n + 1$ and $n + 2$ are a source(s) and sink(t) node and $n + 3$ to $2n + 3$ are



(a) Initial residual graph, an edge (1,4) is added in iteration 1



(b) Residual graph if two more 0 edges are added: (2,4) and (1,5)

Figure 4.3.: Snapshot of residual graph

fake vertices included to prevent information leakage. For example, a list of fake vertices will be returned by the SSSD algorithm if there is no path between the $[s]$ and $[t]$. The source node $n+1$ is connected to all the nodes of Q (i.e., nodes 1 to $n/2$) and all the nodes in R (i.e., nodes numbered $n/2+1$ to n) are connected to sink node $n+2$. The fake nodes form a simple cycle and are not connected to the original nodes or source/sink nodes.

We also define an indicator vector $[A]$ of size $2n+3$ to denote nodes in Q. I.e., the elements of A from 1 to $n/2$ will be 1 and others will be set to 0. Similarly, we create indicator vectors for $[B]$ to represent the nodes in R and $[S], [T], [F]$

to represent the source, sink and fake nodes respectively. The solid lines in Figure 4.3a shows the initial state of the residual graph $[M]$. These indicator vectors will be consistently permuted with the matrices $[C], [W], [M]$ such that no party can identify the location of their input nodes, but can still perform secure node testing like $[cond] = ([A_v] \stackrel{?}{=} 1)$. $[cond]$ will be set to an encryption of 1 if v belongs to Q and an encryption of 0 otherwise.

1: $[C] = [W]$
2: $[M] = [0]^{\{2n+3 \times 2n+3\}}$
3: $[A] = [1]^{n/2}[0]^{(3n+6)/2}$
4: $[B] = [0]^{n/2}[1]^{n/2}[0]^{n+3}$
5: $[S] = [0]^{n}[1][0]^{n+2}$
6: $[T] = [0]^{n+1}[1][0]^{n+1}$
7: $[F] = [0]^{n+2}[1]^{n+1}$
8: **for** $i = 1$ to $n/2$ **do**
9: $\quad [M_{n+1,i}] = [1]$
10: **end for**
11: **for** $i = n/2 + 1$ to n **do**
12: $\quad [M_{i,n+2}] = [1]$
13: **end for**
14: **for** $i = n + 3$ to $2n + 2$ **do**
15: $\quad [M_{i,i+1}] = [1]$
16: **end for**
17: $[M_{2n+3,n+3}] = [1]$

The values in the residual graph matrix $[M]$ and auxiliary indicator vectors $[A], [B], [S], [T]$ and $[F]$ are encrypted using a non-deterministic scheme. I.e, two encryptions of 1 or 0 produce different cipher texts. Since the initialization step is fixed, both parties know the indices of the nodes in the vector/matrix and the values present in the residual matrix and auxiliary indicator vectors. A permutation/re-encryption at the beginning of step 2 hides this. The plain

Table 4.1.: Plain-text view of the initial state of $[M]$. Actual values are non-deterministically encrypted, e.g., (1,1)=$\mathcal{E}(0)$=439, (4,4)=$\mathcal{E}(0)$=227.

	1	2	3	4	5	6	s	t	f_0	f_1	...	f_6
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	1	0	0	...	0
5	0	0	0	0	0	0	0	1	0	0	...	0
6	0	0	0	0	0	0	0	1	0	0	...	0
s	1	1	1	0	0	0	0	0	0	0	...	0
t	0	0	0	0	0	0	0	0	0	0	...	0
f_0	0	0	0	0	0	0	0	0	0	1	...	0
f_1	0	0	0	0	0	0	0	0	0	0	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
f_6	0	0	0	0	0	0	0	0	1	0	...	0

Table 4.2.: Plain-text view of the initial state of $[A]$

	1	2	3	4	5	6	s	t	f_0	f_1	...	f_6
A	1	1	1	0	0	0	0	0	0	0	...	0

text views of the adjacency matrix $[M]$ and indicator vector $[A]$ are shown in Tables 4.1 and 4.2 for the graph in Figure 4.1a.

2. The main outline of the secure weighted bipartite matching is given as follows. The first step is to find an augmentative path from the source $[s]$ to sink $[t]$ in the residual graph $[M]$ using the SSSD path algorithm given in [34]. If there exists a valid path, then the residual flow is updated in $[M]$. An existence of a valid path means that we can increase the maximum matching. Otherwise, the minimal vertex cover is found and the weights in the cost matrix $[C]$ are updated to introduce new zero edges in $[M]$. An execution of SSSD, UpdateResidualGraph and MinVertexCover reveals the location of some of the nodes, so the nodes are permuted after every call so that the access to particular rows of $[C]$ or $[M]$ look

random. During secure permutation, apart from permuting the indices of the nodes, it also re-encrypts the values. Therefore, the parties do know the indices of any nodes and any access to particular row of the matrix or a particular element in a vector will look like random access to all the parties. Following the example, after permutation the matrix $[M]$ and vector $[A]$ could look like Table 4.3 and 4.4. The algorithm for an oblivious execution is as follows.

1: **for** $i = 1$ to $\frac{n^2+2n}{4}$ **do**
2: $Permute\ [W], [C], [M], [A], [B], [S], [T], [F]$.
3: $[PT] = SSSD([M], s, [t])$
4: $[valid\_path] = IfValidPath([PT])$
5: $Permute\ [W], [C], [M], [A], [B], [S], [T], [F], [PT]$.
6: $[M] = UpdateResidualGraph(G, [valid\_path], [PT])$
7: $Permute\ [W], [C], [M], [A], [B], [S], [T], [F]$.
8: $([X']\ ,\ [Y']) = MinVertexCover([M], [A], [B])$
9: $([X]\ ,\ [Y]) = IFC([valid\_path], ([X], [Y]), ([X'], [Y']))$
10: $[min'] = findmin([X], [Y])$
11: $[min] = IFC([valid\_path], [0], [min'])$
12: $[C] = UpdateWeights([C], [X], [Y], [min])$
13: $[M] = UpdateZeroEdges([C])$
14: **end for**
15: $[MM] = maximumMatching([W], [M], [A], [B])$

Table 4.3.: Plain-text view of $[M]$ after permutation. Note that row and column ID's are not actually visible, and actual values re-encrypted, e.g., the upper left corner (4,4) = $\mathcal{E}(0) = 186$.

	4	f_1	s	1	5	f_0	f_2	3	t	2	...
4	0	0	0	0	0	0	0	0	1	0	...
$f1$	0	0	0	0	0	0	1	0	0	0	...
s	0	0	0	1	0	0	0	1	0	1	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 4.4.: Plain-text view of $[A]$ after permutation

	4	f_1	s	1	5	f_0	f_2	3	t	2	...
A	0	0	0	1	0	0	0	1	0	1	...

3. **IfValidPath:** SSSD always returns a path $[PT]$ containing $n+1$ tuples/edges to conceal the true length of the path. This is because the maximum length of a valid path in $[M]$ can have at-most $n+1$ edges (there are n+2 nodes excluding the fake nodes). The capacity of the true and fake tuples/edges are [1] and [0] respectively. If there exists a path of length l, then the first l tuples contain the true edges of the path with capacity [1], followed by $n-l+1$ fake tuples with capacity [0]. The rest of the fake path could be $PT_l = \langle [t], [f_3], [0] \rangle$, $PT_{l+1} = \langle [f_3], [f_4], [0] \rangle$ and so on. If there exists no path, SSSD returns a list of fake tuples PT_0 to PT_n with capacity [0]. The first tuple will be $PT_0 = \langle [s], [f_r], [0] \rangle$, followed by the fake tuples $PT_1 = \langle [f_r], [f_{r+1}], [0] \rangle$, $P_2 = \langle [f_{r+1}], [f_{r+2}], [0] \rangle$ and so on. Therefore, we need to validate the augmentation path P returned by SSSD by checking if the first tuple's capacity is [0]. The secret shared variable $[valid\_path]$ has value [1] if there exists a valid path from $[s]$ to $[t]$ and [0] otherwise. In the first iteration, there are no augmentation paths from $[s]$ to $[t]$. Hence, $[valid\_path]$ will be set to [0].

   1: $[valid\_path] = [PT_0.c] \stackrel{?}{=} [0]$

4. **UpdateResidualGraph:** Given a path $[PT]$ and its validity $[valid\_path]$, the flow of the residual graph $[M]$ is updated. For each edge (v_1, v_2) in the path, we open its locations in the adjacency matrix $[M]$ and subtract the capacity from the flow. If the weight/flow of an edge (v_1, v_2) is updated from [1] to [0], we also add the flow to the back-edge (v_2, v_1) to allow another augmenting path in future iterations to undo some of the flow used by the current augmenting path. For example, SSSD returns a valid path from $[s]$ to $[t]$, if run on residual

graph shown in Figure 4.3a. The updated residual graph is shown by the solid lines in Figure 4.3b.

Care must be taken to avoid updating the flow on the edges between fake nodes as this can destroy the structure of fake nodes. This can be achieved by setting the $[capacity]$ to [0], when $[t]$ is reached. Revealing the locations of the edges in residual graph $[M]$ does not leak information because the rows and columns of $[M]$ are permuted/re-encrypted; it is not possible to distinguish between a true node and a fake node, and the access to rows of $[M]$ appears random.

1: $[capacity] = [valid\_path]$
2: **for** $i = 0$ to n **do**
3: Open v_1 and v_2 in PT_i
4: $[M_{v_1,v_2}] = [M_{v_1,v_2}] \ominus [capacity]$
5: $[M_{v_2,v_1}] = [M_{v_2,v_1}] \oplus [capacity]$
6: $[cond] = v_2 \stackrel{?}{=} [t]$
7: $[capacity] = \text{IFC}([cond], [0], [capacity])$
8: **end for**

5. **FindMin and UpdateWeights:** If a perfect matching has not been found yet, then we need to introduce new zero weight edges in the residual graph to increase the maximum matching. This is done by first finding the minimum vertex cover of $[M]$, which returns the indicator vectors $[X]$ and $[Y]$ that represent vertex cover in Q and R. The algorithm for MinVertexCover is given in Section 4.3. Since, we try to find the minimum weighted matching, the minimum (Δ) weight of the uncovered edges (head and tail are not in the vertex cover) is found and the weight of the edges $[C]$ are updated according to equation 1. If the perfect matching has already been found, then $[X]$ will be equal to $[A]$ and $[Y]$ is set to all [0]'s. Steps 10-15 check if all the elements in $[Y]$ are set to [0], if so then $[min]$ is set to [0]. Hence, none of the entries will be updated if a perfect matching is already found. The first for loop computes the minimum of the uncovered

nodes and second loop is used for updating the cost of the edges. Let Λ be the maximum possible edge weight.

1: $[min] = [\Lambda]$
2: **for** $i, j = 1$ to $2|V| + 3$ **do**
3: $\quad [cond_1] = ([A_i] \stackrel{?}{=} [1]) \otimes ([B_j] \stackrel{?}{=} [1])$
4: $\quad [cond_2] = ([X_i] \stackrel{?}{=} [0]) \otimes ([Y_j] \stackrel{?}{=} [0])$
5: $\quad [flag] = [cond_1] \otimes [cond_2]$
6: $\quad [cost] = \textsc{Ifc}([flag], [C_{i,j}], [\Lambda])$
7: $\quad [cond] = [cost] \stackrel{?}{<} [min]$
8: $\quad [min] = \textsc{Ifc}([cond], [cost], [min])$
9: **end for**
10: $[sum] = [0]$
11: **for** $i = 1$ to $2|V| + 3$ **do**
12: $\quad [sum] = [sum] \oplus [Y_i]$
13: **end for**
14: $[cond] = [sum] \stackrel{?}{=} [0]$
15: $[min] = \textsc{Ifc}([cond], [0], [min])$
16: **for** $i, j = 1$ to $2|V| + 3$ **do**
17: $\quad [cond_1] = ([A_i] \stackrel{?}{=} [1]) \otimes ([B_j] \stackrel{?}{=} [1])$
18: $\quad [cond_2] = ([X_i] \stackrel{?}{=} [1]) \otimes ([Y_j] \stackrel{?}{=} [1])$
19: $\quad [cond_3] = ([X_i] \stackrel{?}{=} [0]) \otimes ([Y_j] \stackrel{?}{=} [0])$
20: $\quad [\Delta] = [cond_1] \otimes [min]$
21: $\quad [C_{i,j}] = \textsc{Ifc}([cond_2], [C_{i,j}] \oplus [\Delta], [C_{i,j}])$
22: $\quad [C_{i,j}] = \textsc{Ifc}([cond_3], [C_{i,j}] \ominus [\Delta], [C_{i,j}])$
23: **end for**

For example, if the input to the MinVertexCover is the residual graph shown by only the solid lines in Figure 4.3a, it returns $[X]$ and $[Y]$ with all $[0]'s$. This is because there are no edges between the set R and Q; the minimum vertex cover set is null. The uncovered nodes are the set $R \cup Q$. We can see that

Table 4.5.: Updated cost matrix [C]

	2	4	1	6	3	5
2	0	3	0	4	0	5
4	3	0	0	0	3	0
1	0	0	0	3	0	2
6	4	0	3	0	6	0
3	0	3	0	6	0	6
5	5	0	2	0	6	0

the minimum edge weight between the uncovered edges is 2 (edge (1,4)). The values in $[C]$ will look like Table 4.5. To show the full matrix we have included only nodes from Q and R. In reality, $[C]$ will also have fake nodes with zero weights. This update replaces one of the non-zero edge weights (namely, $(1, 4)$) with [0].

6. **UpdateZeroEdges:** If the weights are updated in the previous step, then at least one of the non-zero weights in $[C]$ has become [0], which in turn introduces a new edge in the residual graph $[M]$. It is also possible that a zero weighted unmatched edge in $[C]$ could become non-zero. Therefore, for each edge (i, j), we add it to the residual graph $[M]$ if and only if the cost of the edge (i, j) is zero and it has not already been matched. Only unmatched zero weights are added to the residual graph because if an edge is already matched, then $[M_{i,j}] = 0$ and $[M_{j,i}] = 1$. In the first iteration, $[C_{1,4}]$ becomes [0], therefore a new zero edge would be introduced in the residual graph(refer Figure 4.3a).

   1: **for** $i, j = 1$ to $2|V| + 3$ **do**
   2: $\quad [cond\_1] = ([A_i] \stackrel{?}{=} [1]) \otimes ([B_j \stackrel{?}{=} [1])$
   3: $\quad [cond\_2] = [C_{i,j}] \stackrel{?}{=} [0]$
   4: $\quad [M_{i,j}] = \textsc{Ifc}([cond\_1], [cond\_2] \ominus [M_{j,i}], [M_{i,j}])$
   5: **end for**

7. **MaximumMatching:** At the end, we have found a minimum weighted perfect matching of the weighted bipartite graph. In order to compute the minimum

sum, we iterate over the edges of residual graph to check if an edge(i, j) is matched (i.e., $[M_{j,i}]$ is [1]) and the cost of the corresponding matched edge $[W_{i,j}]$ is added to minimum sum. At the end of the instruction 7, $[sum]$ contains the true output to the minimum weighted bipartite matching problem but it is still not available to the parties in plain text. In Section 4.2.4, we will describe how much noise needs to be added to this so that the final output is differentially private.

1: $[sum] = [0]$
2: **for** $i, j = 1$ to $2|V| + 3$ **do**
3: $\quad [cond_1] = ([A_i] \stackrel{?}{=} [1]) \otimes ([B_j] \stackrel{?}{=} [1])$
4: $\quad [cond_2] = ([M_{j,i}] \stackrel{?}{=} [1])$
5: $\quad [flag] = [cond_1] \otimes [cond_2]$
6: $\quad [sum] = [sum] + \textsc{Ifc}([flag], [W_{i,j}], [0])$
7: **end for**
8: return $[sum]$

At each iteration, the algorithm tries to increase the number of maximum matches by checking if there exists a path between $[s]$ and $[t]$ in the residual graph. If there is a path, then the maximum matching is increased. Otherwise, a new zero edge is introduced in the residual graph by finding a minimum of the uncovered edges and updating the weights in the cost matrix $[C]$. Finding the minimum weighted matching in the insecure version requires a maximum of $\frac{n^2}{4}$ iterations. The data-oblivious algorithm is similar to that of the insecure version except that increasing the maximum matching and updating weights to introduce zero edges are mutually exclusive steps. Therefore, the algorithms requires a maximum of $\frac{n^2}{4} + \frac{n}{2}$ iterations of find minimum weighted perfect matching.

4.2.2 Complexity Analysis

The SSSD algorithm for finding an augmentative path takes at most $O(|V|^2)$. Step 3 takes a constant amount of time as it invokes constant number of secure equality operations. Step 4 takes a linear amount of time to update the residual graph because the length of the path is $|V|$. Steps 5-7 run in $O(|V|^2)$ time as it visits each element of the matrix. The costliest step is the data-oblivious matrix permutation which takes $O(|V|^2 \log |V|)$. Therefore each iteration of the algorithm is bounded by $O(|V|^2 \log |V|)$. Since the number of iterations needed to find a perfect matching is $O(|V|^2)$, the overall complexity of the algorithm is $O(|V|^4 \log |V|)$.

To determine the practicality of the algorithm, we give a rough estimate of the run time of the algorithm in terms of number of secure operations (multiplication plus comparison operations). In a linear secret sharing scheme, multiplication and comparison dominate the run time as additions and subtractions can be done locally. We estimate clock time based on [49], which showed that a secure scalar product of 100K length vector can be done in 550ms. Hence, on average a single multiplication takes $5.5\mu s$. The total run time of the algorithm (steps 1 to 15) would be approximately $(5.5 \times C \times \frac{|V|^2+2|V|}{4})$ μs, where C is the total number of secure operations done in steps 2-13 of the algorithm. Step 4 (IfValidPath) does a single secure operation. Step 13 (UpdateZero edges) takes $(2|V|+3)^2$ iterations and performs 6 operations in each iteration (3 comparisons and 3 multiplications). So, in total Step 13 has $6(2|V|+3)^2$ operations. Similarly, we can compute for Steps 4, 6, 8-15, which can be upper bounded by $77(2|V|+3)^2$ secure operations. Step 2 (SSSD) takes $(2|V|+3)^2$ iterations and needs 35 secure operations in each iteration. So, the total number of operations are $35(2|V|+3)^2$. Step 2, 5 and 7 (oblivious permutation) can be implemented by oblivious sorting, which roughly does $(2|V|+3)^2 \log(2|V|+3)$ iterations and has 17 operations in each iterations (1 secure comparison and 16 multiplications). So the total number of secure operations of steps 2, 5 and 7 are $51 \times (2|V|+3)^2 \log(2|V|+3)$. So, C can be upper bounded by $163 \times (2|V|+3)^2 \log(2|V|+3)$. For $|V| = 10$, the

algorithm should take around 20 seconds. For $|V| = 50$, the algorithm should take around 200 minutes.

4.2.3 Security

Theorem 4.2.1 *The minimum weighted bipartite matching algorithm is secure with respect to Definition 4.1.1.*

To prove that the algorithm (call $\mathcal{A}$) given in Section 4.2 is secure, we need to show for any input bipartite graph $G = (V, E)$ with n nodes (1) the sequence of execution is the same as any bipartite graph $G' = (V', E')$ with n nodes and (2) the distribution of memory accesses of $\mathcal{A}$ in G is indistinguishable with the distribution of memory accesses of $\mathcal{A}$ in a random bipartite graph G' with n nodes. All the steps in the algorithm executes the same set of instructions for any bipartite graph with n vertices because all conditional statement has been serialized and all control loops are conditioned on the number of vertices. To prove the memory accesses are indistinguishable we analyze each step. The initialization is a standard step and accesses the same memory locations for any bipartite graph of size n. The main for loop executes for $\frac{n^2+2n}{4}$ times as it is a complete bipartite graph with $\frac{n}{2}$ nodes in each set. Step 1 (permutation) and 2 (SSSD) are data-oblivious and the proofs for them are given in [34, 40]. Step 3 is data-oblivious because it always executes one instruction and checks the capacity of the first tuple in the path P. In step 4, the number of memory access is the same as the path P is always of length n but the memory accesses are different. Data-obliviousness is achieved by means of the crucial permutation step done before opening the vertices in path P. Hence the data accesses appear random. Data-obliviousness for finding the minimum vertex cover is shown in Section 4.3. Steps 5, 6 and 7 all access the same memory locations for any graph of the same size. Therefore, all the steps in the algorithm either access the same memory location or the distribution of the location of memory access are indistinguishable. Hence the algorithm is secure with respect to Definition 4.1.1.

4.2.4 Differentially Private Weighted Bipartite Matching

The global sensitivity of the Weighted bipartite matching under edge privacy (neighboring graphs differ by a single edge weight) is Λ, where Λ is the maximum possible value that an edge weight can take. To avoid adding large noise, we can instead compute smooth sensitivity of minimum weighted bipartite matching that satisfies (ϵ, δ) differential privacy. The maximum local sensitivity of a weighted bipartite graph G with n nodes at distance k for any $k \geq \frac{n}{2} - 1$ is Λ. Therefore the smooth sensitivity is equal to

$$S^*(G) = \max \begin{cases} e^{-\beta k} \max\limits_{d(G,G')=k} LS(G') & \text{if } k \leq \frac{n}{2} - 2 \\ e^{-\beta(\frac{n}{2}-1)}\Lambda & \text{if } k \geq \frac{n}{2} - 1 \end{cases}$$

Computing the max local sensitivity at distance k for $k \leq \frac{n}{2} - 2$ could take exponential time. Therefore, we upper bound the maximum change in minimum weighted bipartite matching by $\min(XWM(G) - NWM(G), \Lambda)$, where XWM/NWM is the maximum/minimum weighted bipartite matching of G respectively. Given a secure data-oblivious algorithm $\mathcal{A}$ for computing NWM from weighted bipartite graph G, it is possible to compute XWM from G using $\mathcal{A}$. This is done constructing another graph G' where the number of nodes is the same as G and replacing $w(e)$ in G with $w'(e) - w(e)$ in G', where $w'(e)$ is the maximum weight of an edge in $[W]$. The edges of NWM in G' are the same as the edges of XWM in G. Therefore, we can compute the upper bound of smoothed local sensitivity of minimum weighted bipartite matching in polynomial time. For weighted bipartite graphs for which difference between XWM and NWM is smaller than Λ, we can achieve better utility by adding noise magnitude proportional to the smooth sensitivity and achieve (ϵ, δ) differential privacy. The upper bounded smooth sensitivity is as follows

$$S^*(G) = \max(\min(\text{XWM(G) - NWM(G)}, \Lambda), e^{-\beta(\frac{n}{2}-1)}\Lambda)$$

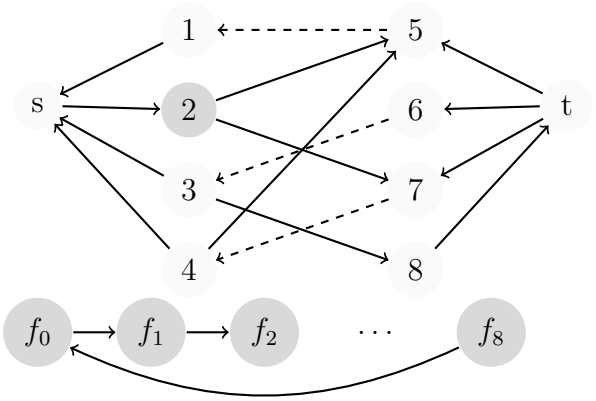
Since the values of n, Λ, β are known, we can securely compute the smooth sensitivity (i.e., $[S^*(G)]$). We can use the protocols specified in [21, 36] to generate a uniformly random noise from Laplace distribution with smooth sensitivity. The securely generated random noise can then be added to the true output $[MM]$ to obtain a differentially private minimum weighted bipartite matching $\overline{[MM]}$, which can then be opened by the parties.

The utility of the minimum weighted matching sum $\overline{[MM]}$ depends upon the amount of noise introduced. If GS is used, then the error/variance of the result is $2\frac{\Lambda^2}{\epsilon^2}$, which can be quite large if Λ is large. In case of LS, the error is dependent upon the input graph $2\left(\frac{S^*(G)}{\epsilon}\right)^2$. In the worst case, $S^*(G)$ is equal to Λ.

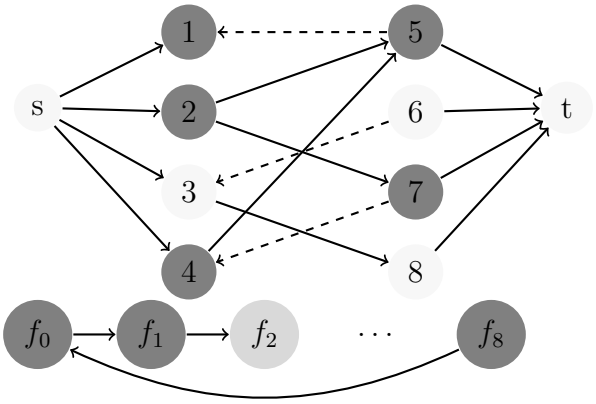
4.3 Minimum Vertex Cover for Bipartite Graph

A vertex cover of a graph $G = (V, E)$ is the set of vertices such that each edge $e \in E$ on the graph is incident to at least one vertex of the set. A vertex cover is minimum if there are no other vertex covers that have fewer vertices. The problem of finding a minimum vertex cover for a graph is NP-hard. But, a polynomial algorithm exists for finding a min vertex cover in a bipartite graph due to the equivalence between vertex cover and maximum matching. Given a bipartite graph G, maximum matching M and two partitions A and B, the minimum vertex cover is found as follows (1) find all the vertices that are reachable from any of the unmatched vertices of A. (2) Let $A' \subseteq A$ and $B' \subseteq B$ are the reachable vertices. Then, $(A \setminus A') \cup B'$ is the minimum vertex cover.

The algorithm below shows how to compute the minimum vertex cover securely given a secret shared maximum matching bipartite flow graph $[M]$ and auxiliary indicator vectors $[A], [B]$ and $[F]$ computed by the previous algorithm. A possible input of this algorithm is shown in Figure 4.4a. The matched edges are represented by dashed edges (e.g., edge (1,5) is matched and it is represented by a dashed backward-edge (5,1) in [M] by $[M_{5,1}] = 1$).



(a) Bipartite graph with maximum matching



(b) After n iterations: Nodes 3, 5 and 7 form the min vertex cover

Figure 4.4.: Minimum vertex cover

1. Let $[P]$ be a encrypted vector of size $2|V| + 3$ with the encrypted field *color*. Initialize the color of all the nodes to *white*.

2. Randomly select a node from the fake vertices. Let it be $[f_r]$. This can be done by assigning random values $[r_i]$ to each element of the node and picking the index i, which has the maximum random value $[r_i]$ and $[F_i]$ set to 1. Step 4 shows the pseudo-code to randomly pick an element from a vector obliviously.

3. Find all unmatched vertices in the set $[A]$ ($[A]$ can also be considered as a set; $[A_v] = [1]$ denoting the presence of node v in $[A]$ and $[0]$ otherwise). This is

done by assigning $[P_i.color] = [gray]$ iff $i \in A$, $[M_{i,j}] = [1]$ (existence of an edge) and $[M_{k,i}] = [0]$, $\forall k \in B$ (node i is not matched with any k for all $k \in B$). In Figure 4.4a, node 2 is the only unmatched node and its color is set to *gray*.

1: **for** $i = 1$ to $2|V| + 3$ **do**
2: $[matched] = [0]$
3: **for** $j = 1$ to $2|V| + 3$ **do**
4: $[cond_1] = ([A_i] \stackrel{?}{=} [1]) \otimes ([B_j] \stackrel{?}{=} [1])$
5: $[cond_2] = ([M_{j,i}] \stackrel{?}{=} [1])$
6: $[matched] = [matched] \vee ([cond_1] \otimes [cond_2])$
7: **end for**
8: $[P_i.color] = \text{IFC}([matched], [white], [grey])$
9: **end for**

4. We want to find all the nodes that are reachable from the unmatched vertices in set $[A]$. Deterministically picking a node to explore can reveal information, therefore we randomly pick a non-fake node whose color is *gray* (i.e., $[P_i.color] = [gray]$) and assign it to $[v]$. This can be done by assigning random values $[R_i]$ to each node and picking the index $[i]$ that has the highest random value. If there are no such nodes, we set $[v] = [f_r]$, where $[f_r]$ is randomly picked fake node in step 2.

1: $[v] = [0]$
2: $[max] = [0]$
3: **for** $i = 1$ to $2|V| + 3$ **do**
4: $[cond_1] = ([A_i] \stackrel{?}{=} [1]) \vee ([B_i] \stackrel{?}{=} [1])$
5: $[cond_2] = [P_i.color] \stackrel{?}{=} [gray]$
6: $[r] = [cond_1] \otimes [cond_2] \otimes [R_i]$
7: $[cond_3] = [r] \stackrel{?}{>} [max]$
8: $[max] = \text{IFC}([cond_3], [r], [max])$
9: $[v] = \text{IFC}([cond_3], [i], [v])$

10: **end for**

11: $[cond] = [v] \stackrel{?}{=} [0]$

12: $[v] = \text{IFC}([cond], [f_r], [v])$

5. Open the index $[v]$ to access the row M_v. Opening the location does not reveal any information because the nodes are permuted so the access looks random. Also, the algorithm never access any row of $[M]$ twice. Then, expand the list of the reachable nodes from v. For all the vertices i, if $[M_{v,i}] = [1]$ and $[P_i.color] = [white]$, set $[P_i.color] = [gray]$. Then the color of P_v is set to *black* to denote that the children of v have been explored.

   1: $open([v])$

   2: **for** $i = 1$ to $2|V| + 3$ **do**

   3: $[cond_f] = [M_{v,i}] \stackrel{?}{=} [1]$

   4: $[cond_w] = [P_i.color] \stackrel{?}{=} [white]$

   5: $[P_i.color] = \text{IFC}([cond_f] \otimes [cond_w], [gray], [P_i.color])$

   6: **end for**

   7: $[P_v.color] = [black]$

6. Repeat steps 4-5 $n - 1$ times.

7. After Step 6, all nodes that are reachable from unmatched nodes of $[A]$ are marked *black* and unreachable nodes are marked *white*. If $[f_8]$ was the randomly selected fake node in step 2, the state of $[M]$ is shown in Figure 4.4b. Let $[X]$ and $[Y]$ are indicator vectors to specify vertex cover nodes that are in $[A]$ and $[B]$ respectively. If $X = A \cap \{white\_nodes\}$ and $Y = B \cap \{black\_nodes\}$, then $X \cup Y$ is the minimum vertex cover of $[M]$. It can be securely realized as follows.

   1: **for** $i = 1$ to $2|V| + 3$ **do**

   2: $[cond_c] = [P_i.color] \stackrel{?}{=} [white]$

   3: $[X] = IFC([cond_c] \otimes [A_i], [1], [0])$

   4: $[Y] = IFC((1 - [cond_c]) \otimes [B_i], [1], [0])$

   5: **end for**

8. Return $[X]$ and $[Y]$. Since this algorithm is called as a subroutine by the algorithm in Section 4.2, the vertex covers are not opened here.

4.3.1 Complexity Analysis

The overall complexity of the secure algorithm to find the minimum vertex cover of a bipartite graph is $O(|V^2|)$. This can be proved by analyzing every step of the algorithm. Step 1 and 2 take linear time $O(|V|)$. Step 3 takes $O(|V|^2)$ time as it access every element of the adjacency matrix. Steps 4 and 5 take linear time $O(|V|)$ to select and expand a random gray colored node. Since, steps 4 and 5 are the executed $|V|$ times, the total run time time $O(|V|^2)$. Step 7 also does a linear amount of work to find the vertex covers.

4.3.2 Security

Theorem 4.3.1 *The minimum vertex cover algorithm is secure with respect to Definition 4.1.1.*

To prove that the algorithm (call $\mathcal{B}$) is secure, we need to show, for any input bipartite graph $G = (V, E)$ with n nodes and its corresponding maximum matching residual matrix $[M]$ and indicator vectors $[A], [B]$ and $[F]$, (1) the sequence of execution is the same as any bipartite graph $G' = (V', E')$ with n nodes and maximum matching $[M']$ and indicator vectors $[A]', [B']$ and $[F']$; and (2) the distribution of memory accesses of $\mathcal{B}$ in G is indistinguishable with the distribution of memory accesses of $\mathcal{B}$ in a random bipartite graph G' with n nodes. Steps 1, 2 and 3 are initialization steps and have the same executions and memory accesses for any bipartite graph G with maximal matching $[M]$ with n nodes. Step 4 used for picking a random node also accesses the same memory locations of $[A], [B]$ and $[P]$. Step 5 executes the same instructions but can access different memory locations for different bipartite graphs. Step 5 provides data-obliviousness because of the permutation: any access to a row

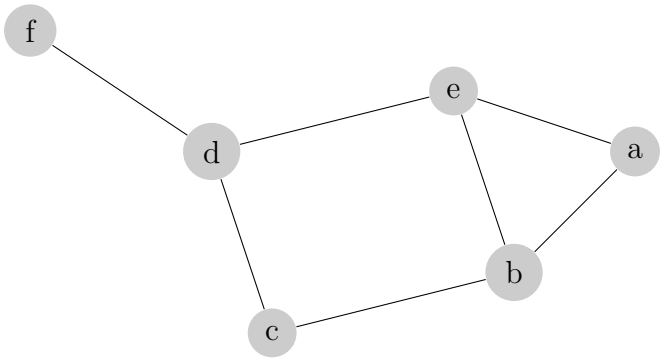
of $[M]$ is equally likely. Also, each row of the matrix is accessed only once in the whole execution. Step 4 and 5 are executed n times for any bipartite graph of size n. Step 7 used for finding the minimal vertex cover executes the same instructions and accesses the same memory for any bipartite graph of size n. Therefore, the algorithm executes the same number of instructions and all the instructions in the algorithm either access the same memory location or the distribution of the location of memory access are indistinguishable. Hence the algorithm is secure with respect to Definition 4.1.1.

4.4 Privacy-Preserving Articulation Points

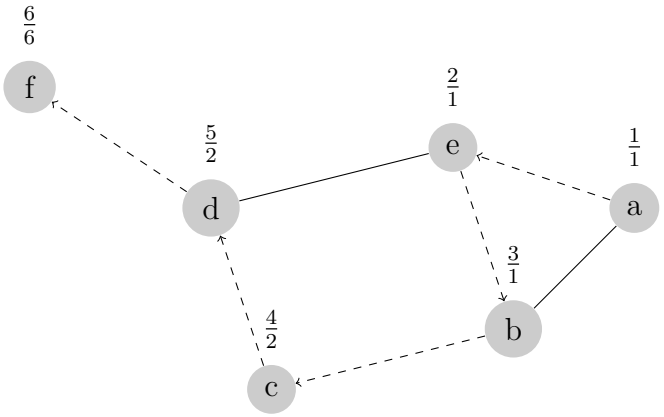
A node $v \in V$ in a undirected graph $G = (V, E)$ is an articulation point (or cut vertex), iff removing it and edges incident on it will increase the number of connected components in the graph. Finding them is useful in designing reliable networks as failure of a single node would split the network into two or more disconnected components.

In the secure version, given a secret shared adjacency matrix $[M]$ of G, we want to identify articulation points $v \in V$ such that the information about $[M]$ is not leaked. Steps 1-6 of the below algorithm constructs a secure depth first tree (DFT) of the graph G. Steps 7 and 8 identifies the articulation points in the algorithm using dfn and low. Figure 4.5 shows an example of an undirected graph with a plausible dfn/low value for each node.

1. Let $[P]$ be a secret shared vector with five fields *visited*, *parent*, *depth*, dfn and *low*, where *visited* denotes if the node has been already visited in the depth first tree (DFT), *depth* is the depth of the node in DFT. *parent* specifies the parent of node, dfn denotes the order in which the node was visited in the DFT and *low* of node v denotes the earliest visited node that can be reached from the descendent of v.



(a) An undirected graph



(b) DFT with dfn and low

Figure 4.5.: Articulation points

2. Initialize the counter $time = 0$. Set the *visited* and *depth* variable of all the nodes to 0 and 1 respectively. Assign a special value $\perp$ to all *parent* variable.

3. Since we are building a DFT, we pick an unvisited random node that has the highest depth and set it has the current working node ($[v]$). The first for loop finds the maximum depth from all the previously unvisited nodes and it is stored in the variable $[max]$. Then it creates a temporary protected array $[T]$ that assigns 1 if the node is unvisited and has the maximum depth. In order to select a random element from the potential node in T, random values are assigned to each element in the array. Then, the node with maximum value is picked as the working node $[v]$.

1: $[max] = 0$
2: **for** $i = 1$ to $|V|$ **do**
3: $\quad [c_1] = [P_i.depth] \overset{?}{>} [max]$
4: $\quad [c_2] = [P_i.visited] \overset{?}{=} 0$
5: $\quad [max] = IFC([c_1] \otimes [c_2], [P_i.depth), [max])$
6: **end for**
7: $[i_{max}] = 0$
8: $[rmax] = 0$
9: **for** $i = 1$ to $|V|$ **do**
10: $\quad [c_1] = [P_i.depth] \overset{?}{=} [max]$
11: $\quad [c_2] = [P_i.visited] \overset{?}{=} 0$
12: $\quad [T_i] = [c_1] \otimes [c_2] \otimes [RAND]$
13: $\quad [c] = [T_i] \overset{?}{>} [rmax]$
14: $\quad [rmax] = IFC([c], [T_i], [rmax])$
15: $\quad [i_{max}] = IFC([c], i, [i_{max}])$
16: **end for**
17: $[v] = [i_{max}]$

4. Reveal the location of $[v]$ in the adjacency matrix $[M]$ and let the row of the node be M_v. Increment the counter *time* and assign dfn and *low* number to time. Then set the *visited* flag of v to 1.

1: $[time] = [time] + 1$
2: $[P_v.dfn] = [time]$
3: $[P_v.low] = [time]$
4: $[P_v.visited] = 1$

5. Extend the depth first search tree by discovering unvisited nodes from current node v. When a unvisited node u is discovered then parent of u is set to v, the depth of v is set to depth of v plus one. If the node u is already visited, then

the edge (u, v) is a back edge and v can reach a lower numbered node u in the DFS tree. Therefore the *low* of v is updated to dfn number of u.

1: **for** $i = 1$ to $|V|$ **do**
2: $[c_1] = [M_{v,i}] \stackrel{?}{=} [1]$
3: $[c_2] = [P_i.visited] \stackrel{?}{=} [1]$
4: $[c_3] = [P_v.parent] \stackrel{?}{=} [i]$
5: $[c_4] = [c_1] \otimes (1 - [c_2])$
6: $[P_i.depth] = IFC([c_4], [P_v.depth] + 1, [P_i.depth])$
7: $[P_i.parent] = IFC([c_4], [v], [P_i.parent])$
8: $[c_5] = [c_1] \otimes [c_2] \otimes (1 - [c_3])$
9: $[c_6] = [P_i.dfn] \stackrel{?}{<} [P_v.low]$
10: $[P_v.low] = IFC([c_5] \otimes [c_6], [P_i.dfn], [P_v.low])$
11: **end for**

6. Repeat steps 3-6 $n - 1$ times

7. Compute the *low* function for all the nodes in their descending order of discovery. For every node v, if any of its children u can get to a lowest numbered node v' in DFS tree, then the *low* of v is set to *low* of u because v can also get to v' by following the tree edge (v, u).

1: **for** v in descending order of dfn **do**
2: **for** $i = 1$ to $|V|$ **do**
3: $[c_1] = [P_i.parent] \stackrel{?}{=} [v]$
4: $[c_2] = [P_v.low] \stackrel{?}{>} [P_i.low]$
5: $[c_3] = [c_1] \otimes [c_2]$
6: $[P_v.low] = IFC([c_3], [P_i.low], [P_v.low])$
7: **end for**
8: **end for**

8. Let $[A]$ be a protected binary vector with 1/0 representing if the corresponding element is an articulation point or not. A root node is an articulation point if

it has more than one children in the DFS tree. The first for loop checks for the number of children of root node r and sets A_r appropriately. For the rest of the nodes, a node v is an articulation point if its dfs no is less than or equal to low no of any its children (i.e., $dfs(v) \leq low(u)$, where u is the child of v). The second for loop

1: **for** $r = 1$ to $|V|$ **do**
2: $[c_1] = [P_r.depth] \stackrel{?}{=} [1]$
3: $[t] = 0$
4: **for** $i = 1$ to $|V|$ **do**
5: $[c_2] = [P_i.parent] \stackrel{?}{=} [r]$
6: $[t] = [t] + [c_2]$
7: **end for**
8: $[c_3] = [t] \stackrel{?}{>} [1]$
9: $[A_r] = [c_1] \otimes [c_3]$
10: **end for**
11: **for** $v = 1$ to $|V|$ **do**
12: **for** $i = 1$ to $|V|$ **do**
13: $[c_1] = [M_{v,i}] \stackrel{?}{=} [1]$
14: $[c_2] = [P_v.dfn] \stackrel{?}{<} [P_i.low]$
15: $[c_3] = [c_1] \otimes [c_2]$
16: $[A_v] = IFC([c_3], [1], [A_v])$
17: **end for**
18: **end for**

4.4.1 Complexity Analysis

Steps 1-3 of the algorithm take linear time as it involves creating a protected vector P with five fields, initializing them and randomly selecting a potential node to explore from the depth first tree respectively. Step 4 takes a constant amount of

time as it just initializes the fields of the working node. Step 5 takes linear time as it analysis each element of the vector $[M_v]$ and updates the depth first search tree with new nodes. Steps 3-5 are repeated $n-1$ times so that the DFS tree is built completely. Therefore total time is bounded by $O(|V|^2)$. Steps 7 takes $O(|V|^2)$ time as it goes through every element of the adjacency matrix $[M]$ to update the *low* function. Step 8 also takes $O(|V|^2)$ time as it access the protection vector $[P]$ for every node v in the graph. Permuting the rows and columns of the matrix (optional step and it is called before step 1 only if the node labels reveal any information) takes time $O(|V|^2 \log |V|)$ and it overwhelms the running time of the other steps in the algorithm. Hence the overall complexity of the algorithm is $O(|V|^2 \log |V|)$. If the permutation step is skipped then the algorithm takes $O(|V|^2)$ time to run.

4.4.2 Security

Theorem 4.4.1 *The articulation points detection algorithm is secure with respect to Definition 4.1.1.*

As before, we will analyze each step of the algorithm to prove that it is secure. All the steps in the algorithm execute the same set of instructions for any graph with fixed size of n vertices because all conditional statement has been serialized and all control loops are conditioned on the number of vertices. To show that the memory accesses are indistinguishable for G and G' with a fixed node size n, we analyze each step of the algorithm. Steps 1 and 2 are initialization steps and accesses the same memory locations for any graph with n nodes. Step 3 also accesses the same memory locations of the protected vector $[P]$ and $[T]$ for any graph with n nodes. In steps 4 and 5, the local of memory accesses differ and depends upon the current working node v, but the memory accesses are indistinguishable because the node $[v]$ is selected uniformly at random and the row $[M_v]$ is never accessed twice during the execution of the algorithm. Similar to steps 4 and 5, step 6 also accesses different locations but each node v is accessed in the reverse order in which they were opened. If the

distribution of memory accesses in step 4 and 5 is indistinguishable with memory access of G' then so is step 6. The location of memory access of step 8 are the same for any graph of size n as it accesses $[P]$ for each node in the graph. Therefore, all the steps in the algorithm either access the same memory location or the distribution of the location of memory access are indistinguishable. Hence the algorithm is secure with respect to Definition 4.1.1.

4.5 Relaxed Data-Obliviousness

There exist problems for which a secure algorithm that satisfies the Definition 4.1.1 is inefficient. One such problem is frequent itemset mining. With the massive amount of data available, it is desirable for a client to, for example, encrypt its data on a cloud server to reduce privacy risks. However, this introduces new challenges if the client wants to mine frequent itemset from the encrypted data as the access patterns along with the execution path could leak sensitive information about the data to the server.

A simple but inefficient way to fix this problem is to let the server run a secure algorithm on an encrypted dataset to compute the frequency of all the itemsets and return the encrypted results to client. Then, the client can filter the false positives to obtain frequent itemsets. This algorithm satisfies the Definition 4.1.1 but it is impractical as the runtime is exponential.

Therefore, we introduce a new relaxed notion of data-obliviousness based on differential privacy.

Definition 4.5.1 (ϵ-Data-Obliviousness) *Let d denote the input to an algorithm. Also, let $A(d)$ denote the sequence of memory access that the algorithm makes. The algorithm is considered ϵ-data-oblivious if for two inputs d and d' that differ by a single element, the total number of instructions executed and access patters of $A(d)$ and $A(d')$ are ϵ-indistinguishable to each party carrying out the computation.*

The above definition relaxes the Definition 4.1.1 in two ways. 1) The indistinguishability holds only against any two neighboring inputs and not against all possible inputs of the same size. 2) An adversary seeing $A(d)$ and $A(d')$ cannot distinguish the two inputs d, d' except with some small probability specified by the privacy parameter ϵ.

4.5.1 ϵ-Data-Oblivious Frequent Itemset Mining

The goal of frequent itemset mining [51] is to find sets of items that are frequently occurring in a dataset (specified by a threshold ϕ). It has applications in marketing, placing frequently items together, etc. [7] used frequent itemset mining to determine the top categories in a text. A number of cryptographic approaches have been proposed to find frequent itemsets from a private dataset like [52–55]. However, these algorithms are not data-oblivious, therefore a data-dependent execution can leak information. I.e. the parties executing the algorithm on an encrypted dataset can learn additional information (e.g., a boundary separating the frequent and infrequent itemsets). With sufficient background information this can lead to privacy breaches. We now propose an efficient data-oblivious frequent itemset mining algorithm that satisfies our new relaxed notion of data-obliviousness, ϵ-data-obliviousness.

Table 4.6.: Notations

$[x]$	Secret shared/Encrypted x
D	Database
$\|D\|$	Size of D
D_i	i^{th} document/tuple in database D
L^k	Set of itemsets at iteration k
L_i^k	i^{th} itemset in L^k
$L_i^k.bv$	Bit vector field of L_i^k
$L_i^k.itemset$	Itemset of L_i^k

Algorithm 5 ϵ-data-oblivious frequent itemset mining

Input: A dataset $[D] \in \mathcal{D}$, items $[\mathcal{I}]$ represented as an indicator vector, privacy budget ϵ, ϕ - threshold , d - maximum size of the itemset.
Output: Frequent itemets.

$\epsilon_1 = \frac{\epsilon}{2 \times d}$
for $z = 1$ to $|[\mathcal{I}]|$ **do**
    $[L_z^1.itemset] = [\mathcal{I}_z]$
    $[L_z^1.bv] = [0]^{|D|}$
end for
for $x = 1$ to $|D|$ **do**
    for $y = 1$ to $|D_x|$ **do**
        for $z = 1$ to $|[L^1]|$ **do**
            $[c] = [D_{x,y}] \stackrel{?}{=} [L_z^1.itemset]$
            $[L_z^1.bv[x]] = \text{IFC}([c], [1], [L_z^1.bv[x]])$
        end for
    end for
end for
$count = \text{DPCOUNT}([L^1], \epsilon_1))$
$[\widehat{L}^1] = \text{DPSAMPLE}([L^1], count, \epsilon_1)$
for $k = 2$ to d **do**
    $[L^k] = \text{CANDGEN}([\widehat{L}^{k-1}])$
    $count = \text{DPCOUNT}([L^k], \epsilon_1))$
    $[\widehat{L}^k] = \text{DPSAMPLE}([L^k], count, \epsilon_1)$
end for

Given a private dataset $[D]$, privacy budget ϵ and a threshold ϕ, the algorithm to compute the frequent itemsets using ϵ-data-oblivious frequent itemset mining algorithm is shown in Algorithm 5. Some of the notations used in this section are shown in Table 4.6.

A secret shared itemset L_i^k is represented by three fields: 1) An inverted index represented by an indicator vector (bv) of size $|D|$. A 0/1 in the i^{th} bit of bv indicates the absence/presence of an item in the i^{th} document respectively. 2) Similarly, an itemset is represented by a indicator vector with 1 denoting the presence of an item in the itemset. 3) An integer *count* to represent the total number of occurrences of an itemset in a corpus. All three fields are secret shared and are not known to the server/parties executing the algorithm.

Algorithm 6 Candidate generation

Input: $[L^k]$ itemsets with corresponding frequencies and bit vector.
Output: Candidate $[L^{k+1}]$ itemsets.

1: **function** CANDGEN()
2: $[L_{k+1}] = \emptyset$
3: **for** $i = 1$ to $|L^k|$ **do**
4: **for** $j = i + 1$ to $|L^k|$ **do**
5: $[c]$ = ISNEIGHBOR($[L_i^k]$,$[L_j^k]$)
6: **if** $c == 1$ **then**
7: $T.bv = [L_i^k.bv] \odot [L_j^k.bv]$
8: $T.itemset = [L_i^k.itemset] \vee [L_j^k.itemset]$
9: $[L^{k+1}] = [L^{k+1}] \bigcup T$
10: **end if**
11: **end for**
12: **end for**
13: return $[L^{k+1}]$
14: **end function**
15: **function** ISNEIGHBOR($[L_i]$,$[L_j]$)
16: $count = 0$
17: **for** $i = 1$ to $|L_i.itemset|$ **do**
18: $[c] = [L_i.itemset(i)] \oplus [L_i.itemset(i)]$
19: $[count] = [count] + [c]$
20: **end for**
21: $[c] = [count] \stackrel{?}{=} [2]$
22: Open and return $[c]$
23: **end function**

Steps 2-5 of Algorithm 5 initializes the *itemset* indicator vector of 1-itemsets of the dataset and its corresponding inverted index to 0. Steps 6-13 parses the private dataset and obliviously computes the inverted index of 1-itemsets. The sub functions used in the algorithm are *DpCount*, *DpSample*, *IsNeighbor* and *CandGen*. Given a set of k-itemsets, a threshold ϕ and privacy ϵ parameter, *DpCount* gives a differentially private count of the number of k-itemsets that are above the threshold ϕ. Given an L^k along with the field $[L_i^k.count]$, *DpSample* uses the algorithm given in [21] to securely sample *count* elements from $[L^k]$ based on the scoring function $[L_i^k.count]$. The function *CandGen* (Algorithm 6) takes as input a set of differentially private k-itemsets and generates $(k + 1)$-itemset candidates. It iterates over the k-itemsets

and considers every pair possible pair for candidate generation. For each pair, it first checks if the itemsets are neighbors. If so, a new $(k+1)$-itemset is generated with its *itemset* field set to the inclusive-or ($\vee$) of the *itemset* field of the neighbors. Then, the new itemset's *bv* field is set to the intersection of the inverted index field *bv*.

The function $isNeighbor$ is used to obliviously check if two secret shared itemsets are neighbors. It basically finds the *xor* of the indicator vector *itemset* to check if two itemsets differ by two elements. The result is returned as plain text. One can argue that it leaks additional information: if two k-itemsets are neighbors. However, since the processing is done on an ϵ differentially private output, the post processing still satisfies differential privacy.

Algorithm 7 Differentially private count

1: **function** DpCount(L^k, ϕ, ϵ)
2: $[res] = 0$
3: **for** $i = 1$ to $|L^k|$ **do**
4: $[L_i^k.count] = 0$
5: **for** $j = 1$ to $|D|$ **do**
6: $[L_i^k.count] = [L_i^k.count] + [L_i^k.bv(j)]$
7: **end for**
8: **end for**
9: **for** $i = 1$ to $|L^k|$ **do**
10: $[c] = [L_i^k.count] \overset{?}{>} \phi$
11: $[res] = \text{IFC}([c], [res] + 1, [res])$
12: **end for**
13: $[res] = [res] + [x \sim Lap(0, \frac{1}{\epsilon})]$
14: open $[res]$ and return it.
15: **end function**

The function *DpCount* takes as input $[L^k]$ (the k-itemsets along with its fields *bv* and *itemset*), threshold count ϕ and privacy budget ϵ. Steps 3-8 of Algorithm 7 iterates each k-itemset in $[L^k]$ and computes the count of each itemset by summing the inverted index vector. Then steps 9-12 compute the number of k-itemsets whose count is above the threshold and then releases a noisy result after adding a noise

sampled from $Lap(0, \frac{1}{\epsilon})$. [21] gives a secure algorithm to sample a random variate from a Laplace distribution with a given μ and λ for the multi-party setting.

The itemsets that were sampled during each iteration are sent to the client for decryption. It is possible that some of the results returned by server may be infrequent but the client can discard them after decryption. The execution is randomized, therefore the server does not learn significant additional information. Note that no noise is added to the frequencies. False negatives can be introduced but the algorithm does not leak information because of the memory accesses except with some probability specified by the privacy parameter ϵ.

4.5.2 Security

Theorem 4.5.1 *The frequent itemset mining algorithm is secure with respect to Definition 4.5.1.*

As before, we will analyze each step of the algorithm to prove that it is secure. Let d and d' be two neighboring datasets of size n that differ by a single item. Steps 2-12 of Algorithm 5 executes the same set of instructions for any two neighboring datasets d and d' because all conditional statements have been serialized and all control loops are conditioned on either the number of items in the domain or size of the dataset. To prove that the number of instructions executed by the Algorithm 5 is ϵ-indistinguishable, we show that the sub-functions functions $DpCount$, $DpSample$, $IsNeighbor$ and $CandGen$ invoked by Algorithm 5 are ϵ-indistinguishable. The function $IsNeighbor$ always executes the same number of instructions for datasets d and d' as the control loop is dependent upon the number of items in the universe. In case of $DpCount$, $DpSample$ and $CandGen$, the number of instructions that they execute is dependent upon size of L^k, which may not be fixed for d and d'. The number of k-itemsets to be selected in the k^{th} iteration is based on $DpCount$. Since, $DpCount$ returns a differentially private count of the number of k-itemsets that have a frequency above the threshold ϕ using the Laplace mechanism, the values $DpCount(d)$

and $DpCount(d')$ are ϵ-indistinguishable. Therefore, we can conclude that the number of instructions executed by the Algorithm 5 itself is ϵ-indistinguishable.

Steps 1-12 of Algorithm 5 access the same memory location for datasets d and d'. To show that the location of memory access are ϵ-indistinguishable, we prove that the location of memory access in each of the functions $DpCount$, $DpSample$, $IsNeighbor$ and $CandGen$ are ϵ-indistinguishable. At iteration k, the function loops over each itemset in k-itemsets and accesses the fields of k-itemset to perform some computation. In each iteration, the functions $DpCount$ and $DpSample$ are used for selecting differentially private k-itemsets, the memory locations returned by $DpSample(d)$ and $DpSample(d')$ are ϵ-indistinguishable. Therefore, all the steps in the algorithm either access the same memory location or the distribution of the location of memory access are ϵ-indistinguishable. Hence the algorithm is secure with respect to Definition 4.5.1.

5 PRIVACY-PRESERVING CLASSIFICATION

A particular issue with text analysis is that high dimensionality poses high costs: computational for cryptographic techniques, and in terms of added noise for randomization mechanisms such as ϵ-differential privacy. One way to address these costs is through feature selection, moving from a high-dimensional feature space to only a few critical features. Unfortunately, the process of feature selection has the potential to reveal private information.

Differential privacy addresses this, by reducing the confidence in any information released. For example, the selection (or non-selection) of a feature under differential privacy could be a result of random chance; preventing making strong inferences about individuals based on the inclusion of the individual's data in the dataset. We thus propose differentially private feature selection.

The main contributions in this section are as follows

1. We derive the global sensitivity of various feature selection techniques and show that some of them are very sensitive to small changes in a corpus, and is therefore not suitable for differential private feature selection. We also propose modifications to existing techniques and show that they are less sensitive to individual changes to a database and can perform better in a differentially private setting.

2. We provide empirical evaluation to show that for techniques with low sensitivity, feature selection can be effective while satisfying differential privacy. We also evaluate the differentially private feature selection in practical setting by building differentially private classifiers: naïve Bayes, support vector machine and decision trees.

In this section, we assume that the data owner has access to the sensitive database and can run these differentially private algorithms to get the private classifiers. The

feature selection techniques can also be used to select split points in decision tree learning. In Section 5.4, we extend the results of [56] showing that using low sensitivity feature selection improves decision tree accuracy.

5.1 Related Work

Differentially private decision tree algorithms have been proposed in [56, 57]. [56] proposes an interactive algorithm, in which the data miner poses queries to a private database for building a differentially private decision tree. Section 5.4 uses their algorithm for evaluating different privacy-preserving feature selection techniques. [56] proposes to build a decision tree in the non-interactive setting by releasing a generalized dataset that satisfies differential privacy. The noisy dataset is then used for building the decision tree.

[58] shows that the global sensitivity of χ^2 statistic is asymptotically constant when the marginal totals are equally distributed. In this chapter, we do not make any assumptions on the distribution of the dataset and show that sensitivity grows as a function of the size of the dataset.

Algorithms for building a differentially private naïve Bayes (DP-NB) classifier are shown in [59–61]. [59] builds a DP-NB classifier to infer private attributes accurately from data containing categorical attributes. [60] extends the classifier to include numerical attributes. [61] develops protocols for building a DP-NB classifier over horizontally and vertically distributed data. However, these algorithms work on micro-data, which does not need differentially private feature selection. Section 5.3.1 shows a differentially private naïve Bayes classifier for unstructured data with private feature selection.

5.2 Differentially Private Feature Selection

We now derive the upper bounds on the sensitivity for several well-known feature selection techniques, which will then be used with the exponential or Laplace mech-

anism for selecting features privately. Note that global sensitivity is based on the impact *any possible feature* could have on the result, not just those features present in the dataset D.

Let $D \in \mathcal{D}$ be a corpus containing N documents (each document d_i is represented by a list of features). Let w_i be a feature in $\mathcal{D}$ and c_j be a category present in D, then N_{w_i,c_j} denote the number of documents of category c_j that contains the feature w_i, $N_{\overline{w_i},c_j}$ denote the number of documents that belong to category c_j but does not contain the feature w_i, $N_{w_i,\overline{c_j}}$ denote the number of documents that contains the feature w_i but does not belong to category c_j, and $N_{\overline{w_i},\overline{c_j}}$ denote the number of documents that neither has the feature w_i nor belong to category c_j. Also, let N_{c_j} denote the number of documents that belongs to class c_j and $N_{\overline{c_j}}$ denote the number of documents that belongs to a class other than c_j. Similarly, we can define for N_{w_i} and $N_{\overline{w_i}}$. Table 5.1 shows a 2×2 contingency table built from a database D for feature w_i. We will use this table in the rest of the section for analyzing the global sensitivity of the feature selection methods for binary classification task.

Table 5.1.: 2×2 contingency table

	c_j	$\overline{c_j}$	
w_i	N_{w_i,c_j}	$N_{w_i,\overline{c_j}}$	N_{w_i}
$\overline{w_i}$	$N_{\overline{w_i},c_j}$	$N_{\overline{w_i},\overline{c_j}}$	$N_{\overline{w_i}}$
	N_{c_j}	$N_{\overline{c_j}}$	N

5.2.1 Term Weights

A simple technique for feature selection is to select the features based on their weight (e.g., inverse document frequency). IDF measures the importance of a feature,

i.e., whether the feature is common or rare across all documents in D. A smoothed IDF of w_i is computed by

$$IDF(D, w_i) = \log \frac{N}{1 + N_{w_i}}$$

Without loss of generality, let us assume that the count of N_{w_i} increases by 1 in the neighboring dataset D'. Then, the global sensitivity of the IDF weighting is $\log \frac{N+1}{N}$.

$$\begin{aligned} GS_{IDF} = & \log \frac{N+1}{2 + N_{w_i}} - \log \frac{N}{1 + N_{w_i}} \\ & \leq \log \frac{N+1}{N} \end{aligned}$$

5.2.2 Chi-Squared Statistic

The χ^2 statistic measures the lack of independence between a feature w_i and a category c_j, which is compared with the χ^2 distribution to measure the extremeness. For Table 5.1, the statistic is computed as follows.

$$\chi^2(D, w_i, c_j) = \frac{N(N_{w_i,c_j} N_{\overline{w_i},\overline{c_j}} - N_{w_i,\overline{c_j}} N_{\overline{w_i},c_j})^2}{(N_{w_i})(N_{\overline{w_i}})(N_{c_j})(N_{\overline{c_j}})}$$

The range of the chi-squared statistic is between 0 and $N(\ell - 1)$, where N is the number of observations and ℓ is the minimum of the number of rows and columns in the contingency table. A naïve analysis shows that the global sensitivity of the χ^2 statistic for a 2×2 contingency table is not more than $N + 1$. While we do not show that this bound is tight, we can show that the noise needed to satisfy differential privacy is at least $\frac{N}{2}$.

Table 5.2.: Contingency tables that differ by 1

(a) A

	c_j	$\overline{c_j}$
w_i	$N-1$	0
$\overline{w_i}$	0	1

(b) B

	c_j	$\overline{c_j}$
w_i	$N-1$	0
$\overline{w_i}$	1	1

To prove the lower bound for the global sensitivity is at least $\frac{N}{2} + \frac{1}{2N}$, we will derive the local sensitivity using the two neighboring databases given in Table 5.2a and 5.2b. Therefore,

$$\begin{aligned} LS_{\chi^2}(A) &= \max_D ||\chi^2(A) - \chi^2(D')||_1 \\ &\geq |\chi^2(A) - \chi^2(B)| \\ &= \frac{N(N-1)^2}{(N-1)^2} - \frac{(N+1)(N-1)^2}{2N(N-1)} \\ &= \frac{2N^2 - (N^2-1)}{2N} = \frac{N^2+1}{2N} \\ &= \frac{N}{2} + \frac{1}{2N} \end{aligned}$$

Therefore, the global sensitivity is at least $\frac{N}{2} + \frac{1}{2N}$. While one could argue that the datasets shown in Tables 5.2a and 5.2b are unlikely, differential privacy considers all possible neighboring datasets present in the universe and adds noise proportional to the maximum change in query value. Techniques such as [35], which satisfies a weaker security notion, can be used to provide better utility as they add noise proportional to the dataset that is being published. In the rest of section, we consider the global sensitivity of the feature selection techniques.

Table 5.3.: Multi-class contingency tables that differ by 1

(a) A for $k > 2$

	c_1	c_2	c_3	...	c_k
w_i	$N-k+1$	0	0	...	0
$\overline{w_i}$	0	1	1	...	1

(b) B neighbor of A

	c_1	c_2	c_3	...	c_k
w_i	$N-k+1$	0	0	...	0
$\overline{w_i}$	1	1	1	...	1

Table 5.4.: Category specific contingency tables

(a) A_{c_1}

	c_1	$\overline{c_1}$
w_i	$N-k+1$	0
$\overline{w_i}$	0	$k-1$

(b) B_{c_1}

	c_1	$\overline{c_1}$
w_i	$N-k+1$	0
$\overline{w_i}$	1	$k-1$

(c) A_{c_j}

	c_j	$\overline{c_j}$
w_i	0	$N-k+1$
$\overline{w_i}$	1	$k-2$

(d) B_{c_j}

	c_j	$\overline{c_j}$
w_i	0	$N-k+1$
$\overline{w_i}$	1	$k-1$

Similarly, we derive the lower bound for global sensitivity of χ^2 for $k > 2$ (multi-class). For $k > 2$, we can compute the χ^2 statistic between the term w_i and each category c_j and then combine the category-specific scores as follows.

$$\chi^2(D, w_i) = \sum_j \frac{N_{c_j}}{N} \chi^2(D, w_i, c_j)$$

$$N \times \chi^2(D, w_i) = \sum_j N_{c_j} \chi^2(D, w_i, c_j)$$

We show that global sensitivity of χ^2 is at least $\frac{N-k+1}{k}$ for $k > 2$. We use the multi-class contingency tables given in 5.3a and 5.3b for deriving the lower bound on global sensitivity. The category specific contingency tables of 5.3a and 5.3b are shown in Tables 5.4(a-d). A_{c_j} and B_{c_j} denote the class-specific (i.e., class c_j) contingency tables for the databases A and B respectively.

$$
\begin{aligned}
LS_{\chi^2}(B, w_i) \geq\ & \chi^2(B, w_i) - \chi^2(A, w_i) \\
=\ & \sum_{c_j \in B} N_{c_j} \chi^2(B, w_i, c_j) - \sum_{c_j \in A} N_{c_j} \chi^2(A, w_i, c_j) \\
=\ & \frac{(N-k+1)}{N} \frac{(N)(N-k+1)^2(k-1)^2}{(N-k+1)(k-1)^2(N-k+1)} \\
& + \sum_{j=2}^{k} \frac{1}{N} \frac{(N)(N-k+1)^2}{(N-k+1)(k-1)(N-1)} \\
& - \frac{(N-k+2)}{N+1} \frac{(N+1)(N-k+1)^2(k-1)^2}{(N-k+2)(k)(k-1)(N-k+1)} \\
& - \sum_{j=2}^{k} \frac{1}{N+1} \frac{(N+1)(N-k+1)^2}{(N-k+1)(k)(N)} \\
=\ & (N-k+1) + \sum_{j=2}^{k} \frac{(N-k+1)}{(k-1)(N-1)} \\
& - \frac{(N-k+1)(k-1)}{k} - \sum_{j=2}^{k} \frac{(N-k+1)}{(k)(N)} \\
=\ & \frac{(k)(N-k+1) - (k-1)(N-k+1)}{k} \\
& + \sum_{j=2}^{k} \frac{(N-k+1)}{(k-1)(N-1)} - \sum_{j=2}^{k} \frac{(N-k+1)}{(k)(N)} \\
=\ & \frac{(N-k+1)}{k} + \frac{(N-k+1)}{N-1} - \frac{(k-1)(N-k+1)}{(k)(N)} \\
>\ & \frac{(N-k+1)}{k}
\end{aligned}
$$

The last inequality holds because $\frac{(N-k+1)}{(N-1)} > \frac{(k-1)(N-k+1)}{(k)(N)}$.

5.2.3 Odds Ratio

Odds Ratio quantifies how strongly the presence or absence of a feature w_i is associated with the presence or absence of a category c_j in the corpus. For a 2×2 contingency table, OR is estimated as follows.

$$OR(D, w_i, c_j) = \frac{Pr(w_i|c_j)Pr(\overline{w}_i|\overline{c}_j)}{Pr(\overline{w}_i|c_j)Pr(w_i|\overline{c}_j)} = \frac{N_{w_i,c_j}N_{\overline{w_i},\overline{c_j}}}{N_{w_i,\overline{c_j}}N_{\overline{w_i},c_j}}$$

Table 5.5.: Smoothed contingency tables that differ by 1

(a) A

	c_j	$\overline{c_j}$
w_i	$N - \frac{1}{2}$	$\frac{1}{2}$
$\overline{w_i}$	$\frac{1}{2}$	$\frac{3}{2}$

(b) B

	c_j	$\overline{c_j}$
w_i	$N - \frac{1}{2}$	$\frac{1}{2}$
$\overline{w_i}$	$\frac{3}{2}$	$\frac{3}{2}$

The global sensitivity of odds ratio is unbounded. The difference between the odds ratio of a database D with a zero $N_{w_i,\overline{c_j}}$ value and a neighboring database D' with non-zero $N_{w_i,\overline{c_j}}$ value is infinite. In case of a smoothed OR (0.5 added to each cell), we can show that the global sensitivity is at least $4N-2$ for a 2×2 contingency table. The smoothed contingency tables of Table 5.2a and 5.2b are shown in Table 5.5a and Table 5.5b. From smoothed contingency tables, we have

$$\begin{aligned} LS_{OR}(B, w_i, c_j) \geq\ & OR(B, w_i, c_j) - OR(A, w_i, c_j) \\ =\ & \frac{\frac{3}{2}(N - \frac{1}{2})}{\frac{1}{4}} - \frac{\frac{3}{2}(N - \frac{1}{2})}{\frac{3}{4}} \\ =\ & 6N - 3 - 2N + 1 \\ =\ & 4N - 2 \end{aligned}$$

Table 5.6.: Smoothed multi-class contingency tables that differ by 1

(a) A for $k > 2$

	c_1	c_2	c_3	$\ldots$	c_k
w_i	$N - k + \frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\ldots$	$\frac{1}{2}$
$\overline{w_i}$	$\frac{1}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\ldots$	$\frac{3}{2}$

(b) B neighbor of A

	c_1	c_2	c_3	$\ldots$	c_k
w_i	$N - k + \frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\ldots$	$\frac{1}{2}$
$\overline{w_i}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\ldots$	$\frac{3}{2}$

Table 5.7.: Smoothed category specific contingency tables

(a) A_{c_1}

	c_1	$\overline{c_1}$
w_i	$N - k + \frac{3}{2}$	$\frac{k-1}{2}$
$\overline{w_i}$	$\frac{1}{2}$	$\frac{3(k-1)}{2}$

(b) B_{c_1}

	c_1	$\overline{c_1}$
w_i	$N - k + \frac{3}{2}$	$\frac{k-1}{2}$
$\overline{w_i}$	$\frac{3}{2}$	$\frac{3(k-1)}{2}$

(c) A_{c_j}

	c_j	$\overline{c_j}$
w_i	$\frac{1}{2}$	$N - k + \frac{3}{2} + \frac{k-2}{2}$
$\overline{w_i}$	$\frac{3}{2}$	$\frac{3(k-2)}{2} + \frac{1}{2}$

(d) B_{c_j}

	c_j	$\overline{c_j}$
w_i	$\frac{1}{2}$	$N - k + \frac{3}{2} + \frac{k-2}{2}$
$\overline{w_i}$	$\frac{3}{2}$	$\frac{3(k-1)}{2}$

Similarly, we derive the lower bound for global sensitivity of OR for $k > 2$. For $k > 2$, we can compute the OR between the term w_i and each category c_j and then combine the category-specific scores as follows.

$$OR(D, w_i) = \sum_j \frac{N_{c_j}}{N} OR(D, w_i, c_j)$$

$$N \times OR(D, w_i) = \sum_j N_{c_j} OR(D, w_i, c_j)$$

The smoothed contingency tables of Table 5.3a and 5.3b are shown in Table 5.6a and Table 5.6b. The category specific contingency tables are shown in Tables 5.7(a-d). From smoothed contingency tables, we have

$$\begin{aligned}
LS_{OR}(B, w_i) \geq & \sum_j N_{c_j} OR(B, w_i, c_j) - \sum_j N_{c_j} OR(A, w_i, c_j) \\
= & (n-k+2)\frac{(n-k+\frac{3}{2})(\frac{3(k-1)}{2})}{\frac{k-1}{4}} - (n-k+3)\frac{(n-k+\frac{3}{2})(\frac{3(k-1)}{2})}{\frac{3(k-1)}{4}} \\
& + \sum_{j=2} 2 \quad \frac{\frac{1}{2}(\frac{3(k-2)}{2}+\frac{1}{2})}{\frac{3}{2}(n-k+\frac{3}{2}+\frac{(k-2)}{2})} - \frac{\frac{1}{2}(\frac{3(k-1)}{2})}{\frac{3}{2}(n-k+\frac{3}{2}+\frac{(k-2)}{2})} \\
= & (n-k+2)(n-k+\frac{3}{2})(6) - (n-k+3)(n-k+\frac{3}{2})(2) \\
& + \sum_{j=2} 2 \quad \frac{(\frac{3(k-2)}{2}+\frac{1}{2})}{3(n-k+\frac{3}{2}+\frac{(k-2)}{2})} - \frac{\frac{(k-1)}{2}}{(n-k+\frac{3}{2}+\frac{(k-2)}{2})} \\
= & 4n^2 + 12n - 8nk + 4k^2 - 12k + 9 + \sum_{j=2} -\frac{4}{3\,(2n-k+1)} \\
= & 4n^2 + 12n - 8nk + 4k^2 - 12k + 9 - \frac{4(k-1)}{3\,(2n-k+1)}
\end{aligned}$$

As n increases, the local sensitivity is dominated by the $4n^2$ term.

5.2.4 GSS Coefficient

Galavotti et al. [62] proposed a simplified χ^2 statistic that removed factors that emphasized rare words and rare categories. For a binary classification task, it is computed as follows

$$\begin{aligned}
GSS(D, w_i, c_j) &= Pr(w_i, c_j)Pr(\overline{w}_i, \overline{c}_i) - Pr(w_i, \overline{c}_i)Pr(\overline{w}_i, c_j) \\
&= \frac{N_{w_i,c_j} N_{\overline{w_i},\overline{c_j}} - N_{\overline{w_i},c_j} N_{w_i,\overline{c_j}}}{N^2}
\end{aligned}$$

A positive/negative value of GSS coefficient denote positive/negative relationship and zero means no relationship between the feature w_i and class c_j. Maximizing the above equation is equivalent to maximizing $N^2 \times GSS(D, w_i, c_j)$. We propose to use the absolute value of the numerator to compute GSS so that the range of GSS is in the interval $[0, \frac{N^2}{4}]$, which captures both positive and negative relationship.

Therefore, we will be maximizing the equation $|N_{w_i,c_j}N_{\overline{w_i},\overline{c_j}} - N_{\overline{w_i},c_j}N_{w_i,\overline{c_j}}|$. The global sensitivity of this equation is N for a 2×2 contingency table (Table 5.1). To prove the global sensitivity, we will assume that the neighboring database differs in the element N_{w_i,c_j} by 1.

$$\begin{aligned}
GS_{GSS(w_i,c_j)} = & \ |(N_{w_i,c_j}+1)N_{\overline{w_i},\overline{c_j}} - N_{w_i,\overline{c_j}}N_{\overline{w_i},c_j}| \\
& - |N_{w_i,c_j}N_{\overline{w_i},\overline{c_j}} - N_{w_i,\overline{c_j}}N_{\overline{w_i},c_j}| \\
\leq & \ N_{w_i,c_j}N_{\overline{w_i},\overline{c_j}} + N_{\overline{w_i},\overline{c_j}} - N_{w_i,\overline{c_j}}N_{\overline{w_i},c_j} \\
& - N_{w_i,c_j}N_{w_i,\overline{c_j}} + N_{w_i,\overline{c_j}}N_{\overline{w_i},c_j} \\
= & \ N_{\overline{w_i},\overline{c_j}} \\
\leq & \ N
\end{aligned}$$

For $k > 2$, GSS coefficient can be computed by first finding the GSS coefficient for each category c_j and then computing the average of category-specific scores.

$$GSS(D, w_i) = \sum_j \frac{N_{c_j}}{N} GSS(D, w_i, c_j)$$

Maximizing the above equation is equivalent to maximizing $GSS(D, w_i) = N^3 \times GSS(D, w_i)$. The following analysis is based on the (general) multi-class contingency Table 5.8. Without loss of generality, we will assume that the neighboring database

D' differ in the element N_{w_i,c_1} (i.e., D' has count $N_{w_i,c_1}+1$). The category specific contingency tables of neighboring databases D and D' are shown in Tables 5.9(a-d).

Table 5.8.: Multi-class contingency table (D)

	c_1	c_2	c_3	...	c_k
w_i	N_{w_i,c_1}	N_{w_i,c_2}	N_{w_i,c_3}	...	N_{w_i,c_k}
$\overline{w_i}$	$N_{\overline{w_i},c_1}$	$N_{\overline{w_i},c_2}$	$N_{\overline{w_i},c_3}$	...	$N_{\overline{w_i},c_k}$

Table 5.9.: Category specific contingency tables

(a) D_{c_1}

	c_1	$\overline{c_1}$
w_i	N_{w_i,c_1}	$N_{w_i}-N_{w_i,c_1}$
$\overline{w_i}$	$N_{\overline{w_i},c_1}$	$N_{\overline{w_i}}-N_{\overline{w_i},c_1}$

(b) D'_{c_1}

	c_1	$\overline{c_1}$
w_i	$N_{w_i,c_1}+1$	$N_{w_i}-N_{w_i,c_1}$
$\overline{w_i}$	$N_{\overline{w_i},c_1}$	$N_{\overline{w_i}}-N_{\overline{w_i},c_1}$

(c) D_{c_j}

	c_j	$\overline{c_j}$
w_i	N_{w_i,c_j}	$N_{w_i}-N_{w_i,c_j}$
$\overline{w_i}$	$N_{\overline{w_i},c_j}$	$N_{\overline{w_i}}-N_{\overline{w_i},c_j}$

(d) D'_{c_j}

	c_j	$\overline{c_j}$
w_i	N_{w_i,c_j}	$N_{w_i}-N_{w_i,c_j}+1$
$\overline{w_i}$	$N_{\overline{w_i},c_j}$	$N_{\overline{w_i}}-N_{\overline{w_i},c_j}$

$$GS_{GSS(w_i)} = |GSS(D', w_i) - GSS(D, w_i)|$$

$$= \sum_{c_j \in D} N_{c_j} GSS(D, w_i, c_j) - \sum_{c_j \in D} N_{c_j} GSS(D', w_i, c_j)$$

$$
\begin{aligned}
&= N_{c_1}\ N_{w_i,c_1}(N_{\overline{w_i}} - N_{\overline{w_i},c_1}) - N_{\overline{w_i},c_1}(N_{w_i} - N_{w_i,c_1}) \\
&\quad + \sum_{j=2}^{k} N_{c_j}\ N_{w_i,c_j}(N_{\overline{w_i}} - N_{\overline{w_i},c_j}) - N_{\overline{w_i},c_j}(N_{w_i} - N_{w_i,c_j}) \\
&\quad - \ (N_{c_1}+1)\ (N_{w_i,c_1}+1)(N_{\overline{w_i}} - N_{\overline{w_i},c_1}) - N_{\overline{w_i},c_1}(N_{w_i} - N_{w_i,c_1}) \\
&\quad\quad + \sum_{j=2}^{k} N_{c_j}\ N_{w_i,c_j}(N_{\overline{w_i}} - N_{\overline{w_i},c_j}) - N_{\overline{w_i},c_j}(N_{w_i} - N_{w_i,c_j}+1)
\end{aligned}
$$

Let, $V = N_{c_1}$ $\quad P_j = N_{c_j}$

$W = N_{w_i,c_1}$ $\quad Q_j = N_{w_i,c_j}$

$X = N_{w_i} - N_{w_i,c_1}$ $\quad R_j = N_{w_i} - N_{w_i,c_j}$

$Y = N_{\overline{w_i},c_1}$ $\quad S_j = N_{\overline{w_i},c_j}$

$Z = N_{\overline{w_i}} - N_{\overline{w_i},c_1}$ $\quad T_j = N_{\overline{w_i}} - N_{\overline{w_i},c_j}$

$$
\begin{aligned}
&\leq\ V\ \ (W)(Z) - (Y)(X) - (V+1)\ \ (W+1)(Z) - (Y)(X) \\
&\quad + \sum_{j=2}^{k}\ P_j\ \ (Q_j)(T_j) - (S_j)(R_j) - (Q_j)(T_j) + (S_j)(R_j+1) \\
&= \quad -(V)(Z) - (W)(Z) - Z + (Y)(X)\ \ + \sum_{j=2}^{k}\ (P_j)(S_j) \\
&\leq \quad -(V)(Z) - (W)(Z) - Z\ \ + \sum_{j=2}^{k}(N_{c_j})N \\
&\leq\ NN_{c_1} + Z(1+W) + N\sum_{j=2}^{k} N_{c_j} \\
&\leq N\sum_{j=1}^{k} N_{c_j} + \frac{N}{2}(1+\frac{N}{2}) \\
&= \frac{5}{4}N^2 + \frac{N}{2}
\end{aligned}
$$

The first inequality uses the fact that $|a| - |b| \leq |a - b|$. The second inequality holds because $S_j \leq N$ and $(Y)(X) \geq 0$. The third inequality uses the fact that

$Z \leq N$. For the last inequality, the expression $Z(1+W)$ subjected to the constraint $W + Z \leq N$ reaches the maximum value at $W = Z = \frac{N}{2}$.

5.2.5 Bray-Curtis Dissimilarity

Bray-Curtis dissimilarity (BCD) is used to quantify the distance between two samples. The sum of the absolute differences between the counts is divided by the sum of the abundances in the two samples to get BCD.

$$BCD(X,Y) = \frac{{}_i|X_i - Y_i|}{{}_iX_i + \quad {}_iY_i}$$

We now show how BCD can be used to measure the independence of a feature w_i and category c_j. Let $O^D_{w_i,c_j}$ denote the observed frequencies in the contingency table built from the corpus D for feature w_i and category c_j; $E^D_{w_i,c_j}$ be the expected frequency for feature w_i and category c_j under the null hypothesis (i.e., w_i and c_j are independent). The dissimilarity becomes zero if and only if w_i and c_j are independent. Higher values indicate that the null hypothesis should be rejected (i.e., the occurrence of w_i and c_j are not independent.) Therefore, we want to select features that maximize the following equation.

$$BCD(D,w_i) = \frac{1}{2N}\sum_{x,y} O^D_{x,y} - E^D_{x,y}$$

where $x \in \{w_i, \overline{w}_i\}$ denotes the presence or absence of a feature w_i and $y \in \{c_1, c_2, \ldots, c_k\}$ for $2 \times k$ contingency table. $E_{x,y}$ is computed as $N \times Pr(x) \times Pr(y)$. For example, $E_{w_i,c_j} = \frac{N_{w_i} N_{c_j}}{N}$. Maximizing the above equation is equivalent to maximizing $\quad {}_{x,y}\ O^D_{x,y} - E^D_{x,y}$.

We will use Table 5.8 to prove that the sensitivity of BCD is $2k$, where k is number of categories in the database D. Without loss of generality, let us assume

that the value of N_{w_i,c_1} increases by 1 in the neighboring database D', then the global sensitivity of BCD can be proved as follows.

$$
\begin{aligned}
GS_{BCD}(w_i) &= |BCD(D', w_i) - BCD(D, w_i)| \\
&= \sum_{x,y} O^D_{x,y} - E^D_{x,y} \; - \sum_{x,y} O^D_{x,y} - E^D_{x,y} \\
&= \quad (N_{w_i,c_1} + 1) - \frac{(N_{w_i}+1)(N_{c_1}+1)}{(N+1)} \\
&\quad + \sum_{j=2}^{k} N_{w_i,c_j} - \frac{(N_{w_i}+1)N_{c_j}}{(N+1)} \\
&\quad + N_{\overline{w_i},c_1} - \frac{N_{\overline{w_i}}(N_{c_1}+1)}{(N+1)} + \sum_{j=2}^{k} N_{\overline{w_i},c_j} - \frac{N_{\overline{w_i}}N_{c_j}}{(N+1)} \Bigg) \\
&\quad - \Bigg(N_{w_i,c_1} - \frac{N_{w_i}N_{c_1}}{N} + \sum_{j=2}^{k} N_{w_i,c_j} - \frac{N_{w_i}N_{c_j}}{N} \\
&\quad + N_{\overline{w_i},c_1} - \frac{N_{c_1}N_{\overline{w_i}}}{N} + \sum_{j=2}^{k} N_{\overline{w_i},c_j} - \frac{N_{\overline{w_i}}N_{c_j}}{N} \Bigg) \\
&= \quad N_{w_i,c_1} + 1 - \frac{(N_{w_i}+1)(N_{c_1}+1)}{(N+1)} - N_{w_i,c_1} - \frac{N_{w_i}N_{c_1}}{N} \\
&\quad + \sum_{j=2}^{k} \quad N_{w_i,c_j} - \frac{(N_{w_i}+1)N_{c_j}}{(N+1)} - N_{w_i,c_j} - \frac{N_{w_i}N_{c_j}}{N} \\
&\quad + N_{\overline{w_i},c_1} - \frac{(N_{\overline{w_i}})(N_{c_1}+1)}{(N+1)} - N_{\overline{w_i},c_1} - \frac{N_{c_1}N_{\overline{w_i}}}{N} \\
&\quad + \sum_{j=2}^{k} \quad N_{\overline{w_i},c_j} - \frac{N_{\overline{w_i}}N_{c_j}}{(N+1)} - N_{\overline{w_i},c_j} - \frac{N_{\overline{w_i}}N_{c_j}}{N} \\
&\leq \; 1 - \frac{(N_{w_i}+1)(N_{c_1}+1)}{(N+1)} + \frac{(N_{w_i})(N_{c_1})}{N} \\
&\quad + \sum_{j=2}^{k} - \frac{(N_{w_i}+1)(N_{c_j})}{(N+1)} + \frac{(N_{w_i})(N_{c_j})}{N} \\
&\quad + - \frac{(N_{c_1}+1)(N_{\overline{w_i}})}{(N+1)} + \frac{(N_{c_1})(N_{\overline{w_i}})}{N} \\
&\quad + \sum_{j=2}^{k} - \frac{(N_{c_j})(N_{\overline{w_i}})}{(N+1)} + \frac{(N_{c_j})(N_{\overline{w_i}})}{N}
\end{aligned}
$$

$$
\begin{aligned}
= & \frac{N(N+1) - N(N_{w_i}+1)(N_{c_1}+1) + (N+1)(N_{w_i}N_{c_1})}{N(N+1)} \\
& + \sum_{j=2}^{k} \frac{-N(N_{w_i}+1)(N_{c_j}) + (N+1)(N_{w_i})(N_{c_j})}{N(N+1)} \\
& + \frac{-(N)(N_{c_1}+1)(N_{\overline{w_i}}) + (N+1)(N_{c_1})(N_{\overline{w_i}})}{N(N+1)} \\
& + \sum_{j=2}^{k} \frac{-N(N_{c_j})(N_{\overline{w_i}}) + (N+1)(N_{c_j})(N_{\overline{w_i}})}{N(N+1)} \\
\leq & \frac{(N - N_{w_i})(N - N_{c_1})}{N(N+1)} + \sum_{j=2}^{k} \frac{(N_{c_j})(N_{w_i} - N)}{N(N+1)} \\
& + \frac{(N_{\overline{w_i}})(N_{c_1} - N)}{N(N+1)} + \sum_{j=2}^{k} \frac{N_{c_j}N_{\overline{w_i}}}{N(N+1)} \\
\leq & \frac{2kN^2}{N(N+1)} \leq 2k
\end{aligned}
$$

The first inequality uses the fact $|a|-|b| \leq |a-b|$ and the second to last inequality holds because each term in the numerator is bounded by N^2.

5.2.6 Information Gain

Information gain [63] measures the number of bits of information obtained for prediction by knowing the presence or absence of a term/feature in a document.

$$
\begin{aligned}
IG(D, w_i) = & -\sum_{j} Pr(c_j) \log Pr(c_j) \\
& + \sum_{j} \sum_{w \in \{w_i, \overline{w}_i\}} Pr(c_j, w') \log \frac{Pr(c_j, w')}{Pr(w')}
\end{aligned}
$$

Maximizing the above equation is equivalent to maximize the following equation.

$$IG'(D, w_i) = -\sum_j \sum_{w \in \{w_i, \overline{w}_i\}} Pr(c_j, w') \log \frac{Pr(c_j, w')}{Pr(w')}$$

$$IG'(D, w_i) = -\sum_j \sum_{w \in \{w_i, \overline{w}_i\}} \frac{N_{w,c_j}}{N} \log \frac{N_{w,c_j}}{N_w}$$

$$N * IG'(D, w_i) = -\sum_j \sum_{w \in \{w_i, \overline{w}_i\}} N_{w,c_j} \log \frac{N_{w,c_j}}{N_w}$$

[56] showed that the global sensitivity of the above equation is equal to $\log(N+1) + \frac{1}{\ln 2}$.

5.2.7 Mutual Information

Mutual Information [63] between a feature w_i and a category c_j is computed as follows.

$$I(w_i, c_j) = \log \frac{Pr(w_i, c_j)}{Pr(w_i)Pr(c_j)}$$

To measure the global goodness of a feature, the category specific scores are combined as:

$$\begin{aligned} MI(D, w_i) &= \sum_j Pr(c_j) I(w_i, c_j) \\ &= \frac{1}{N} \sum_j N_{c_j} I(w_i, c_j) \end{aligned}$$

We will instead maximize $N \times MI(D, w_i)$. It is easy to show that the global sensitivity is unbounded for the above equation. If we compute a smoothed MI (0.5 is added to each cell), then we can show that the global sensitivity is at most $N \log(3) + \log(N+1) + \frac{1}{\ln 2}$. The following proof uses the fact that $x \log \frac{x+1}{x}$ goes to $\frac{1}{\ln 2}$ as x goes to ∞, and 0 when x goes to 0.

The following analysis is based on the Table 5.8. Without loss of generality, we will assume that the neighboring database D' differ in the element N_{w_i,c_1} (i.e., D' has count $N_{w_i,c_1}+1$).

$$
\begin{aligned}
GS_{MI}(w_i) = \; & MI'_{avg}(D,w_i) - MI'_{avg}(D',w_i) \\
= \; & \sum_j N'_{c_j} I'(w_i,c_j) - \sum_j N_{c_j} I(w_i,c_j) \\
= \; & (N_{c_1}+1)\log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} - (N_{c_1})\log\frac{NN_{w_i,c_1}}{(N_{w_i})(N_{c_1})} \\
& + \sum_{j=2} \quad (N_{c_j})\log\frac{(N+1)(N_{w_i,\overline{c_j}})}{(N_{w_i}+1)(N_{c_j})} - (N_{c_j})\log\frac{NN_{w_i,\overline{c_j}}}{(N_{w_i})(N_{c_j})} \\
= \; & N_{c_1}\log\frac{(N+1)(N_{w_i,c_1}+1)(N_{w_i})(N_{c_1})}{(N)(N_{w_i,c_1})(N_{w_i}+1)(N_{c_1}+1)} + \log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} \\
& + \sum_{j=2}(N_{c_j})\log\frac{(N+1)(N_{w_i})}{(N)(N_{w_i}+1)} \\
= \; & N_{c_1}\log\frac{(N_{w_i,c_1}+1)(N_{c_1})}{(N_{w_i,c_1})(N_{c_1}+1)} + \log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} \\
& + \sum_{j=1}(N_{c_j})\log\frac{(N+1)(N_{w_i})}{(N)(N_{w_i}+1)} \\
= \; & N_{c_1}\log\frac{(N_{w_i,c_1}+1)}{(N_{w_i,c_1})} - N_{c_1}\log\frac{(N_{c_1}+1)}{(N_{c_1})} + \log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} \\
& + \sum_{j=1}(N_{c_j})\log\frac{(N+1)}{(N)} - \sum_{j=1}(N_{c_j})\log\frac{(N_{w_i}+1)}{(N_{w_i})} \\
= \; & N_{c_1}\log\frac{(N_{w_i,c_1}+1)}{(N_{w_i,c_1})} - N_{c_1}\log\frac{(N_{c_1}+1)}{(N_{c_1})} + \log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} \\
& + (N)\log\frac{(N+1)}{(N)} - \sum_{j=1}(N_{c_j})\log\frac{(N_{w_i}+1)}{(N_{w_i})} \\
\leq \; & N_{c_1}\log\frac{(N_{w_i,c_1}+1)}{(N_{w_i,c_1})} + \log\frac{(N+1)(N_{w_i,c_1}+1)}{(N_{w_i}+1)(N_{c_1}+1)} + (N)\log\frac{(N+1)}{(N)} \\
\leq \; & N\log 3 + \log(N+1) + \frac{1}{\ln 2}
\end{aligned}
$$

5.3 Feature Selection: Empirical Evaluation

We now present an empirical evaluation to demonstrate the impact of selecting a low sensitive feature selection method when using differential privacy. We present χ^2 as a baseline, as this is a feature selection technique with a solid theoretical grounding (but unfortunately, high sensitivity.) We compare with Bray-Curtis dissimilarity and information gain, as they are the only options with sub-linear sensitivity. GSS is included because its sensitivity (N) is much smaller compared to its range ($[0, \frac{N^2}{4}]$).

We used the 20 newsgroup dataset from the UCI repository [64] for empirical evaluation, which contains approximately 20,000 documents distributed across 20 different newsgroups (some of them are related, e.g., comp.sys.ibm.pc.hardware and comp.graphics). We created a subset D from [64] by grouping all the computer/recreation related documents ($|D| = 5287$) with the label +1/-1 respectively and used the exponential mechanism for privately selecting (unigram) features.

Intuitively, the exponential mechanism scores all the features in $\mathcal{F}$ (feature universe) according to the database D and selects features each with probability proportional to $e^{\frac{\epsilon \times g(D, w_i)}{2 \times m \times \Delta g}}$, where m is the number of features by which the two neighboring databases differ[1] and g is quality function that is used to measure a feature.

In this dissertation, we consider the whole vocabulary of [64] as $\mathcal{F}$. Higher values of m denote stronger privacy as an adversary will not be able to distinguish the presence/absence of m features except with some probability indicated by ϵ. The top 20 features selected by the non-private and differentially private χ^2 statistic are shown in Table 5.10. Similarly, the Tables 5.11 and 5.12 show the top 20 features selected by the non-private and differentially private feature selection techniques BCD and GSS respectively. Differentially private χ^2 performs poorly due to high sensitivity.

For a more complete comparison of the effectiveness of different differentially private feature selection techniques, we compute the overlap of top 100 features selected privately with the top 100 features selected by the non-private χ^2 statistic. Figure 5.1 shows the percentage overlap of the features for different values of m and ϵ. We can

[1]Neighboring databases have the same number of documents but differ by m features

see that BCD and GSS perform well compared to other techniques in the differentially private setting. An interesting result that we see is the performance of information gain in the private setting. While IG has relatively low sensitivity, the scores were all in the range [-5247, -4755]. As a result, the scores were not very discriminatory, and a small amount of noise relative to the overall possible range of scores turns out to be significant, resulting in a significant change in the top 100 features. In spite of the high sensitivity, GSS performed well because the scores were highly discrim-

Table 5.10.: Top 20 features (unigrams) selected using χ^2 statistic

χ^2 statistic (Non-private)	DP χ^2 statistic ($\epsilon = 0.5, m = 1$)
windows , team	prob , stage
car , dod	vonda , formatting
game , year	subjects , queries
season , bike	zoroastrian , assassination
program , players	morandini , inclination
hockey , card	manfredo , inqmind
play , writes	finances , daystar
software , baseball	correlation , citroen
league , graphics	eashtar , oxidizer
dos , cars	versatile , ramdisk

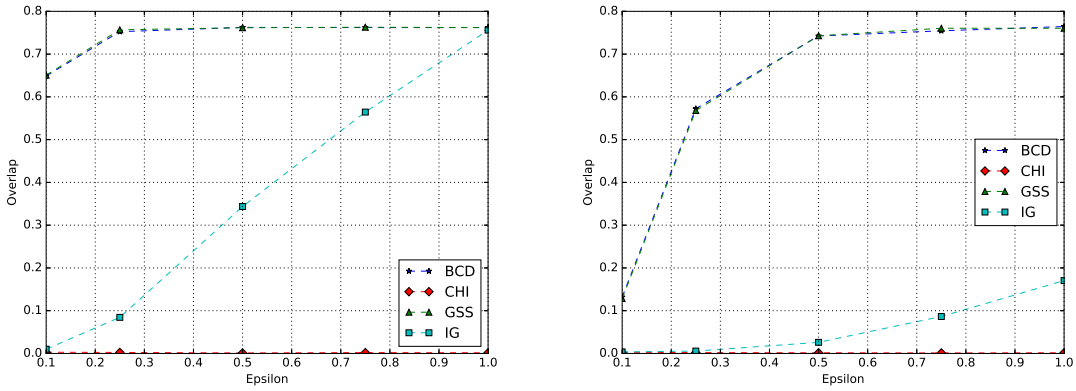
Table 5.11.: Top 20 features (unigrams) selected using BCD

BCD (Non-private)	DP-BCD ($\epsilon = 0.5, m = 1$)	DP-BCD ($\epsilon = 0.5, m = 5$)
writes, article	writes , article	writes , dod
windows, team	windows , team	article , windows
year, car	year , car	team , year
game, dod	game , dod	car , funniest
program, apr	apr , program	wasn , files
good, system	system , problem	computer , ftp
software, problem	software, problem	good , game
card, play	good , file	program , play
file, season	card , play	guzman , psi
bike, graphics	season , graphics	avenue , system

Table 5.12.: Top 20 features (unigrams) selected using GSS

GSS (Non-private)	DP-GSS ($\epsilon = 0.5, m = 1$)	DP-GSS ($\epsilon = 0.5, m = 5$)
article, windows	writes, article	windows, writes
team, year	windows , team	article, season
car, game	car, year	year, software
dod, program	game, dod	graphics, don
apr, good	apr , program	team, trumpeted
system, software	good, card	admits, moreillon
problem, card	play, software	adulteries, play
play, file	system, file	deftwmrc, tractatus
season, bike	problem, season	board, prc
graphics, computer	games, graphics	game, xauth

inatory (range $[3, 1552584]$). The global sensitivity of χ^2, OR, and MI are all high and performed equally poorly. Therefore, we do not include the overlap of OR, MI over χ^2 in our comparison graph. In the next section, we compare the performance of differentially private feature technique in a practical setting.



(a) $m = 1$ (b) $m = 3$

Figure 5.1.: Overlap of 100 private & top 100 non-private χ^2 features

5.3.1 Differentially Private Naïve Bayes Classifier

Naïve Bayes is a classification algorithm that is used to predict the class C of a given instance $W = (W_1, W_2, \ldots, W_n)$. It computes $P(C|W)$ by making a conditional assumption that W_i is independent of other W_j given C. Formally, the problem is to find c_j such that

$$\underset{c_j}{\operatorname{argmax}} Pr(C = c_j) \prod_i Pr(W_i = w_i | C = c_j)$$

To make predictions, we need to estimate $P(c_j)$ and $P(w_j|c_j)$ from a training dataset. To learn a differentially private naïve Bayes classifier (DP-NB), it is enough to release the differentially private counts of N_{w_i,c_j} and N_{c_j}. Since, the amount of noise added to N_{w_i,c_j} and N_{c_j} is proportion to the number of parameters learned, we use private feature selection for estimating the parameters of top m features.

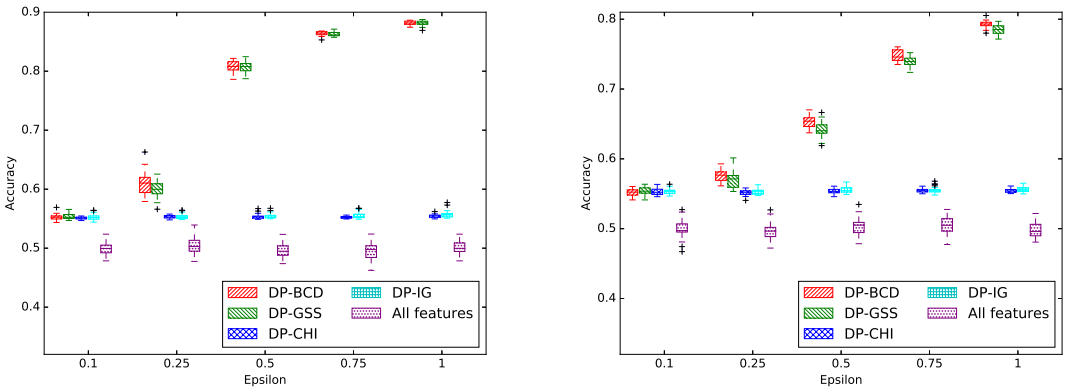
To access the effectiveness of the differentially private feature selection techniques, we compare the performance of differentially private naïve Bayes (achieves document privacy[2]) classifier with private feature selection to the baseline non-private naïve Bayes (NP-NB) classifier. We created two datasets from the 20 newsgroup dataset. COMP/REC dataset consists of 2907 documents related to computer and 2380 recreational documents. The SCI/POL dataset consists of 2372 science documents and 1949 documents related to politics. The binary classification task was to identify the type of each document. In both cases, we consider the 20 newsgroup dataset as the domain and the set of all unigrams present in the domain as feature universe $\mathcal{F}$. The top m features from $\mathcal{F}$ were selected based on the private database D and a feature selection technique g. The baseline accuracy (computed using 5-fold cross validation) of the NP-NB classifier for the datasets are shown in Table 5.13. The first column shows the accuracy (%) of NP-NB when all the features were used for classification. The second column shows the accuracy of NP-NB when the top 50 features were used for classification with the feature selection techniques CHI, IG, BCD and GSS.

[2]Neighboring databases differ by a single document

The algorithm for building a DP-NB is shown in Algorithm 8. Step 1 allocates the privacy budget for releasing N_{c_j}. Step 2 splits the rest of the privacy budget in to two halves. One half of the privacy budget is used for privately selecting the top m features and the other half is used for learning the probabilities of the top m features. Step 4 of the algorithm computes the statistic based on the given feature selection technique g. Then, step 6 selects the top m features each with probability proportional to $e^{\frac{\epsilon \times statistic(D,w_i)}{2 \times m \times \Delta g}}$. The noisy class counts N_{c_j} are computed in Step 7. Steps 8-12, compute the noisy posterior probabilities ($P_{i,j}$) for each of the top m features, where $P_{i,j}$ denotes the differentially private probability of the feature w_i given category c_j.

Table 5.13.: Accuracy (in %) of non-private naïve Bayes classifier

	All Features	**Top 50 Features**			
		CHI	**IG**	**BCD**	**GSS**
COMP/REC	98.4	90.1	90.0	89.3	89.3
SCI/POL	95.8	83.7	83.7	82.5	82.5



(a) COMP/REC dataset (b) SCI/POL dataset

Figure 5.2.: Accuracy of differentially private naïve Bayes classifier with top 50 features; x axis shows the values of ϵ in log scale and y axis denoting the accuracy

Algorithm 8 Differentially private naïve Bayes classifier

Input: A dataset $D \in \mathcal{D}$, universal feature set $\mathcal{F}$, privacy budget ϵ, the number of features to learn m, feature selection technique g and its sensitivity Δg
Output: Differentially private parameters $\hat{P}_{i,j}$ and $\hat{N}_{c_j}$ of naïve Bayes classifier

1: $\epsilon_1 = 0.05 \times \epsilon$
2: $\epsilon_2 = 0.475 \times \epsilon$
3: **for** $w_i \in \mathcal{D}$ **do**
4: Compute $Statistic[i]$ based on feature selection technique g
5: **end for**
6: Sample m features using Exponential mechanism with $Statistic[i]$ as the scoring function, Δg as the sensitivity and ϵ_2 as the privacy budget. i.e., $Pr(w_i) \propto e^{\frac{\epsilon_2 \times statistic[i]}{2 \times m \times \Delta g}}$
7: $\hat{N}_{c_j} = N_{c_j} + Lap(\frac{1}{\epsilon_1})$, where $j \in \{0, 1\}$
8: **for** $i = 1$ to k **do**
9: **for** $j \in \{0, 1\}$ **do**
10: $\hat{P}_{i,j} = N_{w_i,c_j} + Lap(\frac{m}{\epsilon_2})$
11: $\hat{P}_{i,j} = \frac{\hat{P}_{i,j}}{\hat{N}_{c_j}}$
12: **end for**
13: **end for**
14: Publish $\hat{P}_{i,j}$ and $\hat{N}_{c_j}$

Figures 5.2a and 5.2b shows the accuracy of the DP-NB classifier with differentially private feature selection techniques DP-GSS, DP-BCD, DP-CHI on COMP/REC and SCI/POL dataset respectively. We also plotted the results when every feature (All-Features) was used for building a DP-NB classifier. For All-Features, the differentially private counts for every feature in $\mathcal{F}$ were learned using D. No budget was spent on feature selection, but the noise added to each $P_{i,j}$ in step 10 is proportional to $Lap(\frac{|\mathcal{F}|}{2 \times \epsilon_2})$. The x axis shows the privacy budget ϵ in log scale and accuracy in y axis. Each data point in the box plot is the accuracy computed using 5-fold cross validation. The experiment was re-run 25 times and the average accuracy in each of 25 trials were shown as a box plot. For $\log \epsilon \geq 0.25$, the DP-NB classifier with DP-BCD/DP-GSS feature selection technique, performs better than DP-χ^2, DP-IG or All-Features.

5.3.2 Differentially Private Regularized SVM

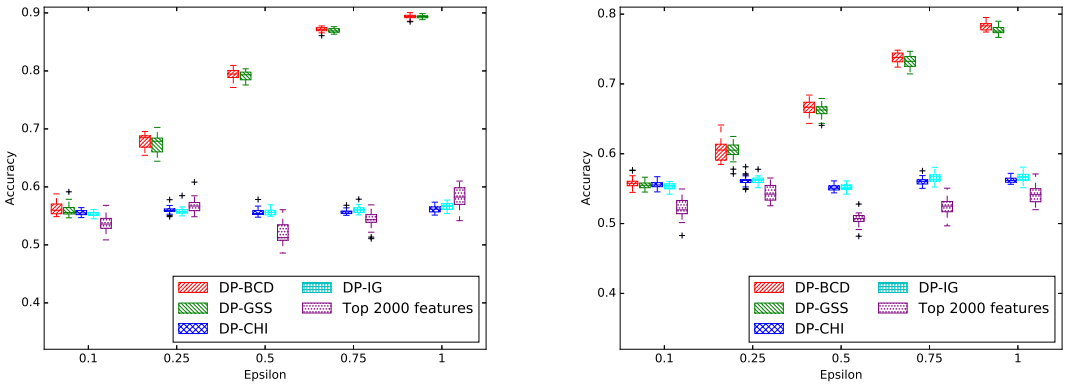
For a binary classification task, a support vector machine (SVM) [65] constructs a hyperplane that maximizes the margin between the two classes. There also exist techniques such as the kernel trick for SVM, which have been shown to perform well on non-linearly separable data. [66] proposed a differentially private algorithm for regularized SVM based on objective perturbation, which involves adding noise to the objective function prior to minimizing.

We evaluate the performance of non-private SVM (NP-SVM) and differentially private SVM (DP-SVM) with private feature selection using the datasets COMP/REC and SCI/POL. The baseline accuracy (computed using 5-fold cross validation) of the non-private SVM classifier for the datasets are shown in Table 5.14. The first column shows the accuracy (%) of NP-SVM when all the features were used for classification. The second column shows the accuracy (%) of SVM when the top 50 features were used for classification with the feature selection techniques CHI, IG, BCD and GSS.

Table 5.14.: Accuracy (in %) of non-private SVM

	All Features	**Top 50 Features**			
		CHI	**IG**	**BCD**	**GSS**
COMP/REC	97.6	91.0	91.1	91.1	91.1
SCI/POL	96.3	83.6	83.7	82.2	82.2

Figures 5.3a and 5.3b shows the accuracy of the DP-SVM classifier with differentially private feature selection techniques DP-GSS, DP-BCD, DP-CHI on COMP/REC and SCI/POL dataset respectively. We were not able to plot the results for All-Features as the implementation of [66] didn't scale well beyond 2000 features. Therefore, we have plotted the box plots for DP-SVM with differentially private top 2000 features. The x axis shows the privacy budget ϵ in log scale and accuracy in y axis. Each data point in the box plot is the accuracy computed using 5-fold cross validation. The experiment was re-run 25 times and the average accuracy in each of 25 trials were



(a) COMP/REC dataset (b) SCI/POL dataset

Figure 5.3.: Accuracy of differentially private regularized SVM classifier with top 50 features; x axis shows the values of ϵ in log scale and y axis denoting the accuracy

shown as a box plot. For $\log \epsilon \geq 0.25$, the DP-SVM with DP-BCD/DP-GSS feature selection technique, performs better than DP-χ^2, DP-IG or top 2000 features.

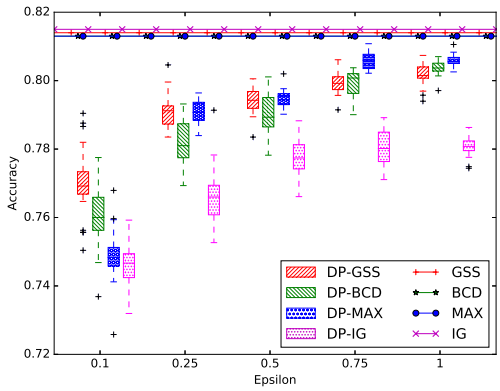
SVM vs Naïve Bayes Classifier

In case of the non-private version, the performance of SVM was comparable to or better than non-private NB except when All-Features was used with COMP/REC dataset. In case of the differentially private version, the performance of DP-SVM is better than DP-NB if the top 50 features were selected using DP-BCD or DP-GSS for $\epsilon \leq 0.25$. For $\epsilon > 0.25$, the accuracies are similar[3].

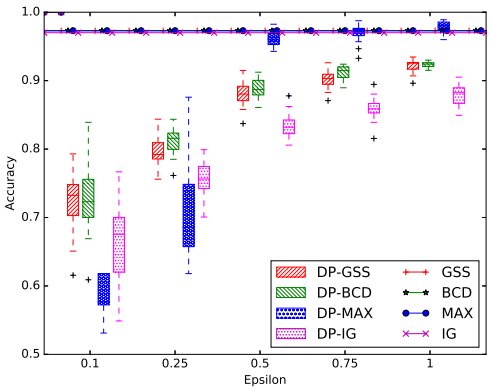
5.4 Differentially Private Decision Trees

A decision tree is learnt from a training dataset using the top-down approach. Initially the whole dataset is present in the root node. Beginning from the root node,

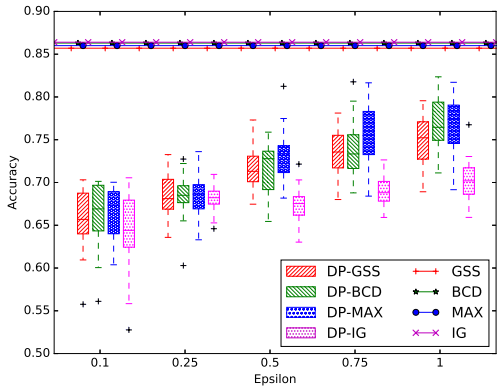
[3]We believe that DP-SVM with All-Features will perform better than DP-NB with All-Features because the DP-SVM with top 2000 features performed better than DP-NB when the features were selected using non-private χ^2 feature selection technique. However, we were not able to validate it due to scalability issues with the implementation.



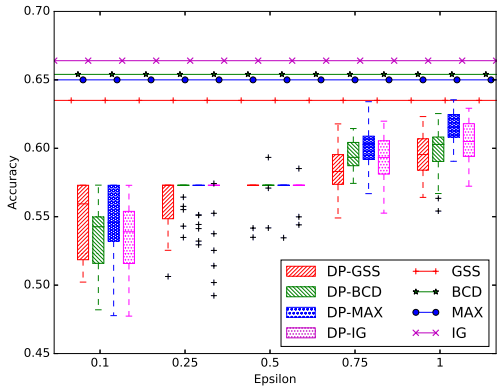
(a) Adult data set



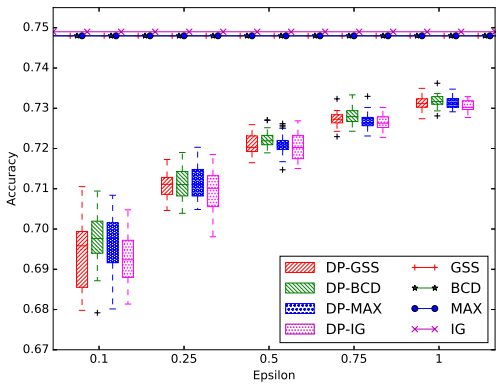
(b) Mushroom data set



(c) Car data set



(d) Contraceptive method choice data set



(e) Connect-4 data set

Figure 5.4.: Accuracy of differentially private decision trees

the data is partitioned into subsets (intermediate nodes) such that similar instances are present in each node. This is achieved by selecting a best splitting attribute for partitioning. The algorithm continues to recurse on each subset (intermediate node), considering only attributes never selected before. It stops if every element in a node belongs to the same class or if there are no more attributes to be selected. The number of instances of each class in a leaf node is saved for future decision making of a test dataset.

[56] showed that a low sensitivity criteria for split point selection, MAX, significantly improved differentially private decision tree accuracy. We extend their results, comparing BCD and GSS against the measures they used. We built a differentially private decision tree (DP-DT) classifier based on the algorithm proposed in [56]. The key differences in constructing a DP-DT are as follows. 1) At each node, the best splitting attribute should be chosen privately. This is done by scoring each attribute using a quality function (like information gain, gini impurity etc.) and using the exponential mechanism to chose the attribute privately. 2) To check the stopping criteria, [56] uses a heuristic, which requires each class count be larger on average than the standard deviation of the noise in the noisy instance count of the subset. 3) The number of instances of each class in a leaf node should be released in differentially private manner.

Apart from information gain and gini index, [56] also evaluated a DP-DT classifier on the MAX operator described in [67]. MAX is computed as follows.

$$MAX(D, A) = \sum_{A \in A} \max_{c \in \{c_0, c_1\}} (N_{A,c})$$

Given a database D and an attribute A, MAX computes the sum of the highest class frequencies over the values of A. The sensitivity of the MAX function is 1. [56] got the best results using MAX operator as it had the lowest sensitivity among the scoring functions. In this section, we also evaluate the performance of BCD and GSS

as a scoring function to select the splitting attribute privately and compare their performance against IG and MAX.

We evaluated the performance of the private decision tree on adult, mushroom, car and connect-4 datasets from the UCI data repository [64]. The datasets were first cleaned by removing any missing values and discrete attributes with at most 16 values were used for tree construction. A non-private decision tree classifier [68] (NP-DT) and differentially private decision tree classifier [56] (DP-DT) with GSS, BCD, MAX and IG as the scoring functions were implemented to compare their performance in a private setting. The accuracy of decision trees were computed using 5-fold cross validation and the average accuracy value over the rounds are shown in Figure 5.4(a-e). For the DP-DT, the experiment was repeated 25 times; the average accuracy over the 5-fold cross validated rounds for each of the 25 trials is given as a box plot.

1. Adult: The Adult dataset is drawn from the United States Census [64]. It is composed of 30,718 instances after the removal of instances with missing values. The binary classification task is to predict whether an adult income is greater than 50,000 a year based on the discrete attributes workclass, education, marital-status, relationship, race and sex. Figure 5.4a shows the accuracy of NP-DT and DP-DT with GSS, BCD, MAX and IG as the scoring function. When privacy is not an issue, the decision tree with IG has an accuracy of 81.5%, which is marginally better than the other scoring functions ($\sim$81.3%). In case of DP-DT, built using a differential private feature selection, we can see that the accuracy of differentially private GSS, BCD and MAX are at least 1% better than the differentially private IG for $\epsilon \leq 0.5$. As we increase ϵ, the gap widens and there is about 2% gap for $\epsilon = 1$. For $\epsilon \leq 0.5$, DP-GSS performs the best and for $\epsilon \geq 0.75$ DP-MAX is the best performing scoring function.

2. Mushroom: The Mushroom dataset consists of 8124 samples corresponding to 23 species of gilled mushrooms, categorized as edible or poisonous. The classification task is to predict whether whether a mushroom sample is poisonous

or edible depending upon 22 characteristics such as shape, color, surface, etc. When all the attributes were considered for building a decision tree, NP-DT with GSS, BCD and MAX performed much better than IG (100% vs 97.5%). Therefore, we restricted ourselves to a subset (attribute ids 14-22) so that the performance of the non-private versions were close (97.3% vs 97.1%). When differential privacy was used, DP-GSS and DP-BCD scoring functions performed better than DP-IG and DP-MAX for $\epsilon \leq 0.25$. But, for $\epsilon \geq 0.5$, the performance of DP-MAX was better than DP-GSS, DP-BCD by about 6%.

3. Car: The Car evaluation dataset was derived from a simple hierarchical decision model [69]. It consists of 1728 examples with 4 class labels (unacceptable, acceptable, good and very good). For our binary classification task, we considered 'unacceptable' as class '0' and rest of the them as class '1' and used six features on price, technology, and comfort of a car for building the decision tree. The accuracy of the NP-DT with GSS (85.7%) was slightly lower than IG and BCD (86.4% and 86.3% respectively). For $\epsilon \leq 0.25$, the accuracy of the private scoring functions were almost the same. For $\epsilon \geq 0.5$, there is a significant performance improvement when DP-MAX was used and we can see at least 1-3% gain when it is used as the splitting criteria.

4. Connect-4: The connect-4 dataset consists of all legal 8-ply positions in the game of connect-4 in which neither player has won yet with the class label indicating the outcome of the first player (win, loss or draw) [64]. It is composed of 67557 samples with 42 attributes describing which player has each position of the board. In our experiments, we discarded the 'draw' instances and considered 'win' as class '1' and loss as class '0' and the binary classification task is to predict if the first player won or lost the game. When every attribute was considered for building a NP-DT, BCD performed much better than IG. Therefore, we restricted ourselves to a subset (first 10 attribute ids a1-a6, b1-b4), so that the performance of the NP-DT with all the scoring functions were close

(∼74.9%). Unlike the other datasets, the gains of using a DP-GSS, DP-BCD or DP-MAX were not huge, but they were still able to beat DP-IG by 0.1%- 0.5% and the median of the accuracies was consistently better.

In summary, the DP-DT with DP-GSS and DP-BCD perform better than DP-DT with DP-IG as the scoring function when the size of the dataset is large and the performance is similar when the dataset size is small. An interesting result we see with DP-MAX scoring function is that, although it is a low sensitive function, the accuracy of DP-DT with DP-GSS is better or same as that of DP-MAX for small values of epsilon ($\epsilon \leq 0.25$). Smaller values of epsilon means better privacy. But as the value of epsilon is increased, the performance of DP-MAX is better DP-BCD and DP-GSS except on the connect-4 dataset.

6 CONCLUSION

In this dissertation, we considered the problem of privacy-preserving text analysis with differential privacy. Providing differential privacy in a secure two-party computation is challenging with malicious adversaries because the solution of each party adding noise to the other party's output to guarantee differential privacy does not work. Even rational adversaries may behave maliciously (can add a predetermined large noise) to gain exclusive access to the result, and the very noise that provides privacy protection also limits detection of malicious behavior.

We presented a secure two-party protocol for pseudo-random sample generation from an arbitrary Laplace distribution using garbled circuits in a malicious setting. A direct consequence of this is the ability to build protocols for differentially private analysis with verifiable noise. As long as one of the parties behave honestly (i.e. generates a standard uniform sample), a malicious party will not be able to influence the final result. Unfortunately, this protocol is expensive, limiting its use to off-line settings.

We also present a much more efficient protocol that succeeds against *rational* adversaries: parties where the cost of getting caught behaving maliciously outweighs the benefits. We demonstrate this in the context of a simple two-party protocol.

The idea of a two-party distributed sampling protocol given in Section 3.3 can be extended to the multi-party case; the rational adversary approach in Section 3.4 is more challenging. We leave the question of building efficient protocol for multi-party case for future work.

In Chapter 4, we investigated the problem of developing data-oblivious algorithms. In Section 4.2, we presented a data-oblivious secure two-party algorithm for weighted bipartite matching based on the Hungarian algorithm. We show how to create a differentially private version that provides a guarantee of differential privacy to both

parties. The data-oblivious algorithm presented in Section 4.2 has an overhead of $O(\log |V|)$ compared to an insecure version (plus the constant factor imposed by computing on encrypted data.) We also presented a data-oblivious algorithms for computing the minimum vertex cover in bipartite graphs and detecting articulation points in undirected graphs. We then introduced ϵ-data-obliviousness, a relaxed notion of data-obliviousness that helps us to develop efficient protocols for data-dependent algorithms like frequent itemset mining.

Finally, we consider the problem of privacy-preserving classification. Private data analysis is a challenging task when confronted with high dimensional data such as text. While feature selection can alleviate these problems, it must be done carefully to avoid introducing new privacy leaks.

We showed that some of the feature selection techniques can be effective while still satisfying differential privacy. Others, however, provide a nearly random selection when differential privacy is satisfied. This work demonstrates that protecting privacy requires more than simply applying privacy protection methods to existing data analysis techniques. Careful selection of analysis techniques that both perform the desired analysis, and do so in a way that inherently limits privacy risks, can significantly improve results when privacy protection methods are used. While we deal only with appropriateness of feature selection techniques, the basic ideas are relevant to any data analysis task where various techniques may be appropriate for the task.

REFERENCES

REFERENCES

[1] Venkatesan T Chakaravarthy, Himanshu Gupta, Prasan Roy, and Mukesh K Mohania. Efficient techniques for document sanitization. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 843–852. ACM, 2008.

[2] Wei Jiang, Mummoorthy Murugesan, Chris Clifton, and Luo Si. t-plausibility: Semantic preserving text sanitization. In *Proceedings of the International Conference on Computational Science and Engineering*, volume 3, pages 68–75. IEEE, 2009.

[3] Balamurugan Anandan, Chris Clifton, Wei Jiang, Mummoorthy Murugesan, Pedro Pastrana-Camacho, and Luo Si. t-plausibility: Generalizing words to desensitize text. *Transactions on Data Privacy*, 5(3):505–534, 2012.

[4] David Sánchez, Montserrat Batet, and Alexandre Viejo. Automatic general-purpose sanitization of textual documents. *IEEE Transactions on Information Forensics and Security*, 8(6):853–862, 2013.

[5] David Sánchez, Montserrat Batet, and Alexandre Viejo. Utility-preserving sanitization of semantically correlated terms in textual documents. *Information Sciences*, 279:77–93, 2014.

[6] Balamurugan Anandan and Chris Clifton. Significance of term relationships on anonymization. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 253–256. IEEE Computer Society, 2011.

[7] Chris Clifton, Robert Cooley, and Jason Rennie. Topcat: Data mining for topic identification in a text corpus. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):949–964, 2004.

[8] Ronen Feldman and Haym Hirsh. Exploiting background information in knowledge discovery from text. *Journal of Intelligent Information Systems*, 9(1):83–97, 1997.

[9] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, FOCS '86, pages 162–167. IEEE, 1986.

[10] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[11] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer Science and Business Media, 2010.

[12] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '99, pages 223–238. Springer, 1999.

[13] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of the 20th Annual International Conference on the Theory and Application of Cryptographic Techniques*, EUROCRYPT '01, pages 280–300. Springer, 2001.

[14] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[15] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP '06, pages 1–12, Venice, Italy, 2006. Springer-Verlag.

[16] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference*, TCC '06, pages 265–284. Springer, 2006.

[17] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103. IEEE, 2007.

[18] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, STOC '07, pages 75–84. ACM, 2007.

[19] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Proceedings of the 29th Annual International Cryptology Conference*, CRYPTO '09, pages 126–142, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *Proceedings of the 28th Annual International Cryptology Conference*, CRYPTO '08, pages 451–468. Springer, 2008.

[21] Fabienne Eigner, Matteo Maffei, Ivan Pryvalov, Francesca Pampaloni, and Aniket Kate. Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 316–325. ACM, 2014.

[22] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 735–746. ACM, 2010.

[23] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proceedings of the Annual Network and Distributed System Security Symposium*, NDSS '11. Internet Society, 2011.

[24] Chris Clifton and Balamurugan Anandan. Challenges and opportunities for security with differential privacy. In *Proceedings of the 9th International Conference on Information Systems Security*, pages 1–13. Springer, 2013.

[25] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[26] Gilles Barthe, George Danezis, Benjamin Grégoire, César Kunz, and Santiago Zanella-Béguelin. Verified computational differential privacy with applications to smart metering. In *Proceedings of the 26th IEEE Computer Security Foundations Symposium*, CSF '13, pages 287–301. IEEE, 2013.

[27] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[28] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Proceedings of the 28th Annual International Cryptology Conference*, CRYPTO '08, pages 554–571. Springer, 2008.

[29] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[30] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A system for secure multi-party computation. In *Proceedings of the 15th ACM SIGSAC Conference on Computer and Communications Security*, CCS '08, pages 257–266. ACM, 2008.

[31] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Security Symposium*, USENIX '11.

[32] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Security Symposium*, USENIX '12.

[33] Michael A Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

[34] Marina Blanton, Aaron Steele, and Mehrdad Alisagari. Data-oblivious graph algorithms for secure computation and outsourcing. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIACCS '13, pages 207–218. ACM, 2013.

[35] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proceedings of the 25th Annual International Conference on the Theory and Application of Cryptographic Techniques*, EUROCRYPT '06, pages 486–503. Springer, 2006.

[36] Balamurugan Anandan and Chris Clifton. Laplace noise generation for two-party computational differential privacy. In *Proceedings of the 13th Annual Conference on Privacy, Security and Trust, 2015*, PST '15, pages 54–61. IEEE, 2015.

[37] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies–Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[38] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *Proceedings of the International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

[39] Michael T Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1262–1277. Society for Industrial and Applied Mathematics, 2010.

[40] Michael T Goodrich. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in o (n log n) time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 684–693. ACM, 2014.

[41] Marina Blanton and Siddharth Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *Proceedings of the 20th European Symposium on Research in Computer Security*, ESORICS '15, pages 384–406. Springer, 2015.

[42] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In *Financial Cryptography and Data Security*, FC '13, pages 239–257. Springer, 2013.

[43] Justin Hsu, Zhiyi Huang, Aaron Roth, Tim Roughgarden, and Zhiwei Steven Wu. Private matchings and allocations. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 21–30. ACM, 2014.

[44] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 299–310. ACM, 2013.

[45] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[46] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '14, pages 506–525. Springer, 2014.

[47] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 377–396. Springer, 2013.

[48] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[49] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the 13th European Symposium on Research in Computer Security*, ESORICS '08, pages 192–206. Springer, 2008.

[50] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[51] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, volume 1215 of *VLDB '94*, pages 487–499, 1994.

[52] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 639–644. ACM, 2002.

[53] Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

[54] Tamir Tassa. Secure mining of association rules in horizontally distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):970–983, 2014.

[55] Justin Zhan, Stan Matwin, and LiWu Chang. Privacy-preserving collaborative association rule mining. In *Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy*, DBSEC '05, pages 153–165. Springer, 2005.

[56] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 493–502. ACM, 2010.

[57] Noman Mohammed, Rui Chen, Benjamin Fung, and Philip S Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 493–501. ACM, 2011.

[58] Stephen E Fienberg, Aleksandra Slavković, and Carline Uhler. Privacy preserving GWAS data sharing. In *Proceedings of the 11th IEEE International Conference on Data Mining Workshops*, ICDMW '11, pages 628–635. IEEE, 2011.

[59] Graham Cormode. Personal privacy vs population privacy: Learning to attack anonymization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1253–1261. ACM, 2011.

[60] Jaideep Vaidya, Basit Shafiq, Anirban Basu, and Yuan Hong. Differentially private naive Bayes classification. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies-Volume 01*, pages 571–576. IEEE Computer Society, 2013.

[61] Mengdi Huai, Liusheng Huang, Wei Yang, Lu Li, and Mingyu Qi. Privacy-preserving naive Bayes classification. In *Proceedings of the International Conference on Knowledge Science, Engineering and Management*, pages 627–638. Springer, 2015.

[62] Luigi Galavotti, Fabrizio Sebastiani, and Maria Simi. Experiments on the use of feature selection and negative evidence in automated text categorization. In *Research and Advanced Technology for Digital Libraries*. Springer, 2000.

[63] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 4th International Conference on Machine Learning*, ICML '97, pages 412–420, 1997.

[64] M. Lichman. UCI machine learning repository, 2013.

[65] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[66] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

[67] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. CRC press, 1984.

[68] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[69] Marko Bohanec and Vladislav Rajkovic. Knowledge acquisition and explanation for multi-attribute decision making. In *Proceedings of the 8th International Workshop on Expert Systems and their Applications*, pages 59–78, 1988.

VITA

VITA

Balamurugan Anandan received his BE in computer science from Kongu Engineering College, India in 2005 and MS in computer science from University of Illinois at Chicago in 2009. His research interests are in the intersection of data mining and privacy, specifically focussing on developing privacy-preserving protocols for text mining. He is also affiliated with the Center for Education and Research in Information Assurance and Security (CERIAS). He received his PhD in computer science in May 2017.