# SECURE MULTIPARTY PROTOCOL FOR DIFFERENTIALLY-PRIVATE DATA RELEASE

by

Anthony Harris

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Mathematics

Boise State University

May 2018

BOISE STATE UNIVERSITY GRADUATE COLLEGE

**DEFENSE COMMITTEE AND FINAL READING APPROVALS**

of the thesis submitted by

Anthony Harris

Thesis Title: Secure Multiparty Protocol for Differentially-Private Data Release

Date of Final Oral Examination: 12 March 2018

The following individuals read and discussed the thesis submitted by student Anthony Harris, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

| | |
|---|---|
| Gaby Dagher, Ph.D. | Chair, Supervisory Committee |
| Liljana Babinkostova, Ph.D. | Member, Supervisory Committee |
| Marion Scheepers, Ph.D. | Member, Supervisory Committee |

The final reading approval of the thesis was granted by Gaby Dagher, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

Dedicated to my parents, Regina and Aaron

## ACKNOWLEDGMENTS

I would like to thank the Mathematics Department for providing the opportunity to continue my education. I would also like to thank my advisor Dr.Dagher for introducing me to the field of cryptology and providing the foundation to pursue a Ph.D, as well as the committee members for their helpful feedback. Finally, I would like to thank my family and friends for their enduring support.

# AUTOBIOGRAPHICAL SKETCH

As I continued the Master's program at Boise State University, my interests began to converge towards cryptology and digital privacy. As our world continues to advance in technology, society will need to place a higher emphasis on cyber security and privacy. I am grateful for dedicating my research in a fast-paced and dynamic field.

# Abstract

In the era where big data is the new norm, a higher emphasis has been placed on models which guarantees the release and exchange of data. The need for privacy-preserving data arose as more sophisticated data-mining techniques led to breaches of sensitive information. In this thesis, we present a secure multiparty protocol for the purpose of integrating multiple datasets simultaneously such that the contents of each dataset is not revealed to any of the data owners, and the contents of the integrated data do not compromise individual's privacy. We utilize privacy by simulation to prove that the protocol is privacy-preserving, and we show that the output data satisfies $\epsilon$-differential privacy.

# Contents

# List of Tables

# List of Figures

# LIST OF ABBREVIATIONS

**RVP** – Random Value Protocol

**MMC** – Mix and Match Count Protocol

**MAIN** – Secure Multiparty Protocol for Differentially-Private Data Release

**Class,Cls** – Classifier Attribute

# LIST OF SYMBOLS

$P_j$ , denotes a party-member

$D_j$ , denotes dataset owned by party $P_j$

$\mathcal{D}$ , collection of all datasets

$\hat{D}$ , the integrated datasets

$A_i$ , denotes an attribute

$A^w$ , denotes a winning attribute

$[\![m]\!]$ , denotes the encrypted version of message $m$

$\epsilon$ epsilon symbol, used as the privacy parameter

$\mathcal{R}_j$ , denotes the individual records owned by party $P_j$

$\phi_j$ Lowercase Phi symbol, denotes the set of attribute-score pairs $\{A_i\}$ that party $P_j$ owns

$\Phi_j$ Uppercase Phi symbol, denotes the set of attribute-score pairs $\{(A_i, U_i)\}$ that party $P_j$ owns

$\mathbb{T}_{A_i}$ , denotes the taxonomy of attribute $A_i$

$\mathbb{T}_j$ , denotes the group taxonomy of $P_j$

$\mathrm{T}$ , denotes the intersected taxonomy of every attribute

$P_j^*$ , denote party $P_j$'s sub-partitioning tree

$P^*$ , denotes a partitioning tree

$\mathcal{N}$ , denotes the number of numerical attributes

$\mathcal{S}$ , denotes the number of specializations in the main algorithm

$\vec{V_{jk}}$ , denotes party-member $P_j$'s $kth$ attribute vector

$\vec{P_j}$ , denotes party-member $P_j$'s standardized attribute-vector

$\vec{C}$ , denotes the encrypted count-vector

$\mathcal{T}$ , denotes the Mix and Match Table

$\stackrel{c}{\equiv}$ , denotes computational indistinguishability

$\vec{x}$ , denotes the list of initial inputs $(x_1, x_2, \ldots, x_n)$

$S_j$ , denotes simulator $S_j$

$f(\vec{x})$ , denotes the ideal functional that takes $\vec{x}$ as input

$\Pi$ , Uppercase pi symbol, denotes a protocol

# Chapter 1

# INTRODUCTION

Recent technology has enabled both the size and storage of data to grow exponentially. Although data is abundant and widely available, it is advantageous for multiple data owners to integrate their respective data. By doing so, the integrated data becomes an enhanced version of the original distributed data, in terms of information and usability. This suggests cooperating parties have access to far better data compared to those working independently. For cooperation to exist, the parties must ensure that the integration protocol is both correct and privacy-preserving. Our protocol aims to establish a secure and private multiparty computation that allows for an efficient means of data integration.

## 1.1 Motivation

Given a collection of private datasets, parties have the ability to enrich their data analytics by simply working together. However, due to legal constraints or trust issues, parties may be unwilling to participate in collaborative computation. If the integrated dataset $\hat{D}$ can be provably shown to not leak any information during or after construction, then parties can act in their best interests without consequence. When parties have access to $\hat{D}$, they also have access to superior data relative to their original data. As a result, data mining techniques will be optimized with respect to $\hat{D}$.

## 1.2 Challenges & Concerns

Integrating datasets is not a new concept. Cryptography has advanced to the point where we can encode, combine, and decode information with relative ease. However it is not enough that a dataset's content remain confidential. Even if sensitive information (name, social security, address, etc.) is removed prior to integration and the datasets are integrated securely, that does not guarantee the contents of $\hat{D}$ are private. In fact, $\hat{D}$ is typically less private the more content it contains. When $\hat{D}$ is both diverse and abundant with respect to its content, that is normally seen as good. After all, such a dataset can be easily mined for data. Unfortunately, the tradeoff for *good* datasets is privacy. Simply put, the more information a dataset has on record about an individual, the more unique that individual becomes. In return, unique individuals are more identifiable, which puts their privacy at risk. There have been several methods developed to minimize this issue, including $k$-anonymity [30], $l$-diversity [21], $t$-closeness [17]. However these methods failed to both formalize privacy mathematically and guarantee that the privacy of an individual is preserved within an arbitrary dataset. Thus one of the biggest challenge in this paper is securely integrating datasets in a manner such that privacy is preserved. In 2014, Mohammed $et.al$ derived an algorithm which integrates datasets in a secure and private manner. Their algorithm was limited to a two-party scenario. The authors stated that their algorithm had the potential of multiparty extension. However this extension was limited by a few factors: Distributed Exponential Mechanism [25], Yao Protocol [20], Random Value Protocol [4], and the Secure Scalar Product Protocol [1]. Each of these protocols were designed specifically for two-party communication. In this paper, we attempt to extend the functionalities of the respective protocols in a manner that is conducive to deriving a secure multiparty differentially-private dataset.

## 1.3 Contributions

Our main contribution involve deriving a protocol which securely integrates multiple datasets in a differentially-private fashion. This achievement is highlighted by the following:

- Creating an exponential mechanism that is applicable in multiparty setting. This process was achievable by the Multiparty Exponential Mechanism 4.1.

- Extending the functionality of the Random Value Protocol(RVP) [4] to be applicable in multiparty party setting, instead of a two-party setting.

- Creating a protocol that allows secure exchange of messages and computation among multiple users. This process was achievable through the Distributed Comparision 4.2. Distributed Comparison encodes and decodes messages through ElGamal encryption [37] and was designed to substitute the functionality of the Yao Protocol [20], which only offered secure computation in a two-party setting.

- Creating a protocol that allows a party to convert their records into binary-vectors with respect to one or more attributes. From there, the parties securely intersect their binary-vectors among themselves which is later used to create a differentially-private dataset. This process was made possible be the Secure & Private Attribute Counting Exchange 5.2 (SPACE) protocol. SPACE was designed to substitute the functionality of Secure Scalar Product Protocol [1], which is conceptually similar but only applicable in the two-party setting.

- Our proposed protocol has the capacity of securely construct a differentially-private dataset from other datasets owned by two or more parties. However, our protocol can easily be reduced to a one-party setting analogous to DiffGen [26].

## 1.4 Thesis Statement

The objective of this thesis is to answer the following question: **Given multiple data owners, how can they securely integrate their respective datasets such that the privacy is maintained and the output model is privacy-preserving?**

Given multiple datasets $D_1, D_2, \ldots, D_n$ owned by $P_1, P_2, \ldots, P_n$ respectively, and a privacy budget $\epsilon$, the goal of this thesis is to design a protocol that securely publishes an anonymized and integrated dataset $\hat{D}$ for the purpose of statistical analysis such that:

1. The protocol is secure (privacy-preserving) in the semi-honest adversarial model.
2. The output is $\epsilon$-differentially private.

## 1.5 Organization of the Thesis

This Thesis is organized as follows:

- Chapter 2 provides basic background knowledge which functions as the backbone of the thesis.
- Chapter 3 discusses relevant literature relating to secure computation, privacy, as well as various data publishing and data mining techniques.
- Chapter 4 discusses how the parties can securely derive a winning attribute $A^w$ in a differentially-private manner, using the Multiparty Exponential Mechanism 4.1.
- Chapter 5 will detail the main algorithm, where the parties collectively use $A^w$ to securely derive a differentially-private dataset $\hat{D}$.
- We conclude the thesis in Chapter 6 by detailing the future work, summary, and closing remarks.

# Chapter 2

# BACKGROUND

## 2.1 Privacy & Security

A dataset $D_j$ owned by party $P_j$ is described as a collection of attributes, classifiers, individuals, and individual records. For each dataset, the classifiers and individuals are both shared and known publicly. However, attributes and individual records are private to their respective party. An attribute (age, sex, income, etc.), described as $A_i$, is owned by a specific party and describes a single element in the individual record of an individual. Although each attribute is private, the parties are aware of which attributes belong to which party. However, the other parties $P_k$ do not know which attribute value corresponds to an individual records contained in $D_j$. Individual records (Person 001: 36 years old, male, 36K, etc.) quantifies how each individual within a dataset is detailed with respect to an attribute. Classifiers (loan approval, home owner, educated, etc.) also categorize individuals in the dataset, the outcome of which is public knowledge. Typically, the parties want to collaborate in such a way that they can predict the outcome of a classifier for a new individual not in the dataset, given the 'attribute-profile' of that same individual (Did Person $x$ get approved for a loan given their attribute-profile?). Before parties begin collaboration, each individual is assigned a common attribute identifier ($ID$). This $ID$ is used in place of an individual's name for the sake of anonymity. Although their name is omitted from the data, that does not necessarily mean their data is private. An individual's privacy can be compromised if their 'attribute-profile' is too unique relative to other individuals in the dataset [25]. A common side-effect of large datasets is that the more attributes it acquires, the more unique and identifiable the individuals in the dataset become. To address this issue, we will employ differential-privacy as a means to protect the privacy of every individual in every dataset. An algorithm is said to be differentially private if by looking at the output, one cannot tell whether any individual's data was included in the original dataset or not. In other words, a differentially private-algorithm guarantees that its outcome hardly changes when a single individual joins or leaves the dataset.

Beyond privacy we also want our protocol to be computationally secure. We have the option of having our protocol be secure with respect to one of the following models: honest, semi-honest, and malicious. For this thesis, we are only concerned with achieving security in the semi-honest setting. Semi-honest security assumes every party will follow the protocol exactly as described. As the parties conduct the protocol they will attempt to learn or reveal any information about the other participants. For a computation to be semi-honest secure, requires that no new information about any party is revealed or deduced during or after the execution of the protocol. A secure protocol requires the scheme to be mathematically secure prior to implementation.

**Example 2.1.1.** Imagine hospital $P_1$, health-insurance company $P_2$, and credit-card company $P_3$ all have distinct attributes. $P_1$ owns dataset $D_1 = \{ID, Classifier, Sex, Salary, \mathcal{R}_1\}$, $P_2$ owns $D_2 = \{ID, Classifier, Education, \mathcal{R}_2\}$, and $P_3$ owns $D_3 = \{ID, Classifier, Job, \mathcal{R}_3\}$, as shown in the Table 2.1. For party $P_j$, $\mathcal{R}_j$ corresponds to the individual records that $P_j$ owns. Each party will also have the outcome of the classifier normally described as "$Class$" in the literature, which indicates whether a specific health-procedure is approved or not. In

Table 2.1: Vertically-partitioned raw data owned by parties $P_1$, $P_2$ and $P_3$

| **Shared** | | $P_1$ | | $P_2$ | $P_3$ |
|---|---|---|---|---|---|
| ID | Classifier | **Job** | **Education** | **Salary** | **Sex** |
| 1 | Yes | Artist | No College | \$30k | Male |
| 2 | No | Unemployed | Undergrad | \$0k | Male |
| 3 | Yes | Unemployed | No College | \$0k | Female |
| 4 | No | Unemployed | Graduate | \$0k | Male |
| 5 | No | Professional | Undergrad | \$40k | Female |
| 6 | No | Professional | Graduate | \$55k | Female |
| 7 | No | Artist | Undergrad | \$40k | Female |
| 8 | No | Artist | undergrad | \$25k | Male |
| 9 | No | Professional | No College | \$30k | Female |
| 10 | Yes | Artist | Graduate | \$40k | Female |

Table 2.1, each column corresponds to a specific attribute, while each row corresponds to a single individual and their respective individual records. Notice individuals 1 and 4 have unique profiles, given that they are respectively the only author and lawyer within the dataset. This means individuals 1 and 4 are vulnerable linking attacks where by occupation alone the parities (or a data miner) can deduce their identities with absolute certainty, assuming the data in the dataset was sufficiently rich. To avoid attribute-based linking attacks, differential-privacy will be introduced later as a privacy measure. ■

## 2.2 Cryptographic Primitives

In this section we will detail all the encryption methods and mechanisms used throughout the thesis. These encryption methods were picked based on their versatility, specifically the ability to foster secure communication between two or more parties.

**Exponential ElGamal [3]**

This primitive allows messages to be jointly encrypted and decrypted among any number of users. The encryption is both semantically secure and homomorphically additive. Each party uses their private keys jointly to encrypt and decrypt message $m$. The Exponential ElGamal sees the most amount of use during Protocol 4.2 (Distributed Comparison) and sees significant application in Protocol 5.2 (SPACE). For brevity, we denote the encryption of a message $m$ as the ciphertext $[\![m]\!]$. Given generator $g$, prime $p$, group public-key $A$, group ephemeral-key $r$, and public backdoor $g^r$

$$[\![m]\!] := (A^r \cdot g^m \ \ mod\, p, \ g^r \ \ mod\, p) \tag{2.1}$$

There are many times within the paper when we mention "homomorphic addition". Homomorphic addition means that the "$\times$" operator can be understood as the "$+$" operator. For instance we can homomorphically add $a$ and $b$ as follows $g^a \times g^b \ mod\, p = g^{a+b} \ mode\, p$. Homomorphic operations are consistently applied when dealing with ElGamal encryption. In a similar fashion, homomorphic encryption is described as follows: $[\![a+b]\!] = [\![a]\!] \times [\![b]\!]$.

**Random Value Protocol (RVP) [4]**

RVP allows two parties to generate a random value $R$, where $R$ has been chosen uniformly within a predefined integer-based interval. This interval starts at 0 and ends at some positive integer-value $\sigma = \sigma_1 + \sigma_2$, where $R \in [0, \sigma]$. $R$ is not known by either party but it is shared between them. More specifically, $P_1$ has $R_1 \in [0, \sigma_1]$, and $P_2$ has $R_2 \in [0, \sigma_2]$ where $\sigma_i$ are both integers. $R_1$ and $R_2$ are considered random 'shares' of $R$, where $R = R_1 + R_2$ and $R, R_1, R_2 \in [0, \sum_{i=1}^{2} \sigma_i]$. For the Multiparty Exponential Mechanism 4.1 RVP was generalized to include $n$ parties.

**Mix and Match [12]**

Mix and Match uses a logic table to securely determine how many times an encrypted value occurs in a given set of values. This primitive will indirectly identify the unique values in the set as well as the multiplicity of their occurrence. However, the unique values and their respective multiplicities will remain encrypted throughout the scheme. The Mix and Match primitive is applied in Protocol 5.2.

**Mix Network [13]**

A mix network allows a list of encrypted messages to be jointly shuffled and re-encrypted such that no party knows the original arrangement of the encrypted messages. There are several ways of acquiring a mix network, where most require that each step to be verifiable by each of the party members. Mix network is applied in Protocol 5.2.

## 2.3 Privacy Model

In this section we introduce the notion of privacy. Typically, when people attempt to define privacy they do so holistically. But for the sake of consistency and correctness it is critical that privacy be defined mathematically. By doing so, there exists a consistent means of confirming or denying the privacy of a protocol. As to date, differential privacy is the"gold standard" of privacy. Differential privacy provides a well-defined, mathematical definition of privacy. This model makes no assumptions about the knowledge of an adversary. This ensures an adversary learns nothing new about an individual, whether or not their record is in the dataset [7].

**Definition 2.3.1.** $\epsilon$-Differential Privacy: [7]

A randomized mechanism $M$ is $\epsilon$-differentially private if for all datasets $D_1$ and $D_2$ (where they differ at most one element), and for all possible anonymized dataset D.

$$Pr[M(D_1) = D] \leq e^{\epsilon} \times Pr[M(D_2) = D] \tag{2.2}$$

■

A standard means to achieve differential privacy is to add random noise to the true output of the dataset. The noise is calibrated according to the sensitivity of the function. The sensitivity of a function is the maximum difference of its outputs from two datasets that differ only in one record. The sensitivity of a utility function is defined as follows

**Definition 2.3.2.** Sensitivity: [7]

For any function $f : D \to \mathbb{R}$, the sensitivity of $f$ is

$$\Delta f = \max_{D_1, D_2} |f(D_1) - f(D_2)| \tag{2.3}$$

For all $D_1$, $D_2$ differing by at most one record. ■

**Example 2.3.1.** Imagine there are two datasets, $D_1$ and $D_2$. $D_1$ contains the individual records $\{A_1, B_1, C_1, ..., Y_1\}$ with respect to an attribute and $D_2$ contains the individual records $\{A_2, B_2, C_2, ..., Z_2\}$ with respect to another attribute. Notice $D_1$ and $D_2$ differ by exactly one record. Since the two datasets are sufficiently close in size, sensitivity can be applied. Define function $f$, which counts the elements in a given set.

$$\Delta f = \max_{D_1, D_2} |f(D_1) - f(D_2)| = |25 - 26| = 1$$

$f$ described in this manner, is typically called the 'count function'. However $f$ is arbitrary and can be whatever function we like. ■

### 2.3.1 Exponential Mechanism

When implementing differential-privacy, it is typically done to numerical attributes where numerical noise is added to the dataset. By adding noise to the dataset, the privacy of each individual within the dataset is preserved. However, it is possible for $P_j$ to own a dataset which contains numerical or categorical attributes. In the case of categorical attributes, it makes no sense to add noise directly. Luckily, there exists an 'exponential mechanism' that achieves differential privacy whenever it makes no sense to add noise [24]. An exponential mechanism is a differentially private method to select an element from a set with high utility. The utility function $u$, takes dataset $D \in D^n$ and takes some value $A \in \mathcal{A}$ as input, outputting a real value in return. Or in other words, $u : (D^n \times \mathcal{A}) \rightarrow \mathbb{R}$. For the utility function, a higher value corresponds to better data utility [24]. The exponential mechanism creates a probability distribution over the range $\mathcal{A}$, where we then sample aa value $A_i \in \mathcal{A}$ [24]. For this paper, $D^n$ represent the datasets among all parties, $\mathcal{A}$ represents the set of attributes among all parties, and $u(D, A)$ returns utility-score $U$, where $D$ and $A$ come from the same party. Our objective regarding the exponential mechanism is to create an $(A, U)$ pair for all attributes, then probabilistically select an $A$ with high utility-score.

**Definition 2.3.3.** Exponential Mechanism [7]:

For a set $D$, a set of possible outputs $T$, and scoring function $U(D, t)$, privacy-budget $\epsilon > 0$, a mechanism is an exponential mechanism if the probability of selecting $t \in T$ is proportional to $exp(\frac{\epsilon \cdot u(D,t)}{2\Delta u})$, where $\Delta u = \max\limits_{t, D_1, D_2} |u(D_1, t) - u(D_2, t)|$ and $D_1, D_2 \in D$ differ by at most one element. ■

**Theorem 2.3.1.** The exponential mechanism preserves $\epsilon$-differential privacy ■

Using the above theorem and definition, we now have the tools necessary tools to verify whether our protocol is indeed private or not. If a protocol implements the exponential mechanism correctly, then the protocol is $\epsilon$-differentially private. Therefore if our protocol applies an exponential mechanism throughout some execution, then privacy is satisfied with respect to that execution.

### 2.3.2 Laplace Mechanism

Recall each party has a collection of both categorical and numerical attributes. Among all the parties we would like to probabilistically select a winning attribute $A^w$ which has the highest utility. For our protocol, utility is loosely described relative to how unique the individuals are in $D_j$, with respect to one of $P_j$'s attribute. For instance, if $P_j$ had the attribute $A_1$ which corresponds to job, where the majority of the individuals within the $P_j$'s dataset had the same job, then $A_1$ would be assigned a low utility score. Conversely, if the majority of the individuals in the dataset had different jobs, then $A_1$ would be assigned a high utility score. Once we appropriately create the attribute-score pair $(A_i, U_i)$ for each attribute, we implemented the exponential mechanism to select a winning-attribute $A^w$ for use. However, that is not the end of the story regarding privacy. The Multiparty Exponential Mechanism 4.1 is a sub-protocol of the MAIN Protocol 4.1, meaning we need to examine the MAIN protocol in its totality to appropriately assess privacy. When approaching final procedures of the MAIN Protocol we need to account for numerical values. We must ensure that prior to release, the numerical data is also differentially private. This is achievable through the Laplace Mechanism, which is designed specifically for numerical data. Conceptually, in order for a dataset to maintain privacy, it cannot contain any raw data. Instead, the dataset owner needs to add random noise to the numerical data in a manner such that it converts the raw data into noisy data, which is statistically similar to the raw data. Although the noise is random, it is controlled in an algorithmic way. If the noise is too large, the data becomes useless. Conversely, if the noise is too small, the privacy of the data is not preserved. The noise itself is derived from the Laplace random variable, whose distribution depend on the sensitivity of function $f$.

**Theorem 2.3.2.** *[26]*

For any function $f : D \to \mathbb{R}^d$, the mechanism $M$ that adds independently generated noise with distribution Lap($\Delta f/\epsilon$) to each of the $d$ outputs satisfies $\epsilon$-differential privacy. ■

Thus, as long as the appropriate amount of noise is added to the raw data prior to publishing, we can ensure that the integrated dataset $\hat{D}$ is $\epsilon$-differentially private.

### 2.3.3 Composition Theorems

What makes differentially privacy nice is that it allows compositions of several mechanism (or protocols), while keeping track of privacy. This allows us to keep track and tally the usage of our privacy budget, throughout the execution of the protocol. A privacy budget is a fixed value, usually denoted as $\epsilon$. If there is a mechanism which requires some of the overall privacy budget (like $\epsilon'$), then we would deduct some of the overall privacy budget (or what is currently available) by $\epsilon'$. That being said we are never allowed to consume more of the privacy budget than we originally started with. There currently exist two well-known theorems which we will use later on in the paper to manage and keep track of our privacy budget.

**Theorem 2.3.3.** *Sequential Composition [26]*

Let each mechanism $M_i$ provide $\epsilon_i$-differential privacy on dataset $D$. Then mechanism $\mathbf{M}(M_1, M_2, \ldots, M_n)$ provides $(\sum_i^n \epsilon_i)$-differential privacy for dataset $D$. ■

**Theorem 2.3.4.** *Parallel Composition [26]*

Let each mechanism $M_i$ provide $\epsilon$-differential privacy. Given a sequence of $M_i(D_i)$ over a set of disjoint datasets $D_i$ (i.e $\{M_1(D_1), M_2(D_2), \ldots, M_n(D_n)\}$), where $D = \bigcup_{i=1}^{n} D_i$ and $\emptyset = D_a \bigcap D_b$ for all $D_a, D_b \in D$, as well as a mechanism $\mathbf{M}(M_1, M_2, \ldots, M_n)$, then $\mathbf{M}$ satisfies max$(\epsilon_i)$-differential privacy. ■

### 2.3.4 Information Gain

In terms of an adequate utility function we recommend 'information gain'. Although there exists many utility functions to choose from. Information Gain ($IG$) takes a dataset and quantifies it based on how well it can be homogenized relative to a set of attributes and a classifier attribute. In this case, the set of attributes can be whatever you like (age, sex, weight, etc.). Similarity, a classifier attribute can also be arbitrary. What distinguishes the set of attributes from a classifier is that the set of attributes are used to predict the outcome of a classifier. For example, we can use the age, sex and weight of people within a dataset to predict whether someone (who is not in the dataset) has a bachelors degree or not. Doing this requires examining the 'entropy' of each partition, relative to some attribute. A partition in this case refers to how the dataset is grouped or "broken up" into smaller datasets, all of which consist of unique elements. For some partition $D_k$ and dataset $D$, where $D_k \subseteq D$, the entropy function $E$ is defined as follows

$$E(D_k) = -\frac{a}{a+b}log_2(\frac{a}{a+b}) - \frac{b}{a+b}log_2(\frac{b}{a+b})$$

if $log(0)_2$, then $E(D_k) := 0$

where $a$ is the number of individuals in partition $D_k$ that satisfies the classifier attribute and $b$ is the number of individuals in partition $D_k$ that do not satisfy the classifier attribute. Using entropy we can compute the information gain ($IG$) on dataset $D$ with $c$ many partitions such that $D_i \subset D$ is defined as follows,

$$IG(D) = E(D) - \sum_{i=1}^{c} E(D_i)$$

Typically, one can get many $IG$ values depending on how the $D$ is partitioned. Thus, the maximum $IG$ value of $D$ normally takes priority. For numerical attributes, partitions are usually defined through a 'split point'. A split point is a value which splits a single set into two. For instance, consider the interval $I = [a, b]$, where $a$ and $b$ are integers and $a \leq b$. Assume there exists a split point $c \in [a, b]$. The original set $I$ is now split into two, $I_1 = [a, c]$ and $I_2 = (c, b]$. The contents of $I_1$ and $I_2$ will determine the respective information gain. Thus information gain is highly dependent on the chosen split point.

**Example 2.3.2.** Let us assume we have the following dataset $D$

| ID | AGE | Education(classifier) |
|---|---|---|
| 1 | 22 | Yes |
| 2 | 18 | No |
| 3 | 26 | Yes |
| 4 | 23 | Yes |
| 5 | 22 | Yes |
| 6 | 31 | No |
| 7 | 20 | No |
| 8 | 26 | Yes |
| 9 | 26 | Yes |
| 10 | 19 | No |

Notice dataset $D$ contains the "Age" attribute, which we will call $A$, along with the classifier "Education". For this example let us assume the split point is $t = 22$. Given $t$, we partition $A$ into two sets $A_1$ and $A_2$, where $A_1$ contains the individuals that are 22 or older $A_1 = \{1, 3, 4, 5, 6, 8, 9\}$ and $A_2$ contains the individuals that are younger than 22, $A_2 = \{2, 7, 10\}$. We start by determining the entropy of attribute $A$ as $E(A) = -\frac{6}{6+4}log_2(\frac{6}{6+4}) - \frac{4}{6+4}log_2(\frac{4}{6+4}) = 0.97095$. Now we compute the entropy of each partition. $E(A_1, t) = -\frac{6}{6+1}log_2(\frac{6}{6+1}) - \frac{1}{6+1}log_2(\frac{1}{6+1}) = 0.50167$ and $E(A_2, t)$ is assumed to be zero since the $A_2$ is perfectly homogenized relative to the classifier. Complete homogenization of a dataset is assumed to have an entropy of 0. In general, any time we compute $log_2(0)$ is present in our computation the $IG$ for the respective dataset is assigned a value of zero. Finally we compute the information gain of $A$, $IG(A, t) = 0.97095 - (0.59167 + 0) =$

$0.37928$. If we assume that this information gain is maximum possible information among all possible split points, then $u(A, t) = 0.37928$ ■

## 2.4 Security Model

This section investigates the notion of security, both holistically and mathematically. We aim to investigate the concepts which will enable us to mathematically determine whether or not a particular protocol is secure or not.

### 2.4.1 Types of Security

There are three types of security: honest, semi-honest, and malicious. Each security type offers different assumptions regarding the behavior of each party. These assumed behaviors determine the security level of the protocol in question. The lowest level of security occurs when the parties assume honest behavior. For honest parties, they follow the protocol exactly as it is described. While the honest parties are executing the protocol, they do not attempt to learn anything from each other. This means that even if information is leaked and accessible, the parties will not be tempted to acquire that information. For security to be achieved in the honest setting, a protocol only needs to to be executable by the participating parties involved. However, for parties in the semi-honest setting their behavior is somewhat similar, but varies slightly. The main distinction between honest and semi-honest parties is that semi-honest parties actively seek new information about each other. For a protocol to be secure in the semi-honest setting, it needs to be both executable by all parties involved as well as not leak any information in the process. Following the execution of the protocol, the parties should not learn anything they did not know prior to the protocol execution.

If information is preserved throughout the duration of the execution, then the protocol is secure in the semi-honest setting. Finally, there are parties with malicious behavior. Malicious adversaries can essentially do as they want. They can deviate from the protocol, input false values, make no inputs at all, etc. Parties in the malicious-setting have the freedom to do whatever they feel like, given that the protocol does not have a mechanism to prevent deviant behavior. They will also attempt to learn new information about the participants throughout the execution of the protocol. A protocol is secure in the malicious setting if the protocol is both executable and does not leak any information, given that the parties will actively seek to deviate from the protocol. In this context, a protocol which is secure in the malicious case acquires the highest level of security. It may be possible to assume a variation of behaviors among different parties. For example, assume a protocol has three parties where one is honest and the other two are respectively semi-honest and malicious. In this case, the protocol could only be shown to be secure in the malicious setting. Security is hierarchal, where behavior of at least one party which corresponds to the highest security among all the parties will set the standard for security.

### 2.4.2 Computational Indistingushability

Our protocol is provably secure in semi-honest model for multiple parties. Semi-honest security relies on the concept of computational indistinguishability, where the information that $P_i$ sends or receives cannot meaningfully be distinguished from the information that $P_j$ sends or receives. In this section, we will highlight a few important mathematical components relevant to semi-honest security.

**Definition 2.4.1.** Computational Indistinguishability [19]

A probability ensemble defined to be a sequence of random variables. If one were to flip a fair coin, then the outcome of the coin represents a binomial distribution which consist of

1 (heads) or 0 (tails). Let $Z = z_1 z_2 \ldots z_n$ be the probability ensemble of the coin, where $z_i$ represents the outcome of the $ith$ coin flip. Two probability ensembles $X$ and $Y$ can be assigned a computational distance through the following function.

$$|Pr[D(X,\omega) = 1^*] - Pr[D(Y,\omega) = 1^*]| \tag{2.4}$$

where $\omega$ is the condition (or event) being tested and $1^*$ is a boolean value that indicates whether the condition had successfully occurred. $X$ and $Y$ are said to be computationally indistinguishable, or described equivalently as $X \overset{c}{\equiv} Y$, where for any function an efficient-algorithm $D$ there exists a negligible function $epsilon(n) > 0$, there exists some $n > N$ such that

$$|Pr[D(X,\omega) = 1^*] - Pr[D(Y,\omega) = 1^*]| < \epsilon(n) \tag{2.5}$$

where $N \in \mathbb{N}$ and $D$ is typically referred to as the 'distinguisher' ■

**Example 2.4.1.** The probability ensembles of $X$ and $Y$ are defined below, where we will assume $X$ represents a biased blue coin and $Y$ represents a biased red coin. $x_i$ and $y_i$ represent the respective outcomes of each coin flip, where the value 1 designates an outcome of heads and 0 designates tails. In this scenario let us define $\omega_1$ as representing a blue coin that outputs $\{1^n\}$ (all heads).

$$X(Blue) = \begin{cases} \frac{1}{3} & \mathit{if}\, x_i = 1 \\ \frac{2}{3} & \mathit{if}\, x_i = 0 \end{cases} \quad Y(Red) = \begin{cases} \frac{2}{3} & \mathit{if}\, y_i = 1 \\ \frac{1}{3} & \mathit{if}\, y_i = 0 \end{cases}$$

The distinguisher $D$ is defined below. $Z$ is also a probability ensemble which represents the outcome $\omega_1$

$$D(distinguisher) = \begin{cases} 1^*(success) & \textit{if } Z = \omega_1 \\ 0^*(failure) & \textit{else} \end{cases}$$

$|Pr[D(X, \omega_1) = 1^*] - Pr[D(Y, \omega_1) = 1^*]|$

$= |(\frac{1}{3})^n - 0|$

$= (\frac{1}{3})^n$

$\lim_{n\to\infty}(\frac{1}{3})^n = 0$ ■

Since the red coin can never yield a successful outcome (being blue, while having all heads), its respective probability is 0. The probability that the blue coin yields a successful output approaches zero as the number of coin flips increase, meaning the computational distance among the two coins becomes arbitrarily small. Although the computational distance in this instance is arbitrarily close , we cannot conclude that $X \overset{c}{\equiv} Y$. Why? $X$ and $Y$ can only be computationally indistinguishable if the computational distance is arbitrarily small for **any** distinguisher $D$. We only showed the equation is satisfied with one specific distinguisher function. Establishing computational indistinguishability requires quite a bit of effort. For this example, a simple way of distinguishing the coins is by color. One could construct $D_2(Z, \omega_2)$, where $\omega_2$ represents a blue coin that acquires an outcome of heads heads or tails. ■

### 2.4.3 Semi-honest Security

The formalization of semi-honest security involves inserting a theoretical simulator $S_j$ into protocol $\Pi$. The simulator wants to simulate communication between itself and $P_j$ with respect to $\Pi$. $S_j$ will communicate on the behalf of the other parties in such a manner that $P_j$ will not be suspicious of $S_j$'s messages. In other words, the messages that $S_j$ sends on the behalf of $P_k$ (some other party not $P_j$) should be indistinguishable relative to what $P_k$ would have actually sent. Another component we need to consider is a trusted third-party, $f$. The trusted third-party knows $\Pi$ and all its operations with respect to each party. In an ideal setting, each party $P_j$ will privately provide their initial-input $x_j$ to $f$. In return $f$ would privately reveal the final output $f_j(\vec{x}) = f_j(x_1, x_2, \dots, x_n)$ to $P_j$. In the literature, $f$ is commonly referred to as the 'functionality' within the ideal model. The ideal model differs significantly from the real model, where both models serve a distinct purpose. Within the real model, the parties conduct $\Pi$ among each other, with an initial input of values. In the ideal model, parties are conducting $\Pi$ with either the trusted third-party or the simulator, given that each party has an initial set of inputs. When the parties computes $\Pi$ with the trusted third-party, that can be loosely described as a perfectly secure procedure. If the final output in the real model is indistinguishable from the final output in the ideal model **and** $S_j$ can properly simulate messages to $P_j$ in an indistinguishable manner, then $\Pi$ will be deemed to be secure in the semi-honest setting [19]. The ideal model loosely represents an idealized execution of $\Pi$, with respect to security. If the ideal execution of $\Pi$ is indistinguishable from the real execution of $\Pi$, then $\Pi$ is secure. Prior to simulation $S_j$ has access to $P_j$'s initial input $x_j$, final output $f_j(\vec{x})$, and random tape $r_j^*$.

**Definition 2.4.2.** *Multiparty Semi-honest Security [19]:*

Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ be a probabilistic polynomial-time functionality and let $\Pi$

be a protocol where $f_j(\vec{x})$ denotes the $jth$ element of $f(\vec{x})$. Let $\Pi$ be a $n$ party protocol for computing $f$, where $n \geq 2$. Let the view of $P_j$ during an execution of protocol $\Pi$ on $\vec{x}$ denote $View_j^{\Pi}(\vec{x})$ as equivalent to $(x_j, r_j^*, m_{j1}, m_{j2}, \ldots, m_{jt})$, where $r_j^*$ represents the outcome of $P_j$'s internal random tape and $m_{ji}$ represents the $ith$ message that $P_j$ received. The output of $P_j$ during an execution of $\Pi$ on denoted $Output_1^{\Pi}(\vec{x})$ is implicit in the party's view of the execution. We say that $\Pi$ securely computes $f$ if there exist probabilistic polynomial time algorithm denoted $S_j$ for all $j \in [1, 2, \ldots, n]$, such that

$$\{(S_j(x_j, f_j(x, y)), f(\vec{x}))\} \overset{c}{\equiv} \{(View_1^{\Pi}(x_j, f_j(\vec{x})), Output^{\Pi}(\vec{x}))\}$$

where $\overset{c}{\equiv}$ denotes computational indistinguishability and $\vec{x} = (x_1, x_2, \ldots, x_n)$. ∎

Down below, we will detail every component of the above equation. When we say "real protocol" we only are referring to an execution of protocol $\Pi$ with respect to the parties $P_1, P_2, \ldots, P_n$. An execution of $\Pi$ with respect to the functionality $f$ or the simulator $S_j$, can be loosely described as the "ideal protocol". A simulation proof basically compares the real protocol with the ideal protocol to confirm or deny the security level of $\Pi$.

- General Notation of Multiparty Semi-honest Security

  1. $S_j$ is a simulator that attempts to simulate the roles of all parties that are not $P_j$. We will describe all parties being simulated by $S_j$ as $P_k$. $S_j$ wants to deceive $P_j$ by making it think that it is communicating with the other parties. Prior to simulation $S_j$ is provide access to $P_j$'s initial input $x_j$, final output $f_j(\vec{x})$, and its random tape $r_j^*$.

2. $View_j^{\Pi}(x_j, f_j(\vec{x})) = (x_j, r_j^*, m_{j1}, m_{j2}, \ldots, m_{jt})$, represents the values that are sent to $P_j$ from $P_k$ throughout the execution of protocol $\Pi$. The view is not influenced by the simulator. However, the view will be influenced by all the parties within the real protocol.

   – $x_j$ is the initial inputs that $P_j$ receives prior to execution of the protocol.

   – $m_{ji}$ is the ith message that $P_j$ receives from an entity that does not include itself.

   – $r_j^*$ is the random-tape of $P_j$. Random-tape records the outcome of every action done by $P_j$ which depends on chance (outcome of a coin flip, dice roll, etc.).

3. $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x}))$ is the output of the 'ideal functionality' $f$. $f$ takes an initial input from each party, described as $\vec{x}$. $f$ then complies the final output $f(\vec{x})$ with respect to $\Pi$, while internally and privately executing the intermediate steps of $\Pi$. $P_j$ will then privately acquire the output $f_j(\vec{x})$ from $f$. The output of $f$ is assigned without the influence of a simulator. Excluding the initial inputs provided, the output for $f$ is assigned without the influence of the parties.

4. $Output^{\Pi}(\vec{x}) = (Output_1^{\Pi}(\vec{x}), Output_2^{\Pi}(\vec{x}), \ldots, Output_n^{\Pi}(\vec{x}))$ describes the final output of each respective party following a complete execution of $\Pi$. $P_j$'s final output is described as $Output_j^{\Pi}(\vec{x})$. The final output value may or may not be private, depends on the instruction of $\Pi$. The final output value is derived without the influence of a simulator. However, the final output will be influenced by all parties involved since $Output^{\Pi}(\vec{x})$ is a direct reflection of the real protocol.

5. $S_j(x_j, f_j(x, y)$ represents the message(s) that $S_j$ sends to $P_j$ throughout the execution of $\Pi$. The message(s) sent by $S_j$ depends on $P_j$'s initial input $x_j$ in $\Pi$ as well as $P_j$'s final output $f_j(x, y)$ in $\Pi$.

6. If the simulator $S_j$ can simulate output messages that are indistinguishable from the view of $P_j$ (the messages it receives) in the real protocol, while at the same time the functionality's final output $f(\vec{x})$ is indistinguishable from the final output of the real protocol, then $\Pi$ is secure in the semi-honest setting.

$S_j$ is a simulator attempting to execute protocol $\Pi$ on the behalf of $P_k$, given that it knows $P_j$'s initial-input $x_j$, final output $f_j(\vec{x})$, and random tape $r_j^*$. At the same time, the trusted third-party $f$ outputs a final output of $\Pi$ to each of the parties when given the initial-input of each party $\vec{x}$. If a simulator can indistinguishably simulate the message(s) that each party would have sent relative to the real protocol and the final output of $f$ is indistinguishable from the final output of the real protocol, then $\Pi$ is secure. However, if such a situation is not achievable, then this implies $\Pi$ inadvertently leaks extra information during its execution and is hence insecure [19].

# Chapter 3

## LITERATURE REVIEW

The literature review is organized by topic, where we detail various methods involved in constructing a dataset as well as various methods relating to data publishing and data mining. From there we compare and contrast the protocols within the academic literature with our own(See Table 3.1). There are two distinctions when it comes to secure computation of datasets: single-party and multiparty. Single-party computation suggests there is only a single-party that computes a dataset in a secure manner. There have been a few single-party algorithms that securely derives new datasets with respect to classification analysis [26] [10] [16]. Each algorithm uses a 'top-down specialization' to organize and classify records contained in a particular dataset, a method which we similarly implemented. Specifically, Fung $et.al$ [10] and Mohammed $et.al$ [26]$et.al$ organize attributes into their respective 'taxonomy' (See Figure 5.1). The taxonomy of an attribute (like 'Education') contains the possible value-types (like 'Any Education', 'College', 'No College', etc.) that the attribute can acquire. Each taxonomy is organized by generality, where the most general value-type ('Any Education') would be located at the top of the taxonomy and the other specialized value-types ('College', 'No College') are located below. On the other hand, a multiparty scheme deals with two or more parties communicating in a secure manner such that multiple datasets can be integrated into a single dataset [27] [28] [25]. Each respective scheme similarly uses classification methods to organize all the records into a single dataset. One can achieve different objectives even with similar classification schemes. For instance, through classification analysis, Mohammed $et.al$ [27] [28] was able to derive the dataset for the purpose of data mining, where the data miner and the dataset owner(s) communicate directly to derive the final dataset. Another paper published by Mohammed $et.al$ [25], uses a similar classification scheme for the purpose of data publishing, where the data miner extracts information from the final dataset without needing to consult the dataset owner(s). Since there are multiple parties, cryptographic methods are employed to guarantee secure transfer of information. Depending on the scheme, these cryptographic methods are applicable to two parties [4] [1] [20] or party numbers exceeding two [37] [12]. Ultimately, the versatility of the general cryptographic scheme will dictate the set amount of parties that can conduct secure communication. That is why we created multiple cryptographic protocols with the ability of fostering secure communication between two or more parties. By doing so, secure computation will not be inhibited by the amount of parties within the MAIN protocol 5.1.

## 3.1 Privacy-preserving Data Processing

For an interactive dataset, each dataset is owned by a private entity. In this case, the data miner is looking to derive information from a dataset but only through the permission of the dataset owner. The data miner then poses a series of questions (known as queries) about the dataset using some private mechanism. The interactive approach is known as **privacy-preserving data mining**(PPDM) [6] [18] [31]. For example, Clifton [6] uses four private mechanisms (Secure sum, Secure Set Union, Secure Size Set Intersection, and Scalar Product Protocol), designed with the purpose of data mining. These four mechanisms also functions as a cryptographic mechanism, where sensitive information is masked throughout the protocol. Since the data miner has to request information from the dataset owner, the dataset owner must personally account for the computational cost of each query. Although the dataset owner has complete control of the information that the data miner can extract, the owner may need to invest in a state-of-the-art device that can easily handle high computation loads.

For a non-interactive dataset, the data requested is first anonymized by the owner then released to the data miner. Once the anonymized data is released to the miner the owner no longer has any control over the data. This non-interactive approach is also known as **privacy-preserving data publishing**(PPDP) [9] [23] [33]. Since the dataset is released, the owner avoids the responsibility of accounting for the computational complexity associated in extracting information from the dataset. The complexity costs now rest on the data miner. Although the owner will no longer need to execute computation following a data publishing, the fact that the dataset is completely available to the miner can cause concern. If the dataset was derived in a way that makes it vulnerable in an unforeseeable manner, whether mathematically or errors relating to software/hardware implementation, then the privacy of the individuals in the dataset may be in jeopardy. Since data publishing is open to a wider array of attacks by an adversary, the owner must rely on a protocol that is provably secure prior to execution.

*Single-Party and Multiparty Models.*

In the multiparty setting, the objective is to integrate datasets owned among various owners, while not revealing information to other parties. The integrated dataset will be used by the data miner to acquire information through queries [36] [32] [8]. For clarity, 'multiparty' in the data mining perspective usually refers to at least two or more parties. This should not be confused with our papers standard of 'multiparty', which is defined of having at least three or more parties. For a single party setting, there is only one dataset owned by a single owner [5] [34] [11]. The data miner will then extract information from the dataset through queries. In either setting, the security of the protocol must be established prior to execution.

*Vertical and Horizontal Partitioning.*

When data processing scheme consists of two or more parties we consider how the datasets are distributed and organized. When each party holds its own unique set of attribute values for all individuals while other parties hold their other unique set of attribute values for the same individuals, the datasets that are comprised of these unique attribute values are 'vertically partitioned' [22](See Fig 5.2.4). Vertically partitioned datasets are most applicable when parties typically have unique information about the individuals that they are unwilling to reveal or share. On the other hand, parties can have access to the same attribute features while the individuals in each dataset are unique. Such an arrangement is known as a "horizontal partition" [35] [15]. Horizontal partitioning is useful when the parties have access to the same type of data that corresponds to different individuals. For our proposed algorithm, each dataset will be vertically partitioned (See Table 2.1).

## 3.2 Multiparty Privacy-preserving Data Publishing

Our protocol specifically focuses on a multiparty, non-interactive datasets. This was chosen because it was the most efficient method of integrating and releasing data in a secure and private manner. Down below we highlight different means of acquiring a multiparty non-interactive dataset. Mohammed $et.al$ [25] uses a multi-layered protocol which allows a data miner to analyze a intergrated dataset with high data-utility, while also perturbing the output of the integrated dataset in a differentially-private manner. Their scheme allows the true contents of the integrated dataset to be hidden, while also allowing the data miner to extract data analytics which are statistically close to the integrated dataset. Pettai $et.al$ [29] incorporates a secure and differentially-private release of data similar to [25]. However the dataset can be derived from $n$ many parties instead of two. This is done through the use of secret-shares between the parties. The dataset is secret-shared between the computing parties, and so are the parameters of the query. A secret-share owned by a party is a value that is unknown to all other parties. The secret-shares of all the parties can collectively operate in such a way that they derive a single desired value. Each query posed has a corresponding answer, which is also comprised of secret-shares. After the data miner makes the query, the secret-shares with respect to the answer will be securely distribute the the other parties. The shares of the answer will then be sent back to the data miner to be recombined and analyzed in a differentially-private manner. Kairouz $et.al$ [14] approach secure and differentially-private mechanisms from a theoretical standpoint. Their protocol focuses on the scenario where each party possesses a single bit of information. With multiple bits of information among each of the parties, they proved the existence of a differentially-private protocol with a fixed accuracy implies the existence of a protocol with the same level of privacy and same level of accuracy for a specific functionality that only depends on one bit of each of the parties.

Table 3.1: Comparative evaluation of main features in related query processing approaches (properties in columns are positioned as beneficial with fulfilment denoted by ● and partial fulfilment by ○)

| Approach | Party Size | | | Privacy Model | | Data Processing | | | |
|---|---|---|---|---|---|---|---|---|---|
| | One | Two | Multi | Differential-Privacy | Partition-Based | Vertical-Partition | Horizontal-Partition | PPDM | PPDP |
| Diff-Gen [26] | ● | | | ● | | ● | | | ● |
| Two-Party Diff-Gen [25] | ○ | ● | | ● | | ● | | | ● |
| Kaiorouz [14] | | | ● | ● | | ○ | ○ | | ● |
| Pettati [29] | | | ● | ● | | | ● | | ● |
| Mohammed [27] | ● | | | | ● | | | ● | |
| Fung [10] | ● | | | | ● | | | | ● |
| Our Algorithm [Protocol 5.1] | ○ | ● | ● | ● | | ● | | | ● |

# Chapter 4

# SECURE DISTRIBUTED MULTIPARTY EXPONENTIAL MECHANISM

## 4.1 Protocol Overview

The objective of the Multiparty Exponential Mechanism is to probabilistically select an attribute with the best data utility. We want our mechanism to output the winning attribute $A^w$ in a secure and private manner, with the cooperation of at least two parties. To maintain security we use our cryptographic primitives, ElGamal and RVP, to produce a secure means of communication among the parties. To maintain privacy we use the definition of the exponential mechanism to select $A^w$ in a differentially-private manner. The following section will go into detail about how this process is achieved. It will first describe the algorithm in its entirety, followed by the algorithm itself. From there, each process of the Multiparty Exponential Mechanism 4.1 will be documented and detailed with respect to when it appears in the algorithm.

Each party $P_j$ begins with their respective datasets $D_j$, which contains: attributes $A_k$, a classifier $A^c$, identification for the individuals $ID$, and records $\mathcal{R}_j$ . Using a predesignated utility-function $u$, $P_j$ uses their dataset $D_j$ and an attribute they own $A_k$ as input to privately derive a utility score $U_k$, where $u(D_j, A_k) = U_k$. Each party privately constructs their respective attribute-score pair described as $(A_k, U_k)$ for each attribute they own. From there, $P_j$ uses every $U_k$ they own to derive its respective $\sigma_j$, where $\sigma_j$ is a fixed value. $P_j$ then derives their respective $L_j$, where the mechanism used to create $\sigma_j$ is similarly used to create $L_j$. However, $L_j$ differs from $\sigma_j$ by the fact that $L_j$ is not a fixed value and can be increased if a winning-attribute is not declared within a procedural loop. Using their respective $\sigma_j$, the parties conduct the extended Random Value Protocol to derive their respective $R_j$, where $R_j$ is a fixed value. $P_j$ privately derives the following pair $(L_j, R_j)$, then encrypts each value described as $(\llbracket L_j \rrbracket, \llbracket R_j \rrbracket)$. We assume the sum of all $L_j$'s be equal to $L$ and the sum of all $R_j$'s equal to $R$. The parties collectively and securely derive $\llbracket R \rrbracket$ and $\llbracket L \rrbracket$ through homomorphic addition. From there, the parties determine the encrypted value $\llbracket R - L \rrbracket$ in a secure and collective fashion. The parties collectively and securely decrypt $\llbracket R - L \rrbracket$, then determine if $R - L \leq 0$. If true, an attribute $A_k$ in some dataset is declared a winning-attribute, where $A_k \rightarrow A^w$. Else if $R - L > 0$, then no winning attribute is declared. If no winning-attribute is declared, then some particular $L_j$ is increased and the parties re-verify $R - L \leq 0$ in a similar fashion. Based on how $R$ and $L$ are designed, a winning-attribute will eventually be declared. This protocol will have at least one loop execution and at most $z$ loop executions, where $z$ is the total amount of attributes distributed amongst the parties. The overall objective of the protocol is to probabilistically select a single attribute $A_k$ among the multiple parties that has highest data-utility (quantified as $U_k$) in a secure and differentially-private manner. It should be stressed that the values of $U_k, \sigma_j, L_j$, and $R_j$ are only know to $P_j$, however $R$ and $L$ are values that remain unknown to all parties. Down below we will present our protocol which securely selects an attribute among multiple parties in a differentially-private manner.

**Multiparty Exponential Mechanism**

**Input:** Set of pairs $(A_i, U_i) : 1 \leq i \leq m_n$ where $U_i$ is the utility score of attribute $A_i$, Privacy Budget $\bar{\epsilon}$

**Output:** Winner attribute $A^w$

1. Each party $P_j : 1 \leq j \leq n$ computes his share $R_j$ with all other parties as follows:
   (a) Each party $P_j$ computes the total of the exponential mechanism of all attribute it owns:
   $$\sigma_j := \sum_{k=m_{j-1}+1}^{m_j} exp(\frac{\bar{\epsilon} \cdot U_k}{2.\triangle u})$$
   (b) Each pair $(P_i, P_j) : i < j$ and $1 \leq i, j \leq n$ jointly execute the RVP protocol [4], where $P_i$ inputs $\sigma_i$ and generates (sub) share $R_{i,j}$, and $P_j$ inputs $\sigma_j$ and generates (sub) share $R_{j,i}$ such that $R_{i,j}, R_{j,i} \in [0, \sigma_i + \sigma_j]$.
   (c) Each party $P_j$ adds all its (sub) shares to compute its final random share $R_j$:
   $$R_j := \frac{1}{2(n-1)} \sum_{i=1}^{n} R_{j,i} \quad : \quad i \neq j$$
2. Initialize the party index to the first party: $p = 1$
3. For $i = 1$ to $m_n$
   (a) If $i > m_p$ then $p := p + 1$
   (b) Each party $P_j : 1 \leq j < p$ computes:
   $$L_j := \sum_{k=m_{j-1}+1}^{m_j} exp(\frac{\bar{\epsilon} \cdot U_k}{2.\triangle u})$$
   (c) The active party $P_p$ computes:
   $$L_p := \sum_{k=m_{p-1}+1}^{i} exp(\frac{\bar{\epsilon} \cdot U_k}{2.\triangle u})$$
   (d) Each party following $P_p$ ($P_j : p < j \leq n$) computes: $L_j = 0$.
   (e) All parties jointly execute Protocol 4.2 to determine if attribute $A_i$ is a winner, where each party $P_j$ inputs to the protocol $(R_j, L_j)$.
      i. If the return value of Protocol 4.2 is $\gamma \leq 0$, then $A_i \rightarrow A^w$ and exit.
      ii. Otherwise, if $i = m_{n-1}$, then $A_{i+1} \rightarrow A^w$ and exit.

Protocol 4.1: Multiparty Exponential Mechanism

**Distributed Comparison**
**Input:** Integers $L_j$ and $R_j$ for each party $P_j : 1 \leq k \leq n$
**Output:** Comparison result $\gamma$
**Initial Step:** All parties agree on a large prime $p$ and generator $g$ such that $g \in Z_p^*$.

1. Each party $P_j$ randomly select private key $a_j \in_R [2, \ldots, p-2]$, and then generates its public key $A_j := g^{a_j} \ mod\, p$.
2. All parties collectively compute the public group-key $A := \prod_{k=1}^{n} A_j = g^{\Sigma_{k=1}^{n} a_j} \ mod\, p$.
3. Each party $P_j$ privately chooses ephemeral keys $r_j$, $\acute{r}_j$ from $[2, \ldots, p-2]$ and then encrypts $R_j$ and $L_j$:

$$[\![R_j]\!] = (A^{r_j} \cdot g^{R_j} \ mod\, p,\ g^{r_j} \ mod\, p) \quad , \quad [\![L_j]\!] = (A^{\acute{r}_j} \cdot g^{L_j} \ mod\, p,\ g^{\acute{r}_j} \ mod\, p)$$

4. All parties jointly perform the following:
   (a) Homomorphically compute the total sum of $R_j : 1 \leq j \leq n$:

$$[\![R]\!] = \prod_{j=1}^{n} [\![R_j]\!] = (A^{\Sigma_{j=1}^{n} r_j} \cdot g^{\Sigma_{j=1}^{n} R_j} \ mod\, p,\ g^{\Sigma_{j=1}^{n} r_j} \ mod\, p)$$

$$[\![R]\!] = (A^{r} \cdot g^{R} \ mod\, p,\ g^{r} \ mod\, p)$$

   (b) Homomorphically compute the total sum of $L_j : 1 \leq j \leq n$:

$$[\![L]\!] = \prod_{j=1}^{n} [\![L_j]\!] = (A^{\Sigma_{j=1}^{n} \acute{r}_j} \cdot g^{\Sigma_{j=1}^{n} L_j} \ mod\, p,\ g^{\Sigma_{j=1}^{n} \acute{r}_j} \ mod\, p)$$

$$[\![L]\!] = (A^{\acute{r}} \cdot g^{L} \ mod\, p,\ g^{\acute{r}} \ mod\, p)$$

   (c) Homomorphically subtract $L$ from $R$:

$$[\![R - L]\!] = [\![R]\!]/[\![L]\!] = (A^{r-\acute{r}} \cdot g^{R-L} \ mod\, p,\ g^{r-\acute{r}} \ mod\, p)$$

   (d) Jointly decrypt $[\![R - L]\!]$:

$$\alpha = (A^{r-\acute{r}} \cdot g^{R-L}) / \prod_{j=1}^{n} (g^{r-\acute{r}})^{a_j} = g^{R-L} \ mod\, p.$$

5. Apply discrete-log algorithm on $\alpha$ to compute $\gamma = R - L$.
6. Return $\gamma$.

Protocol 4.2: Distributed Comparison

## 4.2 Protocol Details

The protocol selects a single attribute $A_k$, among the party-members. The selected attribute is known as the $qth$ winning-attribute $A_q^w$. Each attribute has a corresponding utility-score $U_k$, creating the attribute-score pair $(A_k, U_k)$. Although there are 'n' parties, there may be more than 'n' attributes. However, there can never be less attributes than there are party-members. For this paper we will assume there exists '$z$' attribute-scores pairs, $\{(A_1, U_1), (A_2, U_2), ..(A_z, U_z)\}$, which are distributed among the parties. Given the score of each attribute, the exponential mechanism will use $U_k$ to select an $A_k$ owned by $P_j$. Down below, $A_q^w$ is selected with the following probability, where $\Delta u$ is the sensitivity of the chosen utility function.

$$\frac{exp(\frac{\epsilon \cdot U_k}{2\Delta u})}{\sum_{i=1}^{z}(\frac{\epsilon \cdot U_i}{2\Delta u})} \tag{4.1}$$

*Computation*

A basic implementation of the exponential mechanism begins with the unit-interval [0,1]. First partition the unit-interval into $z$ sub-intervals, where the length of the $k$th sub-interval is equivalent to equation (1). Since the length of the $k$th sub-interval is determined by $U_k$, we can assign attribute-score pair $(A_k, U_k)$ to that sub-interval, where $(A_k, U_k)$ is owned by some party $P_j$. We then sample a random number uniformly from [0,1]. The random value will fall within one of the many possible $kth$ sub-intervals, Since the $kth$ sub-interval corresponds to $(A_k, U_k)$, $A_k$ will then be declared as the $qth$ winning-attribute $A_q^w$. Although this method is relatively simple and intuitive, this scheme unfortunately requires a "secure division protocol", which does not currently exists for our specific purposes. Instead we use a method similar to what was mentioned above, where we partition the interval $[0,\sum_{i=1}^{z} exp(\frac{\epsilon \cdot U_i}{2\Delta u})]$ into $z$ sub-intervals instead. Similarly, the $kth$ sub-interval uniquely corresponds to some attribute-score pair $(A_k, U_k)$. In this case, each $kth$ sub-interval is equal to length $exp(\frac{\epsilon \cdot U_k}{2\Delta u})$. Finally, we sample a random number uniformly from the interval $[0,\sum_{i=1}^{z}(\frac{\epsilon \cdot U_i}{2\Delta u})]$. This random number will be contained among one of the $kth$ sub-intervals, which corresponds to the $(A_k, U_k)$ attribute score-pair. We declare $A_k$ as the $qth$ winning-attribute $A_q^w$. By doing this computation $A_q^w$ would be selected in a manner consistent with the exponential mechanism. As a result, $A_q^w$ was selected in a differentially private manner.

### 4.2.1 Indexing Attributes

Assume there are $n$ parties and $z$ attributes distributed among the parties. Party $P_j$, will own a non-empty set $\hat{\Phi}_j$ which contains $M_j$ many attributes. Party $P_j$ indexes the first attribute they own as $m_{j-1} + 1$, where $m_0 := 0$. $P_j$ indexes the second attribute they own as $m_{j-1} + 2$. This incrementation continues until $P_j$ indexes their last attribute as $m_j$. One could easily convince themselves that $P_j$ owns $M_j = m_j - (m_{j-1} + 1) + 1$ many attributes. For clarity of notation, we see that $P_1$ owns the following set of attributes, $\hat{\Phi}_1 = \{A_1, A_2, \ldots, A_{m_1}\}$. Similarly, $P_2$ owns $\hat{\Phi}_2 = \{A_{m_1+1}, A_{m_1+2}, \ldots, A_{m_2}\}$. In general $P_j$ owns the following attributes, $\hat{\Phi}_j = \{A_{m_{j-1}+1}, A_{m_{j-1}+2}, \ldots, A_{m_j}\}$, where $|\hat{\Phi}_j| = M_j$. If $P_j$ owns only one attribute, then $m_j = m_{j-1} + 1$. Thus from an outside perspective (the reader's perspective), there exists a set $\hat{\Phi}$ which list all the set of attributes owned by all parties, $\hat{\Phi} := \{\hat{\Phi}_1, \hat{\Phi}_2, \ldots, \hat{\Phi}_n\}$. However from $P_j$'s perspective, they are only aware of the $\hat{\Phi}_j$ it privately constructed. Since each party has its own $\hat{\Phi}_j$ we can specifically select the attributes contained in $\hat{\Phi}_j$ in an algorithmic manner.

### 4.2.2 Indexing Attribute-Score Pairs

Recall that each attribute $A_k$ corresponds to a utility-score $U_k$, described as the attribute-score pair $(A_k, U_k)$. We will extend $\hat{\Phi}_j$ by incorporating the utility score $U_k$. In this case, $\Phi_j := \{(A_{m_{j-1}+1}, U_{m_{j-1}+1}), (A_{m_{j-1}+2}, U_{m_{j-1}+2}), \ldots, (A_{m_j}, U_{m_j})\}$, where $|\Phi_j| = M_j$. And in similar fashion, $\Phi := \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$. This notation is seen in summation limits in Protocol 4.1, where each party $P_j$ derives their $\sigma_j$ and $L_j$ from $\Phi_j$.

### 4.2.3 Extended RVP & R-Shares

The purpose of Random Value Protocol (RVP) is to acquire a Random Number R within a closed interval. We first designate two parties to conduct the RVP protocol, $P_a$ and $P_b$. Each party will own a set of attribute-score pairs, where $P_a$ owns $\Phi_a$ and $P_b$ owns $\Phi_b$. Each party then computes $\sigma_a := \sum_{i=m_{(a-1)+1}}^{m_a} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$, $\sigma_b := \sum_{i=m_{(b-1)+1}}^{m_b} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$ respectively. RVP takes $\sigma_a$ and $\sigma_b$ as input, RVP($\sigma_a$,$\sigma_b$), then outputs random-value shares $R_a$ and $R_b$ to the respective party. Once the shares are securely distributed we acquire three key properties: RVP$(\sigma_a, \sigma_b) \rightarrow (R_a, R_b)$, $R = R_a + R_b$, and $R_a, R_b, R \in [0, \sigma_a + \sigma_b]$. $P_a$ knows the value of $R_a$, but is unaware of the value of both $R_b$ and $R$. Similarly, $P_b$ knows the value of $R_b$, but is unaware of the value of both $R_a$ and $R$. If there were only two parties in the protocol, the RVP would be as simple as letting $a = 1$ and $b = 2$. However, since we are focusing on a multiparty setting we must extend the protocol to accommodate $n$ parties while using the three key properties to direct our intutions. For two distinct parties ($P_a$ and $P_b$) among $n$ parties, we apply the following operation RVP($\sigma_a$,$\sigma_b$). However, we slightly adjust the mapping as follows, RVP$(\sigma_a, \sigma_b) \rightarrow (R_{(a,b)}, R_{(b,a)})$, where $R_{(a,b)}$ is $P_a$'s random-value sub-share, $R_{(b,a)}$ is $P_b$'s random-value sub-share, and $(R_{(a,b)}, R_{(b,a)}) = (R_a, R_b)$. Since $P_a$ and $P_b$ are really just operating under the original RVP, their sub-shares will inherit the following property, $R_{(a,b)}, R_{(b,a)} \in [0, \sigma_a + \sigma_b]$. Our first objective is to guarantee that each party $P_i$ has a respective share $R_i \in [0, \sum_{k=1}^{n} \sigma_k]$. However, if each party conducts the RVP amongst themselves, that implies $P_j$ will have $n-1$ sub-shares to work with. If we add all of $P_i$'s $n-1$ sub-shares we can acquire the share $R_i'$. However, $R_i' := \sum_{j=1}^{n} R_{i,j}$ $i \neq j$, will be too large where $R_i' \notin [0, \sum_{k=1}^{n} \sigma_k]$. We can avoid this by multiplying $R_i'$ by a constant. If $R_i := \frac{1}{2(n-1)} R_i'$ and $R := \sum_{k=1}^{n} R_k$, then we acquire the following property: $R_i, R \in [0, \sum_{k=1}^{n} \sigma_k] \; \forall \; i \in \{1, \ldots, n\}$

**Theorem 4.2.1.** *The functionality of the two-party Random Value Protocol can be extended to accomadate $n$ parties, where $n \geq 2$*

*Proof.* :

To extend the Random Value Protocol (RVP) the party does pairwise RVP with each other. When $P_k$ does an RVP operation with $P_j$, $P_k$ receives $R_{k,j} \in [0, \sigma_k + \sigma_j]$ and $P_j$ has $R_{j,k} \in [0, \sigma_j + \sigma_k]$. From there, we have all we need to extend RVP which will go as follows:

Given $R_{(i,j)} \in [0, \sigma_i + \sigma_j]$ where $i \in \{1, \ldots, n\}$, assume the following two values:

1.) $R_i' := \sum_{j=1}^{n} R_{(i,j)} = \sum_j R_{(i,j)}$, where $R_{(i,i)} = 0$

$\implies R_i' = \sum_j R_{(i,j)}$, where $j \in \{1, \ldots, n\} \setminus \{i\}$

2.) $R' := \sum_{i=1}^{n} R_i'$

The sum in (1) represents exactly $n-1$ $R_{(i,j)}$'s, where we purposefully avoided $R_{(i,i)}$ . Based on the interval $[0, \sigma_i + \sigma_j]$, $R_{(i,j)}$ corresponds to exactly one $\sigma_i$ and exactly one $\sigma_j$, where $j \neq i$. Thus, party $P_i$ can create a new interval whose length is equivalent to $n-1$

$\sigma_i$'s, plus all the $\sigma_k$'s from the other party members $P_k$, $k \neq i$

$R'_i \in [0, (n-1)\sigma_i + \sum_j \sigma_k]$

$R'_i \in [0, (n-2)\sigma_i + \sum_{i=1}^n \sigma_i]$

$R' \in [0, \sum_{i=1}^n (n-2)\sigma_i + \sum_{i=1}^n \sum_{i=1}^n \sigma_i]$

$R' \in [0, \sum_{i=1}^n (n-2)\sigma_i + n \sum_{i=1}^n \sigma_i]$

$R' \in [0, \sum_{i=1}^n (2n-2)\sigma_i]$

$R' \in [0, (2n-2) \sum_{i=1}^n \sigma_i]$

$R' \in [0, 2(n-1) \sum_{i=1}^n \sigma_i]$

$R := \frac{R'}{2(n-1)}$

$R \in [0, \frac{1}{2(n-1)}(2(n-1) \sum_{i=1}^n \sigma_i)]$

$R \in [0, \sum_{i=1}^n \sigma_i]$

Conforming to the range of the original RVP protocol for all $n \in \mathbb{N}$ greater than 1.

$\square$

**Example 4.2.1.** *:*

*Let n = 3*

*RVP*$(\sigma_1, \sigma_2) \to (R_{(1,2)}, R_{(2,1)})$

*RVP*$(\sigma_1, \sigma_3) \to (R_{(1,3)}, R_{(3,1)})$

*RVP*$(\sigma_2, \sigma_3) \to (R_{(2,3)}, R_{(3,2)})$

$R'_1 = R_{(1,2)} + R_{(1,3)}$

$R_{(1,2)} \in [0, \sigma_1 + \sigma_2]$, $R_{(1,3)} \in [0, \sigma_1 + \sigma_3]$ *($P_1$'s sub-shares)*

$R'_2 = R_{(2,1)} + R_{(2,3)}$

$R_{(2,1)} \in [0, \sigma_2 + \sigma_1]$, $R_{(2,3)} \in [0, \sigma_2 + \sigma_3]$ *($P_2$'s sub-shares)*

$R'_3 = R_{(3,1)} + R_{(3,2)}$

$R_{(3,1)} \in [0, \sigma_3 + \sigma_1]$, $R_{(3,2)} \in [0, \sigma_3 + \sigma_2]$ *($P_3$'s sub-shares)*

$R'_1 \in [0, 2\sigma_1 + \sigma_2 + \sigma_3]$ *(range of $P_1$'s share)*

$R'_2 \in [0, \sigma_1 + 2\sigma_2 + \sigma_3]$ *(range of $P_2$'s share)*

$R'_3 \in [0, \sigma_1 + \sigma_2 + 2\sigma_3]$ *(range of $P_3$'s share)*

$R' \in [0, 4\sigma_1 + 4\sigma_2 + 4\sigma_3]$ *(range of random value $R'$)*

$R = \frac{R'}{4} = \frac{R'}{2(n-1)}$

$R \in [0, \sigma_1 + \sigma_2 + \sigma_3]$ *(range of random value $R$)*

$R \in [0, \sum_{i=1}^{3} \sigma_i]$ ■

**Example 4.2.2.** *:*

Assume there are three parties $P_1$, $P_2$, and $P_3$. Also assume $P_1$ owns $\Phi_1 = \{(A_1, U_1), (A_2, U_2), (A_3, U_3)\}$, $P_2$ owns $\Phi_2 = \{(A_4, U_4)\}$, and $P_3$ owns $\Phi_3 = \{(A_5, U_5), (A_6, U_6)\}$. Each party uses their respective attribute-score pairs $(A_k, U_k)$ to derive $\exp(\frac{\epsilon \cdot U_k}{2\Delta u})$ for each $U_k$ they own. $P_1$ derives the following three values: 234.562, 12.523, and 232.352 (in that order). $P_2$ acquires 232.378, while $P_3$ acquires 24.398 and 2.193 (in that order). RVP only takes input with integer values, thus the parties initially agree on the value $a \in \mathbb{N}$, then multiply their values by $10^a$, followed by the floor function. If the parties agree on $a = 0$, $P_1$ now has the the following values: 234, 12, and 232. $P_2$ has 232, while $P_3$ has 24 and 2. From there, the parties privately and respectively derive $\sigma_1 = 478 = 234 + 12 + 232$, $\sigma_2 = 232$, and $\sigma_3 = 26 = 24 + 2$. $P_1$ conducts the RVP with

$P_2$ and $P_3$ separately, acquiring a random share $R'_1$ from the interval $[0, 2\sigma_1 + \sigma_2 + \sigma_3] = [0, 1214]$. $P_2$ similarly conducts the RVP with $P_1$ and $P_3$ separately, acquiring its random share $R'_2$ from the interval $[0, \sigma_1 + 2\sigma_2 + \sigma_3] = [0, 968]$. Likewise, $P_3$ acquires its random share $R'_3$ from the interval $[0, \sigma_1 + \sigma_2 + 2\sigma_3] = [0, 762]$. Given that $R' = R'_1 + R'_2 + R'_3$, we can see $R' \in [0, 2944]$. However the objective is to collectively derive the random values $R, R_1, R_2$, and $R_3$ which lie in the interval $[0, \sigma_1 + \sigma_2 + \sigma_3] = [0, 736]$. This is achievable if for $n = 3$, $R = \frac{R'_1}{2(n-1)} + \frac{R'_2}{2(n-1)} + \frac{R'_3}{2(n-1)} \in [0, 736]$. ■

### 4.2.4 L-Shares

Recall party $P_j$ owns $\Phi_j$, the set which contains all of its attribute-score pairs. $P_j$ uses $\Phi_j$ to derive $\sigma_j = \sum_{i=m_{(j-1)}+1}^{m_j} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$. We will similarly use $\Phi_j$ to derive $L_j$. The difference between $\sigma_j$ and $L_j$ is $\sigma_j$ is a constant, while $L_j$ can vary depending on the amount of loops executed within Protocol 4.1. Per loop, $L_j$ can acquire one of the values from the following set $\{0, \sum_{i=m_{(j-1)}+1}^{m_{(j-1)}+1} exp(\frac{\epsilon \cdot U_i}{2\Delta u}), \sum_{i=m_{(j-1)}+1}^{m_{(j-1)}+2} exp(\frac{\epsilon \cdot U_i}{2\Delta u}), \ldots, \sigma_j\},$. We can think of $L_j$ as a sub-summation of $\sigma_j$, where both formulas are fundamentally the same except the upper summation-limit of $L_j$ does unit increments from $m_{(j-1)} + 1$ to $m_j$. If $L_j$ is zero and it changes value following a loop, then the value will equal $\sum_{i=m_{(j-1)}+1}^{m_{(j-1)}+1} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$. If $L_j \notin \{0, \sigma_j\}$, and $L_j$ changes value following a loop, then the upper summation-limit will increment by one, or in other words $\sum_{i=m_{(j-1)}+1}^{m_{(j-1)}+x} exp(\frac{\epsilon \cdot U_i}{2\Delta u}) \rightarrow \sum_{i=m_{(j-1)}+1}^{m_{(j-1)}+(x+1)} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$ Thus $L_j$ can acquire up to $M_j$ many unique non-zero values, where $0 \leq L_j \leq \sigma_j$. Prior to starting Protocol 4.1, all $L_j$'s are initialized to 0. However, once the protocol begins the $L_j$'s are initialized as follows: $L_1 = \sum_{i=m_0+1}^{m_0+1} exp(\frac{\epsilon \cdot U_i}{2\Delta u})$ and $L_k = 0$ where $k \neq 1$.

### 4.2.5 Selecting the Winning-Attribute

Each party $P_j$ privately computes their own respective $L$-shares and $R$-shares, $(L_j, R_j)$. Using the Distributed Comparison 4.2, $P_j$ encrypts its shares as $(\llbracket L_j \rrbracket, \llbracket R_j \rrbracket)$ then with the other parties, collectively and securely computes $\llbracket R \rrbracket = \llbracket \sum_{i=1}^{n} R_i \rrbracket$ and $\llbracket L \rrbracket = \llbracket \sum_{i=1}^{n} L_i \rrbracket$ through Protocol 4.2. It should be noted that value of $R$ is a constant, while the value of $L$ increases after every iteration (or loop). Once $R$ and $L$ are derived, Protocol 4.2 securely computes $\gamma := R - L$. If $\gamma \leq 0$, then we acquire a winning-attribute. Otherwise, we increment the upper summation-limit of some $L_j$ and determine $\gamma$ again. If $L_j$ is incremented in a manner where its upper summation-limit goes from $(m_{(j-1)} + x) \longrightarrow (m_{(j-1)} + (x+1))$ resulting in $\gamma \leq 0$, then the corresponding attribute indexed as $A_{m_{(j-1)}+(x+1)}$ is $qth$ winning winning-attribute $A_q^w$.

**Example 4.2.3.** *:*

(Continuation from example 4.2.2)

Recall $P_1$ owns $\Phi_1 = \{(A_1, U_1), (A_2, U_2), (A_3, U_3)\}$ and $\sigma_1 = 478 = 234 + 12 + 232$, $P_2$ owns $\Phi_2 = \{(A_4, U_4)\}$ and $\sigma_2 = 232$, and $P_3$ owns $\Phi_3 = \{(A_5, U_5), (A_6, U_6)\}$ and $\sigma_3 = 26 = 24 + 2$. Through the extended RVP, parties collectively derive $R \in [0, 736]$, where $R = R_1 + R_2 + R_3$. From there we need to securely verify if $R \leq L$ through Protocol 4.2. Given $L = L_1 + L_2 + L_3$, $L_1$ is initialized as 234, $L_2$ is initialized as 0, and $L_3$ is initialized as 0. We first check if $R \leq (234 + 0 + 0)$. If so, $P_1$ wins and $A_1$ is the winning attribute. Otherwise we increment $L_1$ and get $L_1 = 234 + 12$. Now we determine if $R \leq ((234 + 12) + 0 + 0)$ is true. If so, $P_1$ wins and $A_2$ is the winning attribute. Continuing this trend we will sequentially check the following: If $R \leq (234 + 12 + 232) + 0 + 0$ ($P_1$ wins, $A_3$ is the winning attribute), $R \leq 478 + 232 + 0$

($P_2$ wins, $A_4$ is the winning attribute), $R \leq 478 + 232 + 24$ ($P_3$ wins, $A_5$ is the winning attribute), $R \leq 478 + 232 + (24 + 2)$ ($P_3$ wins, $A_6$ is the winning attribute). Based on how $R$ and $L$ were designed, a winner must be declared in this procedure. ■

### 4.2.6 Multiparty Exponential Mechanism Summary

For the Multiparty Exponential Mechanism, we were able to select a winning-attribute owned by $P_j$ that probabilistically has a high utility. Selecting an attribute with a high utility is advantageous when considering the MAIN Protocol 5.1, which will be discussed in the next chapter. For now just think of utility as measuring how well an attribute can partition (or group) individuals in a dataset. An attribute with low utility suggests that the majority of the individuals are identical with respect to that attribute. We were not only able to select an attribute with high utility, we were able to do so in a differentially-private manner. As a result, the partitioned dataset(s) (discussed in the next chapter) will preserve the privacy of the each individual record. The process of selecting a winning-attribute involves communication (Random value Protocol and Distributed Comparison) between multiple parties in a secure manner. Initially, $P_j$ has a collection of attributes $\hat{\Phi_j}$ then privately derives its collection of attribute-score pairs $\Phi_j$. $P_j$ then uses its utility-scores from $\Phi_j$ to privately derive its $\sigma_j$. Once each party acquires its respective $\sigma_j$, they use it as input in the Extended Random Value Protocol (RVP). The Extended RVP allows the parties to acquire their respective $R_j$'s, where $R_j$ is fixed throughout the duration of the protocol. $P_j$ will then use the utility scores from $\Phi_j$ to derive its respective $L_j$, where only one $L_j$ can be updated per iteration and the value of $L_j$ depends on how many iterations have already occured. After each party acquires its respective $(L_j, R_j)$, they use the Distributed Comparison to encrypt each value $(\llbracket R_j \rrbracket, \llbracket L_j \rrbracket)$, add the values respectively $(\llbracket L \rrbracket, \llbracket R \rrbracket)$, subtract the values $[\![R-L]\!]$, and finally decrypt $R-L$. Once $R-L$ is revealed, a winning attribute will be selected depending on which $L_j$ was most recently updated.

## 4.3 Protocol Analysis

### 4.3.1 Security Analysis

For Protocol 4.1(Exponential Mechanism), there are only two instances when the parties communicate. These communications occur during RVP and Protocol 4.2(Distributed Comparison). The security of RVP was previously demonstrated in its paper of origin [4], meaning we only need to demonstrate the security for the Distributed Comparison. For the rest of the security analysis it will go as follows: Establishing Key Values for the Distributed Comparison, Axiom 4.3.1, Lemma 1, The Diffie-Hellman Assumption 4.3.1, Lemma 2, and Theorem 4.3.1

#### Establishing Key Values For the Distributed Comparison

1. We have the following for $P_j$
   - Axillary Inputs: Prime $p$, generator $g$, bit length $n$, protocol blueprint, and the Diffie-Hellman Assumption. For simplicity we described the axillary inputs as $z^*$.
   - $x_j = ((R_j, L_j), z^*)$
   - $r_j^* = (a_j, r_j, \acute{r}_j)$
   - $m = (\vec{A}_k, A, [\![\vec{R}_k]\!], [\![\vec{L}_k]\!], [\![R]\!], [\![L]\!], [\![R-L]\!], \alpha, R-L)$
   - $\vec{\Omega}_k$ represents a set of $\Omega_k$'s, where $\Omega_k$ is owned by $P_k$

- $View_j^{\Pi}(x_j, f_j(\vec{x})) = (x_j, r_j^*, m)$
- $Output_j^{\Pi}(\vec{x}) = R - L$
- $f_j(\vec{x}) = R - L$

2. We have the following simulated output-messages for $S_j$

- $S_j(x_j, f_j(\vec{x})) = (x_j, r_j^*, \vec{A_{S_j(k)}}, A, [\![\vec{R_{S_j(k)}}]\!], [\![\vec{L_{S_j(k)}}]\!], [\![R^*]\!], [\![L^*]\!], [\![R^* - L^*]\!]$ $, \alpha^*, R^* - L^*)$,
- $R^* := R_{S_j} + R_j = (\sum_{k\in\Psi} R_{S_j(k)}) + R_j$, for $j \notin \Psi = [1, \ldots, n]$
- $L^* := L_{S_j} + L_j = (\sum_{k\in\Psi} L_{S_j(k)}) + L_j$
- $\alpha^*$ is the decryption of $[\![R^* - L^*]\!]$ before applying the discrete-log algorithm
- $X_{S_j(k)}$ represents the simulated value of $X_k$, where $X_k$ is owned by $P_k$ in the real protocol

**Axiom 4.3.1.** *If there exists a sub-operation $\pi^*$ in $\Pi$, where $P_j$ and $P_k$ both conduct $\pi^*$ correctly and expect the same sub-output with respect to $\Pi$ (i.e $m_a \in Output_j^{\pi^*\in\Pi}(\vec{x}) \bigcup Output_k^{\pi^*\in\Pi}(\vec{x})$), then $m_a$ is a message that is contained in the view of each party. Or equivalently,*
$m_a \in View_j^{\Pi}(\vec{x}) \bigcup View_k^{\Pi}(\vec{x})$

The purpose of the axiom is to highlight a common scenario that arises in our protocol, where values are encrypted throughout the process and later decrypted. Once the value(s) are decrypted, the parties will privately see the decrypted value (such as $\alpha$ or $R-L$). Given $\Pi$ is executed in a semi-honest setting, each party knows that every other party will see the same value. Or in other words, every party knows that every other party has the value $m_a$ derived from $\pi^* \in \Pi$, given a semi-honest setting. Thus, it makes no sense for the parties to inform each other about the value of $m_a$. But suppose $P_j$ actually sent $m_a$ to $P_k$. By doing so $m_a$ would be contained within the view of $P_k$ (i.e $m_a \in View_k^{\Pi}(\vec{x})$). However, $P_k$ did not learn anything new once it received the message. But at the same time, if $P_j$ did not send anything, $P_k$ would still know that $P_j$ has $m_a$. Therefore, it makes no difference whether $P_j$ actually sent $m_a$ or not, meaning it makes no difference if $m_a$ is added to the view of each party. This assertion directly implies $m_a$ is also part of the output-message(s) of each party.

**Lemma 1.** Given the initial-input $x_j$ of each party $P_j$, the final-output of the trusted third-party $f$ is indistinguishable from the final-output of the real protocol. Or equivalently $f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$, where $\Pi$ is the Distributed Comparison Protocol and $\vec{x} = (x_1, x_2, \ldots, x_n)$.

*Proof.* For the ideal functionality $f$, when given input $\vec{x}$ we get the following output: $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})) = (R - L, R - L, \ldots, R - L)$. For the real protocol, when given the input $\vec{x}$ we similarly get the following output: $Output^{\Pi}(\vec{x}) = (Output_1^{\Pi}(\vec{x}), Output_2^{\Pi}(\vec{x}), \ldots, Output_n^{\Pi}(\vec{x})) = (R - L, R - L, \ldots, R - L)$. This is because the protocol is deterministic even though the encryptions are probabilistic. The encryptions in $\Pi$ only serve to hide the values, not alter them. Thus, given $(R_1, L_1), (R_2, L_2), \ldots, (R_n, L_n)$ we simply compute the value $R - L = \sum_{j=1}^{n} R_j - \sum_{j=1}^{n} L_j$, which is algebraically deterministic. Since the outcome of $\Pi$ is deterministic based on its initial inputs, this implies the functionality $f$ and the real protocol are similarly deterministic based on its initial inputs. Since both $f$ and and the real protocol have the same inputs, they must have identical outputs. Identical terms are trivially indistinguishable.

$\therefore f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$

□

**The Diffie-Hellman Assumption** [2]

For the next Lemma will need to invoke the Diffie-Hellman assumption. Under modular exponentiation with a sufficiently large prime, the Diffie-Hellman assumption implies that given $g$ and $p$, the triple $(g^a \ mod\, p, g^b \ mod\, p, g^c \ mod\, p)$ is indistinguishable from the triple $(g^a \ mod\, p, g^b \ mod\, p, g^{ab} \ mod\, p)$, where $a, b, c \in_R \{2, \ldots, p-2\}$ and $c \neq ab$. Or in other words $(g^a modp, g^b modp, g^{ab} modp) \stackrel{c}{\equiv} (g^a modp, g^b modp, g^c modp)$, which is equivalent to $|Pr[D(g^a\, modp, g^b\, modp, g^{ab}\, modp) = 1^*] - Pr[D(g^a\, modp, g^b\, modp, g^c\, modp)]| < \epsilon(n)$. In a similar fashion, we can easily see $g^{ab} \ mod\, p \stackrel{c}{\equiv} g^c \ mod\, p$. For example, this suggests $(A_j = g^{a_j}\, modp) \stackrel{c}{\equiv} (A_k = g^{a_k}\, modp)$. Thus $P_j$ and $P_k$ are not compromising their private-keys when collectively computing the public-group key $A$ along with its cryptographic derivatives ($[\![L_j]\!], [\![R_j]\!]$, ect.), which makes since due to the fact that ElGamal encryption is a common encryption standard. It is very important to note that the majority of values computed within the Distributed Comparison can be algebraically manipulated such that they look like $g^{\beta_x} \ mod\, p$, where $\beta_x$ is some positive integer. The only computed values from the Distributed Comparison which cannot be algebraically described using modular exponentiation is the inital-input values, ephemeral keys, $\alpha$, and $R - L$.

**Lemma 2.** The output messages of the simulator during an ideal (or simulated) execution of $\Pi$ is indistinguishable from the messages that $P_j$ would have received ($P_j$'s view) during a real execution of $\Pi$. Or equivalently $S_j(x_j, f_j(\vec{x}) \stackrel{c}{\equiv} View_j^{\Pi}(\vec{x})$, where $\Pi$ is the Distributed Comparison Protocol.

*Proof.* Using proof by contradiction, let us assume that

$$S_j(x_j, f_j(\vec{x})) \overset{c}{\not\equiv} View_j^{\Pi}(\vec{x})$$

$$\implies S_j(x_j, R-L)) \overset{c}{\not\equiv} (x_j, r_j^*, A_k, [\![R_k]\!], [\![L_k]\!], [\![R]\!], [\![L]\!], [\![R-L]\!], \alpha, R-L)$$

Denote $\{g^\beta\}$ as a set of values that can be represented as $g^\beta \ \ mod\, p$

$$\implies S_j(x_j, R-L)) \overset{c}{\not\equiv} (x_j, r_j^*, \{g^\beta\}, \alpha, R-L)$$
$$= (x_j, r_j^*, A_{S_j}, A, [\![R_{S_j}]\!], [\![L_{S_j}]\!], [\![R^*]\!], [\![L^*]\!], [\![R^*-L^*]\!], \alpha^*, R^*-L^*) \overset{c}{\not\equiv} (x_j, r_j^*, \{g^\beta\}, \alpha, R-L)$$
$$= (x_j, r_j^*, \{g^\Gamma\}, \alpha^*, R^*-L^*) \overset{c}{\not\equiv} (x_j, r_j^*, \{g^\beta\}, \alpha, R-L)$$

$x_j$ and $r_j^*$ are identical for $P_j$ and $S_j$. Identical terms are trivially indistinguishable.

$$\implies (\{g^\Gamma\}, \alpha^*, R^*-L^*) \overset{c}{\not\equiv} (\{g^\beta\}, \alpha, R-L)$$
$= (g^c \ \ mod\, p, \alpha^*, R^*-L^*) \overset{c}{\not\equiv} (g^b \ \ mod\, p, \alpha, R-L)$, where $b \in \beta$ and $c \in \Gamma$

From the Diffie-Hellman assumption recall $g^c \ \ mod\, p \overset{c}{\equiv} g^b \ \ mod\, p$. Thus we can make a further reduction

$$(\alpha^*, R^*-L^*) \overset{c}{\not\equiv} (\alpha, R-L)$$

Given $\alpha$, you can directly derive $R-L$ through the discrete-log algorithm. This suggests that $\alpha$ is a redundant representation of $R-L$. Since $\alpha$ is redundant, we can easily make another reduction.

$(R^* - L^*) \overset{c}{\not\equiv} (R - L)$

$R_j$, $L_j$, and $R - L$ are fixed constants where $R^* - L^* = (R_{S_j} + R_j) + (L_{S_j} + L_j)$ is dictated by the simulator $S_j$. Recall that $S_j$ also has access to $(L_j, R_j)$ and $R - L$. The simulator's goal is to have $R - L = R^* - L^*$, which can be achieved algebraically as follows:

$$
\begin{aligned}
R - L = R^* - L^* & \\
&= (R_{S_j} + R_j) - (L_{S_j} + L_j) \\
&= (R_{S_j} - L_{S_j}) + (R_j - L_j)
\end{aligned}
$$

$\implies R_{S_j} - L_{S_j} = (R - L) - (R_j - L_j)$

$\implies (\sum_{k \in \Psi} R_{S_j(k)}) - (\sum_{k \in \Psi} L_{S_j(k)}) = (R - L) - (R_j - L_j)$

Let $(R - L) - (R_j - L_j) := C$

where $C$ is a non-negative fixed constant and known ahead of time by $S_j$

$\implies R_{S_j} - L_{S_j} = C$

$\implies \sum_{k \in \Psi}(R_{S_j(k)} - L_{S_j(k)}) = C$

In order for $R^* - L^*$ to be equal to $R - L$, $S_j$ needs to appropriately choose its simulated values $R_{S_j(k)}$ and $L_{S_j(k)}$. By doing so, $S_j$ can derive its $R_{S_j}$ and $L_{S_j}$. $S_j$ needs to derive $R_{S_j}$ and $L_{S_j}$ such that their difference equals $C = (R - L) + (L_j - R_j)$. Since $S_j$ dictates the values of each $R_{S_j(k)}$ and $L_{S_j(k)}$, this is easily achievable. Assuming $S_j$ behaves optimally in the semi-honest setting, we conclude $R^* - L^* = R - L$ for all simulations. However this contradicts $(R^* - L^*) \overset{c}{\not\equiv} (R - L)$, meaning our original assumption $S_j(x_j, f_j(x_j, x_k)) \overset{c}{\not\equiv} View_j^{\Pi}(x_j, f_j(x_j, x_k))$ is false.

Thus $S_j(x_j, f_j(\vec{x})) \overset{c}{\equiv} View_j^{\Pi}(\vec{x})$

□

**Theorem 4.3.1.** *The Multiparty Exponential Mechanism is secure in the semi-honest multiparty setting*

*Proof.* It is given that RVP is secure, meaning we only need to verify the security of the Distributed Comparison Protocol. Lemma 1 proves $f(\vec{x}) \overset{c}{\equiv} Output^{\Pi}(\vec{x}))$ for the Distributed Comparison Protocol. Using Axiom 4.3.1, Lemma 2 proves $S_j(x_j, f_j(\vec{x})) \overset{c}{\equiv} View_j^{\Pi}(\vec{x})$ for the same protocol. This directly implies the protocol satisfies the below equation, making it secure in the the semi-honest multiparty setting.

$$\{(S_j(x_j, f_j(\vec{x})), f(\vec{x}))\} \overset{c}{\equiv} \{(View_1^{\Pi}(x_j, f_j(\vec{x})), Output^{\Pi}(\vec{x}))\}$$

□

### 4.3.2 Complexity Analysis

**Lemma 3.** The total encryption and communication costs among $n$ parties for Protocol 4.2 is $\mathcal{O}(n^2\xi)$ and $\mathcal{O}(n^2(\zeta + K))$ respectively.

*Proof.* Complexity accounts for the amount of 'loops' as well as the operations involved within each loop. To fix an upper bound on complexity, we will examine the worst case scenario, where the winner candidate-attribute happens to be the last attribute ($A_{m_n}$) owned by $P_n$. Since there are $z$ attributes, we can assume $m_n = z$. This implies there will be at least $z$ loops in Protocol 4.1. For the encryption cost, RVP is given as $\mathcal{O}(\xi)$ [4]. Since we extended the RVP operation, $P_j$ does the RVP operation with the other $n-1$ party-members, meaning $P_j$ conducts $\binom{n}{2} = \frac{n(n-1)}{2}$ RVP operations. Thus for $P_j$, the encryption complexity for RVP is $\mathcal{O}(\frac{n(n-1)}{2} \times \xi) = \mathcal{O}(n^2 \times \xi)$. The encryption cost for the Distributed Comparison is measured by how many exponentiations occur, where $(x^a)^b$ is considered a single exponentiation. For Protocol 4.2, line 2, there are a total of $n$ exponentiations. Lines 3, 4a, 4b, 4c, and 4d have the following respective total exponentiations: $2n$, $3n$, $3n$, 3, and 3. Since we loop through Protocol 4.2 $z$ times, the encryption complexity cost is $\mathcal{O}(z \times (8n+6)) = \mathcal{O}(zn)$. Therefore the total Encryption cost for RVP and Distribution Comparison is $\mathcal{O}(n^2\xi) = \mathcal{O}(n^2 \times \xi + zn)$.

For the communication cost, RVP is given as $O(\zeta)$ [4]. Thus by similar reasoning, we can see $\mathcal{O}(n^2 \times \zeta)$. The communication cost on line 2, line 3, line 4a, line 4b, line 4c, and line 4d have the following respective total communication: $n(n-1)K, 2(n(n-1))K, 2(n(n-1))K$, 0 and $nK$, where $K$ is the key size of the message). Thus, the total communication cost for RVP and Distributed Comparison for $P_j$ is $\mathcal{O}((n^2 \times (\zeta + K))) = \mathcal{O}((n^2 \times \zeta) + n^2K)$. $\square$

### 4.3.3 Correctness Analysis

**Lemma 4.** *Assuming all parties are semi-honest, Protocol 4.1 correctly implements the exponential mechanism for all parties.*

*Proof.* Protocol 4.1 selects its candidate-attribute $A_i$ with probability $\propto exp(\frac{\epsilon U_i}{2\Delta U})$. Each party $P_j$, computes $exp(\frac{\epsilon U_i}{2\Delta U})$ for their respective candidate-score pairs $\Phi_j$. All parties use the exponential mechanism to collectively build the interval [0, $\sum_{i=1}^{z} exp(\frac{\epsilon U_i}{2\Delta U})$]. We can partition the interval into discrete sub-intervals, where each sub-interval corresponds to a candidate-attribute with length equal to $exp(\frac{\epsilon U_{(i)}}{2\Delta U})$. Since the random number 'R' lies uniformly between [0, $\sum_{i=1}^{z} exp(\frac{\epsilon U_i}{2\Delta U})$], the probability of choosing a particular candidate-attribute is given as $\frac{exp(\frac{\epsilon U_i}{2\Delta U})}{\sum_{i=1}^{z} exp(\frac{\epsilon U_i}{2\Delta U})}$. Thus Protocol 4.1, selects an $A_i$ in a manner consistent with the Exponential Mechanism. $\square$

# Chapter 5

# SECURE MULTIPARTY PROTOCOL FOR DIFFERENTIALLY-PRIVATE DATA RELEASE

## 5.1 Protocol Overview

Recall $P_j$ owns its dataset $D_j$ where $D_j$ contains the attributes owned by $P_j$ as well as the individual records $\mathcal{R}_j$. The individual records in $D_j$ are only known by $P_j$. These records correspond to whether a particular individual satisfies an attribute owned by $P_j$. There are two types of attributes, class attribute $A^c$ and predictor attributes $A^p$. Class attributes $A^c$ are categorical, publicly available, and represent a parameter that the data miner attempts to predict. On the other hand, predictor attributes are private and uniquely distributed among the parties. Predictor attributes $A^p_{ij}$ can be numerical or categorical and are used to predict the outcome of a class attribute. We can equivalently describe $P_j$'s dataset $D_j$ as $D_j(ID, A^c, A^p_{1j}, A^p_{2j}, \ldots, A^p_{M_j j}, \mathcal{R}_j)$, where $ID$ assigns the individuals in the dataset an anonymous username. There are $n$ many $D_j$'s that form the set $\mathcal{D}$, where $\mathcal{D} := \{D_1, D_2, \ldots, D_n\}$. Our proposed algorithm seeks to integrate all $D_j$'s to produce an anonymized, differentially-private data set $\hat{D}$. Since producing $\hat{D}$ also requires a privacy parameter $\epsilon$, a pre-determined amount of specializations $\mathcal{S}$, and knowing the total amount of numerical attributes $\mathcal{N}$, we can equivalently describe it as $\hat{D}(\mathcal{S}, \epsilon, \mathcal{N}, \mathcal{D})$. It should be noted that $\mathcal{S}$ corresponds to how many times Protocol 4.1 is used.

The parties begin with their respective datasets $D_j$. Each dataset contains public identification $ID$ for each individual, a public Classifier $A^c$, a collection of publicly known attributes $\vec{A}$, and private individual records $\mathcal{R}_j$. Thus we can represent $D_j$ as $D_j(ID, A^c, \vec{A}, \mathcal{R}_j)$. Each attribute $A_i \in \vec{A}$ that a party owns has a corresponding taxonomy $\mathbb{T}_{A_i}$. A taxonomy describes all the values that $A_i$ can acquire in terms of classification. Each taxonomy begins as a root node, which is the most general value (or classification) that $A_i$ can acquire. All taxonomies have the capacity to become specialized, acquiring new child nodes in the process. For the categorical attributes, the manner in which specialization occurs is predetermined. For numerical attributes, specialization is computed using the exponential mechanism. The parties know which taxonomy is specialized based on the winning-attribute selected through Protocol 4.1. Each party privately organizes (or groups) their private records $\mathcal{R}_j$ (as well as $ID$s) with respect to all the taxonomies of all the attributes they own. By doing this, the party create their own Sub-Partitioning Tree $\mathcal{P}_j^*$. Each $\mathcal{P}_j^*$ similarly has a root node, where all the records they own are classified into one group. Since $\mathcal{P}_j^*$ is dependent on each collective taxonomy, if a taxonomy $\mathbb{T}_{A_i}$ is updated, then $\mathcal{P}_j^*$ is specialized with respect to how a $\mathbb{T}_{A_i}$ was updated, assuming $P_j$ owns attribute $A_i$. Each child node in $\mathcal{P}_j^*$ is regarded as a "sub-partition". The parties will then agree on a particular set of sub-partitions (excluding records) that are distributed among themselves. This particular set of sub-partitions will be known as a "leaf node". With respect to this protocol, each $P_j$ will select the attribute-value(s) contained in a single leaf-node with respect to their $\mathcal{P}_j^*$. $P_j$ will then transformed each attribute value contained in the leaf node into binary attribute-vectors described as $\vec{V_{j_k}}$, based on the records they own. The binary attribute-vectors owned by $P_j$ can then be simplified into a single binary standardized attribute-vector $\vec{P_j}$. The parties want to determine how many records are contained in (or satisfies) all the pre-specified sub-partitions distributed among themselves. The amount of records satisfying all the pre-specified sub-partitions is called the "True Count", $TCount$. To acquire $TCount$, the parties use their respective $\vec{P_j}$'s to derive a single count-vector $\vec{C}$, where the contents of $\vec{C}$ are securely encrypted and $\vec{C}$ indirectly contains $TCount$. Each party jointly uses an encrypted-table $\mathcal{T}$ to derive $[\![TCount]\!]$. The parties will jointly and securely generate encrypted random noise, described as $[\![Laplace]\!]$. The parties do the following addition to acquire the encrypted "Noisy Count", $[\![NCount]\!] = [\![TCount + Laplace]\!] = [\![TCount]\!] \times [\![Laplace]\!]$. Finally the parties jointly decrypt and reveal the encrypted Noisy Count, $NCount$. $NCount$ is then assigned to the differentially-private dataset $\hat{D}$, with respect to the pre-specified sub-partitions. Or in other words, $\hat{D}$ will contain many different $NCount$'s with respect to the combination of pre-specified sub-partitions that the parties are interested in. When $\hat{D}$ is released, the data miner will acquire an integrated dataset, which is both secure and differentially private.

## 5.2 Protocol Details

### 5.2.1 Attribute Taxonomy

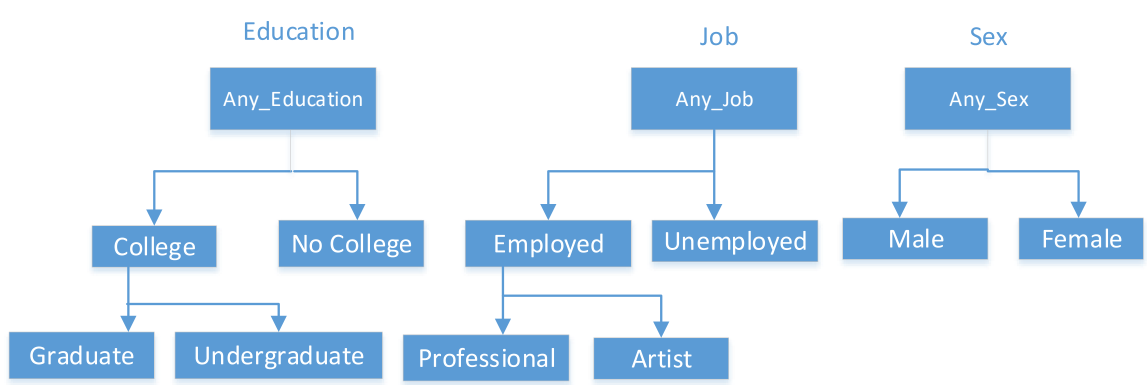


Figure 5.1: Attribute Taxonomies

**Main Protocol:** **Input:** Raw datasets $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$, privacy budget $\epsilon$, total number of numerical attributes $\mathcal{N}$, and the number of specializations $\mathcal{S}$

**Output:** Differentially-private dataset $\hat{D}$

1. Parties construct the root node $\mathbf{T}_0 = \bigcap_{j=1}^{n} Root(\mathbb{T}_j)$ of the unified taxonomy $\mathbf{T}$, where all attributes are initially unspecialized
2. Each party determines $\epsilon^* \leftarrow \frac{\epsilon}{4(\mathcal{S} \cdot \mathcal{N})}$
3. Each party determines the split-value for each $A_i^n$ they own with probability $\propto \exp(\frac{\epsilon^*}{2\Delta u} \times u(\mathcal{D}, A_i^n))$
4. Each party computes the utility-score for each respective attribute they own, through the utility function $u(\mathcal{D}, A_i)$.
5. For $q = 1$ to $\mathcal{S}$ do the following:
   (a) All parties jointly execute Protocol 4.1 (Multiparty Exponential Mechanism) on $\mathbf{T}_{q-1}$ to determine the $(q-1)th$ winning attribute $A_{q-1}^w$.
   (b) The party owning the winning attribute $A_{q-1}^w$ performs the following:
      i. Derives $\mathbf{T}_q$ as follows:
         A. If $A_{q-1}^w$ is *categorical*, specialize $A_{q-1}^w$ on $\mathbf{T}_{q-1}$ with respect to its taxonomy $\mathbb{T}_{A_{q-1}^w}$.
         B. If $A_{q-1}^w$ is *numerical*, specialize $A_{q-1}^w$ on $\mathbf{T}_{q-1}$ with respect to its split value $v_a$.
      ii. Instruct other parties on how to specialize.
   (c) Each party determines the split-value for each $A_i^n$ they own with probability $\propto \exp(\frac{\epsilon^*}{2\Delta u} \times u(T, A_i^n))$.
   (d) Each party computes the utility-score of each attribute $A_i \in \mathbf{T}_q$ from utility function $u(\mathcal{D}, A_i)$.
6. For each leaf partition $P_{leaf}(\mathbf{T}) \in \mathbf{T}_{\mathcal{S}}$, all parties execute Protocol 5.2 (SPACE) to securely compute its noisy count $NCount$.
7. Return $\hat{D} = \{(P_{leaf}(\mathbf{T}), NCount)\}$.

Protocol 5.1: Main Protocol

Before acquiring $\hat{D}$, we begin with $\mathcal{D}$. Recall each dataset $D_j \in \mathcal{D}$, contains a collection of attributes and records. For categorical attributes, they have a pre-defined taxonomy on how an attribute (like 'Job') can be re-classified into sub-attributes (Engineer, Doctor, Lawyer, etc.), as well as how those sub-attributes can also be re-classified into other

**Secure & Private Attribute-Counting Exchange (SPACE) Protocol**
**Input:** Leaf partition $P^*_{leaf}$, privacy budget $\hat{\epsilon} = \frac{\epsilon}{2}$.
**Output:** Noisy count $NCount$ of $P^*_{leaf}$.

1. Each party $P_k$ computes its standardize attribute-vector for partition $P_{leaf}$, and then encrypts each element using exponential ElGamal: $\vec{P}_k = \langle [\![X_{k,1}]\!], [\![X_{k,2}]\!], \ldots, [\![X_{k,d}]\!] \rangle$, where $d$ is the total number of records.
2. All parties homomorphically compute count vector: $\vec{C} := \langle [\![C_1]\!], [\![C_2]\!], \ldots, [\![C_d]\!] \rangle$, where: $[\![C_j]\!] := (g^{\sum_{k=1}^{n} X_{k,j}} \cdot A^{(\sum_{k=1}^{n} r_k)} \ mod \ p, g^{\sum_{k=1}^{n} r_k} \ mod \ p)$.
3. Apply the Mix and Match protocol $\mathcal{M}(\vec{C})$ (Protocol 5.3) to compute the exponential ElGamal encryption of true count $[\![TCount]\!]$ of records satisfying all the attributes in $P_{leaf}$.
4. Each party $P_k$ computes two gamma variables: $Y_{1,k} \sim Gamma(n, 1/\hat{\epsilon})$ and $Y_{2,k} \sim Gamma(n, 1/\hat{\epsilon})$, and then encrypts $Y_k = Y_{1,k} - Y_{2,k}$ using the group public key $A$: $[\![Y_k]\!] := (g^{Y_k} \cdot A^{r_k} \ mod \ p, g^{r_k} \ mod \ p)$.
5. All parties homomorphically compute: $[\![Y]\!] := \prod_{k=1}^{n} [\![Y_k]\!] = (g^{\sum_{k=1}^{n} Y_k} \cdot A^{\sum_{k=1}^{n} r_k} mod p, g^{\sum_{k=1}^{n} r_k} mod p) = (g^Y \cdot A^r mod p, g^r mod p)$. where $Y \sim Laplace(0, 1 \backslash \hat{\epsilon})$
6. Compute the encryption of noisy count $[\![NCount]\!] := [\![TCount]\!] \times [\![Y]\!]$.
7. All parties jointly decrypt $[\![NCount]\!]$ described as $\alpha$ and then apply the discrete-log algorithm to determine the noisy count $NCount$.
8. Jointly decrypt $[\![NCount]\!]$:
$$\alpha = (A^{r-\acute{r}} \cdot g^{NCount}) / \prod_{k=1}^{n} (g^{r-\acute{r}})^{a_k} = g^{R-L} \ mod \, p.$$
9. Apply discrete-log algorithm on $\alpha$ to compute $NCount$.
10. Return $NCount$.

Protocol 5.2: SPACE

sub-attributes (Chemical Engineer, Pediatrician, State Attorney, etc.). In this case 'Job', is commonly referred to as the 'root' node (indexed as $Root$), since it encapsulates all of the other classifications. The first set of sub-attributes mentioned earlier can be referred to as '1st order child-nodes' (indexed as $Child_1$). The second set of sub-attributes that were mentioned can be similarly described as '2nd order child-nodes' (indexed as $Child_2$). In

**Mix and Match Count Protocol**
**Input:** Encrypted count vector $\vec{C} := \langle [\![C_1]\!], [\![C_2]\!], \ldots, [\![C_d]\!] \rangle$.
**Output:** Encrypted true count $[\![TCount]\!]$

1. All parties agree on an initial Mix and Match table $\mathcal{T}$ encrypted with their group public key $A$:

| Input | Output |
|---|---|
| $[\![0]\!]$ | $[\![0]\!]$ |
| $[\![1]\!]$ | $[\![0]\!]$ |
| $[\![2]\!]$ | $[\![0]\!]$ |
| $\vdots$ | $\vdots$ |
| $[\![n]\!]$ | $[\![0]\!]$ |

2. For each element $[\![C_j]\!]$ in the count vector $\vec{C}$, where $1 \leq j \leq d$:
   (a) All parties jointly apply Mix Network protocol to randomly shuffle the rows and re-randomize all ciphertexts in $\mathcal{T}$.
   (b) For each row in $\mathcal{T}$:
      i. Jointly apply plaintext equality test (PET) between $[\![C_j]\!]$ and the input ciphertext in that row.
      ii. If there is a match (PET is satisfied), homomorphically increment the corresponding output by 1 by multiplying the corresponding ciphertext with $[\![1]\!]$, and then go back to Step 2.
3. All parties jointly apply (PET) between $[\![n]\!]$ and each ciphertext in the input column of $M$.
4. When a match is found, return the corresponding output ciphertext (which represents the encrypted true count $[\![TCount]\!]$).

Protocol 5.3: Mix and Match Count

this case, a taxonomy can also be thought of as a collection of values, where all values have a common relation with a single value, which is the root node. Given a set $X$ containing values that are all connected by a single root node, the root node of $X$ is defined as $Root(X) = Child_0(X)$, while the $kth$ order child-node of $X$ is defined as $Child_k(X)$. The taxonomy of each categorical attribute is publicly available prior to Protocol 5.1. Since the taxonomy of categorical attributes are predetermined and public, no privacy budget is required to generate the sub-attributes of the root node. However, that is not true for numerical attributes $A_i^n$. The taxonomy of each $A_i^n$ must be generated using an exponential mechanism. This can be done by first examining the numerical domain of $A_i^n$. We use the utility function $u(\mathcal{D}, A_i^n)$ to assign a utility-score to each element in the domain. We group all the elements in the domain which were assigned the same utility-score into a set $I_a$. For each unique utility-score assigned through $u(\mathcal{D}, A_i^n)$, there exists an $I_a$ which contains the elements that were assigned the same utility-score. We then use the exponential mechanism to select $I_a$ amongst the other $I$'s, where the $I_a$ which contains the highest utility-score is exponentially more likely to be chosen. Finally, we randomly select an element $v_a \in I_a$, where $v_a$ is now regarded as the 'split-value' of $A_i^n$. The split-value dictates the taxonomy of $A_i^n$. We can also get a visual intuition in regards to attribute taxonomies by examining Figure 5.1

**Example 5.2.1.** Assume $P_j$ owns the numerical attribute $A_1^n$ which has integer values from 1 to 10, where [1,10] is the root node of $A_1^n$. Based on $u(\mathcal{D}, A_i^n)$, let us say $P_j$ determined the elements contained in $\{1, 5, 6\}$ each have the utility-score $U_1$, the elements in $\{2, 3, 9, 10\}$ each have have the utility-score $U_2$, and the elements in $\{4, 7, 8\}$ each have the utility-score $U_3$. $P_j$ then defines the following sets $I_1 = \{1, 5, 6\}$, $I_2 = \{2, 3, 9, 10\}$, and $I_3 = \{4, 7, 8\}$. $P_j$ uses the exponential mechanism to select $I_3$. $P_j$ then randomly selects the split-value $v_3 \in I_3$. In this example, assume $v_3 = 4$. Now that $v_3$ is acquired, $P_j$ takes the root node [1,10] and splits it into two 1st order child-nodes [1,4] and [5,10]. The root node along with the subsequent child nodes become the updated taxonomy of $A_1^n$, which is publicly available. Now we have the following,

$\mathbb{T}_{A_1^n} = \{[1, 10]_{Root}, [1, 4]_{Child_1}, [5, 10]_{Child_1}\}$ ■

To explicitly detail how to derive a split value from a numerical attribute $A_i^n$ mathematically, the steps are detailed below

- Construct a discrete interval for $A_i^n$ containing all possible values of $A_i^n$, described as $I_{A_i^n}$
- Partition $I_{A_i^n}$ into subsets $I_1, \ldots, I_k$ such that all elements $v_a$ within some subset $I_a$ have the same utility-score where $U_a = u(\mathcal{D}, v_a)$
- Use exponential mechanism to select an interval $I_a$ with privacy budget $\epsilon'$ and probability:

$$\frac{\exp(\frac{\epsilon'}{2\Delta u} \times u(\mathcal{D}, v_a(A_i^n))) \times |I_a|}{\sum_{b=1}^{k}(\exp(\frac{\epsilon'}{2\Delta u} \times u(\mathcal{D}, v_b(A_i^n))) \times |I_b|)}$$

  where $v_a \in I_a$, $v_a(A_i^n)$ designates $v_a$ is derived from $A_i^n$, and $|I_a|$ represents the number of values in the subset.
- Uniformly select a value $v_a \in I_a$ to be the split-value for $A_i^n$.

Since the taxonomy of $A_i^n$ is dependent on the exponential mechanism, whenever the taxonomy of a numerical attribute is updated some of the privacy budget is consumed. The exponential mechanism will use $\epsilon'$ of the total privacy budget, where $\epsilon' < \epsilon$. The taxonomy of every attribute begins as a root node, which is the most generalized classification with respect to that attribute. If the attribute is selected through Protocol 4.1, the root node is updated and specialized into sub-classifications(or child-nodes). And as mentioned earlier, depending on whether the attribute is categorical or numerical will dictate whether its taxonomy is predetermined or computed. For any attribute $A_i$, its taxonomy is defined as $\mathbb{T}_{A_i}$. $A_i$ can acquire varying classification values depending on what part of the taxonomy you are looking at. A specific classification value is described as "$A_i.value$", where each $A_i.value$ is initialized as the the root node of its taxonomy, $Root(\mathbb{T}_{A_i})$. $A_i.value$ can be understood to be a set of values, which corresponds to a specific location within the taxonomy. For instance, the attribute "Job" that we will describe as $A_1$ can acquire a series of classification values. For $A_1.value$, assume $value \in \{Any\_Job_{Root}, Employeed_{Child_1}$ $, Unemployed_{Child_1}, rofessional_{Child_2}, Artist_{Child_2}\}$, each value is an element of $\mathbb{T}_{A_1}$ where $Root(\mathbb{T}_{A_1}) = Any\_Job$. Based on $\mathbb{T}_{A_1}$, the $Child_1$ values a generated from the $Root$ value. However, the $Child_2$ are generated from $Employeed_{Child_1}$, while $Unemployed_{Child_1}$ generates nothing.

### 5.2.2 Partitioning Process

Taxonomies are only concerned with how to classify records. Partitions are very similar, however they include and incorporate the private records of the datasets. For some $A_i$ owned by $P_j$, there exists a $\mathbb{T}_{A_i}$ that classifies and organizes all records in $\mathcal{R}_j$, as well as $ID$. In other words, a taxonomy allows $P_j$ the ability to group its records and $ID$'s with respect to an attribute. For simplicity within this section, assume that each record is accompanied by its respective $ID$. $P_j$ will need to classify the individual records it owns, with respect to all the taxonomies they own. The intersection of all the taxonomies that $P_j$ owns is equivalently described as $P_j$'s "group taxonomy", $\mathbb{T}_j := \bigcap_{i=m_{j-1}+1}^{m_j} \mathbb{T}_{A_i}$, where each $\mathbb{T}_{A_i} \in \mathbb{T}_j$ is initialized as a root node. The details of how taxonomies are intersected and implemented are highlighted in **Definition 5.2.1** and **Example 5.2.2**. Since the group taxonomy $\mathbb{T}_j$ functions as an intersection of all the taxonomies owned by $P_j$, $\mathbb{T}_j$ is also a taxonomy. When $P_j$ groups the individual records with respect to the values within $\mathbb{T}_j$, we have a "sub-partitioning tree", $\mathcal{P}_j^*$. There exists $n$ many of these partitioning

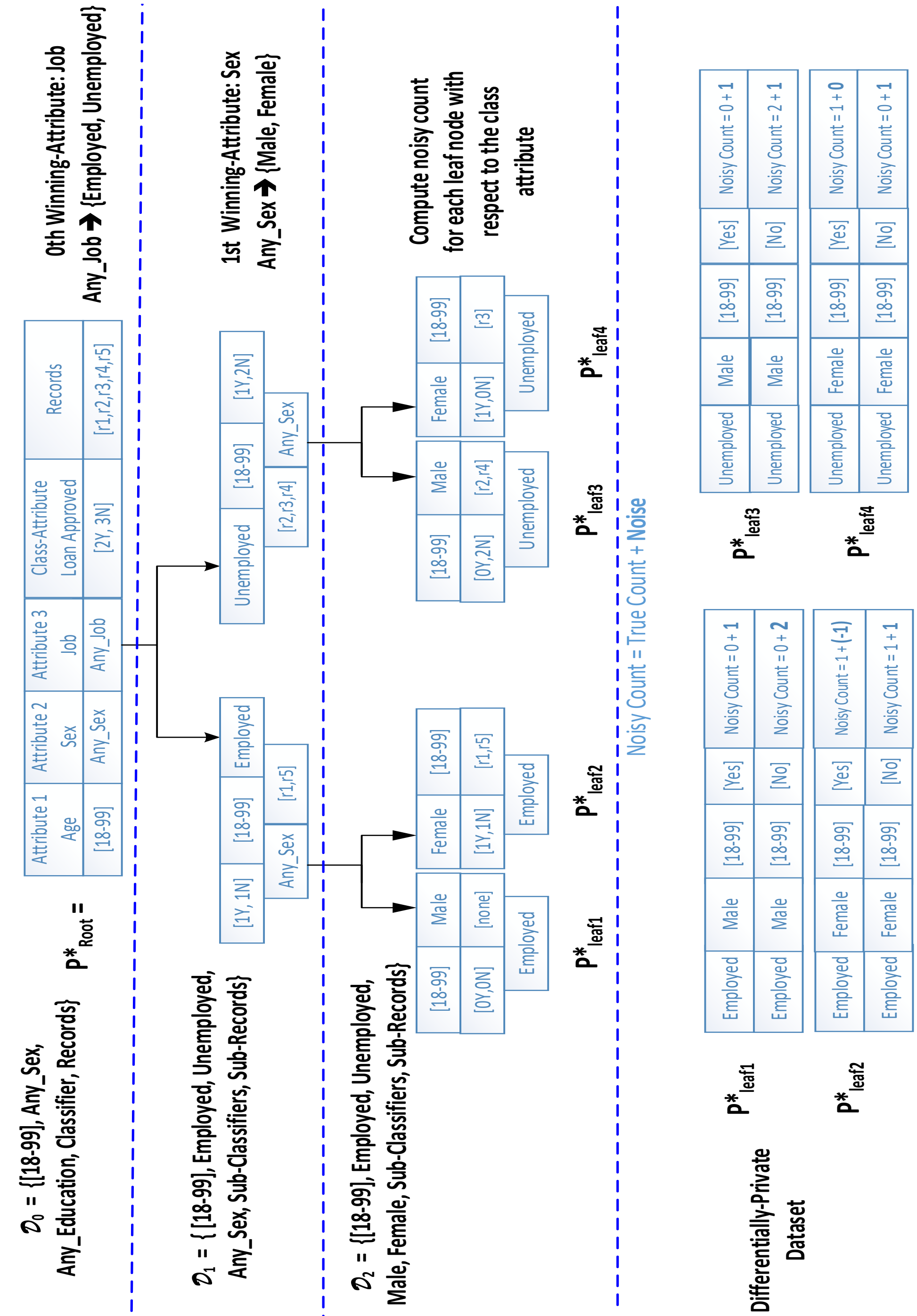


Figure 5.2: Partitioning Tree

trees, privately owned by each respective party. From a theoretical standpoint, we can construct a single Partitioning Tree $\mathcal{P}^* := \bigcap_{i=1}^{n} \mathcal{P}_i^*$, where the word "partition" (defined in **Definition 5.2.2**), directly corresponds to the partitions of $\mathcal{P}^*$(See Figure 5.2). These partitions are analogous to the root node and child nodes of $\mathcal{P}^*$. It is also very important to understand that $\mathcal{P}^*$ does not actually exist in practice. This is because the records must remain private and if some $P_j$ has access to $\mathcal{P}^*$, then $P_j$ would have direct access to everyone's records. Although no party has direct access to $\mathcal{P}^*$, we will eventually see why it is useful. From the reader's perspective, the root node of $\mathcal{P}^*$ will encapsulate the most general values of each attribute, as well as all the private records of each party. We describe this root as $\mathcal{D}_0 := \mathcal{P}^*_{root}$. From there, we run Protocol $4.1$ to acquire $A_0^w$ owned by some $P_j$, we specialize (or partition) $\mathcal{P}^*$ with respect to $A_0^w$. From our original root node, we now have 2 or more 1st order child-nodes. These 1st order child-nodes will be contained in $\mathcal{D}_1$. Similarly, we run Protocol $4.1$ again to acquire $A_1^w$, where each of the 1st order child-nodes, will produce 2nd order child-nodes (it is possible for child-nodes to be empty). All the non-empty 2nd order child-nodes would similarly be contained in $\mathcal{D}_2$. Since there are $\mathcal{S}$ many specializations, the process described will occur $\mathcal{S}$ many times. After the $\mathcal{S}$th specializations, the partitions that correspond to the $\mathcal{S}$th order child-nodes are called 'leaf nodes' or $\mathcal{P}_{leaf}$, which function as input for Protocol 5.2. In general, the set of $kth$ order child-nodes $Child_k(P^*)$ in the Partitioning Tree $P^*$ is described as $\mathcal{D}_k = Child_k(P^*) \subseteq \mathcal{P}^*$. Although the reader can see how $P^*$ can organize all the values and records contained in $\mathcal{D}$, the individual parties do not have this perspective. We can avoid this problem by examining a version of $P^*$ without records. When the records are removed from $P^*$, it is actually a taxonomy, which we designate as the unified taxonomy $\mathbf{T}$. $\mathbf{T}$ is really just an intersection of all the $\mathbb{T}_j$'s following the final $\mathcal{S}$th specialization. In the same way $P^*$ has kth-order child nodes $Child_k(P^*)$, $\mathbf{T}$ has corresponding $kth$ order child-nodes $\mathbf{T}_k = Child_k(\mathbf{T}) \subseteq \mathbf{T}$ that do not contain records.

**Definition 5.2.1.** **=** ***Taxonomic Intersection***

A taxonomic intersection between two taxonomies $\mathbb{T}_1$ and $\mathbb{T}_2$, described as $\mathbb{T} = \mathbb{T}_1 \cap \mathbb{T}_2$ is defined as follows:

Let $Child_k(\mathbb{T}_i)$ represent the $kth$ order non-empty child-node(s) of $\mathbb{T}_i$, where $Child_0(\mathbb{T}_i) := Root(\mathbb{T}_i)$. For $Child_a(\mathbb{T}_1)$ and $Child_b(\mathbb{T}_2)$, assume $0 \leq max(a) \leq max(b)$. Or in other words, the taxonomy of $\mathbb{T}_2$, is at least as tall (or long) as the taxonomy of $\mathbb{T}_1$. Also let $k_i \in Child_c(\mathbb{T}_i)$ be an attribute value in $\mathbb{T}_i$, where arbitrary attribute-values assume non-empty intersections (i.e $k_s \cap k_t \neq \emptyset$)

If $k_1 \in Child_a(\mathbb{T}_1)$ and $k_2 \in Child_b(\mathbb{T}_1)$, then $Child_a(\mathbb{T}_1) \cap Child_b(\mathbb{T}_2) := \{k_1 \cap k_2\}$

The $kth$ order child-node of $\mathbb{T}$ is given as,

$$Child_k(\mathbb{T}) := \begin{cases} Child_k(\mathbb{T}_1) \cap Child_k(\mathbb{T}_2) & \textit{if}\, 0 \leq k \leq a \\ Child_a(\mathbb{T}_1) \cap Child_k(\mathbb{T}_2) & \textit{if}\, a < k \leq b \end{cases}$$

**Definition 5.2.2.** **=** ***Partition***. A partition $\mathcal{C}$ is a tuple:
$[A_1, A_2, \ldots, A_{m_n}, A^c, Recs]$, where:

- $\{A_1, A_2, \ldots, A_{m_n}\}$ is the set of all attributes in $T$, where each attribute $A_i : 1 \leq i \leq m_n$ can be assigned a single value $A_i.value$ from its corresponding taxonomy tree $\mathbb{T}_{A_i}$. Any record assign to partition $\mathcal{C}$ must satisfy these values.
- $A^c$ attribute represents the count of each class value from the records assigned to $\mathcal{C}$.

- $Recs$ attribute contains the list of records assigned to $\mathcal{C}$.

**Example 5.2.2.** This example highlights how to intersect taxonomies $\mathbb{T}_{A_i}$ as well as constructing a Sub-Partitioning Tree $P_j^*$. Imagine party 1 has a dataset which contained three attributes $\{A_1 = Job, A_2 = Sex, A_3 = Age\}$ and a total of 5 records. For those who want to acquire a visual regrading Partitioning Trees or Sub-Partitioning Trees in Figure 5.2 We will define the values of each attribute as follows:

For $A_1.value$, $value \in \mathbb{T}_{A_1}$

$\mathbb{T}_{A_1} = \{Any\_Job_{root}, Employeed_{child_1}, Unemployed_{child_1}\} = \{Job, E, U\}$

For $A_2.value$, $value \in \mathbb{T}_{A_2}$

$\mathbb{T}_{A_2} = \{Any\_Sex_{root}, Male_{child_1}, Female_{child_1}\} = \{Sex, M, F\}$

For $A_3.value$, $value \in \mathbb{T}_{A_3}$

$\mathbb{T}_{A_3} = \{[18, 99]_{root}\} = \{Age\}$

Based on the values above we can see that both $A_1$ and $A_2$ were specialized once and $A_3$ was not specialized at all. Let us assume that $A_1$ is specialized first, followed by $A_2$. The order of specialization will dictate the development of the final taxonomy $\mathbb{T}_1$

For $\mathbb{T}_1 = \mathbb{T}_{A_1} \cap \mathbb{T}_{A_2} \cap \mathbb{T}_{A_3} = \mathbb{T}_{Job} \cap \mathbb{T}_{Sex} \cap \mathbb{T}_{Age}$

| $\mathbb{T}_1$ *(public)* | | |
|---|---|---|
| $Root(\mathbb{T}_1)$ | $Child_1(\mathbb{T}_1)$ | $Child_2(\mathbb{T}_1)$ |
| $Job \cap Sex \cap Age$ | $E \cap Sex \cap Age$ | $E \cap M \cap Age$ |
| – | $U \cap Sex \cap Age$ | $E \cap F \cap Age$ |
| – | – | $U \cap M \cap Age$ |
| – | – | $U \cap F \cap Age$ |

Observe $\mathbb{T}_1$ has 7 values (or classifications). If $P_1$ partitions (or assigns) its records $\mathcal{R}_1$ with respect to the values of $\mathbb{T}_1$, then $P_1$ can easily derive its Sub-Partitioning Tree $\mathcal{P}_1^*$. Think of $\mathcal{P}_1^*$ as party 1 'injecting' its records into $\mathbb{T}_1$. Let us say $\mathcal{R}_1 = \{r_1, r_2, r_3, r_4, r_5\}$, then for this example assume $P_1$ can construct the following:

| $\mathcal{P}_1^*$ *(private)* | | |
|---|---|---|
| $Root(P_1^*)$ | $Child_1(P_1^*)$ | $Child_2(P_1^*)$ |
| $\{r_1, r_2, r_3, r_4, r_5\}$ | $\{r_1, r_5\}$ | $\emptyset$ |
| – | $\{r_2, r_3, r_4\}$ | $\{r_1, r_5\}$ |
| – | – | $\{r_2, r_4\}$ |
| – | – | $\{r_3\}$ |

Based on $\mathcal{P}_1^*$, the reader (as well $P_1$) can make various conclusions such as the individuals that correspond to record 1 and record 5 are employed, female, and between the ages of 18 and 99 years old. For instance, the records corresponding to an employed female are $r_1$ and $r_5$

Since there were only 2 specializations among $P_1$'s attributes, that implies $\mathcal{S} = 2$. Thus $Child_2(P_1^*) = \{\mathcal{P}_{leaf_1}^*, \mathcal{P}_{leaf_2}^*, \mathcal{P}_{leaf_3}^*\} = \{\{r_1, r_5\}, \{r_2, r_4\}, \{r_3\}\}$, where empty child-nodes are disregarded. For simplicity, this toy example lacks a classifier attribute. The classifier attribute (which is public) would be treated exactly like the other attributes. Once the $P^*_{leaf}$'s appear, $P_1$ will determine which records in each $P^*_{leaf}$ satisfies the classifier attribute. ■

### 5.2.3 Computing Count Vector

(The count vector is the input for step 6 in the MAIN Protocol)

After the Protocol 4.1 (Multiparty Exponential Mechanism) is ran $\mathcal{S}$ times, the parties constructed their respective $P^*_j$. From there, each party will examine their respective leaf partitions $Child_{\mathcal{S}}(P^*_j) \in P^*_j$, and encode each partition. Each leaf partition will be encoded as binary vectors, where each element of each vector is either '1' or '0'.

Following an execution of Protocol 4.1 $\mathcal{S}$ number of times, we acquire the following sequence of winning-attributes: $A^w_0, A^w_1, \ldots, A^w_{\mathcal{S}-1}$. Each $A^w_i$ indicates how the respective owner party $P_j$ specializes their group taxonomy $\mathbb{T}_j$, which is initialized as a root node. After $\mathcal{S}$ many specializations, the parties construct a unified taxonomy $\mathbf{T} := \bigcap_{j=1}^n \mathbb{T}_j$, which is an intersection of each parties respective group taxonomy. Once the parties appropriately intersect their respective group taxonomies they examine the $\mathcal{S}$th order child-nodes of $\mathbf{T}$, described as $Child_{\mathcal{S}}(\mathbf{T})$. For each $P^*_{leaf} \in Child_{\mathcal{S}}(\mathbf{T})$, $P_j$ accounts for all the attributes-values that correspond to the attributes they privately own. $P_j$ will code the attribute-values into binary vectors. For a particular attribute-value, a record which satisfies an attribute-value and classifier attribute will be assigned a value of 1, otherwise the record will be assigned the value 0. The number of records that satisfies all the attribute-values and classifier within a particular $P^*_{leaf}$ is called the true count '$TCount$'. To acquire $TCount$

in a secure and private manner will require the construction of the encrypted count vector $\vec{C} := \langle [\![C_1]\!], [\![C_2]\!], \ldots, [\![C_p]\!], \ldots, [\![C_d]\!] \rangle$. Recall parties $P_1, P_2, \ldots, P_n$ own a unique set of attributes, where $P_j$ owns $M_j$ many attributes. We will assume there are $z$ attributes and $d$ records in total. Given $P^*_{leaf}$, each attribute-value that corresponds to an attribute that $P_j$ owns, can be converted into a **row vector** $\vec{V_{j_k}}$, where $1 \leq k \leq M_j$ and the $ith$ column of $\vec{V_{j_k}}$ corresponds to whether record $i$ satisfied a specific attribute-value and classifier. $P_j$ owns the following **attribute-vector(s)**

$$\vec{V_{j_k}} := \langle x_{1j\,[k]}, x_{2j\,[k]}, \ldots, x_{dj\,[k]} \rangle \tag{5.1}$$

Where $x_{ik} \in \{0, 1\}$, $k \in [1, \ldots, M_j]$

Although $P_j$ owns $M_j$ many attribute-vectors, we would prefer if each party had exactly one vector to reduce complexity. We also would like a single vector to preserve information of all the attribute-vectors that $P_j$ owns. $P_j$ can construct such a vector by taking all the $\vec{V_{j_k}}$'s it owns and multiply the respective columns elements together, creating $\vec{P_j}$. $\vec{P_j}$ is referred to as $P_j$'s **standardized attribute-vector** and is defined as:

$$\vec{P_j} := \langle \prod_{k=1}^{M_j} x_{1j\,[k]}, \prod_{k=1}^{M_j} x_{2j\,[k]}, \ldots, \prod_{k=1}^{M_j} x_{dj\,[k]} \rangle \tag{5.2}$$

To further simplify notation we have

$$\vec{P_j} = \langle X_{1,j}, X_{2,j}, \ldots, X_{d,j} \rangle, X_{ij} \in \{0, 1\} \tag{5.3}$$

Now that all the parties privately constructed their respective $\vec{P_j}$, we would like to add all the respective column components together, acquiring a single vector in the process.

However, if we are going to add the respective column components of each $\vec{P}_j$ then we need to do so in a secure manner. This can be done in a manner similar to Protocol 4.2, where Exponential ElGamal was the basis of secure homomorphic addition. Thus the parties can compute the following ElGamal encryptions,

$$[\![C_p]\!] := ((\prod_{i=1}^{n} g^{X_{pi}}) \cdot A^{(\sum_{i=1}^{n} r_i)}, g^{\sum_{i=1}^{n} r_i}), 1 \leq p \leq d \tag{5.4}$$

where all operations are in mod $p'$. To simplify the notation we have

$$[\![C_p]\!] = ((\prod_{i=1}^{n} g^{X_{pi}}) \cdot A^r, g^r) \tag{5.5}$$

Now we finally have the components to construct the encrypted **count-vector**.

$$\vec{C} := \langle [\![C_1]\!], [\![C_2]\!], \ldots, [\![C_p]\!], \ldots, [\![C_d]\!] \rangle \tag{5.6}$$

The $C_p$ element in $\mathcal{C}$ satisfies all specified attributes-values contained in some $P^*_{leaf}$ if and only if $C_p = n$, where $n$ represents the number of parties. By building $\vec{C}$, the parties now have the means of determining how many records in $\mathcal{D}$ satisfy a pre-determined set of attribute values(i.e a leaf node), along with a classifier.

**Example 5.2.3.** There are three parties, with 6 attributes and 10 records in total. $P_1$ owns job (professional, artist), sex (male, female) and salary ([1-10]). $P_2$ owns the education (As, Bs, Ms, PhD), while $P_3$ owns debt ([1-10]) and health (good, bad). for this example a record that is assigned a value of '1' satisfies both a predesignated attribute value and classifier $A^c$. Let us assume that $A^c$ corresponds to whether an individual was approved for a loan. The parties construct the following attribute-vectors:

$P_1$ owns the job attribute-vector

$\vec{V_{11}} = \vec{A_1} = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 0 \rangle$, *1* if professional

$P_1$owns the sex attribute-vector

$\vec{V_{12}} = \vec{A_2} = \langle 1, 1, 0, 1, 1, 1, 1, 1, 0, 1 \rangle$, 1 if female

$P_1$owns the salary attribute-vector

$\vec{V_{13}} = \vec{A_3} = \langle 1, 0, 0, 1, 1, 1, 0, 0, 0, 0 \rangle$, *1* if salary 5

$P_2$owns the education attribute-vector

$\vec{V_{21}} = \vec{A_4} = \langle 1, 1, 1, 1, 0, 1, 0, 1, 0, 1 \rangle$, *1* if bachelors

$P_3$owns the debt attribute-vector

$\vec{V_{31}} = \vec{A_5} = \langle 1, 0, 1, 1, 0, 1, 1, 1, 0, 0 \rangle$, 1 if debt 4

$P_3$owns the health attribute-vector

$\vec{V_{32}} = \vec{A_6} = \langle 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$, 1 if healthy

Now that each party has an attribute-vector for each attribute, $P_j$ multiplies the respective column elements among the attribute vectors it owns. $P_j$ then constructs its standardized-attribute vector $\vec{P_j}$.

$\vec{P_1} = \langle X_{1,1}, X_{2,1}, \ldots, X_{10,1} \rangle = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 0 \rangle$

corresponds to $professional \cap female \cap salary$

$\vec{P_2} = \langle X_{1,2}, X_{2,2}, \ldots, X_{10,2} \rangle = \langle 1, 1, 1, 1, 0, 1, 0, 1, 0, 1 \rangle$

corresponds to $bachelors$

$\vec{P_3} = \langle X_{1,3}, X_{2,3}, \ldots, X_{10,3} \rangle = \langle 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$

corresponds to $debt\ 4 \cap healthy$

Notice that $\vec{P_j}$ correspond to a partition involving an intersection of multiple attribute values. In this case, the records of $P_1$ are contained in the partition of $\mathcal{P}^*$ which only intersects Professional, Female, and Salary 5. Similarly $\mathcal{R}_2 \in \{Bachelors\} \in \mathcal{P}^*$ and $\mathcal{R}_3 \in \{Debt\,4 \cap Healthy\} \in \mathcal{P}^*$. After each $\vec{P_j}$ is privately derived, the parties homomorphically add their respective column elements among their standardize attribute-vectors. By doing this, the parties collectively construct $\vec{C}$.

$$\vec{C} = \langle [\![3]\!], [\![1]\!], [\![1]\!], [\![2]\!], [\![0]\!], [\![2]\!], [\![0]\!], [\![2]\!], [\![0]\!], [\![1]\!] \rangle$$

From the reader's perspective, we can see record 1 satisfies the intersection of all 6 attributes-values while records 5, 7, and 8 satisfy none of the attribute-values. Since the parties are interested in the number of records satisfying all attribute-values, called the 'True Count' ($TCount$). From the reader's perspective the $TCount$ in this example is equal to 1. In the next section we will described how the parties can manipulated and acquire a representative of $TCount$ given $\vec{C}$. ■

### 5.2.4 Deriving the Encrypted True Count

(Step 6 of the MAIN Protocol, where $\mathcal{C}$ is the input of Protocol 5.3)

After the parties construct their respective $\vec{P_j}$'s, they collaborate to compute $\vec{C}$. Their next objective is to determine $TCount$, where $TCount$ represents how many elements $[\![C_p]\!] \in \mathcal{C}$ satisfy all the attribute values among all parties. Or equivalently, the parties are interested in how many elements in $[\![C_p]\!] = [\![n]\!]$ . However, they do not want to reveal $TCount$, as that would breach the privacy of the records. So instead, the parties would like to derive an encrypted version of the $TCount$, described as $[\![TCount]\!]$. This can be achieved in Protocol 5.3. This protocol use a table $\mathcal{T}$ which contains two equally-sized columns, designated as 'input' and 'output'. The 'input' side of $\mathcal{T}$ contains all of the following values $\{[\![0]\!], [\![1]\!], [\![2]\!], \ldots, [\![n]\!]\}$, where $n$ designates the number of parties. The 'output' side of $\mathcal{T}$ initially has $n + 1$ many $[\![0]\!]$'s.

Table 5.1: General Table $\mathcal{T}$

| Input | Output |
|---|---|
| $[\![0]\!]$ | $[\![0]\!]$ |
| $[\![1]\!]$ | $[\![0]\!]$ |
| $\vdots$ | $\vdots$ |
| $[\![n-1]\!]$ | $[\![0]\!]$ |
| $[\![n]\!]$ | $[\![0]\!]$ |

In order to derive $[\![TCount]\!]$ we must first construct the "count table", $\mathcal{T}$. Refer to table 5.1 for a visual observation. It should be noted that $\mathcal{T}$ is typically not in numerical order, and is normally randomized. The numerical ordering was done so the reader can acquire visual intuition of $\mathcal{T}$. Once $\mathcal{T}$ is initialized, then it is defined as $\mathcal{T}_0$. Each element of $\mathcal{T}_0$ will be encrypted with a single joint public-key $A$ from $\vec{C}$, while using a temporary joint ephemeral-key $r$ (See Protocol 4.2). Party $P_j$ will take $[\![C_p]\!] \in \vec{C}$ and compare it with the elements of the input column, from top (1st row, 1st column) to bottom ($nth$ row, 1st column). $P_j$ will then conduct a Plaintext Equality Test (PET). If we designate the $kth$ input element in $\mathcal{T}$ as $[\![I_k]\!]$ ($kth$ row, 1st column), $P_j$ will verify if $1 = [\![C_p]\!]/[\![I_k]\!]$. If true, PET is verified which implies $C_p = I_k$. Since $C_p$ and $I_k$ are both encrypted $P_j$ only knows that both values are equal. However, $P_j$ does not know what $C_p$ or $I_k$ are numerically. Following verification, $P_j$ looks at the corresponding $kth$ output element, which we will describe as $[\![O_k]\!] \in \mathcal{T}$ ($kth$ row, 2nd column). $P_j$ then homomorphically adds $[\![1]\!]$ to $[\![O_k]\!]$, where $[\![O_k + 1]\!] = [\![O_k]\!] \times [\![1]\!]$. After the output element is incremented, the parties apply a mix-network to the table. This means the rows of $\mathcal{T}$ are randomly "shuffled" row-wise and each element of $\mathcal{T}$ are encrypted with a new joint ephemeral-key $r'$. Thus the $ith$ count table $\mathcal{T}_i$ is updated to $\mathcal{T}_{i+1}$. $P_j$ would similarly conduct PET and increment similarly on $\mathcal{T}_{i+1}$, but using $[\![C_{p+1}]\!] \in \vec{C}$ instead. This process will iterate from $[\![C_1]\!]$ to $[\![C_d]\!]$. After $\mathcal{T}_0$ is updated the $dth$ time as $\mathcal{T}_d$, they will make one final update given as $\mathcal{T}_d$. The parties will similarly examine the input column of $\mathcal{T}_d$. $P_j$ will then conduct PET with $[\![n]\!]$ and $[\![I_k]\!]$, from top to bottom of $\mathcal{T}_d$. Once PET is verified, $P_j$ will examine the corresponding output element $[\![O_k]\!]$. This $[\![O_k]\!]$ will not be modified or incremented in anyway. Once $P_j$ has this particular $[\![O]\!]$, $P_j$ now knows the encrypted true count where $[\![TCount]\!] = [\![O]\!]$.

**Example 5.2.4.** Assume $\vec{P_1} = \langle 1, 0, 1 \rangle$, $\vec{P_2} = \langle 1, 1, 1 \rangle$, $\vec{P_3} = \langle 1, 0, 1 \rangle$. We homomorphically add the columns of the standardized attribute-vectors yielding $\vec{C} = \langle [\![3]\!], [\![1]\!], [\![3]\!] \rangle = \langle [\![C_1]\!], [\![C_2]\!], [\![C_3]\!] \rangle$. Since there are three parties in this example that implies $n = 3$. The parties collectively construct the first table, designated as $\mathcal{T}_0$. Then begin the process with $[\![C_1]\!] = [\![3]\!]$.

**Initialized table $\mathcal{T}_0$**

$$\mathcal{T}_0 = \begin{bmatrix} [\![3]\!] & [\![0]\!] \\ [\![2]\!] & [\![0]\!] \\ [\![1]\!] & [\![0]\!] \\ [\![0]\!] & [\![0]\!] \end{bmatrix}$$

**First input element in $\mathcal{T}_0$ matches $[\![C_1]\!]$, designate as 'yes', increment output**

$$\mathcal{T}_0 = \begin{bmatrix} [\![3]\!] \overset{\textbf{yes}}{\leftarrow} & [\![1]\!] \\ [\![2]\!] & [\![0]\!] \\ [\![1]\!] & [\![0]\!] \\ [\![0]\!] & [\![0]\!] \end{bmatrix}$$

***Shuffle table row-wise and update ephemeral key***

**First input element in $\mathcal{T}_1$ matches $[\![C_2]\!]$, designate as 'yes', increment output** $[\![C_2]\!] = [\![1]\!]$

$$\mathcal{T}_1 = \begin{bmatrix} [\![1]\!] \overset{\textbf{yes}}{\leftarrow} & [\![1]\!] \\ [\![0]\!] & [\![0]\!] \\ [\![3]\!] & [\![1]\!] \\ [\![2]\!] & [\![0]\!] \end{bmatrix}$$

***Shuffle table row-wise and update ephemeral key***

$[\![C_3]\!] = [\![3]\!]$

$$\mathcal{T}_2 = \begin{bmatrix} [\![0]\!] \overset{\textbf{no}}{\leftarrow} & [\![0]\!] \\ [\![1]\!] & [\![1]\!] \\ [\![3]\!] & [\![1]\!] \\ [\![2]\!] & [\![0]\!] \end{bmatrix}$$

$$\mathcal{T}_2 = \begin{bmatrix} [\![0]\!] & [\![0]\!] \\ [\![1]\!] \overset{\textbf{no}}{\leftarrow} & [\![1]\!] \\ [\![3]\!] & [\![1]\!] \\ [\![2]\!] & [\![0]\!] \end{bmatrix}$$

**Third input element in $\mathcal{T}_2$ matches $[\![C_3]\!]$, designate as 'yes', increment output**

$$\mathcal{T}_2 = \begin{bmatrix} [\![0]\!] & [\![0]\!] \\ [\![1]\!] & [\![1]\!] \\ [\![3]\!] \stackrel{\textbf{yes}}{\leftarrow} & [\![2]\!] \\ [\![2]\!] & [\![0]\!] \end{bmatrix}$$

***Shuffle table row-wise and update ephemeral key***

$[\![n]\!] = [\![3]\!]$

$$\mathcal{T}_3 = \begin{bmatrix} [\![3]\!] \stackrel{\textbf{yes}}{\leftarrow} & [\![2]\!] \\ [\![2]\!] & [\![0]\!] \\ [\![1]\!] & [\![1]\!] \\ [\![0]\!] & [\![0]\!] \end{bmatrix}$$

**First input element in $\mathcal{T}_2$ matches $[\![C_3]\!]$, designate as 'yes', acquire corresponding output element** Each party $P_j$ obtains the output $[\![2]\!]$, which corresponded to the input value $[\![n]\!]$, thus $[\![TCount]\!] = [\![2]\!]$. ■

### 5.2.5 Deriving the Noisy Count

(Steps 4-8 in the Protocol 5.2(SPACE))

Once the parties acquire $[\![TCount]\!]$ their final objective is to add random noise to the $[\![TCount]\!]$, using the Laplace Mechanism. In other words, they would like to make the true count noisy, which in turn preserves $\epsilon$-differential privacy. Once noise is added to the encrypted true count, we acquire the encrypted noisy count $[\![NCount]\!] = [\![TCount + Laplace]\!] = [\![TCount]\!] \times [\![Laplace]\!]$. From there, decrypt the encrypted noisy count to get the following $NCount$.

After each party, $P_j$ acquires $[\![TCount]\!]$ they start on Step 4 of Protocol 5.2 to derive the noisy count. To do this, $P_j$ will sample two gamma random variables $Y_{1,j}, Y_{2,j} \sim Gamma(n, 1/\epsilon)$. $P_j$ will then subtract the gamma variables from each other, yielding $Y_j = Y_{1,j} - Y_{2,j}$. Using joint group-key $A$ with ephemeral-key $r_j$, $P_j$ encrypts $Y_j$ as $[\![Y_j]\!]$. The parties collectively multiply their encrypted values, acquiring $[\![Y]\!] := \prod_{i=1}^{n}[\![Y_i]\!]$. In this case, $Y \sim Laplace(0, 1/\epsilon)$ and $[\![Y]\!] := [\![Laplace]\!]$. The goal is to homomorphically add the encrypted true count to the encrypted Laplacian noise to get, $[\![TCount + Laplace]\!] = [\![TCount]\!] \times [\![Laplace]\!]$, where $[\![NCount]\!] = [\![TCount + Laplace]\!]$. Once the parties collectively acquire $[\![NCount]\!]$, they will use their private keys to decrypt, followed by a discrete logarithm-algorithm to acquire $NCount$.

**Example 5.2.5.** ***(Continuation of Example 5.2.4)*** Assume each of the three parties sampled two gamma variables $(Y_{1,j}, Y_{2,j})$, then subtract their values. $P_1$, $P_2$, and $P_3$ respectively own: $Y_1 = 0.45$, $Y_2 = -0.90$, and $Y_3 = 1.15$. From there the parties first encrypt, then homomorphically add those values. The parties collectively derive $[\![Y]\!] = [\![0.70]\!] = [\![0.45 - 0.90 + 1.15]\!]$. From the example, recall the parties acquired $[\![TCount]\!] = [\![2]\!]$. The parties homomorphically add $[\![TCount]\!]$ with $[\![Y]\!]$, acquiring $[\![NCount]\!] = [\![2 + 0.70]\!]$. After the parties have the encrypted noisy count, they jointly decrypt and apply a discrete-log algorithm to reveal $NCount = 2.7$. ■

### 5.2.6 Protocol Summary

The MAIN Protocol is able to construct a differentially private dataset that can be used for data mining. The protocol initially begins with each party owning their respective attributes and agreeing on a privacy budget $\epsilon$. It is already assumed that the taxonomy of the categorical attributes are already established. It is also assumed that all attributes are initially unspecialized as a single root node. For the numerical attributes $A_i^n$, the owner party will determine the corresponding split point. Once the split values are initialized, each party will use a utility function $u(\mathcal{D}, A_i)$ to compute a utility score $U_i$ for each $A_i$ they own. After $P_j$ has its collection of attribute-score pairs $\Phi_j$, the parties begin the Multiparty Exponential Mechanism 4.1 to determine the $qth$ winning attribute $A_q^w$. When $A_q^w$ is determined, the owner of $A_q^w$ will update the corresponding taxonomy $\mathbb{T}_{A^w}$, which in turn updates their group taxonomy $\mathbb{T}_j$, which also in turn updates the root-node of the unified taxonomy $\mathbf{T}_0 \in \mathbf{T}$. It should be noted that all taxonomies are public and $\mathbf{T}$ is an intersection of all taxonomies, and updates relative to how each $\mathbb{T}_{A^w}$ updates. We acquire $\mathcal{S}$ many winning-attributes, meaning that $\mathbf{T}$ will be specialized $\mathcal{S}$ times, where the final 'level' of partitions correspond to $\mathbf{T}_{\mathcal{S}}$. Each partition in $\mathcal{T}_{\mathcal{S}}$ is regarded as a leaf partition $P^*_{leaf}(\mathbf{T})$. The parties will collectivelly examine each $P^*_{leaf}(\mathbf{T})$ and derive an $NCount$ through Protocol 5.2. This $NCount$ represents a differentially private version of the 'true count', described as $TCount$. $TCount$ represents how many records satisfy all the attribute values contained in a specific predetermined set of $P^*_{leaf}(P^*_j)$'s owned among the parties. Once each leaf partition in $\mathbf{T}$ has a corresponding $NCount$, the parties have a collection of $Leaf - NCount$ pairs $\{(P^*_{leaf}(\mathbf{T}), NCount)\}$. This collection of Leaf-NCount pairs is actually our differentially-private dataset $\hat{D}$, therefore each party acquires $\hat{D} = \{(P^*_{leaf}(\mathbf{T}), NCount)\}$ as an output.

## 5.3 Protocol Analysis

### 5.3.1 Security Analysis

In this section we will review the security of the MAIN Protocol. To verify security, we must address moments when the parties specifically communicate with each other. There exist three moments where the parties actively communicate with each other Protocol 4.1 (Multiparty Exponential Mechanism), Protocol 5.2 (Secure & Private Attribute-Counting Exchange), and Protocol 5.3 (Mix and Match Count). We previously verified the security of the Multiparty Exponential Mechanism. Thus we only need to address the other two protocols. We will first begin by confirming whether Mix and Match Count Protocol(MMC) is provably secure. It should be noted that MMC uses a cryptographic primitive described as “Mix Network”. In the academic literature, Mix Network has many variants, some of which are provably secure in a multiparty semi-honest setting. Since the security of Mix Network has been demonstrated in the academic literature, we will not reprove its security in this paper.

**Establishing Key Values For the Mix and Match Count Protocol**

1. We have the following for $P_j$
   - $x_j = (\vec{C}, z^*)$
   - $r_j^* = (a_j, \vec{r_j})$

     Before $\mathcal{T}_i$ is re-randomized, re-shuffled, and updated to $\mathcal{T}_{i+1}$, $P_j$ uses a new ephemeral key $r_{ji} \in \vec{r_j}$, where $|\vec{r_j})| = d + 1$
   - $m = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_d, [\![TCount]\!])$
   - $View_j^{\Pi}(x_j, f_j(\vec{x})) = (x_j, r_j^*, m)$
   - $Output_j^{\Pi}(\vec{x}) = [\![TCount]\!]$
   - $f_j(\vec{x}) = [\![TC\acute{o}unt]\!]$
2. We have the following simulated output-messages for $S_j$

- $S_j(x_j, f_j(\vec{x})) = (x_j, r_j^*, \mathcal{T}_0^*, \mathcal{T}_1^*, \ldots, \mathcal{T}_d^*, [\![TCount^*]\!])$,

**Lemma 5.** Given the initial-input $x_j$ of each party $P_j$, the final-output of the trusted third-party $f$ is indistinguishable from the final-output of the real protocol. Or equivalently $f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$, where $\Pi$ is the Mix and Match Count Protocol and $\vec{x} = (x_1, x_2, \ldots, x_n)$.

*Proof.* When the ideal functionality $f$ is given the initial-input vector $\vec{x}$, we get the following output: $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})) = ([\![TCount]\!], [\![TCount]\!], \ldots, [\![TCount]\!])$. For the real protocol, when given the same initial-input vector $\vec{x}$ we similarly get the following output: $Output^{\Pi}(\vec{x}) = (Output_1^{\Pi}(\vec{x}), Output_2^{\Pi}(\vec{x}), \ldots, Output_n^{\Pi}(\vec{x})) = ([\![TC\acute{o}unt]\!], [\![TC\acute{o}unt]\!], \ldots, [\![TC\acute{o}unt]\!])$. So we need to verify that $[\![TCount]\!] \stackrel{c}{\equiv} [\![TC\acute{o}unt]\!]$. By the Diffie-Hellman assumption, this is trivially true. Recall that ElGamal encryption is semantically secure. Thus given two unique ciphertext, one cannot make any conclusion about plaintext. Since no conclusion about the plaintext can be made, the ciphertext are indistinguishable.

$\therefore f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$

$\square$

**Lemma 6.** *The output messages of the simulator during an ideal (or simulated) execution of $\Pi$ is indistinguishable from the messages that $P_j$ would have received ($P_j$'s view) during a real execution of $\Pi$. Or equivalently $S_j(x_j, f_j(\vec{x})) \stackrel{c}{\equiv} View_j^{\Pi}(\vec{x})$, where $\Pi$ is the Mix and Match Count Protocol.*

*Proof.* Using proof by contradiction, let us assume,

$S_j(x_j, f_j(\vec{x})) \overset{c}{\not\equiv} View_j^{\Pi}(\vec{x})$

$$\implies S_j(x_j, [\![TCount^*]\!]) \overset{c}{\not\equiv} (x_j, r_j^*, \mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_d, [\![TCount]\!])$$
$$\implies S_j(x_j, [\![TCount^*]\!]) \overset{c}{\not\equiv} (x_j, r_j^*, \{\mathcal{T}_i\}, [\![TCount]\!])$$
$$= (x_j, r_j^*, \mathcal{T}_0^*, \mathcal{T}_1^*, \ldots, \mathcal{T}_d^*, [\![TCount^*]\!]) \overset{c}{\not\equiv} (x_j, r_j^*, \{\mathcal{T}_i\}, [\![TCount]\!])$$
$$= (x_j, r_j^*, \{\mathcal{T}_i^*\}, [\![TCount^*]\!]) \overset{c}{\not\equiv} (x_j, r_j^*, \{\mathcal{T}_i\}, [\![TCount]\!])$$

$x_j$ and $r_j^*$ are identical for $P_j$ and $S_j$. Identical terms are trivially indistinguishable.

$$\implies (\{\mathcal{T}_i^*\}, [\![TCount^*]\!]) \overset{c}{\not\equiv} (\{\mathcal{T}_i\}, [\![TCount]\!])$$
$$\implies (\mathcal{T}_i^*, [\![TCount^*]\!]) \overset{c}{\not\equiv} (\mathcal{T}_i, [\![TCount]\!])$$

From the Diffie-Hellman assumption recall $g^c \;\; mod\, p \overset{c}{\equiv} g^b \;\; mod\, p$. Each element in $\mathcal{T}_i$ and $\mathcal{T}_i^*$ can be represented as $g^x \;\; mod\, p$, where $x$ is a positive integer. Thus, each table is indistinguishable from each other, or equivalently $\mathcal{T}_i \overset{c}{\equiv} \mathcal{T}_i^*$. We can make the following reduction.

$([\![TCount^*]\!]) \overset{c}{\not\equiv} ([\![TCount]\!])$

However this contradicts the Diffie-Hellman assumption. Thus our original assumption must have been false.

$\therefore S_j(x_j, f_j(\vec{x})) \overset{c}{\equiv} View_j^{\Pi}(\vec{x})$

□

**Theorem 5.3.1.** *The Mix and Match Count Protocol is secure in the multiparty semi-honest setting*

*Proof.* Let us assume $\Pi$ is the Mix and Match Count Protocol. By Lemma 5, we proved $f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$. And by Lemma 6, we also proved $S_j(x_j, f_j(\vec{x})) \stackrel{c}{\equiv} View_j^{\Pi}(\vec{x})$. Therefore $\Pi$ is secure in the multiparty semi-honest setting which is defined as.

$$\{(S_j(x_j, f_j(\vec{x})), f(\vec{x}))\} \stackrel{c}{\equiv} \{(View_1^{\Pi}(x_j, f_j(\vec{x})), Output^{\Pi}(\vec{x}))\}$$

$\square$

Since we verified the security of MMC, the only thing left to do is verify the security of the SPACE Protocol. This proof will have a similar flow to the other security proofs. the only distinction is we invoke the fact that simulator $S_J$ has access to $P_j$'s random tape, specifically in Lemma 8. This knowledge will allow $S_j$ to decrypt values at will by being able to access the outcome of any event conducted by $P_j$ which depends on chance. We did not invoke this tactic in our other security proofs because it was not necessary to do so.

**Establishing Key Values For the SPACE Protocol**

1. We have the following for $P_j$

   - $x_j = (P_{leaf}, z^*)$
   - $r_j^* = (a_j, \vec{r'_j}, Y_j)$
     $P_j$ uses a unique ephemeral key $r'_{ji} \in \vec{r'_j}$ for each element in $\vec{P_j}$, where $\vec{r'_j} = d$.
   - $m = (\vec{C}, [\![TCount]\!], [\![\vec{Y_k}]\!], [\![Y]\!], [\![NCount]\!], \alpha, NCount)$

- $View_j^{\Pi}(x_j, f_j(\vec{x})) = (x_j, r_j^*, m)$
- $Output_j^{\Pi}(\vec{x}) = NCount$
- $f_j(\vec{x}) = N\acute{C}ount$

2. We have the following simulated output-messages for $S_j$

- $S_j(x_j, f_j(\vec{x})) = (x_j, r_j^*, \vec{C^*}, [\![TCount^*]\!], [\![\vec{Y_{S(k)}}]\!], [\![Y^*]\!], [\![NCount^*]\!], \alpha^*, NCount^*)$,

**Lemma 7.** Given the initial-input $x_j$ of each party $P_j$, the final-output of the trusted third-party $f$ is indistinguishable from the final-output of the real protocol. Or equivalently $f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$, where $\Pi$ is the Secure & Private Attribute-Counting Exchange Protocol and $\vec{x} = (x_1, x_2, \ldots, x_n)$.

*Proof.* For the ideal functionality $f$, when given input $\vec{x}$ we get the following output: $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})) = (NCount, NCount, \ldots, NCount)$. For the real protocol, when given the input $\vec{x}$ we similarly get the following output: $Output^{\Pi}(\vec{x}) = (Output_1^{\Pi}(\vec{x}), Output_2^{\Pi}(\vec{x}), \ldots, Output_n^{\Pi}(\vec{x})) = (N\acute{C}ount, N\acute{C}ount, \ldots, N\acute{C}ount)$. So we need to verify that $NCount \stackrel{c}{\equiv} N\acute{C}ount$. Based on the probabilistic nature of this protocol It should be assumed $NCount \neq N\acute{C}ount$. Recall that $NCount = TCount + Laplace$, where $Laplace$ is a random variable selected from the random distribution $Laplace(0, 1\backslash\epsilon)$, which is notationally equivalent to $Laplace \sim Laplace(0, 1\backslash\epsilon)$. Define $\Theta := TCount + Laplace(0, 1\backslash\epsilon)$. Thus, that implies $NCount$ and $N\acute{C}ount$ are sampled from the same distribution, or equivalently $NCount, N\acute{C}ount \sim \Theta$. Although $NCount \neq N\acute{C}ount$, since the participants are semi-honest and the final-output falls within a probability distribution, the values cannot be distinguished. In other words, assuming $P_j$ already derived $NCount$ after executing $\Pi$ once, given the same inputs from all parties, $P_j$ would expect that its new final-output $NCount_2$ is not equal to $NCount$ for the second execution of $\Pi$. However $NCount_2$ is arbitrary. Thus $NCount \neq NCount_2$ is no different than $NCount \neq NC\acute{o}unt$. This suggests if $P_j$ were sent $NC\acute{o}unt$ or $NCount_2$, $P_j$ would assume either value is valid.

$$\therefore f(\vec{x}) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x})$$

□

**Lemma 8.** The output messages of the simulator during an ideal (or simulated) execution of $\Pi$ is indistinguishable from the messages that $P_j$ would have received ($P_j$'s view) during a real execution of $\Pi$. Or equivalently $S_j(x_j, f_j(\vec{x})) \stackrel{c}{\equiv} View_j^{\Pi}(\vec{x})$, where $\Pi$ is the Secure & Private Attribute-Counting Exchange Protocol.

*Proof.* Using proof by contradiction, let us assume that

$$S_j(x_j, f_j(\vec{x})) \stackrel{c}{\not\equiv} View_j^{\Pi}(\vec{x})$$

$$\implies S_j(x_j, NCount) \stackrel{c}{\not\equiv} (x_j, r_j^*, \vec{C}, [\![TCount]\!], [\![\vec{Y_k}]\!], [\![Y]\!], [\![NCount]\!], \alpha, NCount)$$
$$\implies S_j(x_j, NCount) \stackrel{c}{\not\equiv} (x_j, r_j^*, \vec{C}, \{g^{\beta}\}, \alpha, NCount)$$
$$\implies (x_j, r_j^*, \vec{C^*}, [\![TCount^*]\!], [\![\vec{Y_{S(k)}}]\!], [\![Y^*]\!], [\![NCount^*]\!], \alpha^*, NCount^*) \stackrel{c}{\not\equiv}$$
$$(x_j, r_j^*, \vec{C}, \{g^{\beta}\}, \alpha, NCount)$$
$$\implies (x_j, r_j^*, \vec{C^*}, \{g^{\Gamma}\}, \alpha^*, NCount^*) \stackrel{c}{\not\equiv} (x_j, r_j^*, \vec{C}, \{g^{\beta}\}, \alpha, NCount)$$

$x_j$ and $r_j^*$ are identical for $P_j$ and $S_j$. Identical terms are trivially indistinguishable.

$\implies (\vec{C^*}, \{g^\Gamma\}, \alpha^*, NCount^*) \overset{c}{\not\equiv} (\vec{C}, \{g^\beta\}, \alpha, NCount)$

$\implies (\vec{C^*}, g^c \ \ mod\, p, \alpha^*, NCount^*) \overset{c}{\not\equiv} (\vec{C}, g^b \ \ mod\, p, \alpha, NCount)$

From the Diffie-Hellman assumption recall $g^c \ \ mod\, p \overset{c}{\equiv} g^b \ \ mod\, p$.

$\implies (\vec{C^*}, \alpha^*, NCount^*) \overset{c}{\not\equiv} (\vec{C}, \alpha, NCount)$

$\vec{C^*}$ is similarly composed of sematically secure elements, thus we can make a further reduction

$\implies (\alpha^*, NCount^*) \overset{c}{\not\equiv} (\alpha, NCount)$

Recall that $\alpha$ is actually $NCount$ before the Discrete-Logarithm algorithm is applied. This means that $\alpha$ is a redundant representation of $NCount$. Thus we can make the following reduction.

$\implies (NCount^*) \overset{c}{\not\equiv} (NCount)$

All is left to show is that $S_j$ can derive $NCount^*$ such that $NCount^* = NCount$. Recall that $S_j$ has access to $P_j$'s random tape $r_j^*$. Since $S_j$ knows $P_j$'s random tape, $S_j$ knows that $P_j$ selected $Y_j$. $S_j$ also knows $P_j$'s private key $a_j$ and ephemeral keys $\vec{r_j}$. This suggests $S_j$ can directly derive $P_j$'s $[\![Y_j]\!]$. Now we can proceed with the following algebraic argument.

$[\![NCount]\!] = [\![NCount^*]\!]$

$\implies [\![NCount]\!] = [\![TCount^*]\!] \times [\![Y^*]\!]$

$\implies [\![NCount]\!] = [\![TCount^*]\!] \times [\![\sum_{k\in\Psi} Y_{S(k)} + Y_j]\!]$

Since $S_j$ knows $r_j^*$, $S_j$ has $P_j$'s encryption components(private key $a_j$, ephemeral keys $\vec{r_j}$) as well as the randomly generated $Y_j$ value. $S_j$ also used its own encryption components while simulating each $P_k$. Thus $S_j$ can easily decrypt and algebraically manipulate the homomorphic equation as follows,

$\implies NCount = TCount^* + (\sum_{k\in\Psi} Y_{S(k)} + Y_j)$
$\implies NCount - Y_j = TCount^* + (\sum_{k\in\Psi} Y_{S(k)})$
$\implies (NCount - TCount^*) - Y_j = (\sum_{k\in\Psi} Y_{S(k)})$
Let $C := (NCount - TCount^*) - Y_j$, where $C \in \mathbb{R}$ is a fixed value.

$\implies C = \sum_{k\in\Psi} Y_{S(k)}$

In order for $NCount^*$ to be equal to $NCount$, $S_j$ needs to appropriately choose its simulated values $Y_{S_j(k)}$. Since $NCount$, $TCount^*$, and $Y_j$ are known by $S_j$ this is easily achievable. Note that $S_j$ does not have full control of $NCount$, $TCount^*$, and $Y_j$ since these values are dependent on the intermediate values that $P_j$ executes throughout the protocol. Those intermediate values are either unknown to $S_j$, or fixed. Assuming $S_j$ behaves optimally in the semi-honest setting, we conclude $NCount^* = NCount$ for all simulations. However this contradicts $(NCount^*) \overset{c}{\not\equiv} (NCount)$, meaning our original assumption $S_j(x_j, f_j(x_j, x_k)) \overset{c}{\not\equiv} View_j^{\Pi}(x_j, f_j(x_j, x_k))$ is false.

$\therefore S_j(x_j, f_j(\vec{x})) \overset{c}{\equiv} View_j^{\Pi}(\vec{x})$

$\square$

**Theorem 5.3.2.** The Secure & Private Attribute-Counting Exchange (SPACE) Protocol is secure in the multiparty semi-honest setting.

*Proof.* From Theorem 7.2.1, we proved that the intermediate protocol Mix and Match Count is secure in the multiparty setting. In Lemma 7 we proved $f(\vec{x})) \stackrel{c}{\equiv} Output^{\Pi}(\vec{x}))$, where $\Pi$ represents the SPACE protocol. Also in Lemma 8 we showed $S_j(x_j, f_j(\vec{x})) \stackrel{c}{\equiv} (View_1^{\Pi}(x_j, f_j(\vec{x}))$. Therefore SPACE satisfies the below equation, making it secure in the multiparty semi-honest setting.

$$\{(S_j(x_j, f_j(\vec{x})), f(\vec{x}))\} \stackrel{c}{\equiv} \{(View_1^{\Pi}(x_j, f_j(\vec{x})), Output^{\Pi}(\vec{x}))\}$$

□

**Theorem 5.3.3.** The Multiparty (Main) Protocol is secure in the multiparty semi-honest setting.

*Proof.* Theorem 7.2.2 and 5.2.1 encapsulates every communication-based protocol in the Main Protocol. Since all communication is secure in the multiparty semi-honest setting, then we conclude the Main Protocol is secure in the multiparty semi-honest setting. □

### 5.3.2 Complexity Analysis

**Proposition 5.3.1.** *(Complexity)* The total encryption and communication costs among $n$ parties for the Protocol 5.1 is respectively bounded by $\mathcal{O}(n^2\xi)$ and $\mathcal{O}(dn^3K)$, where $d$ are the number of records, $n$ is the number of parties, and $K$ is the bit length.

Let us begin with Protocol 5.3 (MMC). Recall that $[\![m]\!] := (A^r \cdot g^m modp, g^r modp)$ requires 3 exponentiations. We will assume the worst case scenario, where $\vec{C} := \langle [\![n]\!]_1, [\![n]\!]_2, \ldots, [\![n]\!]_d \rangle$

and for all $i \in [0 \ldots, d]$, each $\mathcal{T}_i$ is defined as,

$$\mathcal{T}_i = \begin{bmatrix} [\![1]\!] & [\![x_1]\!] \\ [\![2]\!] & [\![x_2]\!] \\ \vdots & \vdots \\ [\![n-1]\!] & [\![x_{n-1}]\!] \\ [\![n]\!] & [\![x_n]\!] \end{bmatrix}$$

We can easily deduce that each $\mathcal{T}_i$ has $3 \cdot 2n$ many exponentiations, implying there's $3 \cdot 2n(d+1)$ many exponentiations among all the $\mathcal{T}_i$'s. For each $k \in [0, \ldots, d-1]$, $\mathcal{T}_k$ has exactly $n$ many PET tests, and 1 homomorphic addition. Both a PET and homomorphic addition requires 3 many exponentiations. This means $\mathcal{T}_k$ accounts for $3n \cdot (d-1) + 3 \cdot (d-1)$ many exponentiations, respectively. On the other hand, $\mathcal{T}_d$ has exactly $n$ many PET tests, with 0 homomorphic additions. Thus, $\mathcal{T}_d$ accounts for $3n \cdot 1$ many exponentiations. Therefore, by taking the sum of the previously mentioned values, MMC accounts for $9dn + 3d + 6n$ many exponentiations, where $d >> n$. MMC also uses a Mix Network protocol for the sake of randomization. Since there are several variations of Mix Networks, we assume the complexity of some Mix Network to be $\Omega$. Thus the encryption complexity for MMC is $O(\Omega + 9dn + 3d + 6n) = O(\Omega + dn)$.

For Protocol 5.2(SPACE) each party independently constructs $\vec{P_j}$, which requires $3d$ many exponentiations per party. Then the parties homomorphically compute $\vec{C}$, which is $3d \cdot n$ many exponentiations. After conducting MMC, the parties compute $[\![Y_k]\!]$ and $[\![Y]\!]$, which both require 3 many exponentiations per party. And finally decryption requires 3 many exponentiations per party. Thus, SPACE requires $6d + 6$ many exponentiations per party, meaning its encryption complexity is given as $\mathcal{O}(3dn + 3d + 6) = \mathcal{O}(dn)$. Recall the encryption complexity of Protocol 4.1 is $O(n^2\xi + zn)$. Since the Exponential mechanism is ran $\mathcal{S}$ times, we get the following $\mathcal{O}(\mathcal{S} \cdot n^2\xi + zn)$, which trivially reduces down to $\mathcal{O}(n^2\xi)$. Thus the encryption cost of Protocol 5.1(MAIN) is given by $\mathcal{O}(\Omega + dn + n^2\xi)$, reduces to $\mathcal{O}(\Omega + n^2\xi)$. Since the complexity is low for the Mix Network we can make the final reduction $\mathcal{O}(n^2\xi)$.

For communication cost, lets first examine MMC. Each encrypted element in $\mathcal{T}_i$ has an encryption cost of $n(n-1)K$, where $K$ designates the key size. Thus the communication cost of each $\mathcal{T}_i$ is $2n \cdot n(n-1)K$, where there is $d+1$ many $\mathcal{T}_i$'s. This means that generating all $d+1$ tables requires a communication cost of $2n \cdot n(n-1)K \cdot (d+1)$. For SPACE, line 2, 4, 5 and 6 has the following respective communication costs: $d \cdot n(n-1)K$, 0, $n(n-1)K$, and $nK$. Thus the communication cost for SPACE is $\mathcal{O}(((d+1) \cdot n^3)K)$. Recall the communication cost for Protocol 4.1 is $\mathcal{O}(n^2\zeta + nK)$. And for some Mix Network, we will assume the communication cost is $\Psi$. Thus for Protocol 5.1(MAIN), the communication cost is $\mathcal{O}(\Psi + (d+1)n^3K + n^2\zeta + nK) = \mathcal{O}((dn^3)K)$, since the Mix Network complexity is low.

### 5.3.3 Correctness Analysis

**Theorem 5.3.4.** Assuming all parties are semihonest, Protocol 5.1(MAIN) releases $\epsilon$-differentially private data when all parties hold different attributes for the same set of individuals

The MAIN Protocol has three major components which consumes the original privacy budget $\epsilon$: attribute selection, updating the taxonomy, and computing the noisy count. In this section, we will briefly overview their correctness and then algebraically prove that the MAIN Protocol preserves $\epsilon$-differential privacy.

- *Attribute Selection*

  MAIN selects an attribute distributed among $n$ parties $\mathcal{S}$ number of times using the MultiParty Exponential Mechanism. MAIN uses an exponential mechanism to select a winning-attribute $A^w$ distributed among the parties. Since an exponential mechanism was applied, then by Theorem 2.3.1 (exponential mechanism), $A^w$ is selected in a differentially private manner. The total privacy budget is $\epsilon$, however for the attribute selection we assume that consume $\epsilon_1$ much of the privacy budget per selection, where $\epsilon_1 < \epsilon$.

- *Updating* $\mathbf{T}$

  $\mathbf{T}$ is the taxonomic representation of the partitioning tree $P^*$. $\mathbf{T}$ is publicly available, and updates with respect to the taxonomy of some winning-attribute $A^w$. When a winner is selected, both $\mathbb{T}_{A^w}$ and $A^w$ are publicly announced among the $n$ parties. When categorical attributes are specialize, we do not consume any of the privacy budget. This is because the taxonomy of all categorical attributes are already known in advance, thus there is no privacy to protect. However, for numerical attributes, their taxonomy is dictated by an exponential mechanism. Since we used the exponential mechanism to specialize the taxonomy of a numerical attribute, we consumed some of the privacy budget. In either case, if we are dealing with with categorical or numerical attributes, differential privacy is preserved. Each time a numerical attribute is updated, we assume that we consume $\epsilon_2$ much of the budget, where $\epsilon_2 < \epsilon$.

- *Computing the Noisy Count*

  MAIN reaches the final update after our $\mathcal{S}th$ specialization, when we encounter leaf nodes. All the leaf nodes are contained in outputs $\mathcal{D}_{\mathcal{S}}$, where $P^*_{leaf} \in \mathcal{D}_{\mathcal{S}}$. MAIN indirectly derives and encrypts the true count as $[\![TCount]\!]$ for some $P^*_{leaf}$. For each respective $P^*_{leaf}$, the parties compute a noisy count $NCount$, where $[\![TCount]\!] \rightarrow NCount$. Since the noise being added to $TCount$ is $Lap(1/\epsilon')$, by Theorem 2.3.2 (laplace mechanism), it is differentially private. Let use assume that each time we use the Laplace Mechanism we consume $\epsilon_3$ much of the budget, where $\epsilon_3 < \epsilon$.

Let us by assuming the following:

*1.* Per attribute, the Multiparty Exponential Mechanism, consumes $\epsilon_1 := \frac{\epsilon}{4\mathcal{S}}$ of the privacy budget.

*2.* Per attribute, the Exponential Mechanism used to determine the split value of a numerical attributes consumes $\epsilon_2 := \frac{\epsilon}{4\mathcal{S}\mathcal{N}}$.

*3.* For each $P^*_{Leaf}$, the Laplace Mechanism used to add numerical noise to the leaf partitions of $P^*$ to acquire the noisy count, consumes $\epsilon_3 := \epsilon/2$ of the privacy budget.

**Claim: The cumulative budget for (1)-(3) does not exceed $\epsilon$**

The Multiparty Exponential Mechanism is conducted $\mathcal{S}$ many times. The Exponential Mechanism is also done $\mathcal{S}$ and applied among the $\mathcal{N}$ numerical attributes per iteration. Thus by Theorem 2.3.3(sequential composition), we can sum the respective budgets

$$
\begin{aligned}
&\mathcal{S} \cdot \epsilon_1 + \mathcal{S} \cdot \mathcal{N} \cdot \epsilon_2 \\
\implies &\mathcal{S} \cdot (\tfrac{\epsilon}{4\mathcal{S}}) + \mathcal{S} \cdot \mathcal{N} \cdot (\tfrac{\epsilon}{4\mathcal{S}\mathcal{N}}) \\
\implies &\tfrac{\mathcal{S} \cdot \epsilon}{4\mathcal{S}} + \tfrac{\mathcal{S} \cdot \mathcal{N} \cdot \epsilon}{4\mathcal{S}\mathcal{N}} \\
\implies &\tfrac{\epsilon}{4} + \tfrac{\epsilon}{4} \\
= &\tfrac{\epsilon}{2}
\end{aligned}
$$

Thus (1) and (2) consumes half the privacy budget. As for (3), recall that each leaf partition $P^*_{Leaf}$ uses $\frac{\epsilon}{2}$ of the privacy budget. Although this may seem like we would quickly go over budget, this is justified by Theorem 2.3.4. The leaf partitions are all derived from the original dataset $\mathcal{D}$, but are all disjoint from each other content-wise. Since we are applying the same mechanism to disjoint sets, each set can consume the same budget $\epsilon'$ is overall equivalent to $\epsilon'$-differential privacy. Thus for the Laplace Mechanism we can easily assign a privacy budget of $\frac{\epsilon}{2}$.

$\therefore$ The MAIN Protocol consumes exactly $\epsilon = \frac{\epsilon}{2} + \frac{\epsilon}{2}$ of the privacy budget, meaning it preserves $\epsilon$-differential privacy. ■

# Chapter 6

# CONCLUSION

## 6.1 Summary

In this thesis, we were able to construct a protocol that allows parties to integrate their vertically-partitioned datasets in a secure and private manner. The protocol needed to account for several instances of communication between the parties, where correspondence needed to be provably secure. Using proof by simulation, we were able to prove that all instances of communication were secure in the semi-honest setting for $n$ parties, where $n \geq 2$. To ensure our the integrated dataset $\hat{D}$ maintained privacy, we specialized the dataset using a differential-private exponential mechanism and slightly perturbed numerical data using Laplacian noise. From our efforts we found an efficient means of publishing data in a multiparty setting, while preserving the privacy and confidentiality of the original datasets. Such a certainly would allow organizations to freely cooperate without the risk of leaking sensitive data inadvertently. Preserving data is becoming a more relevant issue as companies who inadvertently leak information risk hurting their brand and/or legal repercussions. Therefore we hope to pave a path where data owners can freely cooperate in a manner which optimizes security, privacy, and data analytics.

## 6.2 Future Work

As far as our future work, we hope to extend the functionality of the MAIN Protocol to be suitable in the malicious setting. This will be more challenging as we need to account for arbitrary situations where any of the parties can deviate from the MAIN Protocol for any reason. The current MAIN Protocol is contingent on the assumption that everyone follows the rules, but attempts to learn information from the other parties. Our protocol is well-suited for the semi-honest setting, but would fail miserably if for example $P_j$ arbitrarily assigned each attribute a ridiculously high utility score. In this situation, $P_j$ would surely have winning attributes nearly every time which would both wreck the privacy and utility of the integrated dataset $\hat{D}$. Understanding the malicious framework would allow a proper execution of the MAIN Protocol in the most general setting.

# Bibliography


[1] Artak Amirbekyan and Vladimir Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *Proceedings of the sixth Australasian conference on Data mining and analytics-Volume 70*, pages 209–214. Australian Computer Society, Inc., 2007.

[2] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[3] Felix Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In *ICISC*, volume 3935, pages 32–47. Springer, 2005.

[4] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 486–497, 2007.

[5] Rui Chen, Noman Mohammed, Benjamin CM Fung, Bipin C Desai, and Li Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[6] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y Zhu. Tools for privacy preserving distributed data mining. *ACM Sigkdd Explorations Newsletter*, 4(2):28–34, 2002.

[7] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[8] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Eurocrypt*, volume 4004, pages 486–503. Springer.

[9] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[10] Benjamin CM Fung, Ke Wang, and S Yu Philip. Anonymizing classification data for privacy preservation. *IEEE transactions on knowledge and data engineering*, 19(5), 2007.

[11] Fosca Giannotti, Laks VS Lakshmanan, Anna Monreale, Dino Pedreschi, and Hui Wang. Privacy-preserving mining of association rules from outsourced transaction databases. *IEEE Systems Journal*, 7(3):385–395, 2013.

[12] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. *Advances in Cryptology—ASIACRYPT 2000*, pages 162–177, 2000.

[13] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking.

[14] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Secure multi-party differential privacy. In *Advances in neural information processing systems*, pages 2008–2016, 2015.

[15] 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.

[16] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Workload-aware anonymization techniques for large-scale datasets. *ACM Transactions on Database Systems (TODS)*, 33(3):17, 2008.

[17] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[18] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology—CRYPTO 2000*, pages 36–54. Springer, 2000.

[19] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Advances in Cryptology-EUROCRYPT 2007*, pages 52–78, 2007.

[20] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[21] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[22] Olvi L Mangasarian, Edward W Wild, and Glenn M Fung. Privacy-preserving classification of vertically partitioned data via random kernels.

[23] David J Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 126–135. IEEE, 2007.

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

[25] Noman Mohammed, Dima Alhadidi, Benjamin CM Fung, and Mourad Debbabi. Secure two-party differentially private data release for vertically partitioned data. *IEEE transactions on dependable and secure computing*, 11(1):59–71, 2014.

[26] 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*, pages 493–501. ACM, 2011.

[27] Noman Mohammed, Benjamin Fung, Patrick CK Hung, and Cheuk-Kwong Lee. Centralized and distributed anonymization for high-dimensional healthcare data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(4):18, 2010.

[28] Noman Mohammed, Benjamin C Fung, and Mourad Debbabi. Anonymity meets game theory: secure data integration with malicious participants. *The VLDB Journal—The International Journal on Very Large Data Bases*, 20(4):567–588, 2011.

[29] Martin Pettai and Peeter Laud. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 421–430. ACM, 2015.

[30] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[31] Vassilios S Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *ACM Sigmod Record*, 33(1):50–57, 2004.

[32] Omar Wahab, Omar Moulay, Hachami , Arslan Zaffar, Mery Vivas, and Gaby G Dagher. Darm: A privacy-preserving approach for distributed association rules mining on horizontally-partitioned data *, 07 2014.

[33] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. ($\alpha$, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing.

In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 754–759. ACM, 2006.

[34] Wai Kit Wong, David W Cheung, Edward Hung, Ben Kao, and Nikos Mamoulis. Security in outsourcing of association rule mining. In *Proceedings of the 33rd international conference on Very large data bases*, pages 111–122. VLDB Endowment, 2007.

[35] Hwanjo Yu, Xiaoqian Jiang, and Jaideep Vaidya. Privacy-preserving svm using nonlinear kernels on horizontally partitioned data. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 603–610. ACM, 2006.

[36] Feng Zhang, Chunming Rong, Gansen Zhao, Jinxia Wu, and Xiangning Wu. Privacy-preserving two-party distributed association rules mining on horizontally partitioned data. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 633–640. IEEE, 2013.

[37] Lidong Zhou, Michael A Marsh, Fred B Schneider, and Anna Redz. Distributed blinding for distributed elgamal re-encryption. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 824–824. IEEE, 2005.