Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data

(Full Version)

Wen-jie Lu
University of Tsukuba
riku@mdl.cs.tsukuba.ac.jp

Shohei Kawasaki
University of Tsukuba
kawasaki@mdl.cs.tsukuba.ac.jp

Jun Sakuma
University of Tsukuba, JST CREST, RIKEN AIP Center
jun@cs.tsukuba.ac.jp

Abstract

In recent years, there has been a growing trend towards outsourcing of computational tasks with the development of cloud services. The Gentry's pioneering work of fully homomorphic encryption (FHE) and successive works have opened a new vista for secure and practical cloud computing. In this paper, we consider performing statistical analysis on encrypted data. To improve the efficiency of the computations, we take advantage of the batched computation based on the Chinese-Remainder-Theorem. We propose two building blocks that work with FHE: a novel batch greater-than primitive, and matrix primitive for encrypted matrices. With these building blocks, we construct secure procedures and protocols for different types of statistics including the histogram (count), contingency table (with cell suppression) for categorical data; k-percentile for ordinal data; and principal component analysis and linear regression for numerical data. To demonstrate the effectiveness of our methods, we ran experiments in five real datasets. For instance, we can compute a contingency table with more than 50 cells from 4000 of data in just 5 minutes, and we can train a linear regression model with more than $40k$ of data and dimension as high as 6 within 15 minutes. We show that the FHE is not as slow as commonly believed and it becomes feasible to perform a broad range of statistical analysis on thousands of encrypted data.

I. Introduction

In recent years, considerable efforts have been made in the field of fully homomorphic encryption. Starting from Gentry's breakthrough work in constructing the first fully homomorphic encryption (FHE) scheme [7], successive innovations and improvements [3]–[6], [24], [27], [28] of fully homomorphic encryption have been proposed. At a high level, FHE enables us to perform addition and multiplication on ciphertexts. Thus it allows us to evaluate any function f on ciphertexts. We can decompose the input into bits and encrypt each bit separately. Since addition and multiplication on $\{0, 1\}$ are equivalent to the AND-gate and the XOR-gate in boolean circuits, we can construct the corresponding boolean circuit for the function f and evaluate the boolean circuit on ciphertexts. Such scheme has become widely recognized as a technology to enable processing of private data without compromising privacy.

Computational resources of cloud computing are completely virtualized, which helps to reduce the operational costs of service providers. However, such virtualization makes it difficult to keep control of data. In many domains; for instance, medical, and financial ones, confidentiality and privacy of data are one of the principal concerns raised in cloud-based applications. FHE schemes provide a natural method to address these concerns by encrypting data in the cloud and performing computations on ciphertexts without decrypting the data. Since FHE schemes theoretically allow evaluating any function on ciphertexts, FHE schemes might enable us to use the cloud for outsourcing computational tasks such as statistical analysis with a guarantee of data privacy.

Statistical analysis usually involves a large scale of data with a large number of dimensions. As a result, conducting statistical analysis in a way that evaluates the corresponding boolean circuits on FHE ciphertexts might be inefficient in practice, in terms of the memory usage and computational time. On the other hand, we can avoid encrypting the data bit-by-bit to obtain more efficient solutions. In [24], [31], and [20], particular encoding methods are used to obtain computationally and spatially efficient solutions on FHE ciphertexts. We remark that these encoding methods are specifically designed for a certain statistical analysis task. Thus it seems difficult to reuse these encoding methods for other tasks.

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
Copyright 2017 Internet Society, ISBN 1-1891562-46-0
http://dx.doi.org/10.14722/ndss.2017.23119

In this paper, we focus on applications of FHE to statistical analysis with three types of data. Our goal is to conduct a wide range of statistical analysis on FHE ciphertexts with computational and space efficiency. To achieve this goal, we need to have *somewhat generic* encodings for statistical analysis and fast computing routines on FHE ciphertexts. In this work, we present efficient procedures for a wide range of statistical analysis using just a few of generic data encodings. Specifically, we use two encodings to conduct descriptive and predictive statistics including the histogram (count, histogram order), contingency table with cell suppression, k-percentile, principal component analysis, and linear regression.

A. Related Works

The first fully homomorphic encryption (FHE) scheme is proposed in [7] while the efficiency of FHE is known as a big question following its invention. During the last few years, considerable effort has been devoted to improving the performances of FHE schemes [3]–[6], [24], [27], [28]. Moreover, packing techniques for example [28] and [24] to name a few, are used for accelerating the computation on ciphertexts and are applied to real applications. In [31] the authors present a specific string matching method with FHE; and in [20], the authors demonstrate a specific method for conducting a χ^2 test with FHE. These methods both leverage unique data encoding methods for particular problems. Thus these methods might be lacking in generality. The generic database query system using FHE [1] can support different aggregation queries.

Several studies that realize evaluating descriptive statistics using FHE have been reported. Evaluating the standard descriptive statistics such as the mean and standard deviation from FHE ciphertexts are presented in [24]. In [29] the authors also show how to compute the co-variance using FHE. Notice that these statistics involve numerical attributes only, while in the statistical analysis we can also have categorical and ordinal data. For categorical and ordinal data, we can implement procedures for statistics such as histogram and k-percentile using the private database query system of [1]. These implementations might require $O(N)$ multiplicative depths on ciphertexts where N is the number of data points, this might be impractical for a large scale of datasets.

For predictive statistics, the earlier study [12] presents the construction of *building* linear classifiers (i.e., the Linear Mean Classifier and Fisher's Linear Discriminant Classifier) from FHE encrypted data. More recently, the work of [2] shows three protocols for private *evaluation* of hyperplane decision classifiers, naive Bayes classifiers and decision tree classifiers on ciphertexts. Notice they focus on the *model evaluation* and the privacy-preserving model building is beyond the scope of [2]. In [29], the authors also present a protocol to obtain a linear regression model from FHE encrypted data using Cramer's rule or matrix inversion. The computational complexity of their method, thus, blows up factorially with the data dimension. In other words, their method is only suitable for data with small dimensions, i.e., less than 6. To the best of our knowledge, no practical FHE solution that trains the linear regression model from data with high dimension has been established.

B. Contribution

In this work, we show that we can evaluate a broad range of statistics for three different kinds of data on FHE ciphertexts using two encodings methods. The evaluation of many descriptive and predictive statistics commonly requires comparison operations and matrix operations. We, thus, propose two building blocks: a novel batch greater-than primitive and a layout consistent matrix primitive. We give concrete implementations using an open-sourced library, i.e., HElib [26]. Our contributions are summarized as follows.

Batch Greater-than Primitive. Comparing encrypted numbers is a common low-level primitive in many cryptographic protocols. In Section IV-B, we leverage the packing technique of [28] and present a batch variant of the greater-than protocol of [11]. Specifically, our bGT primitive requires $O(\lceil(\theta 2^d)/\ell\rceil)$ homomorphic operations to compare θ pairs of d-bit integers while the greater-than protocol of [11] needs $O(\theta 2^d)$ homomorphic operations. Thereby, a large ℓ can translate to a substantial improvement in efficiency.

Layout-Consistent Matrix Primitive. The current routine supports multiplication of encrypted matrices but the layout of the resulting matrix is inconsistent with that of the input [14]. Our matrix primitive, described in Section IV-A, allows one to conduct matrix additions and multiplications without changing the layout of encrypted matrices. We achieved this by arranging matrices in a row-wise manner and coupling the row-wise layout with a replication operation from HElib. Consequently, this layout consistency enables us to conduct iterative algorithms [13], [22] on encrypted matrices and contributes to our methods for predictive statistics.

To show that our building blocks are suitable for secure statistical analysis, in Section IV-C, we give experimental comparisons of our FHE-based bGT and matrix primitive with the garbled circuit (GC) [30] implementations using the state-of-the-art framework [19]. From the experimental results, we can see that our FHE-based primitives, in some common cases, are competitive with the GC counterparts.

Wide Range of Descriptive Statistics. We present practical procedures for conducting the k-percentile queries and contingency tables with cell suppression functionality in Section V-B. We can derive the secure versions of these statistics from the private database query system of [1]. However, it might be impractical to apply it to these descriptive statistics since [1] requires multiplicative depths of $O(N)$ to perform the comparison, where N is the number of the data. On the other hand, our procedures only require a constant multiplicative depth, which is of particular importance for the FHE scheme.

Our procedure for evaluating the contingency table also supports the cell suppression functionality which naturally requires comparisons. With the use of our bGT primitive, we show that we can achieve an efficient procedure for evaluating the contingency tables with cell suppression on ciphertexts without any interaction. Our review of the literature suggests that this report is the first approach to secure evaluation of contingency tables with cell suppression from FHE ciphertexts.

Protocols for Building Predictive Models. We describe procedures for principal component analysis (PCA) and linear regression in Section V-C. Our procedures apply iterative algorithms that involve matrix additions and multiplications only. Thereby, we can evaluate these algorithms on FHE ciphertexts straightforwardly. However, these iterative algorithms require a large message space whereas HElib only offers a limited size of message space. In Section V-C, we also propose the use of Plaintext Precision Expansion (PPE), which provides desired message space by compositing two or more ciphertexts with a limited message space. With our matrix primitives and PPE, we can evaluate the PCA and linear regression with data dimension up to 20 which is 4-times larger than that in [29]. Our review of the literature suggests that this report is the first practical approach to building a linear regression model with high dimensional data from FHE ciphertexts.

II. Preliminaries

We begin by introducing the notations used in this paper. We write $[d]$ to denote the set of positive integers $\{1, \dots, d\}$ and the cardinality of a set $\mathcal{D}$ are marked as $|\mathcal{D}|$. We write $x \overset{\$}{\leftarrow} \mathcal{D}$ to denote that x is sampled uniformly at random from $\mathcal{D}$. A matrix is shown as a bold uppercase roman letter, e.g., $\boldsymbol{A}$. We presume vector $\boldsymbol{v}$ forms a column vector following the convention of statistics. The row vector is represented by the transpose operation, e.g., $\boldsymbol{v}^\top$. Let $\boldsymbol{a}_i^\top$ denote the i-th row of the matrix $\boldsymbol{A}$: elements of a matrix are represented by non-bold lowercase roman letters with subscripts, e.g., a_{ij}. Matrix-vector multiplication and matrix multiplication are denoted as $\boldsymbol{X}\boldsymbol{a}$ and $\boldsymbol{X}\boldsymbol{Y}$, respectively. We denote the element-wise multiplication of vectors as $\boldsymbol{a} \dot{\times} \boldsymbol{b}$ where $(\boldsymbol{a} \dot{\times} \boldsymbol{b})_j = a_j b_j$ for all possible position j. We write $\boldsymbol{a} \ll k$ (resp. $\boldsymbol{a} \gg k$) to denote the left-rotation (resp. right-rotation) of the vector $\boldsymbol{a}$ with an offset k. We use $\mathbf{1}\{\mathcal{P}(x)\}$ to denote the indicator function for the predicate $\mathcal{P}(x)$, that is $\mathbf{1}\{\mathcal{P}(x)\} = 1$ if and only if $\mathcal{P}(x)$ is true, and 0 otherwise. We denote the encryption of a message x as $[\![x]\!]$.

A. Leveled Homomorphic Encryption

In this work, we specifically examine the Ring Learning with Error [21] variant of the Brakerski–Gentry–Vaikuntanathan (BGV) scheme, a leveled homomorphic encryption scheme proposed in [3].

The message space of the BGV's scheme works over a polynomial ring modulo a cyclotomic polynomial $\mathbb{A}_t := \mathbb{Z}_t[x]/\Phi_m(x)$, where $\Phi_m(x)$ is the m-th cyclotomic polynomial. Five algorithms specify the encryption scheme: $\mathsf{KeyGen}, \mathsf{Enc}_{\mathsf{pk}}, \mathsf{Dec}_{\mathsf{sk}}, \mathsf{Add}$ and Mul stand for key generation, encryption, decryption, addition, and multiplication respectively. We write pk and sk to denote the public and private keys, respectively. When the choice of the key is clear, we drop the pk and sk subscripts. KeyGen takes as input three positive integers m, t, and L; outputs a public-private key pair $(\mathsf{pk}, \mathsf{sk})$. Here m and t determine the message space; L indicates the multiplicative depth that the scheme can evaluate. According to the security analysis of [9], to achieve κ-bit security, parameters in the KeyGen should follow

$$\phi(m) > \frac{(L(\log \phi(m) + 23) - 8.5)(\kappa + 110)}{7.2}, \tag{1}$$

where $\phi(\cdot)$ is the Euler function. In a leveled homomorphic encryption, we have additive and multiplicative homomorphisms:

$$\begin{aligned} \mathsf{Dec}(\mathsf{Add}([\![a]\!], [\![b]\!])) &= a + b \mod (\Phi_m(x), t) \\ \mathsf{Dec}(\mathsf{Mul}([\![a]\!], [\![b]\!])) &= a \times b \mod (\Phi_m(x), t), \end{aligned}$$

where messages $a, b \in \mathbb{A}_t$. Also, the BGV's scheme supports scalar addition and multiplication, that is, given a ciphertext $\mathsf{Enc}(x)$ of $x \in \mathbb{A}_t$, we can have operations that output ciphertexts $\mathsf{Enc}(a + x)$ and $\mathsf{Enc}(ax)$ for all $a \in \mathbb{A}_t$. Due to space limitation, we omit the details of functions of the BGV's scheme here. We refer to [3] for more information about these functions.

Packing. One interesting and useful property of the BGV's scheme is that it enables us to pack multiple "messages" into one ciphertext allowing asymptotically efficient computation on encrypted data [28]. The cyclotomic polynomial $\Phi_m(x)$ factors into ℓ irreducible polynomials for some prime modulo t. That is $\Phi_m(x) = \prod_{j=1}^{\ell} F_j(x) \mod t$. We can pack an integer vector $\boldsymbol{u} \in \mathbb{Z}_t^\ell$ into an element $a \in \mathbb{A}_t$ by viewing each element of $\boldsymbol{u}$ as a polynomial (only with the constant term) and then applying the polynomial Chinese-Remainder-Theorem over factors $F_j(x)$. On the other hand, the unpacking function just takes the residual of the polynomial factors as $u_j = a \mod (F_j(x), t)$ for $1 \leq j \leq \ell$.

The efficiency gain of the CRT-packing comes from the *element-wise* addition and multiplication. We briefly demonstrate this property. Let $\mathsf{Pack} : \mathbb{Z}_t^\ell \mapsto \mathbb{A}_t$ be the packing function, and $\mathsf{Unpack} : \mathbb{A}_t \mapsto \mathbb{Z}_t^\ell$ be the unpacking function. Given vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_t^\ell$, the CRT-packing works as follows.

$$\begin{aligned} \mathsf{Unpack}(\mathsf{Pack}(\boldsymbol{x}) + \mathsf{Pack}(\boldsymbol{y})) &= \boldsymbol{x} + \boldsymbol{y} \mod t \\ \mathsf{Unpack}(\mathsf{Pack}(\boldsymbol{x}) \times \mathsf{Pack}(\boldsymbol{y})) &= \boldsymbol{x} \dot{\times} \boldsymbol{y} \mod t. \end{aligned}$$

If we use Pack to encode the integer vectors before applying the encryption function, we can perform ℓ homomorphic additions (resp. multiplications) by just a single application of Add (resp. Mul).

In addition to element-wise addition and multiplication, the CRT-packing also supports manipulations of encrypted vectors. Specifically, we can homomorphically *rotate* an encrypted vector and *replicate* one element of an encrypted vector. Similarly, let $\mathsf{Rotate} : \mathbb{A}_t \times \mathbb{Z} \mapsto \mathbb{A}_t$ be the rotation function and $\mathsf{Replicate} : \mathbb{A}_t \times \mathbb{Z} \mapsto \mathbb{A}_t$ be the replication function. These functions work over the CRT-packing as follows.

$$\mathsf{Unpack}(\mathsf{Rotate}(\mathsf{Pack}(\boldsymbol{x}), k)) = \boldsymbol{u} \in \mathbb{Z}_t^{\ell}$$
$$\mathsf{Unpack}(\mathsf{Replicate}(\mathsf{Pack}(\boldsymbol{x}), k)) = \boldsymbol{v} \in \mathbb{Z}_t^{\ell},$$

where we have $u_j = x_{j+k \mod \ell}$ (rotation) and $v_j = x_k$ (replication) for all $1 \leq j \leq \ell$. For rotation, we can have negative k while we require $1 \leq k \leq \ell$ for the replication. In this work, we take advantage of the CRT-packing and the vector manipulation operations to give efficient solutions of statistical analysis. For instance, we can perform matrix multiplications within a quadratic order of homomorphic operations. We present the matrix primitive in Section IV.

For the sake of simplicity, we write $[\![a]\!] + [\![b]\!]$ and $[\![a]\!] \cdot [\![b]\!]$ to denote $\mathsf{Add}([\![a]\!], [\![b]\!])$ and $\mathsf{Mul}([\![a]\!], [\![b]\!])$, respectively. We also use the rotation operators to indicate the invocation of the Rotate function, i.e., $[\![a]\!] \gg k$ or $[\![a]\!] \ll k$. When we apply the Pack function to vectors with length less than ℓ, we append zeros to the vectors. We usually assume enough spaces for packing vectors but we discuss one exception in Section IV, in which over-sized vectors are divided into smaller parts before applying the CRT-packing.

B. Data Representation

In this paper, we aim to conduct a broad range of statistics of numerical, ordinal, and categorical data. We firstly describe data representations for different types of attributes.

Categorical Attributes. The values of categorical attributes represent some *states* without meaningful order. Let d_c be the number of categorical attributes. We denote the domain of each categorical attribute as

$$\mathcal{C}_j = \{s_1^j, s_2^j, \cdots, s_{|\mathcal{C}_j|}^j\},\ 1 \leq j \leq d_c,$$

where s_k^j is the k-th state of the attribute $\mathcal{C}_j$. The cross-product gives the domain of the categorical attributes $\mathcal{C} := \mathcal{C}_1 \times \cdots \times \mathcal{C}_{d_c}$. Let $\boldsymbol{c}_i^\top \in \mathcal{C}$ be a vector of the categorical data. Then $c_{ij} \in \mathcal{C}_j$ is a categorical value of the j-th categorical attribute.

Ordinal Attributes. Values in an ordinal attribute have a meaningful ranking among them. We designate the number of ordinal attributes as d_o. Similarly, the domain of each ordinal attribute is represented as

$$\mathcal{O}_j = \{\hat{s}_1^j, \hat{s}_2^j, \cdots, \hat{s}_{|\mathcal{O}_j|}^j\},\ 1 \leq j \leq d_o,$$

where $\hat{s}_k^j$ is the k-th state of the attribute $\mathcal{O}_j$. The order of attribute values is given as $\hat{s}_1^j \preceq \cdots \preceq \hat{s}_{|\mathcal{O}_j|}^j$. We also present the domain of the ordinal attributes as the cross-product $\mathcal{O} := \mathcal{O}_1 \times \cdots \times \mathcal{O}_{d_o}$. Let $\boldsymbol{o}_i^\top \in \mathcal{O}$ be the i-th ordinal data. Then o_{ij} is an ordinal value of the j-th ordinal attribute.

Numerical Attributes. In this paper, we presume that all the numerical values are integers since the BGV's scheme can only process integers. We use a fixed point number of finite precision. Given $x \in \mathbb{R}$ and $M \in \mathbb{Z}$, we have $\lfloor Mx \rceil \in \mathbb{Z}$ where $\lfloor \cdot \rceil$ rounds a real number to the nearest integer. Let d_n be the dimension of numerical data and $\boldsymbol{x}_i^\top \in \mathbb{Z}_t^{d_n}$ be the i-th numerical data. The j-th element of each vector is designated as the j-th numerical attribute.

We represent the collections of N categorical, ordinal, and numerical data points respectively as follows.

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{c}_1^\top \\ \vdots \\ \boldsymbol{c}_N^\top \end{pmatrix} \in \mathcal{C}^N\ \boldsymbol{O} = \begin{pmatrix} \boldsymbol{o}_1^\top \\ \vdots \\ \boldsymbol{o}_N^\top \end{pmatrix} \in \mathcal{O}^N\ \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{pmatrix} \in \mathbb{Z}_t^{N \times d_n}$$

C. Data Encoding

The choice of value encoding method can affect the efficiency of function evaluation on ciphertexts dramatically. We introduce some encoding methods that are specifically to categorical and ordinal data.

Indicator Encoding $\mathcal{E}_{\text{id}} : \mathcal{C}_j \to \{0,1\}^{|\mathcal{C}_j|}$. $\mathcal{E}_{\text{id}}$ takes as input an attribute value $s_k^j \in \mathcal{C}_j$ and outputs a vector with all elements zero except the k-th element, which is set to 1. For instance, presuming $|\mathcal{C}_j| = 3$, the indicator encoding of the second state s_2^j will be $\mathcal{E}_{\text{id}}(s_2^j) = [0, 1, 0]$. We construct protocols of the histogram (count) and the contingency table using this encoding.

Staircase Encoding $\mathcal{E}_{\text{st}} : \mathcal{O}_j \to \{0,1\}^{|\mathcal{O}_j|}$. Staircase encoding takes as input an attribute value $\hat{s}_k^j \in \mathcal{O}_j$ and outputs a binary vector. The staircase encoding sets the 1-st to the $(k-1)$-th elements as 0 and sets the k-th to the last elements as 1. For

	s_1^q	$\cdots$	$s_{\lvert\mathcal{C}_q\rvert}^q$	Total
s_1^p	μ_{11}	$\cdots$	$\mu_{1\lvert\mathcal{C}_q\rvert}$	μ'_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$s_{\lvert\mathcal{C}_p\rvert}^p$	$\mu_{1\lvert\mathcal{C}_p\rvert}$	$\cdots$	$\mu_{\lvert\mathcal{C}_p\rvert\lvert\mathcal{C}_q\rvert}$	$\mu'_{\lvert\mathcal{C}_p\rvert}$
Total	μ_1	$\cdots$	$\mu_{\lvert\mathcal{C}_q\rvert}$	N

Fig. 1: A contingency table of two categorical attributes $\mathcal{C}_p$ and $\mathcal{C}_q$ of N data points.

example, presuming the domain size of $|\mathcal{O}_j| = 3$, the staircase encoding of the second state $\hat{s}_2^j$ will be $\mathcal{E}_{\text{st}}(\hat{s}_2^j) = [0, 1, 1]$. We use $\mathcal{E}_{\text{st}}$ for the evaluation of k-percentile.

To apply the CRT-packing to different types of attributes, we process the numerical data with constant magnification and convert the categorical and ordinal data using the indicator encoding and the staircase encoding. For instance, we first process a categorical value c_{ij} with the indicator encoding and then encrypt it as $[\![\mathsf{Pack}(\mathcal{E}_{\text{id}}(c_{ij}))]\!]$. Also, when $\boldsymbol{X}$ is a matrix, let $\mathsf{Pack}(\boldsymbol{X})$ be the vector formed by applying the operation to *each row* of $\boldsymbol{X}$ separately. That is $\mathsf{Pack}(\boldsymbol{X}) = [\mathsf{Pack}(\boldsymbol{x}_1^\top), \mathsf{Pack}(\boldsymbol{x}_2^\top), \ldots]$. In this paper, we represent the encryption of matrices and vectors with the CRT-packing by default. We write $[\![\boldsymbol{x}]\!]$ to denote the ciphertext of vectors instead of using $[\![\mathsf{Pack}(\boldsymbol{x})]\!]$ for the sake of simplicity. Similarly, $[\![\boldsymbol{X}]\!]$ denotes the ciphertext of the matrix $\boldsymbol{X}$.

III. Problem Statement

In this work, we consider statistical functions including the histogram (count and histogram order) and contingency table (with cell suppression) for categorical attributes; the k-percentile for ordinal attributes; and the principal component analysis and linear regression for numerical attributes. We present these statistics in turn.

A. Descriptive Statistics

Single Categorical Attribute. Let $\{c_{1j}, \ldots, c_{Nj}\}$ be the j-th categorical attribute values of N data points. If c_{ij}s are encoded by the indicator encoding, then the summation of vectors yields the histogram.

$$\mathsf{Hist}(\{c_{1j}, \ldots, c_{Nj}\}) = \boldsymbol{h} \text{ where } \boldsymbol{h} = \sum_{i=1}^{N} \mathcal{E}_{\text{id}}(c_{ij}). \tag{2}$$

The histogram query naturally gives the *count* and *histogram order*. The count of the state s_p^j can be given as

$$\mathsf{Count}(\{c_{1j}, \ldots, c_{Nj}\}, p) = \mathbf{1}_p^\top \boldsymbol{h}, \tag{3}$$

where $\mathbf{1}_p$ is an indicator vector of which the elements are 0 except the p-th element is 1.

The histogram order reveals the order of the counts of the histogram $\boldsymbol{h}$. We define this functionality as

$$\mathsf{HistOrder}(\{c_{1j}, \ldots, c_{Nj}\}) = \boldsymbol{k}, \tag{4}$$

where the count of the state $s_{k_x}^j$ is not less than the count of the state $s_{k_y}^j$ for any $1 \le x < y \le |\mathcal{C}_j|$.

Multiple Categorical Attributes. Next, we consider the evaluation of contingency tables of two categorical attributes $\mathcal{C}_p$ and $\mathcal{C}_q$. Evaluation of a contingency table corresponds to counting combinations (s_u^p, s_v^q) for all possible (u, v) pairs. We write μ_{uv} to denote the count of the combination (s_u^p, s_v^q). For instance, one categorical data point $\boldsymbol{c}_i = [\cdots, s_2^p, \cdots, s_3^q, \cdots]$ contributes to the count μ_{23} by 1. An example of the contingency table of attributes $\mathcal{C}_p$ and $\mathcal{C}_q$ is shown in Fig. 1. We define the functionality of contingency table evaluation as

$$\mathsf{ContingencyTable}(\{c_{ip}, c_{iq}\}_{i=1}^{N}) = \boldsymbol{\mu}. \tag{5}$$

In a contingency table, small counts represent rare individuals or cases of the population. For concerns of individual privacy, applications that evaluate contingency tables with private data collected from different sources usually additionally perform *cell suppression* [16], [23] to conceal existence of individuals with rare combination of attribute values. A common practice of the cell suppression is to zero-out the counts that are smaller than a constant threshold $\mathcal{T}$. The functionality of zero-out suppression can be defined as

$$\mathsf{CT\text{-}Suppression}(\{c_{ip}, c_{iq}\}_{i=1}^{N}, \mathcal{T}) = \bar{\boldsymbol{\mu}}, \tag{6}$$

where $\bar{\mu}_s = \mu_s \cdot \mathbf{1}\{\mu_s > \mathcal{T}\}$ for $1 \le s \le |\mathcal{C}_p||\mathcal{C}_q|$. Notice that $\boldsymbol{\mu}$ is the output of $\mathsf{ContingencyTable}$. We describe a novel method to compute $\mathsf{CT\text{-}Suppression}$ in Section V-B.

Ordinal Attributes. For the ordinal attributes, we consider k-percentile. k-percentile is the value that separates given ordinal values into two parts so that the one part with lower values contains k % of the data. For instance, the 50-percentile is also named as the median. Letting $\{o_{1j}, \dots, o_{Nj}\}$ be the j-th ordinal attribute values of N data points, we can sort the ordinal values in ascending order as $o_{\pi(1)j} \preceq \cdots \preceq o_{\pi(N)j}$. Here, π is a permutation function that returns indices in descending order. Using the notation of π, we can define the k-percentile functionality as

$$k\text{-Percentile}(o_{1j}, \dots, o_{Nj}) = o_{N^*j}, \tag{7}$$

where $N^* := \pi(\lceil (k \cdot N)/100 \rceil)$ and $o_{\pi(i)j} \preceq o_{\pi(i+1)j}$ holds for all $1 \le i < N$.

B. Predictive Statistics

Principal Component Analysis. PCA is a statistical procedure that converts a set of numerical observations of possibly correlated variables into a small number of directions that are mutually linearly independent. In PCA, we firstly compute a covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{\mu}^\top \text{ where } \boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i^\top. \tag{8}$$

Then we compute the eigenvalues and eigenvectors of $\boldsymbol{\Sigma}$. Let the eigenvalues of $\boldsymbol{\Sigma}$ be $\lambda_1 \ge \cdots \ge \lambda_{d_n}$, and denote the corresponding eigenvectors as $\boldsymbol{u}_1, \dots, \boldsymbol{u}_{d_n}$. An iterative algorithm (i.e., PowerMethod) can evaluate the k-th eigenvalue λ_k and the corresponding principal component $\boldsymbol{u}_k$ with T iterations.

```
PowerMethod ($\boldsymbol{\Sigma}, \{\lambda_q\}_{q=1}^{k-1}, \{\boldsymbol{u}_q\}_{q=1}^{k-1}$):
1. $\boldsymbol{\Sigma}^k := \boldsymbol{\Sigma} - \sum_{q=1}^{k-1} \lambda_q \boldsymbol{u}_q \boldsymbol{u}_q^\top$.
2. Choose a random vector $\boldsymbol{v}^{(0)} \xleftarrow{\$} \mathbb{Z}_t^{d_n}$.
3. For $0 \le \tau < T$, compute
       $\boldsymbol{v}^{(\tau+1)} = \boldsymbol{\Sigma}_k \boldsymbol{v}^{(\tau)}$.     (9)
4. Output $\boldsymbol{u}_k = \frac{\boldsymbol{v}^{(T)}}{\|\boldsymbol{v}^{(T)}\|}$ and $\lambda_k = \frac{\|\boldsymbol{v}^{(T)}\|}{\|\boldsymbol{v}^{(T-1)}\|}$.
```

Linear Regression. The problem of linear regression is to find a model that predicts values of a numerical target variable from observations of numerical input variables using a linear equation. Let $\{(\boldsymbol{x}_i^\top, y_i)\}_{i=1}^N$ be the given dataset in which $\boldsymbol{x}_i^\top$ are the input variables and y_i is the target variables. The model of linear regression is given as $y \approx \boldsymbol{x}^\top \boldsymbol{w}$. Therein, the model $\boldsymbol{w}$ is obtained by minimizing the least-squares error:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^{N} \|y_i - \boldsymbol{x}_i^\top \boldsymbol{w}\|_2^2.$$

The analytical solution $\boldsymbol{w}^*$ is given as

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{10}$$

where the matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$ are the collections of numerical data. The Eq. 10 is immediately solved if we can evaluate the inverse of $\boldsymbol{X}^\top \boldsymbol{X}$. We leverage a division-free variant of the iterative matrix inversion method from [13] so that we can compute the matrix inversion on FHE encrypted matrices. Let $\boldsymbol{M}$ be a matrix, λ be a real value, and T be the number of iterations. The division-free matrix inversion method works as follows.

```
DF-MatrixInversion ($\boldsymbol{M}, \lambda, T$):
1. Initialize $\boldsymbol{A}^{(0)} = \boldsymbol{M}, \boldsymbol{R}^{(0)} = \boldsymbol{I}, \alpha^{(0)} = \lambda$.
2. For $0 \le \tau < T$, compute
       $\boldsymbol{R}^{(\tau+1)} = 2\alpha^{(\tau)} \boldsymbol{R}^{(\tau)} - \boldsymbol{R}^{(\tau)} \boldsymbol{A}^{(\tau)}$,
       $\boldsymbol{A}^{(\tau+1)} = 2\alpha^{(\tau)} \boldsymbol{A}^{(\tau)} - \boldsymbol{A}^{(\tau)} \boldsymbol{A}^{(\tau)}$,     (11)
       $\alpha^{(\tau+1)} = \alpha^{(\tau)} \alpha^{(\tau)}$.
3. Output $\boldsymbol{R}^{(T)}$.
```

Here $\boldsymbol{I}$ is an identity matrix. This method *approximates* the inverse of the matrix $\boldsymbol{M}$. According to the analysis of [13], $\boldsymbol{R}^{(\tau)}$ converges to $\lambda^{2^\tau} \boldsymbol{M}^{-1}$ quadratically if λ is close to the largest eigenvalue of $\boldsymbol{M}$.

TABLE I: Complexity of our primitives. We write "–" to indicate that the homomorphic operation is not used.

	addition	multiplication	rotation
$[\![\boldsymbol{X}]\!] \cdot [\![\boldsymbol{u}]\!]$	$O(d)$	$O(d)$	$O(d \log d)$
$[\![\boldsymbol{X}]\!] + [\![\boldsymbol{Y}]\!]$	$O(d)$	–	–
$[\![\boldsymbol{X}]\!] \cdot [\![\boldsymbol{Y}]\!]$	$O(d^2)$	$O(d^2)$	$O(d^2 \log d)$
bGT	$O(\lceil(\theta D)/\ell\rceil)$	$O(\lceil(\theta D)/\ell\rceil)$	$O(\log D)$

IV. Building Blocks

In the previous section, we have described the descriptive and predictive statistics that we are going to evaluate. We can see that the evaluations of these statistics require operations including matrix addition, matrix multiplication, and comparison operation. In this section, we present two building blocks for matrix operations and comparison on encrypted values. We give the summary of complexities of our primitives in Table I.

A. Matrix Operations

For our statistical analysis, we process every data in the form of matrices and vectors. Once matrices are encoded and encrypted, it requires expensive homomorphic operations to rearrange the layout of these values. For instance, it requires many homomorphic operations to change a row-wise encrypted matrix to a column-wise encrypted counterpart. To achieve a low computation overhead, it is important for us to keep the layout consistent throughout each matrix operation. We introduce layout-consistent matrix operations for FHE encrypted matrices.

Halevi et al. [14] introduced three layouts to represent matrix as a single ciphertext: the row-major order, the column-major order, and the diagonal-major order. In this work, we consider the row-major order in which rows of the matrix are encrypted separately. It is natural to apply this layout in real applications. For instance, some research agents might independently hold data with a different size but following the same data schema. Recall that we apply the CRT-packing to each row of matrices and then encrypt each row. Thereby, we write $\{[\![\boldsymbol{x}_i^\top]\!]\}_{i=1}^d$ and $\{[\![\boldsymbol{y}_i^\top]\!]\}_{i=1}^d$ to denote the ciphertexts of each row of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. The ciphertext of a vector $\boldsymbol{u} \in \mathbb{Z}_t^d$ is $[\![\boldsymbol{u}]\!]$.

Matrix–vector Multiplication. Halevi et al. [14] introduced a *general* procedure for the matrix–vector multiplication. For the row-major layout, their procedure requires to "sum up" all the slots of the CRT-packing, which might be expensive than the replication operation regarding computational time. However, we give a different routine according to the observation that we only involve *symmetric matrices* in the matrix–vector multiplication (i.e., PCA). We thus can conduct the matrix–vector multiplication as $[\![\boldsymbol{X}\boldsymbol{u}]\!] = \sum_{i=1}^d [\![\boldsymbol{x}_i^\top]\!] \cdot \mathsf{Replicate}([\![\boldsymbol{u}]\!], i)$. The idea of this equation follows that the matrix $\boldsymbol{X}$ being *symmetric*, thus, having the i-th row equals to the i-th column.

Matrix Addition & Multiplication. We can simply conduct the layout-consistent matrix addition as $[\![\boldsymbol{x}_i^\top]\!] + [\![\boldsymbol{y}_i^\top]\!]$ for $1 \le i \le d$ while we need more delicate operations to achieve the matrix multiplication without destroying the row-major layout.

We hope to conduct $\boldsymbol{XY}$ so that we can evaluate the inverse matrix (i.e., Eq. 11) on encrypted matrices. To keep the layout consistent, we use the $\mathsf{Replicate}$ function. We conduct the matrix multiplication on encrypted matrices as

$$\sum_{i=1}^{d} \mathsf{Replicate}([\![\boldsymbol{x}_j^\top]\!], i) \cdot [\![\boldsymbol{y}_i^\top]\!] \text{ for } 1 \le j \le d.$$

We give an example to demonstrate this routine as follows.

$$\underbrace{\begin{bmatrix} [1 & 2] \\ [3 & 4] \end{bmatrix}}_{\boldsymbol{X}} \cdot \underbrace{\begin{bmatrix} [e & f] \\ [g & h] \end{bmatrix}}_{\boldsymbol{Y}} = \begin{bmatrix} \overbrace{1 \cdot [e \quad f] + 2 \cdot [g \quad h]}^{j=1} \\ \underbrace{3 \cdot [e \quad f] + 4 \cdot [g \quad h]}_{j=2} \end{bmatrix}.$$

Also, we can compute $[\![\boldsymbol{u}\boldsymbol{u}^\top]\!]$ on the ciphertext $[\![\boldsymbol{u}]\!]$ in a similar manner. Specifically, the ciphertext of the k-th row of the matrix $\boldsymbol{u}\boldsymbol{u}^\top$ is given as $\mathsf{Replicate}([\![\boldsymbol{u}]\!], k) \cdot [\![\boldsymbol{u}]\!]$. We write $[\![\boldsymbol{u}]\!] \cdot [\![\boldsymbol{u}^\top]\!]$ to denote this operation.

B. Batch Greater-than Primitive

For conducting statistics such as contingency tables, histogram order, and k-percentile, we need comparison operations. To this end, we introduce a novel batch greater-than (bGT) primitive.

Given integers $a, b \in [D]$ for some positive D, we know that $a > b$ if and only if $\exists w \in [D]$ such that $a - b - w = 0$. Thereby, we can construct a straw-man protocol by homomorphically computing $(a - b - w) \cdot r$ for all $w \in [D]$ where the random value r is used to hide $|a - b|$. This straw-man protocol, thus, requires $O(D)$ homomorphic operations and generates $O(D)$ ciphertexts. The idea behind the straw-man protocol follows the greater-than protocol of [11].

Algorithm 1 Batch greater-than primitive.

```
- Input: $[\![\boldsymbol{a}]\!]$, and $[\![\boldsymbol{b}]\!]$, where $\boldsymbol{a}, \boldsymbol{b} \in [D]^\theta$ for $D, \theta \in \mathbb{Z}^+$.
- Output: $[\![\boldsymbol{\gamma}]\!]$ where the length of $\boldsymbol{\gamma}$ is $\theta D$.
- Remark: One can learn $\mathbf{1}\{a_j > b_j\} = \mathbf{1}\{0 \in \{\gamma_{k \cdot \theta + j}\}_{k=0}^{D-1}\}$
1: Compute $[\![\tilde{\boldsymbol{a}}]\!] = \mathsf{Repeat}([\![\boldsymbol{a}]\!], \theta, D)$; $[\![\tilde{\boldsymbol{b}}]\!] = \mathsf{Repeat}([\![\boldsymbol{b}]\!], \theta, D)$.
2: Generate random permutations $\pi_j : [D] \to [D]$ for $0 \le j < \theta$.
3: Compute a $\theta \cdot D$ dimension vector $\boldsymbol{w}$ in which $w_{\alpha(j)} = \pi_j(\alpha)$. Here $\alpha(j) := \theta \cdot \alpha + j$, for $\alpha \in [D]$ and $0 \le j < \theta$.
4: Compute $[\![\boldsymbol{\beta}]\!] = [\![\tilde{\boldsymbol{a}}]\!] - [\![\tilde{\boldsymbol{b}}]\!] - \mathsf{Pack}(\boldsymbol{w})$.
5: Compute $[\![\boldsymbol{\gamma}]\!] = [\![\boldsymbol{\beta}]\!] \cdot \mathsf{Pack}(\boldsymbol{r})$ where $\boldsymbol{r} \xleftarrow{\$} (\mathbb{Z}_t/\{0\})^{\theta \cdot D}$.
6: Output $[\![\boldsymbol{\gamma}]\!]$.
```

We can reduce the computational cost and the number of ciphertexts of the straw-man protocol by using the CRT-packing. Recall that the CRT-packing enables us to pack ℓ integers into one ciphertext and the homomorphic addition and multiplication are then carried out on these ℓ integers simultaneously. Thereby, we can compute $(a - b - w) \cdot r$ with ℓ different w by viewing these w as a vector $\boldsymbol{w}$ and using the Pack function. Moreover, we need to shuffle the positions of each w before packing them since $|a-b|$ will be revealed if the position of w is predictable. This greater-than method, thus, requires $O(\lceil D/\ell \rceil)$ homomorphic operations and generates $O(\lceil D/\ell \rceil)$ ciphertexts which is a considerable improvement for a large ℓ.

Indeed, we can give a generalized batch greater-than method which takes as input $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$ where $\boldsymbol{a}, \boldsymbol{b} \in [D]^\theta$ and outputs $\mathbf{1}\{a_j > b_j\}$ for all $1 \le j \le \theta$. The method described above is a specialization of this with $\theta = 1$. The bGT protocol is shown in Alg. 1. The Repeat function (Step 1) takes as input $[\![\boldsymbol{u}]\!]$, θ, and R. Repeat duplicates the first θ elements of $\boldsymbol{u}$ for R times. For instance $\mathsf{Repeat}([\![\boldsymbol{u}]\!], \theta = 3, R = 2) = [\![[u_1 u_2 u_3 u_1 u_2 u_3]]\!]$.

```
$\mathsf{Repeat}([\![\boldsymbol{u}]\!], \theta, R)$:
1. $[\![\tilde{\boldsymbol{u}}]\!] = [\![\boldsymbol{u}]\!] \cdot \mathsf{Pack}([\underbrace{1 \ldots 1}_{\theta} 00 \ldots])$.
2. $R = (b_\rho \cdots b_1 b_0)_2$ where $b_\rho$ is the most significant bit.
3. For $0 \le i \le \rho$
    a) If $b_i$ is 1 then $[\![\tilde{\boldsymbol{u}}]\!] = [\![\tilde{\boldsymbol{u}}]\!] \gg k$; $[\![\tilde{\boldsymbol{u}}]\!] = [\![\tilde{\boldsymbol{u}}]\!] + [\![\boldsymbol{u}]\!]$
    b) $[\![\boldsymbol{u}]\!] = [\![\boldsymbol{u}]\!] + ([\![\boldsymbol{u}]\!] \gg k)$
    c) $k = k \times 2$
4. return $[\![\tilde{\boldsymbol{u}}]\!]$
```

The Repeat procedure requires $O(\log_2 R)$ homomorphic additions and rotations.

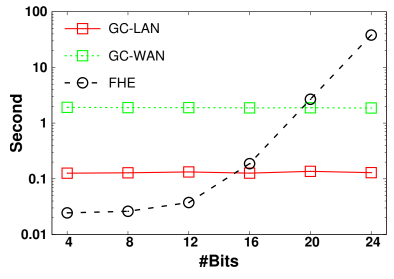
We operate multiple comparisons in a batch manner. Thus we need to avoid collisions of w in different comparisons (Step 3). Moreover, we might do not have enough spaces, i.e., $\ell < \theta D$ for packing. In this case, we can extend the spaces with multiple ciphertexts. The bGT protocol performs comparisons of θ pairs of integers and requires $O(\lceil (\theta D)/\ell \rceil)$ homomorphic operations and generates $O(\lceil (\theta D)/\ell \rceil)$ ciphertexts. In this work, we usually use $\theta = 1$ while we use $\theta > 1$ in the evaluation of the contingency table and k-percentile. We usually use the bGT only in the last step of a larger protocol since we need to decrypt the output of bGT to obtain the comparison result. However, exceptions do exist when we can take advantage of the randomness of the output of bGT. For instance, in this work, we use the bGT as an intermediate step to evaluate Eq. 6. Precisely, at Line 5 – Line 7 of the $\mathsf{PCT\text{-}Suppression}$ protocol in Section V-B, the output of bGT is used to mask the suppressed counts with random values.

C. Comparison with the Garbled Circuit

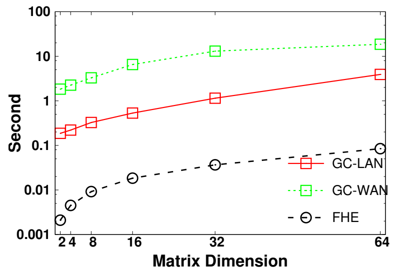
We experimentally compared our proposed primitives with the garbled circuit implementations (Fig. 2).

GC Setting. For GC, we used a state-of-the-art framework, ObliVM [19] which allows us to implement the garbled circuit with a high-level programming language interface. We used two physically separated machines as the circuit generator and the circuit evaluator. The generator and evaluator held random shares of the private inputs. We ran the GC experiments on two network settings: a Local Area Network (two machines located inside the same router) and a Wide Area Network (one machine located in Japan and the other located on the west coast of USA). The network bandwidth of LAN and WAN was about 88 Mbps and 48 Mbps, respectively. In ObliVM, we used the real-mode which provides the garbled-row-reduction [25] and free-XOR [17] optimizations.

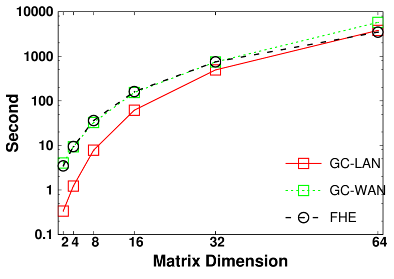
FHE Setting. In the executions of the FHE primitives, we assume an encryptor encrypts the private inputs and uploads the ciphertexts to the server. The server operates the primitives on the ciphertexts and obtains the result. A decryptor downloads the result from the server and gets the plain result after the decryption. For performance measurement, we used the same network (LAN and WAN) as GC. For FHE-based primitives, we implemented using eight parallels. We also used different parameters in



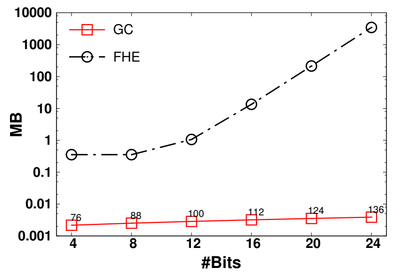
(a) Evaluation Time of GT



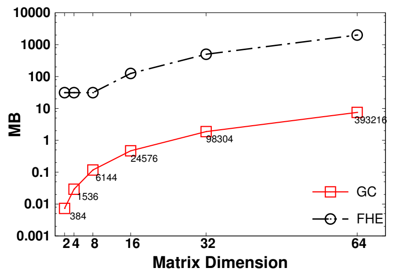
(b) Evaluation Time of $\boldsymbol{X} + \boldsymbol{Y}$



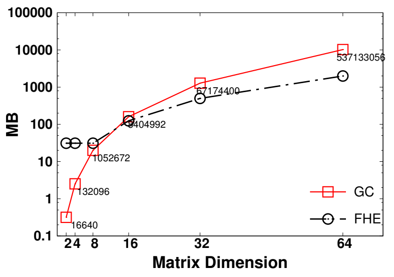
(c) Evaluation Time of $\boldsymbol{XY}$



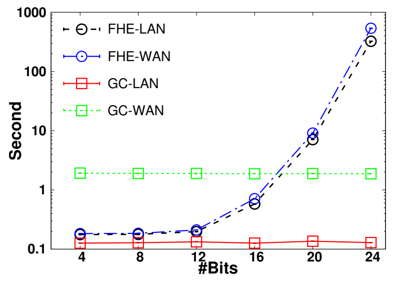
(d) Ciphertext Size of GT



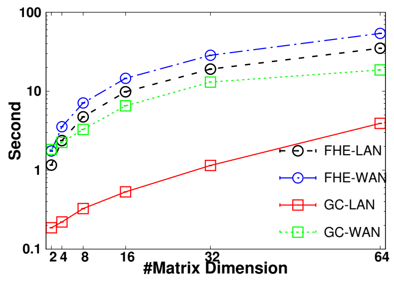
(e) Ciphertext Size of $\boldsymbol{X} + \boldsymbol{Y}$



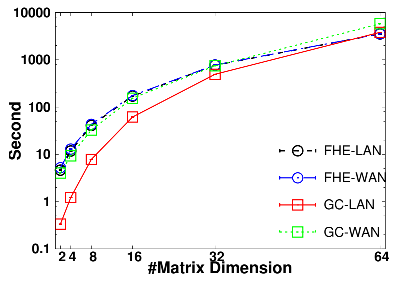
(f) Ciphertext Size of $\boldsymbol{XY}$



(g) Operation Time of GT



(h) Operation Time of $\boldsymbol{X} + \boldsymbol{Y}$



(i) Operation Time of $\boldsymbol{XY}$

Fig. 2: Performance numbers (averaged over 10 runs) of FHE-based and GC-based primitive implementations. LAN and WAN were introduced. For the matrix addition and matrix multiplication, matrices with 32-bits values were used. The numbers on the figure (g) – figure (i) indicate the number of AND-gates in the garbled circuits.

bGT and the matrix primitives. Specifically, we set the parameters of the BGV's scheme $t = 67499$ and $\Phi_m(x)$ with $m = 5227$ (i.e., $\ell = 1742$) for evaluating the batch greater-than primitive. On the other hand, we use $t = 7321^3$ and $m = 27893$ (i.e., $\ell = 78$) for evaluating the matrix primitives.

Performance Measurements. We employed three different performance measurements: *evaluation time*, *ciphertext size*, and *operation time*. The operation time of our FHE-based primitives includes the time of *encryption, upload, evaluation, download*, and *decryption*. The evaluation time includes the time of evaluation only, which is independent of the network bandwidth. For the GC implementations, we measured the time for circuit generation and the time for circuit evaluation.

When we use the FHE primitives as an independent two-party computation, the entire computation time is measured by the operation time. On the other hand, when the FHE primitives are used as building blocks for a more complicated two-party computation, the outputs of the FHE primitives are successively reused without interaction with the other party. In such reuses, encryption, upload, and download are not processed, and thus the server does not need to communicate with encryptors and decryptors. Thus, to measure the efficiency of our FHE primitives, we measured the evaluation time, too. We note that we can not separately evaluate the evaluation time from operation time for GC execution. Therefore, the evaluation time of GC is the same as the operation time in our evaluation.

We also compared the size of ciphertexts that the FHE-based primitives output with the size of network packets exchanged during the execution of the GC-based primitives [1].

Greater-than. Fig. 2a, Fig. 2d, and Fig. 2g show the performances of the FHE-based and GC-based greater-than implementations. As shown in the results, our FHE-based greater-than primitive offers competitive performances to its GC counterpart when

[1]We counted the number of AND-gates (20 bytes each) in the circuit.

TABLE II: Input-output relationships for the stakeholders. We write "–" to indicate no input or output.

Stakeholder	Possess	Input	Output
encryptor	pk	x	–
cloud	pk	–	$[\![z]\!]$
decryptor	pk, sk	–	z

comparing relatively small integers such as integers with less than 16 bits. The complexity of the FHE-based greater-than grows exponentially with the bit length. Thus, it seems inefficient for our greater-than primitive to handle large numbers. Noting that descriptive statistics of ordinal or categorical attributes typically assumes small domains (e.g., $0 \leq \mathit{age} \leq 150$), we consider $12 \sim 16$-bits to be sufficient to meet regular requirements in many cases.

Matrix Addition. Fig. 2b, Fig. 2e, and Fig. 2h show the performances of the FHE-based and GC-based implementations of matrix addition. Since we leverage the CRT-packing for FHE encrypted matrices, the evaluation time of the FHE-based matrix addition increases linearly with the matrix dimension. The FHE-based matrix addition can operate faster than its GC counterpart in terms of evaluation time while the size of ciphertexts generated by the FHE-based matrix addition was two magnitudes larger than that in the GC counterpart. The operation time of the FHE-based matrix addition is thus greater than that of its GC counterpart. We can also see that the evaluation time of the FHE-based matrix addition was smaller than the operation time of the GC (Fig. 2b). In the WAN setting, the operation times of these two implementations were quite close. We emphasize that the performance of the GC-based matrix addition and that of the FHE-based matrix addition are not directly comparable. If the matrix addition itself is the target computation, the GC-based solution works faster. However, when we need successive matrix additions in the middle of a larger computation, the FHE-based implementation can provide competitive performance with its GC counterpart.

Matrix Multiplication. Fig. 2c, Fig. 2f, and Fig. 2i show the performances of FHE-based and GC-based implementations of matrix multiplication. The GC implementation ran slightly faster than the FHE-based one in the LAN environment while in the WAN environment, these two implementations performed almost the same regarding evaluation time. Notice that the number of ciphertexts in the FHE-based matrix multiplication and that of the FHE-based matrix addition were the same due to the layout-consistency of our matrix primitives. On the other hand, the GC-based matrix multiplication exchanged more network packets than that of the GC-based matrix addition. We can see that the evaluation time and operation time of the FHE-based matrix multiplication were almost the same, indicating the time of network communication in FHE-based matrix multiplication is negligible. When we need to operate iterative matrix multiplications, the FHE-based primitive, which requires less network communication time, can offer better performance in terms of operation time.

From the experimental results, we can conclude that our two building blocks are viable for cloud-based applications. We admit that our greater-than primitive might be inefficient for comparing large numbers, but for many statistics, small domains such as sizes of several thousand might be sufficient. Also, we have to transfer hundreds of megabytes of ciphertexts which seems to hinder the performance of our FHE-based matrix primitives. But we are interested in the statistical analysis rather than a single matrix addition or multiplication. As Eq. 11 shows, we need to perform matrix operations iteratively. For the FHE-based matrix primitives, the number of generated ciphertexts is independent of the number of iterations. Thus, after the cloud finishes the analysis, the cost of transferring the FHE ciphertexts might not be the bottleneck. However, the network packets exchanged by the GC-based implementations increases linearly with the number of iterations. In other words, for evaluating complex functions, e.g. functions with a large multiplicative depth or functions with large fan-in, the communication time might become the bottleneck of GC solutions. Moreover, FHE-based solutions enable to delegate the computation to the cloud, and allow the encryptor to perform encryption only.

V. Computing Statistics on Ciphertexts

This section presents the details of evaluating the statistics described in Section III on FHE encrypted data.

A. Security Model

We give an overview of our desired security properties. We consider three stakeholders: encryptor, cloud, and decryptor. We assume all stakeholders behave semi-honestly and the cloud does not collude with the decryptor. Let x be a private input of the encryptor and f be a publicly known function. We consider the following model (Table II). The encryptor sends the ciphertext $[\![x]\!]$ to the cloud for the computation of a particular function f. The cloud operates specified homomorphic operations on $[\![x]\!]$ and sends the resulting ciphertext $[\![z]\!]$ to the decryptor. The decryptor decrypts the resulting ciphertext and learns z but nothing else. The cloud and the encryptor learn nothing at the end of the execution of the protocol. The encryptor sends the encryption of his private input following the data processing of different types of data in Table III. In the following protocol descriptions, we thus omit the encryption phase of the encryptor.

TABLE III: A summary of the form of ciphertexts and statistics

Data Type	Ciphertext Form	Statistics
$c_{iq} \in \mathcal{C}_q$	$[\![\mathsf{Pack}(\mathcal{E}_{\text{id}}(c_{iq}))]\!]$	histogram, count, histogram order and contingency table
$o_{ip} \in \mathcal{O}_p$	$[\![\mathsf{Pack}(\mathcal{E}_{\text{st}}(o_{ip}))]\!]$	k-percentile
$\boldsymbol{x}_i \in \mathbb{Z}^{d_c}$	$[\![\mathsf{Pack}(\boldsymbol{x}_i)]\!]$	PCA and linear regression

B. Descriptive Statistics

Histogram and Count. The evaluations of Eq. 2 (histogram) and Eq. 3 (count) on FHE encrypted categorical data are straightforward using the CRT-packing and indicator encoding. For the collection of categorical data $\boldsymbol{C} \in \mathcal{C}^N$, we can compute the histogram of $\mathcal{C}_p$, i.e. the p-th attribute, as $\sum_{i=1}^N [\![\mathcal{E}_{\text{id}}(c_{ip})]\!]$. Also, we can compute the histograms of multiple attributes simultaneously. For instance $\sum_{i=1}^N [\![\mathcal{E}_{\text{id}}(c_{ip}) \| \mathcal{E}_{\text{id}}(c_{iq})]\!]$ gives the histograms of $\mathcal{C}_p$ and $\mathcal{C}_q$. Moreover, to give the count of specific attribute values, we need one more homomorphic multiplication. For example, $\left(\sum_{i=1}^N [\![\mathcal{E}_{\text{id}}(c_{ip})]\!]\right) \cdot \mathsf{Pack}(\mathbf{1}_3)$ gives the ciphertext of the count for s_3^p, i.e., the third state of the attribute of $\mathcal{C}_p$. Similarly, we can give multiple counts simultaneously.

Histogram Order. The evaluation of Eq. 4 requires computing the order of the counts in the histogram, which indicates that comparisons of encrypted integers are needed. Our method for calculating the histogram order on ciphertexts splits into two stages: one for operating bGT and the other for recovering the histogram order from the outputs of bGT. In the second stage, we need to decrypt the outputs of bGT.

$\mathsf{PrivateHistOrder}$ $(\{[\![\mathcal{E}_{\text{id}}(c_{ij})]\!]\}_{i=1}^N)$:

The cloud:

1. Computes the histogram $[\![\boldsymbol{h}]\!] = \sum_{i=1}^N [\![\mathcal{E}_{\text{id}}(c_{ij})]\!]$.
2. Computes $[\![h_p]\!] = \mathsf{Replicate}([\![\boldsymbol{h}]\!], p)$ for $1 \le p \le |\mathcal{C}_j|$.
3. For all $1 \le u < v \le |\mathcal{C}_j|$ pairs, invokes the Alg. 1 with $D = N$ and $\theta = 1$
$$[\![\boldsymbol{\gamma}_{uv}]\!] = \mathsf{bGT}([\![h_u]\!], [\![h_v]\!]).$$
4. Outputs ciphertexts $\{[\![\boldsymbol{\gamma}_{uv}]\!]\}_{1 \le u < v \le |\mathcal{C}_j|}$.

The decryptor:

5. Constructs a matrix $\boldsymbol{\Delta} \in \{0,1\}^{|\mathcal{C}_j| \times |\mathcal{C}_j|}$ according to the *decryption* of $\{[\![\boldsymbol{\gamma}_{uv}]\!]\}_{1 \le u < v \le |\mathcal{C}_j|}$.
   a) The diagonal of $\boldsymbol{\Delta}$ is set to 0, that is $\delta_{uu} = 0$.
   b) For all (u, v) pairs such that $1 \le u < v \le |\mathcal{C}_j|$, set $\delta_{uv} = \mathbf{1}\{0 \in \boldsymbol{\gamma}_{uv}\}$ and set $\delta_{vu} = 1 - \delta_{uv}$.
6. Outputs a vector $\boldsymbol{k}$ with the value k_l set as the row-index of $\boldsymbol{\Delta}$ which contains exactly $|\mathcal{C}_j| - l$ of 1s for $1 \le l \le |\mathcal{C}_j|$.

$\mathsf{PrivateHistOrder}$ calls the bGT primitive $O(|\mathcal{C}_j|^2)$ times. By operating these comparisons, we have obtained the order of the values of the histogram. According to bGT, if $0 \in \boldsymbol{\gamma}_{uv}$ holds then we know that the count of state s_u^j is larger than that of state s_v^j. In the protocol, the matrix $\boldsymbol{\Delta}$ just acts as a handy helper for us to calculate the histogram order.

Contingency Table. We first present a novel method to evaluate the contingency table (Eq. 5) from ciphertexts and then describe how to achieve the zero-out suppression (Eq. 6).

$\mathsf{PrivateContingencyTable}$ $(\{[\![\mathcal{E}_{\text{id}}(c_{ip})]\!], [\![\mathcal{E}_{\text{id}}(c_{iq})]\!]\}_{i=1}^N)$:

The cloud:

1. Finds the smallest co-prime integers k_1 and k_2 such that $k_1 \ge |\mathcal{C}_p|$ and $k_2 \ge |\mathcal{C}_q|$.
2. For $1 \le i \le N$, computes
$$[\![\boldsymbol{p}_i]\!] = \mathsf{Repeat}([\![\mathcal{E}_{\text{id}}(c_{ip})]\!], k_1, k_2)$$
$$[\![\boldsymbol{q}_i]\!] = \mathsf{Repeat}([\![\mathcal{E}_{\text{id}}(c_{iq})]\!], k_2, k_1).$$
3. Computes and outputs $[\![\boldsymbol{\mu}]\!] = \sum_{i=1}^N [\![\boldsymbol{p}_i]\!] \cdot [\![\boldsymbol{q}_i]\!]$.

The decryptor obtains the contingency table of $\mathcal{C}_p$ and $\mathcal{C}_q$ from the vector $\boldsymbol{\mu}$. Specifically, the count μ_{uv} in the contingency table is given by the x-th element of $\boldsymbol{\mu}$ where $(x-1) \equiv (u-1) \mod k_1$ and $(x-1) \equiv (v-1) \mod k_2$[2]. We present a concrete example in Fig. 3, in which the domain sizes are $|\mathcal{C}_p| = |\mathcal{C}_q| = 2$ and $k_1 = 2, k_2 = 3$. In Fig. 3, the white cells

[2]Indices start from 1.

	$\mathcal{E}_{\text{id}}(c_{ip})$		$\mathcal{E}_{\text{id}}(c_{ip})$		$\mathcal{E}_{\text{id}}(c_{ip})$	
element-wise multi.	$\mathcal{E}_{\text{id}}(c_{iq})$			$\mathcal{E}_{\text{id}}(c_{iq})$		
contribute to	μ_{11}	μ_{22}	–	μ_{21}	μ_{12}	–

Fig. 3: One multiplication gives 2×2 combinations of attributes of $c_{ip} \in \mathcal{C}_p$ and $c_{iq} \in \mathcal{C}_q$ where $|\mathcal{C}_p| = 2$ and $|\mathcal{C}_q| = 2$.

indicate 0 since we use 0-padding in the CRT-packing. Thereby element-wise multiplications on these positions only give 0, and thus no other information except the contingency table are revealed by $\boldsymbol{\mu}$. The idea behind this method, to some extent, is the Chinese-Remainder-Theorem. The coprime duplication plays a major role in the above evaluation.

Following the PrivateContingencyTable procedure, we describe how to achieve the zero-out suppression functionality. Let the suppression threshold be $\mathcal{T} \in \mathbb{Z}^+$ and let $\Sigma := k_1 k_2$.

PCT-Suppression $(\{[\![\mathcal{E}_{\text{id}}(c_{ip})]\!], [\![\mathcal{E}_{\text{id}}(c_{iq})]\!]\}_{i=1}^N, \mathcal{T})$:

The cloud: Step 1 to 3 follows PrivateContingencyTable.

4. Invokes bGT: $[\![\boldsymbol{\gamma}]\!] = \mathsf{bGT}([\![\boldsymbol{\mu}]\!], [\![\mathcal{T}]\!])$ with $D = N$ and $\theta = \Sigma$ for the bGT protocol.
5. Computes $[\![\boldsymbol{\mu}']\!] = [\![\boldsymbol{\mu}]\!] + \mathsf{Pack}(\boldsymbol{\delta})$ where $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{Z}_t^{\Sigma}$.
6. Computes $[\![\boldsymbol{\gamma}']\!] = [\![\boldsymbol{\gamma}]\!] + \mathsf{Pack}(\boldsymbol{r})$ where the length of the vector $\boldsymbol{r}$ is $N\Sigma$, $r_{k\Sigma+x} = \delta_x$ for $0 \le k < N$, and $1 \le x \le \Sigma$.
7. Samples $\boldsymbol{r}^* \xleftarrow{\$} (\mathbb{Z}_t/\{0\})^{N\Sigma}$ and computes $[\![\boldsymbol{\gamma}^*]\!] = [\![\boldsymbol{\gamma}]\!] \cdot \mathsf{Pack}(\boldsymbol{r}^*)$.
8. Outputs $[\![\boldsymbol{\mu}']\!], [\![\boldsymbol{\gamma}']\!]$ and $[\![\boldsymbol{\gamma}^*]\!]$.

The decryptor:

9. Finds out the set $\mathcal{SZ} := \{(s, z = k\Sigma + s) | \gamma^*_{k\Sigma+s} = 0, 1 \le s \le \Sigma, 0 \le k < N\}$.
10. Initializes $\hat{\boldsymbol{\mu}}$ as $\hat{\boldsymbol{\mu}} = \boldsymbol{0}$ and then sets $\hat{\mu}_s = \mu'_z - \gamma'_z$ for $(s, z) \in \mathcal{SZ}$.
11. Outputs $\hat{\boldsymbol{\mu}}$.

We describe the idea of our PCT-Suppression protocol. Without loss of generality, we presume that $\mu_s > \mathcal{T}$ for some specific $1 \le s \le \Sigma$. According to the bGT protocol, we have *one and only one* 0 in the set $\Gamma_s := \{\gamma_{k\Sigma+s}\}_{k=0}^{N-1}$. This enables us to hide numbers. If we have only one 0 in Γ_s, we can recover the value of μ_s from the tuple $\{\mu_s + \delta, \delta + \Gamma_s, r^* \cdot \Gamma_s\}$ with some non-zero random value r^*. Here the mathematic operations are carried out on each element of Γ_s.

On the other hand, if $\mu_s \le \mathcal{T}$, after the execution of the bGT protocol we have $0 \notin \Gamma_s$. We can not recover the value of μ_s from the tuple. Thereby, the suppression is achieved. We aim to hide the rare individuals or cases in the population by zero-outing the counts in contingency tables. Our PCT-Suppression procedure hides the counts in the contingency table with values smaller than $\mathcal{T}$ but enables us to learn other counts that with values larger than $\mathcal{T}$.

k-percentile. To compute the k-percentile on ciphertexts, we leverage the staircase encoding $\mathcal{E}_{\text{st}}$. We conduct the k-percentile of the attribute $\mathcal{O}_j$ on FHE ciphertexts as follows.

Private k-Percentile $(\{[\![\mathcal{E}_{\text{st}}(o_{ij})]\!]\}_{i=1}^N, k)$

The cloud:

1. Computes $[\![\boldsymbol{f}]\!] = \sum_{i=1}^N [\![\mathcal{E}_{\text{st}}(o_{ij})]\!]$.
2. Computes $k' = \lceil (kN)/100 \rceil$.
3. Invokes bGT: $[\![\boldsymbol{\gamma}]\!] = \mathsf{bGT}([\![\boldsymbol{f}]\!], k')$ with $D = N$ and $\theta = |\mathcal{O}_j|$ for the bGT protocol.
4. Outputs $[\![\boldsymbol{\gamma}]\!]$.

The decryptor:

5. Finds out an index $1 \le n^* \le |\mathcal{O}_j|$ s.t. $0 \notin \{\gamma_{k|\mathcal{O}_j|+n^*-1}\}_{k=0}^{N-1}$ and $0 \in \{\gamma_{k|\mathcal{O}_j|+n^*}\}_{k=0}^{N-1}$. If no such n^* exists, sets the value of n^* as
$$n^* = \begin{cases} 1 & \text{if } 0 \notin \boldsymbol{\gamma} \\ |\mathcal{O}_j| & \text{o.w.} \end{cases}$$
6. Outputs $\hat{s}^j_{n^*}$.

The decryptor can derive the k-percentile of the attribute $\mathcal{O}_j$ from $\boldsymbol{\gamma}$. Indeed, we obtain the *cumulative frequencies* of $\{o_{1j}, \ldots, o_{Nj}\}$ in Step 1 due to the use of staircase encoding. For instance, let us consider the ordinal data $\{\hat{s}^j_1, \hat{s}^j_2, \hat{s}^j_3, \hat{s}^j_3, \hat{s}^j_1, \hat{s}^j_2\}$

for $N = 6$. Then the summation in Step 1 gives cumulative frequencies $\boldsymbol{f} = [2, 4, 6]$. To get the k-percentile, we only need to find out, from left to right, the *first* frequency that is larger than $\lceil (kN)/100 \rceil$. In the previous example, we know $\hat{s}_2^j$ is the 50-percentile point because $f_1 < 3 \wedge f_2 \geq 3$. We perform the comparisons using the bGT protocol. Thus, to determine the k-percentile from $\boldsymbol{\gamma}$ we simply find an index $1 \leq n^* \leq |\mathcal{O}_j|$ s.t. $0 \notin \{\gamma_{k|\mathcal{O}_j|+n^*-1}\}_{k=0}^{N-1}$ while $0 \in \{\gamma_{k|\mathcal{O}_j|+n^*}\}_{k=0}^{N-1}$. For the boundary conditions, we can determine that $\hat{s}_1^j$ is the k-percentile point if 0 is absent in $\boldsymbol{\gamma}$. On the other hand if $0 \in \{\gamma_{k|\mathcal{O}_j|+n^*}\}_{k=0}^{N-1}$ for all possible n^*, we know that $\hat{s}_{|\mathcal{O}_j|}^j$ is the k-percentile of the population.

C. Predictive Statistics

Principal Component Analysis. For the evaluation of PCA, we can perform the computation of Eq. 8 and Eq. 9 on ciphertexts directly. Given the collection of numerical data $\boldsymbol{X} \in \mathbb{Z}_t^{N \times d_n}$, we evaluate the *first* principal component with T iterations as follows.

PrivatePCA $(\{[\![\boldsymbol{x}_i^\top]\!], [\![\boldsymbol{x}_i \boldsymbol{x}_i^\top]\!]\}_{i=1}^N, T)$

The cloud:

1. Computes $[\![N\boldsymbol{\mu}]\!] = \sum_{i=1}^N [\![\boldsymbol{x}_i^\top]\!]$.
2. Computes $[\![N^2\boldsymbol{\Sigma}]\!] = N \cdot \sum_{i=1}^N [\![\boldsymbol{x}_i \boldsymbol{x}_i^\top]\!] - [\![N\boldsymbol{\mu}]\!] \cdot [\![N\boldsymbol{\mu}^\top]\!]$.
3. Computes $[\![\boldsymbol{v}^{(\tau+1)}]\!] = [\![N^2\boldsymbol{\Sigma}]\!] \cdot [\![\boldsymbol{v}^{(\tau)}]\!]$ for $0 \leq \tau < T$.
4. Outputs $[\![\boldsymbol{v}^{(T)}]\!]$ and $[\![\boldsymbol{v}^{(T-1)}]\!]$.
5. The decryptor outputs the largest eigenvalue as $\lambda_1 = \|\boldsymbol{v}^{(T)}\| / \|\boldsymbol{v}^{(T-1)}\|$ and the associated eigenvector as $\boldsymbol{u}_1 = \boldsymbol{v}^{(T)} / \|\boldsymbol{v}^{(T)}\|$.

Step 1 and Step 2 follow Eq. 8 except we can not perform the division on ciphertexts. Notice that, in Step 2, the operation $[\![N\boldsymbol{\mu}]\!] \cdot [\![N\boldsymbol{\mu}^\top]\!]$ generates ciphertexts of a matrix. The evaluation in Step 3 is also straightforward using our matrix–vector multiplication primitives described in Section IV-A.

Linear Regression. To conduct the linear regression of Eq. 10, we need to compute the inverse of the design matrix $\boldsymbol{X}^\top \boldsymbol{X}$. To do so, we use the DF-MatrixInversion procedure in Eq. 11. The evaluation of Eq. 11 on ciphertexts are straightforward using our matrix multiplication primitive described in Section IV-A. Given the collection of numerical data $\{(\boldsymbol{x}_i^\top, y_i)\}_{i=1}^N$ and the largest eigenvalue λ_1 of the design matrix, we can evaluate Eq. 10 with T iterations as follows.

PrivateLR $(\{[\![y_i \boldsymbol{x}_i^\top]\!], [\![\boldsymbol{x}_i \boldsymbol{x}_i^\top]\!]\}_{i=1}^N, [\![\lambda_1]\!], T)$

The cloud:

1. Computes $[\![\boldsymbol{X}^\top \boldsymbol{y}]\!] = \sum_{i=1}^N [\![y_i \boldsymbol{x}_i^\top]\!]$ and $[\![\boldsymbol{X}^\top \boldsymbol{X}]\!] = \sum_{i=1}^N [\![\boldsymbol{x}_i \boldsymbol{x}_i^\top]\!]$.
2. Invokes the DF-MatrixInversion procedure
$$[\![\lambda_1^{2^T} (\boldsymbol{X}^\top \boldsymbol{X})^{-1}]\!] = \text{DF-MatrixInversion}([\![\boldsymbol{X}^\top \boldsymbol{X}]\!], [\![\lambda_1]\!], T).$$
3. Outputs $[\![\lambda_1^{2^T} \boldsymbol{w}^*]\!] = [\![\lambda_1^{2^T} (\boldsymbol{X}^\top \boldsymbol{X})^{-1}]\!] \cdot [\![\boldsymbol{X}^\top \boldsymbol{y}]\!]$.
4. The decryptor outputs $\boldsymbol{w}^*$ by dividing $\lambda_1^{2^T} \boldsymbol{w}^*$ with $\lambda_1^{2^T}$.

Notice that the multiplication in Step 3 is a matrix–vector multiplication. The DF-MatrixInversion computes the matrix inversion with a known factor $\lambda_1^{2^T}$. Thereby, our PrivateLR procedure computes the linear regression model $\boldsymbol{w}^*$ with the factor $\lambda_1^{2^T}$.

Plaintext Precision Expansion (PPE). We have described straightforward procedures to conduct the PCA and linear regression on ciphertexts, using our matrix primitives. However, we still have an issue in implementing these procedures. That is, the current implementation of the BGV scheme, i.e., the HElib [26], only allows a maximum of 60-bits plaintext precision which might not be sufficiently large enough for conducting the PCA and linear regression. We show an example of this below.

We take the PCA as an example. Assume that the $d_n \times d_n$ covariance matrix $\boldsymbol{\Sigma}$ (as Eq. 8) is B-bounded, i.e. $|\sigma_{ij}| \leq B$ for all $\sigma_{ij} \in \boldsymbol{\Sigma}$. After T iterations, the output from Eq. 9 is bounded by $d_n^T M^{T+1} B^{T+1}$. Recall that we need to introduce a fixed magnifier M to convert the real values to integers. Presuming that we use $B = 10^2$, $M = 10^3$, and $d_n = 5$, then the estimation above reveals that $T = 3$ iterations are not allowed because $d_n^3 M^4 B^4 \approx 2^{73}$ exceeds 2^{60}, the maximum plaintext precision. As a result, the 60-bits precision makes it possible to perform only a few iterations on ciphertexts. However, the iterative algorithms we used for the PCA and linear regression might not give converged solutions within a few iterations, which means we can obtain only very rough approximations for the PCA and linear regression. To address this, we need to perform more iterations, which requires a higher plaintext precision.

Class	Protocol	Input x	Output z	$f(x)$
model-I	Matrix addition	$\boldsymbol{X}, \boldsymbol{Y}$	$\boldsymbol{X} + \boldsymbol{Y}$	$\boldsymbol{X} + \boldsymbol{Y}$
	Matrix multiplication	$\boldsymbol{X}, \boldsymbol{Y}$	$\boldsymbol{XY}$	$\boldsymbol{XY}$
	Histogram	$\{\mathcal{E}_{\text{id}}(c_{ij})\}_{i=1}^{N}$	$\boldsymbol{h}$	$\boldsymbol{h}$ (Eq. 2)
	Count	$\{\mathcal{E}_{\text{id}}(c_{ij})\}_{i=1}^{N}$	$\mathbf{1}_p^\top \boldsymbol{h}$	$\mathbf{1}_p^\top \boldsymbol{h}$ (Eq. 3)
	PrivateContingencyTable	$\{\mathcal{E}_{\text{id}}(c_{pj}), \mathcal{E}_{\text{id}}(c_{qj})\}_{i=1}^{N}$	$\boldsymbol{\mu}$	$\boldsymbol{\mu}$ (Eq. 5)
model-II	bGT	a, b	$\boldsymbol{\gamma}$	$\mathbf{1}\{a > b\}$
	PCT-Suppression	$\{\mathcal{E}_{\text{id}}(c_{pj}), \mathcal{E}_{\text{id}}(c_{qj})\}_{i=1}^{N}$	$\boldsymbol{\mu}', \boldsymbol{\gamma}', \boldsymbol{\gamma}^*$	$\hat{\boldsymbol{\mu}}$ (Eq. 6)
	Private k-Percentile	$\{\mathcal{E}_{\text{st}}(o_{ij})\}_{i=1}^{N}$	$\boldsymbol{\gamma}$	$\hat{s}_{n^*}^{j}$ (Eq. 7)
	PrivateHistOrder	$\{\mathcal{E}_{\text{id}}(c_{ij})\}_{i=1}^{N}$	$\{\boldsymbol{\gamma}_{uv}\}_{1 \le u < v \le \lvert\mathcal{C}_j\rvert}$	$\boldsymbol{k}$ (Eq. 4)
	PrivatePCA	$\{\boldsymbol{x}_i^\top, \boldsymbol{x}_i \boldsymbol{x}_i^\top\}_{i=1}^{N}$	$\boldsymbol{v}^{(T)}, \boldsymbol{v}^{(T-1)}$	$\boldsymbol{u}_1, \lambda_1$ (Eq. 9)
	PrivateLR	$\{y_i \boldsymbol{x}_i^\top, \boldsymbol{x}_i \boldsymbol{x}_i^\top, \lambda_1\}_{i=1}^{N}$	$\lambda_1^{2^T} \boldsymbol{w}^*$	$\boldsymbol{w}^*$ (Eq. 10)

TABLE IV: Model classification

We introduce PPE to achieve a higher plaintext precision with the application of the Chinese-Remainder-Theorem (CRT) [15]. Let f be the function that we evaluate, and let x be the input of f. Suppose that $f(x) > 2^{60}$. We, thus, cannot directly evaluate f on the ciphertext of x since we cannot offer plaintext with values larger than 2^{60}. To alleviate this problem, we with K distinct plaintext spaces and get K values as $\{f(x) \mod t_k\}_{k=1}^{K}$, where $t_k < 2^{60}$ for all k. According to the CRT, if we have $\gcd(t_k, t_{k'}) = 1$ for all $k \neq k'$, then from the set of values $\{f(x) \mod t_k\}_{k=1}^{K}$, we can uniquely determine the value which is equal to $f(x) \mod t$ for $t = \prod_{k=1}^{K} t_k$. Thereby we can obtain $f(x)$ by using such small t_k's with product is larger than $f(x)$. If we fix the magnitude of t_k, then we can achieve any desired precision by adjusting K for a desired precision. Indeed, PPE is achieved at the expense of increasing both computational and communication cost by a factor K while PPE is totally parallelizable. We can also apply the PPE to the evaluation of the descriptive statistics.

Tuning of Parameters. To obtain a final solution with the desired precision, we need to appropriately determine the magnification constant M, the number of iterations T, and the number ciphertexts used for precision expansion K. Given a desired precision of the final solution, the number of iterations required to reach the precision can be determined by the convergence property of the iterative method (The power method for PCA guarantees geometric convergence. The iterative matrix inversion guarantees quadratic convergence). Given the desired precision and the number of iterations, the bit-length to represent the final solution can be determined. If this bit length is shorter than the plaintext size, PPE is not needed. If the bit length exceeds the plaintext size, precision expansion is introduced so that the plaintext space covers the required bit length. We experimentally surveyed the relations between the desired precision of the final solution, the number of iterations, the magnification constant, and the bit-length to represent the final solution. See Appendix A for the details.

VI. Security Analysis

We also assume that all stakeholders hold the encryption key pk while only the decryptor holds the decryption key sk. We focus on secure outsourcing in this paper. Thus, we do not discuss the phase of key generation and key distribution.

The outline of our secure outsourcing that evaluates deterministic function f proceeds as follows. We consider the following two models for the security analysis.

Model-I ($z = f(x)$)**.** The encryptor encrypts his private input x and sends $[\![x]\!]$ to the cloud. The cloud homomorphically evaluates f on $[\![x]\!]$ and sends $[\![f(x)]\!]$ to the decryptor. The decryptor decrypts $[\![f(x)]\!]$ and obtains $f(x)$.

Model-II ($z \neq f(x)$)**.** The encryptor encrypts his private input x and sends $[\![x]\!]$ to the cloud. The cloud performs specified homomorphic operations on $[\![x]\!]$ and sends the resulting ciphertext $[\![z]\!]$ to the decryptor. The decryptor decrypts $[\![z]\!]$ and obtains z. The decryptor derives $f(x)$ from z by some local post-processing.

We summarize the model classification of the proposed protocols in Table IV. We give the security statements about the protocols.

Theorem 1: We assume all stakeholders behave semi-honestly and assume that the decryptor and the cloud do not collude with each other. Let x be a private input of the encryptor. If the FHE scheme provides semantic security, after execution of the protocol for f, the decryptor learns z but nothing else. The encryptor and the cloud learn nothing.

We give the proof of Theorem 1 in the next paragraph. If $z = f(x)$, Theorem 1 guarantees the security of the protocol for f. If $z \neq f(x)$, we need to show that z reveals nothing but $f(x)$. For some protocols (i.e. bGT, PCT-Suppression, Private k-Percentile, and PrivateHistOrder), we show that z does not reveal any information except $f(x)$. However, in our construction, we allow the protocol of PrivatePCA and PrivateLR to output z that contains information more than $f(x)$ for the sake of efficiency. We discuss these points in the following.

Security Analysis. We give a sketch proof of Theorem 1 and defer the full argument to the full version of our paper. Our proof follows the simulation-based paradigm [10]. Let the view of the encryptor, decryptor, and the cloud during the execution of the protocol be $\mathcal{V}_e$, $\mathcal{V}_d$, and $\mathcal{V}_c$, respectively. Notice that the encryptor does not receive any message from other entities.

Proof of Theorem 1 (Sketch): Let pk be the encryption key used by the encryptor. From the construction of the protocol, the security against the semi-honest encryptor and the semi-honest decryptor are apparent. So, we omit the proofs for the encryptor and decryptor.

Security against a semi-honest cloud follows from the fact that the view of the cloud, $\mathcal{V}_c$, consists of $\{\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(x), \mathsf{Enc}_{\mathsf{pk}}(z)\}$. We can simply construct a simulator $\mathcal{S}_c$ as follow. $\mathcal{S}_c$ firstly randomly chooses values x' and z'. Then $\mathcal{S}_c$ simulates $\mathcal{V}_c$ by $\hat{\mathcal{V}}_c = \{\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(x'), \mathsf{Enc}_{\mathsf{pk}}(z')\}$. Since the FHE provides semantic security by assumption, $\mathcal{V}_c$ and $\hat{\mathcal{V}}_c$ are indistinguishable. Thus, our protocols are secure at the presence of a semi-honest cloud. ■

Security Discussion under Model-II. For protocols classified in the model-II, the decryptor obtains $f(x)$ with some post-processing on z. We show that z reveals nothing except $f(x)$ for certain protocols.

Batch Greater-Than. In $\mathsf{bGT}([\![a]\!], [\![b]\!])$ (we assume that $\theta = 1$), if $a \leq b$ holds then the output z consists of D uniform random values from $\mathbb{Z}_t/\{0\}$ and reveals nothing but the output. If $a > b$, we have one 0 in $\boldsymbol{\gamma}$ at a position selected randomly and values at remaining positions distribute uniformly on $\mathbb{Z}_t/\{0\}$. Thereby, from z the decryptor can only learn $\mathbf{1}\{a > b\}$ but nothing else.

PCT-Suppression. We use bGT to compare Σ values in the contingency table, i.e., $\boldsymbol{\mu}$, with the threshold $\mathcal{T}$. Since these comparisons are independent of each other, we focus on a specific μ_s. $\boldsymbol{\gamma}^*$ is the output from the bGT (each element are multiplied with non-zero random values). If $\mu_s > \mathcal{T}$, we have one 0 in set $\Gamma_s := \{\gamma^*_{k\Sigma+s}\}_{k=0}^{N-1}$ at a random position and remaining values are all random. Otherwise, Γ_s consists of uniform random values on $\mathbb{Z}_t/\{0\}$. Presume that, in the set Γ_s, we have $\gamma^*_{k'\Sigma+s} = 0$. Then the decryptor can learn $\hat{\mu}_s = \mu'_s - \gamma'_{k'}$ which is the desired output. On the other hand if $\mu_s \leq \mathcal{T}$, for all $1 \leq k \leq \Sigma$, value $\mu'_s - \gamma'_k$ is uniformly distributed on $\mathbb{Z}_t/\{0\}$. Consequently, from the output z, the decryptor only learns $\hat{\boldsymbol{\mu}}$ and nothing else.

Private k-Percentile. The output z of the k-percentile protocol comes from the bGT. From z, the decryptor learns that cumulative frequencies before $\hat{s}^j_{n*}$ are less than $\lceil kN/100 \rceil$ and cumulative frequencies of $\hat{s}^j_{n'}$ with $n' > n^*$ are larger than $\lceil kN/100 \rceil$. That is equivalent to knowing that $\hat{s}^j_{n^*}$ is the k-percentile of the population. Since the bGT reveals nothing except the comparison results, the k-percentile protocol reveals to the decryptor no more than that $\hat{s}^j_{n^*}$ is the k-percentile.

PrivateHistOrder. The histogram order protocol invokes bGT $\mathcal{O}(|C_j|^2)$ times to compare $|\mathcal{C}_j|$ values in the histogram and outputs the comparison results. Since the bGT reveals nothing except the comparison results, it is straightforward to see that the $\mathsf{PrivateHistOrder}$ protocol reveals to the decryptor no more than the order of counts in the histogram.

PrivatePCA. In this protocol, the decryptor receives two vectors, $\boldsymbol{v}^{(T)}$ and $\boldsymbol{v}^{(T-1)}$. He learns the largest eigenvalue $\lambda_1 = \|\boldsymbol{v}^{(T)}\| / \|\boldsymbol{v}^{(T-1)}\|$ and the associated eigenvector $\boldsymbol{u}_1 = \boldsymbol{v}^{(T)}/\|\boldsymbol{v}^{(T)}\|$. Precisely speaking, the difference of the direction of $\boldsymbol{v}^{(T)}$ and $\boldsymbol{v}^{(T-1)}$ can contain some information about the inputs. However, due to the geometric convergence property of the power method algorithm, the difference of the directions is negligible after a sufficient number of iterations. We consider that it is worth letting the decryptor perform the division after the decryption for the sake of efficiency.

PrivateLR. In this protocol, the output $z = \lambda_1^{2^T} \boldsymbol{w}^*$. We can see that the only information leaked to the decryptor is the iteration number T. Precisely speaking, T can contain some information about the condition number of $\boldsymbol{X}^\top \boldsymbol{X}$, which is related to the eigenvalues of $\boldsymbol{X}^\top \boldsymbol{X}$. However, it is not likely that the decryptor can recover (a part of) $\boldsymbol{X}$ from T. Thereby, letting the decryptor perform the division after the decryption can lead to a more efficient evaluation.

VII. Experimental Evaluation

We implemented our building blocks and all the procedures that is described in Section V. Our implementations were written in C++, and we used the HElib library [26] for the implementation of the BGV scheme. We compiled our code using g++ 4.9.2 on a machine running Ubuntu 14.04.4 with eight 2.60GHz Intel(R) Xeon(R) E5-2640 v3 processors and 32 GB of RAM. The proposed procedures and the PPE technique are parallelizable. We leveraged 8 parallels in our benchmarks to accelerate the computation.

We used multiple parameter sets in our benchmarks to show the best performance of our procedures. Our choices for selecting the parameters of the HElib are shown in Table V. In this table, we have modulo parameter t_k, the number of slots of the CRT-packing ℓ, levels parameter L, the parameter for cyclotomic polynomial m, the number of coprime moduli K, and the security level κ. We used at most $K = 8$ moduli and for each modulo we set $t_k \approx 2^{36}$ to achieve about 300-bit precision. Specifically, we used parameter set (I) for evaluating the PrivateHistOrder, Private k-Percentile procedures. The evaluations of PrivateContingencyTable and PCT-Suppression used parameter set (II) while the evaluations of PrivatePCA and PrivateLR use the set (III).

We conducted experiments on five datasets from the UCI Machine Learning Repository [18]. For detailed discussions, we focus on one of them, the Adult dataset, which includes 32561 records with 6 numerical attributes, 7 categorical attributes, and 1 ordinal attribute. Specifically, to show the scalability of the PrivatePCA and PrivateLR procedures, we also gave the benchmarks on other four datasets.

	t_k	ℓ	L	m	K	κ
(I)	67499	1742	5	5227	1	90
(II)	8191	4096	10	16384	1	80
(III)	$\approx 2^{36}$	≈ 70	32	27893	≤ 8	110

TABLE V: Parameter sets of the BGV scheme.

Type	Domain	N	Evaluation	Decryption
Hist. Order	$\lvert\mathcal{C}_j\rvert = 8$	500	1.26 $\pm$ 0.145s	1.26s
		1k	1.31 $\pm$ 0.157s	1.23s
		10k	2.72 $\pm$ 0.289s	4.80s
		32k	6.28 $\pm$ 0.484s	13.2s
	$\lvert\mathcal{C}_j\rvert = 16$	500	2.42 $\pm$ 0.439s	3.27s
		1k	2.53 $\pm$ 0.336s	3.30s
		10k	6.24 $\pm$ 0.448s	13.6s
		32k	13.8 $\pm$ 1.38s	41.2s
K-per-centile	$\lvert\mathcal{O}_j\rvert = 100$	500	4.768 $\pm$ 0.12s	3.27s
		1k	9.487 $\pm$ 0.92s	3.11s
		10k	97.515$\pm$ 1.60s	18.6s
		32k	321.285 $\pm$ 21.7s	48.8s

(a) Benchmark (adult dataset) of the PrivateHistOrder, and Private k-Percentile. Values are averaged over 10 runs.

Attributes	N	Evaluation	Decryption
$\lvert\mathcal{C}_p\rvert = 8, \lvert\mathcal{C}_q\rvert = 6$	500	35.69 $\pm$ 1.55s	3.84s
	1k	68.42 $\pm$ 4.17s	7.45s
	2k	155.26 $\pm$ 20.01s	14.83s
	4K	287.02 $\pm$ 10.10s	30.00s

(b) Benchmark of the PCT-Suppresion procedures. Values are averaged over 10 runs.

A. *Experiment Setup*

Parameters of HElib. To achieve the best performance, we need to choose the parameters of the HElib appropriately. We determined the parameters of the HElib based on the three concerns.

1) To provide the desired security level.
2) To offer sufficient spaces of the CRT packing, i.e. ℓ.
3) To operate the homomorphic rotation efficiently.

In our experiments, we used parameters shown in Table V. These parameter sets offer at least 80-bit security level and provide the number of slots up to several thousand. Moreover, homomorphic rotation on these parameters is efficient. From the implementation aspect, we choose m and t_k so that (PAlgebra is a C++ class in HElib)

```
PAlgebra::numOfGens() == 1
&& PAlgebra::SameOrd(0) == true
```

We refer to [8], [9], [26] for the homomorphic rotation.

Error Ratio. Our PrivatePCA and PrivateLR procedures use iterative algorithms and fixed-precision values. It thus introduces error. We write λ^* and $\boldsymbol{w}^*$ to denote the solutions to the PCA and the LR, respectively. We write $\hat{\lambda}$ and $\hat{\boldsymbol{w}}$ to denote the outputs obtained from our PCA and LR procedures. We define the error ratio of our procedures as follows.

$$\text{Error}_{\lambda^*} = \frac{|\lambda^* - \hat{\lambda}|}{\lambda^*} \qquad \text{Error}_{\boldsymbol{w}^*} = \frac{\|\boldsymbol{w}^* - \hat{\boldsymbol{w}}\|_2}{\|\boldsymbol{w}^*\|_2}.$$

This error ratio definition enables us to estimate the loss of accuracy. We experimentally studied the iteration-error tradeoffs in Appendix A.

B. *Experimental Results*

We measured the time of procedure evaluation and time of decryption of the results. We give the standard deviations only for the evaluation time due to the space limitation. We remark that standard deviations for decryption times were negligible in our experiments.

Histogram Order & K-percentile. Table VIa shows the experimental results of the PrivateHistOrder and Priavte k-Percentile procedures. For the histogram order (upper part), we ran the experiments on two categories *workclass* and *education*, which respectively consists of 8 and 16 attribute values. The time of decryption increases linearly with respect to N and it dominates the evaluation time when N is large. This is because we needed to decrypt $\lceil(|\mathcal{C}_j|^2 \cdot N)/\ell\rceil$ ciphertexts. The decryption is totally parallelizable so it can be easily reduced by using more cores.

For the k-percentile procedures, we conducted the experiments with the ordinal attribute *age* from the adult dataset and presumed that the domain size $|\mathcal{O}_j| = 100$ (lower part of Table VIa). As long as $n < \ell$ (i.e., 1742), the time for download and

M	T	K	Evaluation	Decryption
	3	2	67.3 $\pm$ 4.89s	0.876s
10	4	3	99.9 $\pm$ 4.77s	0.848s
	5	3	122 $\pm$ 2.63s	0.874s
	3	3	70.6 $\pm$ 4.19s	0.848s
100	4	4	104 $\pm$ 7.68s	1.27s
	5	4	128 $\pm$ 7.93	1.26s
	3	3	72.7 $\pm$ 2.12s	0.96s
1000	4	4	108 $\pm$ 4.06s	1.25s
	5	5	136 $\pm$ 5.67s	1.43s

(a) PCA (for the first principal component)

M	T	K	Evaluation	Decryption
	1	1	173 $\pm$ 9.12s	0.475s
10	2	3	341 $\pm$ 8.12s	0.428s
	3	5	672 $\pm$ 9.76s	0.618s
	1	2	160 $\pm$ 3.97s	0.397s
100	2	4	400 $\pm$ 27.8s	0.649s
	3	7	787 $\pm$ 10.5s	0.816s
	1	2	164 $\pm$ 8.25s	0.388s
1000	2	4	383 $\pm$ 10.0s	0.622s
	3	8	865 $\pm$ 11.7s	0.944s

(b) Linear Regression (the time of one call of PCA were omitted)

TABLE VI: Benchmarks of the PCA and LR protocol (adult dataset): M stands for the magnification constant; T denotes the number of iterations. K is the expansion factor. Values are averaged over 10 runs.

Data set	d_n	N	PCA (eval/decrypt)	LR (eval/decrypt)
adult	6	32561	141.21 / 2.36	872.82 / 1.59
autompg	7	398	149.80 / 1.82	950.93 / 1.47
wine-equality	12	4898	217.32 / 1.94	3543.76 / 1.68
forestfires	13	513	299.38 / 1.87	3757.99 / 1.59
communities	20	1994	472.98 / 1.86	10871.34 / 1.76

TABLE VII: Experimental results of the PCA and LR protocol using UCI datasets. d_n stands for the number of the numerical attributes. The unit of time is the second.

decryption were steady. When $n > \ell$, the decryption time increased almost linearly with n. To reduce the response time, the analyst can choose the parameters of BGV that offer a larger ℓ.

Contingency Table. Table VIb shows the benchmarks of the PCT-Suppression. We ran the experiments on two categories *workclass* and *relationship*, which respectively consists of 8 and 6 attribute values. The time of evaluation and decryption grow linearly with the number of data N, but this computation is entirely parallelizable in our PCT-Suppression procedure. We can easily accelerate this procedure with a higher level of parallelism.

Most of the decryption time in the PCT-Suppression procedure is the time of decrypting the output of the bGT protocol due to the suppression functionality, while the decryption time in the PrivateContingencyTable procedure is independent of the number of data N.

PCA & Linear Regression. We used three different magnification constants M and three different iteration numbers T to benchmark the PCA protocol (only for the first principal component). The results are shown in Table VIa.

By applying the CRT-packing, the number of ciphertexts to transfer and decrypt during the post-processing phase are $O(\lceil d_n/\ell \rceil)$, which is independent of the number of records N. As shown in Table VIa, the download and decryption time were steady. It took less than three minutes to evaluate one principal component with a low error ratio $\text{Error}_{\lambda^*} < 0.1$ (i.e., $T = 5$, $M = 1000$ in Fig. 4a in Appendix A).

The experimental results of the LR protocol are shown in Table VIb. We omit here the computation time for obtaining the largest eigenvalue λ_1. Similarly, we benchmarked the protocol in nine settings. For the same reason as for the PCA protocol, the time to download and the time to decrypt the output from our LR protocol were negligible. The matrix inversion converges quadratically. We thus achieved a error ratio $\text{Error}_{\boldsymbol{w}^*} < 10^{-3}$ within a few iterations (i.e., $T = 3, M = 1000$ in Fig 4b in Appendix A). For the evaluation time, it took about 17 minutes to achieve this error guarantee.

Extra Experiments for the Predictive Statistics. The extra experimental results of the PCA protocol (the first principal component only) and the LR protocol are shown in Table VII. Here, we used $M = 1000$ and $T = 3$, and we listed the evaluation time and decryption time. We can see that the evaluation time of PCA protocol increases linearly with the input dimension d_n, while the evaluation time of the LR procedure increases quadratically with d_n.

VIII. Conclusions

We proposed to conduct privacy-preserving statistical analysis using fully homomorphic encryption. Also, we introduced two building blocks, matrix operations of encrypted matrices and a novel batch greater-than primitive. With the application of these primitives, we show how to conduct the descriptive and predictive statistics on FHE ciphertexts efficiently. We experimentally demonstrated the utility of our procedures. For example, it took us less than 20 minutes to conduct the model building of a linear regression on about 30k data of 6 features as input. We conclude that with applications of CRT-packing and appropriate data encoding, securely conducting statistical analysis on large-scale datasets using a fully homomorphic encryption is becoming more and more practical.

Acknowledgment. The authors would like to thank the useful and insightful comments from the reviewers and thank professor Takashi Nishide from University of Tsukuba for discussions on security analysis. This work is supported by JST CREST program "Advanced Core Technologies for Big Data Integration". This work is further supported by the JSPS KAKENHI 24680015 and 16H02864.

References

[1] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, 2013, pp. 102–118.

[2] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11*, 2015.

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012, pp. 309–325.

[4] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, 2011, pp. 97–106.

[5] ——, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, 2011, pp. 505–524.

[6] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, 2011, pp. 487–504.

[7] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University (CA, USA), 2009.

[8] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 465–482.

[9] ——, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, 2012, pp. 850–867.

[10] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[11] P. Golle, "A private stable matching algorithm," in *Financial Cryptography and Data Security, 10th International Conference, FC 2006, Anguilla, British West Indies, February 27-March 2, 2006, Revised Selected Papers*, 2006, pp. 65–80.

[12] T. Graepel, K. E. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, 2012, pp. 1–21.

[13] C. Guo and N. J. Higham, "A Schur-Newton method for the matrix pth root and its inverse," *SIAM J. Matrix Analysis Applications*, vol. 28, no. 3, pp. 788–804, 2006.

[14] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, 2014, pp. 554–571.

[15] J. Hoffstein, J. Pipher, and J. H. Silverman, *An introduction to mathematical cryptography*. (Springer, New York), 2008.

[16] N. Kirkendall and G. Sande, "Comparison of systems implementing automated cell suppression for economic statistics," *Journal of Official Statistics*, vol. 14, no. 4, p. 513, 1998.

[17] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II*, 2008, pp. 486–498.

[18] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[19] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, 2015, pp. 359–376.

[20] W.-J. Lu, Y. Yamada, and J. Sakuma, "Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption," *BMC medical informatics and decision making*, vol. 15, no. Suppl 5, p. S1, 2015.

[21] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[22] R. Mises and H. Pollaczek-Geiringer, "Praktische verfahren der gleichungsauflösung." *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, no. 2, pp. 152–164, 1929.

[23] S. U. Nabar and N. Mishra, "Releasing private contingency tables," *Journal of Privacy and Confidentiality*, vol. 2, no. 1, p. 9, 2009.

[24] M. Naehrig, K. E. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, 2011, pp. 113–124.

[25] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM conference on Electronic commerce, Denver, CO, USA*. ACM, 1999, pp. 129–139.

[26] V. S. Shai Halevi, "HELib," http://shaih.github.io/HElib/index.html, accessed: 2014-12-10.

[27] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, 2010, pp. 420–443.

[28] ——, "Fully homomorphic SIMD operations," *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

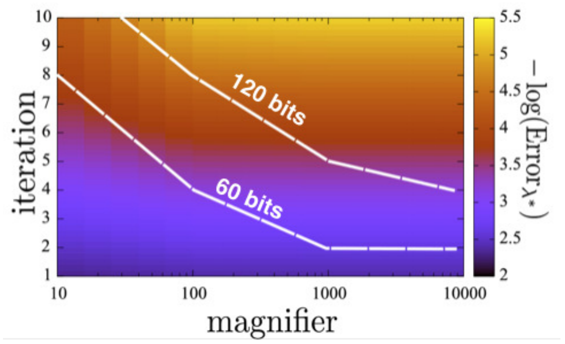
[29] D. Wu and J. Haven, "Using homomorphic encryption for large scale statistical analysis," http://cs.stanford.edu/people/dwu4/FHE-SI_Report.pdf, 2012.

[30] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, 1986, pp. 162–167.

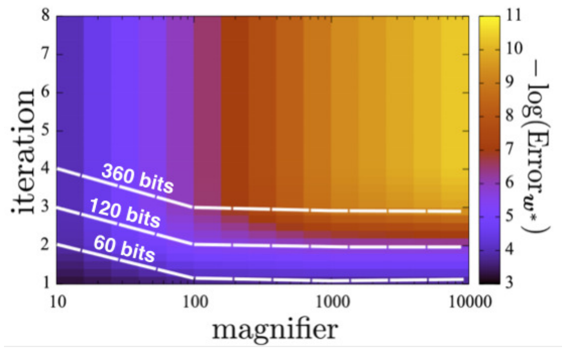
[31] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Secure pattern matching using somewhat homomorphic encryption," in *Proceedings of the 2013 ACM workshop on Cloud computing security workshop, Berlin, Germany*. ACM, 2013, pp. 65–76.

Fig. 4: Tradeoffs between the magnification constant, number of iterations, and error ratios. The dotted lines show the number of bits of values in Eq. 9 and Eq. 10.

(a) Tradeoffs of Error$_{\lambda^*}$



(b) Tradeoffs of Error$_{\boldsymbol{w}^*}$



Procedures	Evaluation	Decryption	Number of Generated Ciphertexts
histogram (count)	$O(N)$	$O(1)$	$O(1)$
histogram order	$O(\lceil(\lvert\mathcal{C}_p\rvert^2 N)/\ell\rceil)$	$O(\lceil(\lvert\mathcal{C}_p\rvert^2 N)/\ell\rceil)$	$O(\lceil(\lvert\mathcal{C}_p\rvert^2 N)/\ell\rceil)$
contingency table (no suppression)	$O(N \log(\lvert\mathcal{C}_p\rvert\lvert\mathcal{C}_q\rvert))$	$O(1)$	$O(1)$
contingency table (suppression)	$O(N \log(\lvert\mathcal{C}_p\rvert\lvert\mathcal{C}_q\rvert))$	$O(\lceil N \log(\lvert\mathcal{C}_p\rvert\lvert\mathcal{C}_q\rvert)/\ell\rceil)$	$O(\lceil N \log(\lvert\mathcal{C}_p\rvert\lvert\mathcal{C}_q\rvert)/\ell\rceil)$
k-percentile	$O(N)$	$O(\lceil N/\ell\rceil\lvert\mathcal{O}_j\rvert)$	$O(\lceil N/\ell\rceil\lvert\mathcal{O}_j\rvert)$
PCA (per principal component)	$O(KTd_n)$	$O(K\lceil d_n/\ell\rceil)$	$O(K\lceil d_n/\ell\rceil)$
linear regression	$O(KTd_n^2 + d_n)$	$O(K\lceil d_n^2/\ell\rceil)$	$O(K\lceil d_n^2/\ell\rceil)$

TABLE VIII: The computation and space complexity of our proposed procedures: N is the number of data; ℓ stands for the number of slots of the CRT-packing; $\mathcal{O}_j$ represents an ordinal attribute; $\mathcal{C}_p, \mathcal{C}_q$ represent categorical attributes; T denotes the number of iterations: K signifies the number of co-prime moduli; and d_n is the number of numerical attributes.

Appendix

A. Iteration-Error Tradeoffs

To study the tradeoffs between the computation time (number of iterations) and the error ratio, we ran a grid-search experiment on the adult dataset. Figure 4a presents the tradeoffs for Error$_{\lambda^*}$ (the base of $\log$ is e). This plot enables us to choose parameters for our system. Suppose the analyst seeks an error ratio of approximately 10^{-2} with 3 digits preserved. The plot indicates that he should use a magnifier that is larger than 1000 for preserving the digits, and to use more than 6 iterations according to the observation that $-\log 10^{-2} \approx 4.6$. Thereby, he requires a plaintext precision of more than 120 bits. In other words, if the moduli $p_k \approx 2^{36}$, the expansion factor $K > 3$ is needed for plaintext precision expansion.

The convergence rate of DF-MatrixInversion depends on the choice of λ, which is preferred to be the largest eigenvalue of the inverse matrix. To demonstrate the effectiveness of our system, we ran another grid-search experiment. In the second grid-search experiment, we ran the regression protocol on the adult dataset in plain, whereas we used the output from our PCA protocol as λ. Similarly, the Figure 4b depicts the tradeoffs for Error$_{\boldsymbol{w}^*}$. From the plot, it is apparent that even with the approximated eigenvalue, our regression algorithm works properly. Given the required error ratio and the number of digits to preserve, the way to decide the expansion factor K is similar to that in the PCA protocol.

B. Asymptotic Analysis

The computational and spatial complexity of the proposed procedures are summarized in Table VIII.

C. Security Proofs

In this subsection, we give formal proofs for the security of the protocols: bGT, PCT-Suppression, PrivateHistOrder, and Private k-Percentile. Our proofs follow the simulation-based paradigm [10].

Proof for a Semi-honest Cloud. For all proposed primitives and protocols, the view of the cloud during a real execution consists of ciphertexts only. Thus the security for a semi-honest cloud reduces to the semantic security of FHE. Let f be one of the proposed protocols (Table IV). We state the security of f against a semi-honest cloud.

Theorem 2: After the execution of the proposed protocol f, the cloud who behaves semi-honestly learns nothing about the private input x.

Proof: We can simply construct an ideal-world simulator $\mathcal{S}^f_{\mathsf{cloud}}$ for the view of the cloud in the real execution of f as follows.

1. $\mathcal{S}^f_{\mathsf{cloud}}$ receives the encryption key pk and the specification about f, such as the number of inputs and the domain of inputs (summarized in Table IV).
2. $\mathcal{S}^f_{\mathsf{cloud}}$ samples and computes $\mathsf{Enc}_{\mathsf{pk}}(\hat{x})$ where $\hat{x}$ is chosen uniformly at randomly according to the specification of f .
3. $\mathcal{S}^f_{\mathsf{cloud}}$ outputs $\mathsf{Enc}_{\mathsf{pk}}(\hat{x})$.

The view of the cloud in the real protocol execution is $\mathcal{V}^f_{\mathsf{cloud}} = \{\mathsf{Enc}_{\mathsf{pk}}(x)\}$ while the simulated view by $\mathcal{S}^f_{\mathsf{cloud}}$ is $\hat{\mathcal{V}}^f_{\mathsf{cloud}} = \{\mathsf{Enc}_{\mathsf{pk}}(\hat{x})\}$. We immediately have $\hat{\mathcal{V}}^f_{\mathsf{cloud}} \approx^c \mathcal{V}^f_{\mathsf{cloud}}$ from the the semantic security of the FHE. ■

Batch Greater-than. We prove that Algorithm 1 securely evaluates the greater-than functionality. Remind that Algorithm 1 is executed by the cloud. The decryptor receives $[\![\gamma]\!]$ (Step 6) from the cloud and decrypts it to learn the greater-than result. For simplicity, we here prove for the case $\theta = 1$. The proof can be readily generalized to the case of $\theta > 1$ without loss of generality.

Theorem 3: For inputs a and $b \in [D]$ (i.e., $\theta = 1$), Algorithm 1 securely evaluates $\mathbf{1}\{a > b\}$. The decryptor learns $\mathbf{1}\{a > b\}$ as the output of Algorithm 1 and nothing else.

Proof: We construct a simulator $\mathcal{S}_{\mathsf{bGT}}$ for the view of the decryptor in the real execution as follows.

1. $\mathcal{S}_{\mathsf{bGT}}$ receives the encryption key pk, the domain size D, and the evaluation result $\mathbf{1}\{a > b\}$ as input.
2. $\mathcal{S}_{\mathsf{bGT}}$ samples a vector $\hat{\boldsymbol{\gamma}}$ uniformly at random from $(\mathbb{Z}_t/\{0\})^D$.
3. If $\mathbf{1}\{a > b\}$ holds, $\mathcal{S}_{\mathsf{bGT}}$ chooses $\hat{k}^*$ uniformly at random from $[D]$ and sets $\hat{\gamma}_{\hat{k}^*} = 0$.
3. $\mathcal{S}_{\mathsf{bGT}}$ outputs $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{Pack}(\hat{\boldsymbol{\gamma}}))$.

The simulated view for the decryptor is $\hat{\mathcal{V}}_{\mathsf{bGT}} = \{\hat{\boldsymbol{\gamma}}\}$. From the construction of Algorithm 1, the view of the decryptor in the real execution is $\mathcal{V}_{\mathsf{bGT}} = \{\boldsymbol{\gamma}\}$. To prove the security of Algorithm 1, we need to show that $\mathcal{V}_{\mathsf{bGT}}$ and $\hat{\mathcal{V}}_{\mathsf{bGT}}$ are indistinguishable.

We first show the distribution of $\boldsymbol{\gamma}$ in the real execution. According to Line 4 and Line 5 of Algorithm 1, we have

$$\gamma_k = (a - b - \pi_0(k)) \cdot r_k \text{ for all } k \in [D],$$

where π_0 is a private random permutation $\pi_0 : [D] \to [D]$ generated by the cloud and r_k is selected uniformly at random from $\mathbb{Z}_t/\{0\}$. In the case of $a \leq b$, we have $a - b - \pi_0(k) \neq 0$ for all $k \in [D]$. The value of γ_k, is thus, uniformly distributed in $\mathbb{Z}_t/\{0\}$ due to the random value r_k. On the other hand (i.e., $a > b$), we have *one and only one* $k^* \in [D]$ such that $\gamma_{k^*} = a - b - \pi_0(k^*) = 0$. Furthermore, the value of k^* distributes uniformly in $[D]$ due to the random permutation π_0. The value of γ_k for all $k \neq k^*$ distributes uniformly on $\mathbb{Z}_t/\{0\}$. This is precisely the same distribution from which $\mathcal{S}_{\mathsf{bGT}}$ samples the $\hat{\boldsymbol{\gamma}}$, and we conclude that $\mathcal{V}_{\mathsf{bGT}} \approx^c \hat{\mathcal{V}}_{\mathsf{bGT}}$. ■

PCT-Suppression. We show the security of the $\mathsf{PCT\text{-}Suppression}$ protocol.

Theorem 4: The $\mathsf{PCT\text{-}Suppression}$ protocol securely computes Eq. 6. At the end of the execution of $\mathsf{PCT\text{-}Suppression}$, the decryptor learns $\bar{\boldsymbol{\mu}}$ and nothing else.

Proof: Due to the independent random permutations π_js of bGT and the independent random values $\boldsymbol{\delta}$ and $\boldsymbol{r}^*$ of $\mathsf{PCT\text{-}Suppression}$, the computation of

$$\bar{\mu}_s = \mu_s \cdot \mathbf{1}\{\mu_s > \mathcal{T}\}$$

is processed independently for each $1 \leq s \leq \Sigma$. Thus, it suffices to show the security for a specific index s. We construct a simulator $\mathcal{S}_{\mathsf{PCT}}$ for the view of the decryptor in the real execution as follows (only for a specified index s).

1. $\mathcal{S}_{\mathsf{PCT}}$ receives the encryption key pk, and the evaluation result $\bar{\mu}_s$ as input.
2. $\mathcal{S}_{\mathsf{PCT}}$ chooses $\hat{\boldsymbol{\gamma}}'_s$ and $\hat{\boldsymbol{\gamma}}^*_s$ uniformly at random from $(\mathbb{Z}_t/\{0\})^\Sigma$.
3 $\mathcal{S}_{\mathsf{PCT}}$ chooses $\hat{\delta}_s$ uniformly at random from $\mathbb{Z}_t$ and sets $\hat{\mu}'_s = \bar{\mu}_s + \hat{\delta}_s$.
4. If $\bar{\mu}_s > 0$ holds (i.e., no suppression), $\mathcal{S}_{\mathsf{PCT}}$
   4.1 chooses $\hat{k}^*$ uniformly at random from $[\Sigma]$,
   4.2 sets the $\hat{k}^*$-th element of $\hat{\boldsymbol{\gamma}}'_s$ as $\hat{\gamma}'_{s\hat{k}^*} = \hat{\delta}_s$,
   4.3 sets the $\hat{k}^*$-th element of $\hat{\boldsymbol{\gamma}}^*_s$ as $\hat{\gamma}^*_{s\hat{k}^*} = 0$.
5. $\mathcal{S}_{\mathsf{PCT}}$ outputs three ciphertexts: $\mathsf{Enc}_{\mathsf{pk}}(\hat{\mu}'_s)$, $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{Pack}(\hat{\boldsymbol{\gamma}}'_s))$, and $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{Pack}(\hat{\boldsymbol{\gamma}}^*_s))$.

The simulated view is $\hat{\mathcal{V}}_{\mathsf{PCT}} = \{\hat{\mu}'_s, \hat{\boldsymbol{\gamma}}'_s, \hat{\boldsymbol{\gamma}}^*_s\}$. According to the construction of $\mathsf{PCT\text{-}Suppression}$, the view of the decryptor in the real execution consists of three components: $\mathcal{V}_{\mathsf{PCT}} = \{\mu'_s, \boldsymbol{\gamma}'_s, \boldsymbol{\gamma}^*_s\}$. More specifically, $\mu'_s = \mu_s + \delta_s$ (Step 5), $\boldsymbol{\gamma}'_s = \boldsymbol{\gamma}_s + \delta_s$

(Step 6) and $\boldsymbol{\gamma}_s^* = \boldsymbol{\gamma}_s \dot{\times} \boldsymbol{r}_s^*$ (Step 7). Recall that $\dot{\times}$ is the element-wise multiplication and $\boldsymbol{\gamma}_s$ is the output (after decryption) of $\mathsf{bGT}(\llbracket \mu_s \rrbracket, \llbracket \mathcal{T} \rrbracket)$. Also, δ_s and $\boldsymbol{r}_s^*$ are chosen uniformly at random from $\mathbb{Z}_t$ and $(\mathbb{Z}_t/\{0\})^N$, respectively. We need to show that $\mathcal{V}_{\mathsf{PCT}}$ and $\hat{\mathcal{V}}_{\mathsf{PCT}}$ are indistinguishable.

We first discuss the distribution of $\mathcal{V}_{\mathsf{PCT}}$ when $\mu_s \leq \mathcal{T}$. In this case, from the proof of Theorem 3, the output of $\mathsf{bGT}(\llbracket \mu_s \rrbracket, \llbracket \mathcal{T} \rrbracket)$ consists of uniform random values from $\mathbb{Z}_t/\{0\}$. Thereby, elements of $\boldsymbol{\gamma}_s'$ and $\boldsymbol{\gamma}_s^*$ are also distributed uniformly on $\mathbb{Z}_t/\{0\}$.

When $\mu_s > \mathcal{T}$, there exists *one and only one* $k^* \in [\Sigma]$ such that $\boldsymbol{\gamma}_{sk^*} = 0$. Consequently, the k^*-th element of $\boldsymbol{\gamma}_s'$ equals to δ_s and $\gamma_{sk^*}^* = 0$. The remaining elements of $\boldsymbol{\gamma}_s'$ and $\boldsymbol{\gamma}_s^*$ still distribute uniformly on $\mathbb{Z}_t/\{0\}$.

Moreover, in both cases, the value of μ_s' distributes uniformly on $\mathbb{Z}_t$. This is precisely the same distributions from which $\mathcal{S}_{\mathsf{PCT}}$ generates the $\hat{\mu}_s'$, $\hat{\boldsymbol{\gamma}}_s'$, and $\hat{\boldsymbol{\gamma}}_s^*$. We conclude that $\mathcal{V}_{\mathsf{PCT}} \approx^c \hat{\mathcal{V}}_{\mathsf{PCT}}$. ■

Histogram Order. We show the security of the PrivateHistOrder protocol. To show the proof, we firstly define an indexing function $\pi : \mathcal{C}_j \rightarrow [|\mathcal{C}_j|]$ such that $k_{\pi(s_p^j)} = p$ for all $1 \leq p \leq |\mathcal{C}_j|$. For instance, when $\boldsymbol{k} = [2, 3, 1]$, this indicates that the indexes of attributes in the histogram sorted by the counts in ascending order is s_2^j, s_3^j and s_1^j. The indexing function π tells the index of a specific state s_p^j in $\boldsymbol{k}$, e.g., $\pi(s_2^j) = 1$.

Theorem 5: The PrivateHistOrder protocol securely evaluates Eq. 4. At the end of the execution of PrivateHistOrder, the decryptor learns $\boldsymbol{k}$ and nothing else.

Proof: We leverage $\mathcal{S}_{\mathsf{bGT}}$ to construct a simulator $\mathcal{S}_{\mathsf{HistOrder}}$ for the view of the decryptor in the real execution of PrivateHistOrder.

1. $\mathcal{S}_{\mathsf{HistOrder}}$ receives the encryption key pk, the number of data N, the domain size $|\mathcal{C}_j|$, and the evaluation result $\boldsymbol{k}$ as input.
2. For all (u, v) pairs where $1 \leq u < v \leq |\mathcal{C}_j|$, $\mathcal{S}_{\mathsf{HistOrder}}$ runs $\mathcal{S}_{\mathsf{bGT}}$ with inputs of pk, N, and $\mathbf{1}\{\pi(s_u^j) > \pi(s_v^j)\}$ (i.e., $\theta = 1$). For each invocation, we denote the output from $\mathcal{S}_{\mathsf{bGT}}$ as $\llbracket \hat{\gamma}_{uv} \rrbracket$.
3. $\mathcal{S}_{\mathsf{HistOrder}}$ outputs the ciphertexts $\{\llbracket \hat{\gamma}_{uv} \rrbracket\}_{1 \leq u < v \leq |\mathcal{C}_j|}$.

The view of the decryptor in the real execution of PrivateHistOrder consists of outputs from independent invocations of bGT. Thereby, this is straightforward to see that the security of the PrivateHistOrder protocol is reduced to the security of bGT which has been shown in Theorem 3. ■

***K*-Percentile.** We show the security of the Private k-Percentile protocol.

Theorem 6: The Private k-Percentile protocol securely computes Eq. 7. At the end of the execution of Private k-Percentile, the decryptor learns $\hat{s}_{k^*}^j$ and nothing else.

Proof: We also leverage $\mathcal{S}_{\mathsf{bGT}}$ to construct a simulator $\mathcal{S}_{\mathsf{percentile}}$ for the view of the decryptor in the real execution of Private k-Percentile.

1. $\mathcal{S}_{\mathsf{percentile}}$ receives the encryption key pk, the number of data N, the domain size $|\mathcal{O}_j|$ and the evaluation result $\hat{s}_{k^*}^j$ as input.
2. $\mathcal{S}_{\mathsf{percentile}}$ generates a vector of boolean values $\boldsymbol{f}$ in which $f_p = \mathbf{1}\{\mathsf{False}\}$ for all $1 \leq p < k^*$ and $f_p = \mathbf{1}\{\mathsf{True}\}$ for all $k^* \leq p \leq |\mathcal{O}_j|$.
3. $\mathcal{S}_{\mathsf{percentile}}$ runs $\mathcal{S}_{\mathsf{bGT}}$ with inputs of pk, N, and $\boldsymbol{f}$ (i.e. $\theta = |\mathcal{O}_j|$) and outputs the resulting ciphertexts from $\mathcal{S}_{\mathsf{bGT}}$.

The view of the decryptor in the real execution of Private k-Percentile consists of the output from *one* invocation of bGT. Thereby, the security of the Private k-Percentile protocol also reduces to the security of bGT. ■