Secure Cloud Computing via Fully Homomorphic Encryption

March 2019

Wen-jie Lu

Secure Cloud Computing via Fully Homomorphic Encryption

Graduate School of Systems and Information Engineering
University of Tsukuba

March 2019

Wen-jie Lu

Abstract

Third-party commercial cloud services such as Amazon Web Service, AliCloud and Azure provide a flexible way of using computing and storage resources on demand. Moreover, using these cloud services can significantly reduce the amount of management work, i.e., users do not need to take care of the firmware, hareware and even the software.

Using the cloud is convenient and economical, however, on the opposite side, it brings potential risks to users' personal information. For example, *Facebook Leak* and *Heartland Payment Systems (HPY)* credit card breach leak out million of personal data and credit card records. These leaks harm user's privacy and lead to a great economic loss. Also, data that transferred and stored in the third-party server is out of the user's control. Users are forced to trust cloud service providers not to spy on their private information and must to believe them not to take record even the user want to delete the data. Potential risks of data privacy and control-less issues are becoming the core concerns when using commercial cloud services. Especially when the data is highly sensitive and confidential, such as genomic data and federal records, it is difficult (even illegal) to transfer these data to an outside cloud server.

In this dissertation, we present three primitives, i.e., Π_{IP} (§ 3), Π_{MP} (§ 4) and Π_{SMP} (§ 5) using *fully homomorphic encryption*. These primitives are delicately designed for computing inner products of encrypted values, but with a different requirement. Using Π_{IP}, we can compute a *single inner product* of long vectors in a very efficient way. On the hand, Π_{MP} enables us to compute *iterative* inner products from multiple vectors, e.g., matrix exponent. In addition, for applications that require a smaller communication overhead, we present the third primitive Π_{SMP}, which is also friendly for a weak decryptor.

The inner product functionality is the fundamental operation of many statistical computations, such as statistic tests, descriptive statistics, predictive statistics, and machine learning. As the concrete applications of the proposed primitives, we present three use-case applications.

- For Π_{IP}, we present an efficient FHE-based protocol to conduct Hardy–Weinberg Equilibrium test and Linkage Equilibrium. More specifically, we consider two participants who provide genomic and clinical data, respectively. Our approach is about 2000× faster than the previous solution [Lauter et al., 2014a].

- For Π_{MP}, we present a framework that equipped with three kinds of encodings and three fundamental building blocks for securely outsourcing a various types of statistical analysis over numerical, categorical and ordinal data. Moreover, we present a protocol to privately outsource the decision tree evaluation to the cloud, which is the first construction of outsourcing such functionality privately, to the best of our knowledge.

- For Π_{SMP}, we present a communication efficient secure matrix multiplication protocol. This protocol enables us to conduct privacy-preserving machine learning (PPML) evaluation in a much more efficient manner. For instance, our matrix multiplication protocol can reduce more than 90% computation time and communication overhead over the state-of-the-art PPML approaches [Liu et al., 2017, Mohassel and Zhang, 2017].

Acknowledgments

Firstly, I would like to express my sincere gratitude to my adviser Prof. Jun Sakuma for the continuous support of my master study, for his passion, motivation, and immense knowledge. His guidance helped me in all the time of research and completing of this dissertation. I could not have imagined having a better adviser and mentor for my Phd study.

My sincere thanks also go to Prof. Kunihiro, Prof. Nishide, and Prof. Yadama who gave me profession advises on the cryptographic filed and the bioinformatics field, which have bought tremendous help to my research and to this dissertation.

I thank my fellow labmates, for all your kindness to a foreign student as me, for sleepless nights we were fighting together, and for all the funny we have had in the last three years.

Last but not the least, I would like to thank my parents, for supporting me to come through my study aboard economically and spiritually. I have to admit that without the help from my parents, I could not finish my study.

Contents

List of Tables

List of Figures

Notation

Notation	Description
$\mathbb{Z}_t$	set of integers $[-t/2, t/2) \cap \mathbb{Z}$
$\mathbb{Z}_t[X]/(X^N+1)$	set of polynomials for a 2-power number N
$P[i]$	the i-th coefficient of the polynomial P
$a \xleftarrow{\chi} \mathcal{S}$	sample a from $\mathcal{S}$ following the distribution χ
$\mathcal{I}\{\mathcal{P}\}$	return 1 if the predicate $\mathcal{P}$ is true. Otherwise return 0
$\boldsymbol{v}, \boldsymbol{v}[i]$	vector and its i-th entry
$\boldsymbol{M}, \boldsymbol{M}[i,j]$	matrix and its (i,j)-th entry
$\boldsymbol{M}[i,:], \boldsymbol{M}[:,j]$	the i-th row and the j-th column of matrix
$\langle \boldsymbol{v}, \boldsymbol{u} \rangle$	inner product of vectors
$\boldsymbol{M}\boldsymbol{u}$	product of matrix and vector
$(\mathsf{sk}, \mathsf{pk}, \mathsf{evk})$	decryption, encryption and evaluation key
$\boldsymbol{u} \circ \boldsymbol{v}$	entry-wise multiplication of two vectors
$\boldsymbol{u} \dot{+} \boldsymbol{v}$	entry-wise addition of two vectors
$\lceil r \rfloor$	rouding the real value r to the nearest integer

Chapter 1

Introduction

1.1 Background

Analyzing data collected from various types of sources allows more accurate analysis results and more insightful decision makings. In practices, these data usually come from different parties. For instance, health information exchange networks (e.g., GaHIN [GaH, 2015], NHIN [NHI, 2015]) have been established for improving public health. The government needs to combine tax records with education records to analyze the efficiency of educational investments by linking two and more databases [XRo, 2018]. In addition, two satellite operators belonging to different nations (e.g., the US and Russia) perform collision predictions by exchanging their satellite's status. On the other hand, third-party commercial cloud services such as Amazon Web Service, AliCloud and Azure provide a flexible and economic way to perform data analysis on large scale data. According to a recent survey [Microsoft, 2016], about one third of organizations and companies work with commercial clouds.

On the opposite side, outsourcing data and computation to the commercial cloud might bring potential risks to users' personal information. For example, *Facebook Leak* and *Heartland Payment Systems (HPY)* credit card breach leak out million of personal data and credit card records. These leaks do harm to user's privacy and lead to a great economic loss. Also, data that transferred and stored in the third-party server is out of the user's control. Users is forced to trust cloud service providers not to spy on their private information, and must trust them not to take record even the user want to delete the data. Potential risks of data privacy and control-less issues are becoming the core concerns when using the commercial cloud services. Especially when the data is highly sensitive and confidential, such as genomic data and federal records, it is difficult (even illegal) to outsource data to an outside cloud. It is necessary to develop advanced techniques to protect the confidentiality of data to enable secure analysis on the cloud environment.

Fully homomorphic encryption (FHE) [Gentry, 2009] that allows to perform arithmetic operations above encrypted values directly is one of promising solution for the privacy issue on the cloud environment. By using FHE, data and values can be encrypted before transferring to the cloud, and all computation and evaluation by the cloud are performed over ciphertexts only. This gives a powerful way to protect the privacy data on the cloud. For example, even the database on the cloud is leaked or hacked, only the encrypted data are released. As long as the decryption key is kept secret, the privacy and confidentiality of the leaked information is unscathed.

However, the two main difficulties of deploying FHE to cloud applications are the large computation overhead and communication overhead of FHE.

Large Computation Overhead. From the theoretical point, FHE allows to evaluate an arbitrary Boolean circuit on encrypted data by using a plaintext space $\mathbb{Z}_2$, which requires to encrypt *each bit* of the input data [Gentry et al., 2012a]. Nevertheless, [Gentry et al., 2012b] have shown that the overhead of evaluating a Boolean gate on FHE ciphertexts is a polylog function aspect to the security level, the practical running time of an FHE-based protocol could be too long to be practical. For example, it might took more than 36 hours to evaluate the AES circuit (which is a small size circuit consists of about 3.0×10^4 gates) on FHE ciphertexts [Gentry et al., 2012a]. Even the following optimizations and programming improvements accelerated this AES performance to 7 minutes, the bit-wise encryption manner seems too far from practical for daily applications such as statistical analysis and machine learning which can consist of millions of Boolean gates. In other words, the bit-wise encryption and Boolean circuit manner is so powerful that we can evaluate any functionality on encrypted data, at the cost of a large computation and storage overhead. On the other hand, to have a better performance, we can evaluate Arithmetic circuits instead of evaluating Boolean circuits, i.e., using a prime field $\mathbb{Z}_t$ for a prime $t > 2$. Many approaches belong to on this line such as e.g., Wu and Haven [2012], Lauter et al. [2014a]. However, the efficiency of these approaches are not convincing enough, e.g., it took Wu and Haven [2012] more than 400 minutes to compute the linear regression from encrypted data with only 5 features.

Large Communication Overhead. When developing FHE protocols for the cloud applications, the communication overhead might require more care than the computation overhead because it is much more difficult for the cloud provider to extend its network bandwidth than adding more machines [Pinkas et al., 2014]. However, most of the current FHE protocols does not take count of the communication efficiency. For example, the method of Liu et al. [2017] aims to evaluate a convolutional neural network over encrypted images, but this method needs to exchange more than 9 GB data just for one single 32×32 pixels RGB image.

1.2 Our Contributions

The existing FHE-based applications have their own performance issue. Some of them require too much computing resources (e.g., thousands hours of computation) while some of them introduce a large communication overhead. These issues are not specified to the applications described but also to many other approaches that use FHE. It is commonly thought by the community that FHE is too expensive to use, e.g., requiring many hours to evaluate circuits with a few hundreds of (Boolean or Arithmetic) gates. However, in this dissertation, we will show that FHE protocols with a reasonably small computation overhead and communication overhead are possible as long as we use appropriate message packings (described in the following chapters).

More specifically, we focus on the functionality of inner product of vectors. The inner product functionality can be represented by an Arithmetic circuit. More importantly, it is the fundamental operation of many statistical computations, such as statistic tests, descriptive statistics, predictive statistics and machine learning. In other words, if we want to develop a practical application for such statistical computations using FHE, communication and computation efficient FHE protocols for the inner product functionality are necessary.

In this dissertation, we present three FHE-based primitives, i.e., Π_{IP} (§ 3), Π_{MP} (§ 4) and Π_{SMP} (§ 5), via existing and newly proposed packing techniques. These primitives are delicately designed for computing inner products of encrypted values, but with a different requirement. Using Π_{IP}, we can compute a *single inner product* of long vectors in a very efficient way. On the hand, Π_{MP} enables us to compute *iterative* inner products from multiple vectors, with a larger computation overhead compared with Π_{IP}. For example, products of K matrices (i.e, $K > 2$) can be reduced to iterative inner products. In addition, for applications that require a smaller communication overhead, we present the third primitive Π_{SMP}, which is also friendly for a weak decryptor. Π_{SMP} is useful for client–server applications, in which the network bandwidth is very limited and the client's computing power is usually much weaker than the server.

Moreover, using the proposed three primitives, we present three concrete applications of using FHE on the cloud. In particular, we deal with three different ways of using the cloud which can cover many real applications; that is *cloud-aid data sharing*, *outsourcing computation* and *cloud-based online service*. Cloud-aid data sharing is used when two weak clients want to cooperate together by vertically joining their data. Because the clients are not powerful enough to conduct the computation, they want to use the computing resources of the cloud. On the other hand, outsourcing computation involves more than two participants, and thus is more more general than the cloud-aid data sharing setting. We can use traditional *secure multi-party* (MPC) solutions but they are less practical because MPC solutions need a quadratic number of communications between the participants [Couteau, 2018] . On the other hand, by collecting participants' data to a center server, we can eliminate the communications between the participants. For the third setting, the cloud-based online service runs between a client and a web server (e.g., AliCloud's face recognition

service), leading two kinds of privacy to be considered. One is the client's data privacy (e.g., the client's facial information). The other is the web server's privacy (e.g., the parameters of the classifier of face recognition).

Our contributions include designing and developing efficient protocols for secure computing on untrusted cloud servers via fully homomorphic encryption. In this thesis, we will show four specific protocols for the three settings introduced above.

- For the cloud-aid data sharing setting, we present an efficient FHE-based protocol to conduct Hardy–Weinberg Equilibrium test and Linkage Equilibrium. More specifically, we consider two participants who provide genomic and clinical data, respectively. Our approach is about 2000× faster than the previous solution [Lauter et al., 2014a].

- For the outsourcing computation setting, we present a framework that equipped with three kinds of encodings and three fundamental building blocks for securely outsourcing a various types of statistical analysis over numerical, categorical and ordinal data. Moreover, we present a protocol to privately outsource the decision tree evaluation to the cloud, which is the first construction of outsourcing such functionality privately, to the best of our knowledge.

- For the cloud-based online service setting, we present a communication efficient secure matrix multiplication protocol. This protocol enables us to conduct privacy-preserving machine learning (PPML) evaluation in a much more efficient manner. For instance, our matrix multiplication protocol can reduce more than 90% computation time and communication overhead over the state-of-the-art PPML approaches [Liu et al., 2017, Mohassel and Zhang, 2017].

Chapter 2

Cryptographic Preliminaries

In this chapter, we review some background of cryptographic primitives and privacy-preserving computation under the simulation-based paradigm [Goldreich, 2009].

2.1 (Decision) Ring Learning with Errors

The security of the homomorphic encryption we used is based on the famous Ring Learning with Errors (RLWE) problem [Lyubashevsky et al., 2010]. We give a definition of the decision-RLWE problem here.

Definition 1. *Let N be a power of 2. Let $\mathbb{A} = \mathbb{Z}[X]/(X^N + 1)$, and $\mathbb{A}_t = \mathbb{A}/t\mathbb{A}$ for some prime integer t. Let $\mathfrak{s}$ be a random element in $\mathbb{A}_t$, and let χ_e be a distribution over $\mathbb{A}_t$ obtained by sampling each coefficient of the polynomial from a discrete Gaussian distribution over $\mathbb{Z}$. We write $A_{\mathfrak{s},\chi_e}$ to denote the distribution obtained by sampling $a \xleftarrow{\mathcal{U}} \mathbb{A}_t$ uniformly at random, choosing $e \xleftarrow{\chi_e} \mathbb{A}_t$, and outputting $(a \cdot \mathfrak{s} + e, a)$. Decision-RLWE is the problem of distinguishing between the distribution $A_{\mathfrak{s},\chi_e}$ and the uniform distribution on $\mathbb{A}_t^2$.*

According to [Lyubashevsky et al., 2010], for certain parameters (e.g., N, t, standard deviation of χ_e, and the distribution of $\mathfrak{s}$), the decision-RLWE problem is as hard as solving a certain famous lattice problem (i.e., shortest vector problem) in the worst case.

2.2 BGV's Homomorphic Encryption Scheme

In our research, we use fully homomorphic encryption (FHE) as one of the building blocks. Specifically, we apply BGV's homomorphic encryption scheme [Brakerski et al., 2012] due to the publicly available implementation, i.e., HElib [Halevi and Shoup, 2017]. We give some details of the BGV's scheme.

2.2.1 Distributions

Three distributions are involved in the construction of the BGV's encryption scheme.

- For a real value $\sigma > 0$, χ_e denotes a distribution over $\mathbb{Z}^N$ which picks its elements independently from the discrete zero-mean Gaussian distribution of variance σ^2. Due to the security concerns, σ is normally chosen as 3.2 [Chen et al., 2017].
- $\mathcal{HWT}$ denotes the uniform distribution over the set $\{-1, 0, 1\}^N$ whose Hamming weight is exactly 64.
- $\mathcal{ZO}$ is the distribution over the set $\{-1, 0, 1\}^N$ which draws ± 1 with probability of 0.25, and draws 0 with probability of 0.5.

Also, we write $\mathcal{U}$ to indicate the uniform distribution. For example, $a \xleftarrow{\mathcal{U}} \mathbb{Z}_t$ is sampling uniformly at random from $\mathbb{Z}_t$.

2.2.2 BGV's Homomorphic Encryption Scheme

The BGV scheme is defined over polynomial rings of the form $\mathbb{A} = \mathbb{Z}[X]/(X^N + 1)$ where N is a power of 2. The plaintext space for the scheme the ring is $\mathbb{A}_t := \mathbb{A}/t\mathbb{A}$, namely polynomials modulo $X^N + 1$ and a prime t.

The ciphertext space for this scheme consists of vectors over $\mathbb{A}_q = \mathbb{A}/q\mathbb{A}$, where q is an odd modulus that evolves along with the homomorphic evaluation. Specifically, the scheme is parameterized by a chain of moduli, $q_0 < q_1 < \cdots < q_L$, and a newly encrypted ciphertext is defined over $\mathbb{A}_{q_L}$. Along with homomorphic multiplication, we switch to smaller and smaller moduli before we get to the last level, i.e., $\mathbb{A}_{q_0}$. We designate ciphertexts that are defined over $\mathbb{A}_{q_i}$, "i-th level ciphertexts". These i-th level ciphertexts are size-2 vectors over $\mathbb{A}_{q_i}$, i.e., $\mathfrak{c} = (\mathfrak{c}_0, \mathfrak{c}_1) \in \mathbb{A}_{q_i}^2$.

BGV's homomorphic encryption scheme consists of five algorithms, i.e., KeyGen, Enc, Dec, Add and Mult. We now review these algorithms.

KeyGen: Three keys are generated. The secret key is sampled according to the hamming weight distribution $\mathsf{sk} := (1, \mathfrak{s})$ where $\mathfrak{s} \xleftarrow{\mathcal{HWT}} \{-1, 0, 1\}^N$. The public key is computed as $\mathsf{pk} = (t \cdot e - a \cdot \mathfrak{s}, a)$ where $e \xleftarrow{\chi_e} \mathbb{A}_{q_L}$ and $a \xleftarrow{\mathcal{U}} \mathbb{A}_{q_L}$. The evaluation key is computed as $\mathsf{evk} = (\mathfrak{s}^2 + t \cdot e' - a' \cdot \mathfrak{s}, a')$ where e' and a' follow the same distribution of e and a, respectively. Note that the arithmetic operations here (i.e., $\cdot, +$ and $-$) are performed over the quotient space $\mathbb{A}_{q_L}$.

Enc: Given the plaintext $m \in \mathbb{A}_t$, the L-th level ciphertext is computed as

$$(\mathfrak{c}_0, \mathfrak{c}_1) := u \cdot \mathsf{pk} \dot{+} (m + t \cdot e_1, e_2) \in \mathbb{A}_{q_L}^2$$

where $u \xleftarrow{\mathcal{ZO}} \{-1, 0, 1\}^N$ and $e_1, e_2 \xleftarrow{\chi_e} \mathbb{A}_{q_L}$.

Dec: To decrypt a l-th level ciphertext $(\mathfrak{c}_0, \mathfrak{c}_1) \in \mathbb{A}_{q_l}^2$, we compute $(\mathfrak{c}_0 + \mathfrak{c}_1 \cdot \mathfrak{s} \bmod q_l) \bmod t$.

Add: Homomorphic addition of two l-th level ciphertexts $\mathfrak{c}, \mathfrak{c}' \in \mathbb{A}_{q_i}^2$, are simply adding the components, i.e., $(\mathfrak{c}_0 + \mathfrak{c}'_0, \mathfrak{c}_1 + \mathfrak{c}'_1)$. We designate the operator $\oplus$ as the homomorphic addition.

Mult: Homomorphic multiplication of two l-th level ciphertexts $\mathfrak{c}, \mathfrak{c}' \in \mathbb{A}_{q_i}^2$ are more complicated than the homomorphic addition. We first compute three values $(\mathfrak{d}_0, \mathfrak{d}_1, \mathfrak{d}_2) = (\mathfrak{c}_0 \cdot \mathfrak{c}'_0, \mathfrak{c}_0 \cdot \mathfrak{c}'_1 + \mathfrak{c}_1 \cdot \mathfrak{c}'_0, \mathfrak{c}_1 \cdot \mathfrak{c}'_1)$. Then, we need the evaluation key evk and compute $(\mathfrak{d}'_0, \mathfrak{d}'_1) = (\mathfrak{d}_0, \mathfrak{d}_1) \dotplus \mathfrak{d}_2 \circ \mathsf{evk}$. Finally, we rescale them and convert the level-l ciphertext to the next level $(\lceil \frac{1}{t} \rfloor \cdot \mathfrak{d}'_0, \lceil \frac{1}{t} \rfloor \cdot \mathfrak{d}'_1) \in \mathbb{A}_{q_{l-1}}^2$ via the modulus-switching [Brakerski et al., 2012]. In other words, the BGV's scheme supports maximally a multiplicative depth of L. Even though, with the bootstrapping technique [Halevi and Shoup, 2015], we can evaluate any depth of multiplicative circuit. However, current bootstrapping techniques are still too expensive to use. For most of the applications, the maximum depth L must be decided in advance. We also designate the operator $\otimes$ as the homomorphic multiplication operation.

2.3 Existing Packings

To make homomorphic encryption more practical and useful, one of the most important methodology is applying an appropriate *packing* for the task at hand. Recall that the plaintext element in the BGV's scheme are polynomials in $\mathbb{A}_t$, and homomorphic operations on encrypted values are transferred in the plaintext space as corresponding operations (i.e,. multiplication and addition) in the ring $\mathbb{A}_t$. On the other hand, the user of homomorphic encryption would instead want to operate computation on integers (real numbers). Packing methods are responsible for converting these integers (real numbers) inputs to elements of $\mathbb{A}_t$, and after the homomorphic operations, the packing methods convert the computed results back to the integer (real number) domain.

There are three packing methods [Chen et al., 2017, Yasuda et al., 2013, Smart and Vercauteren, 2014] that encode information into the coefficients of polynomials.

2.3.1 Packing A Single Integer

Suppose that $(a_d a_{d-1} \cdots a_0)_B$ is the B-radix representation of an integer $a \in \mathbb{Z}$, that is $a = \sum_{i=0}^{d} a_i \cdot B^i$. The integer packing π_{int} simply packs the integer a as the polynomial

$$\pi_{\mathsf{int}}(a) = A = \mathrm{sign}(a) \cdot \sum_{i=0}^{d} a_i X^i.$$

As long as $d < N$ and $B < t$, we can see that $A \in \mathbb{A}_t$.

2.3.2 Packing A Single Floating Point Value

The floating point value is given by an integer part and a fractional party, e.g., $5.875 = 5 + 0.875$. Formally, a floating point value $b \in \mathbb{R}$ can be written as

$$b = \underbrace{\sum_{i=0}^{d} b_i \cdot 2^i}_{\text{integer part}} + \underbrace{\sum_{j=1}^{d'} b'_j \cdot 2^{-j}}_{\text{fractional part}}$$

for $b_i, b'_j \in \{0, 1\}$. The fractional packing works as

$$\pi_{\mathsf{frac}}(b) = B := \sum_{i=0}^{d} b_i X^i + \sum_{j=1}^{d'} -b'_j X^{N-j}.$$

As long as $d + d' \leq N$, then $B \in \mathbb{A}_t$.

The plaintext space of the FHE can only handle at most $\log_2 t$ bits of precision. A larger t introduces more noise than a smaller one during the homomorphic operations. As a result, when a large t is used, we might need to set a large N to prevent decryption failure. On the other hand, the integer packing π_{int} and fractional packing π_{frac} enable us to handle values with more than $\log_2 t$ bits of precision. For example, π_{int} allows to encrypt integer in the range of $[0, t^N)$, which is about $N \log_2 t$ bits of precision. Thereby, we can use a relatively smaller (and thus faster) parameter N for the FHE scheme.

2.3.3 Forward Backward Packing

The technique of Yasuda et al. [2013] and Lu et al. [2015] enables efficient private evaluation of inner products. We give a generalization of this technique. Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_t^N$ be vectors of integers. We introduce two functions π_{fwd} and π_{bwd} that convert a vector of integers to a polynomial:

$$\pi_{\mathsf{fwd}}(\boldsymbol{u}) = \sum_{i=0}^{N-1} \boldsymbol{u}[i] \cdot X^i, \quad \pi_{\mathsf{bwd}}(\boldsymbol{v}) = \sum_{j=0}^{N-1} \boldsymbol{v}[j] \cdot X^{N-1-j}. \tag{2.1}$$

Let α be a scalar. We have the following properties.

$$\pi_{\mathsf{fwd}}(\boldsymbol{u}) + \pi_{\mathsf{fwd}}(\boldsymbol{v}) = \pi_{\mathsf{fwd}}(\boldsymbol{u} + \boldsymbol{v}) \qquad \alpha \cdot \pi_{\mathsf{fwd}}(\boldsymbol{u}) = \pi_{\mathsf{fwd}}(\alpha \cdot \boldsymbol{u})$$
$$\pi_{\mathsf{bwd}}(\boldsymbol{u}) + \pi_{\mathsf{bwd}}(\boldsymbol{v}) = \pi_{\mathsf{bwd}}(\boldsymbol{u} + \boldsymbol{v}) \qquad \alpha \cdot \pi_{\mathsf{bwd}}(\boldsymbol{v}) = \pi_{\mathsf{bwd}}(\alpha \cdot \boldsymbol{v}).$$

In other words, we can operate the vector addition and the vector-scalar multiplication with π_{fwd} and π_{bwd}.

2.3.4 Chinese Remainder Theorem Packing

Aside from the forward backward packing, the CRT-packing presented in [Smart and Vercauteren, 2014] is another technique commonly used for developing efficient FHE-based protocols. CRT-packing leverages the polynomial Chinese Remainder Theorem (i.e., CRT) to convert a polynomial vector to an element of $\mathbb{A}_t$.

The plaintext space of the BGV scheme are elements of $\mathbb{A}_t := \mathbb{Z}_t[X]/(X^N+1)$, and the polynomial X^N+1 factors prime t into ℓ irreducible factors, $\{F_j(X)\}_j$

$$X^N + 1 = F_1(X) \cdot F_2(X) \cdots F_\ell(X) \pmod{t},$$

where all the factors of degree $d = N/\ell$. For the purpose of packing, we view a polynomial $A \in \mathbb{A}_t$ not as polynomial of modulus t but as a polynomial over the extension field $\mathbb{F}_{t^d}$. The plaintext values that encoded in A are its evaluations at ℓ (specific) primitive $2N$-th roots of unity in $\mathbb{F}_{t^d}$.

In literature, factors $\{F_j(X)\}$ are called plaintext slots. We write $\pi_{\mathsf{crt}} : (\mathbb{F}_{t^d})^\ell \to \mathbb{A}_t$ to denote the CRT-packing function, and write π_{crt}^{-1} as the reversing function.

The most important property of this packing is element-wise operations. Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be two length-ℓ vectors of *polynomials*, where $\boldsymbol{p}[j], \boldsymbol{q}[j] \in \mathbb{Z}_t[X]$. The element-wise polynomial addition and multiplication are given as

$$\begin{aligned} \pi_{\mathsf{crt}}(\boldsymbol{p}) + \pi_{\mathsf{crt}}(\boldsymbol{q}) &= \pi_{\mathsf{crt}}(\boldsymbol{p} \dot{+} \boldsymbol{q}) \\ \pi_{\mathsf{crt}}(\boldsymbol{p}) \times \pi_{\mathsf{crt}}(\boldsymbol{q}) &= \pi_{\mathsf{crt}}(\boldsymbol{p} \circ \boldsymbol{q}). \end{aligned} \tag{2.2}$$

Beside the element-wise operation, the CRT-packing also enables "slot-movement". As noted in [Gentry et al., 2012b], for each slot index $i, j \in \{1, 2, \cdots, \ell\}$ there is a mapping $\mathcal{M}_k$ parameterized by $k \in \mathbb{Z}_{2N}^*$, $\mathcal{M}_k : A(X) \mapsto A(X^k) \mod (X^N+1)$, which will exchange the element in i-th slot with the element in the j-th slot. By multiplying binary masking vectors, we can achieve cyclic rotation on the ciphertexts of CRT-packed vectors. It is noteworthy that the mapping $\mathcal{M}_k$ changes the decryption key. Suppose the ciphertext $\mathfrak{c}$'s decryption key is $\mathsf{sk} := (1, \mathfrak{s})$. Then, after the application of $\mathcal{M}_k$ on $\mathfrak{c}$, the corresponding decryption key becomes $\mathsf{sk}' := (1, \mathcal{M}_k(\mathfrak{s}))$. Thereby, during the key generation procedure, we need to generate some "key-switching" keys $\mathcal{M}_k(\mathfrak{s}) \to \mathfrak{s}$.

2.4 Security Model

Our protocols are private under the semi-honest assumption. That is the protocol players follow the protocol specification, but might want to learn extra information during the protocol execution. Our security definitions follow the real world/ideal world paradigm of [Canetti, 2000, Goldreich, 2009]. Specifically, we compare the protocol execution in the real world to an execution in an ideal

world. In the real world, protocol players follow the specification of a protocol Π, and in the ideal world, protocol players have access to a trusted third party (TTP) that evaluates the functionality. The protocol execution is viewed as occurring in the existence of an adversary $\mathcal{A}$ and cooperated with an environment $\mathcal{C} = \{\mathcal{C}_\kappa\}$ which is modeled as a class of polynomial-size circuits parameterized by a security parameter κ. The role of the environment is to choose the input to the protocol execution and to distinguish experiments in the real world and the ideal world. We use the notation of privacy [Ishai et al., 2011].

Definition 2. *Let $\mathcal{F} : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ be a deterministic functionality, and $\mathcal{F}_b(x_1, x_2)$ denote the b-th element of $\mathcal{F}(x_1, x_2)$. Let Π be a two-party protocol for implementing $\mathcal{F}$. The view of the b-th party (i.e., $b \in \{1,2\}$) during an execution of Π on (x_1, x_2), written as $\mathcal{V}_b^{\Pi}(x_1, x_2)$, is $(x_b, r, m_1, \cdots, m_T)$, where r denotes the random coin of the b-th party, and m_i represents the i-th message it has received.*

Protocol Π is said to privately implement the functionality $\mathcal{F}(x, y)$ in the semi-honest model if there exist probabilistic polynomial-time algorithms, denoted $\mathcal{S}_1$ and $\mathcal{S}_2$, such that

$$\{\mathcal{S}_1(x_1, \mathcal{F}_1(x_1, x_2))\}_{x_1,x_2\in\{0,1\}^*} \approx^{\mathrm{c}} \{\mathcal{V}_1^{\Pi}(x_1, x_2)\}_{x_1,x_2\in\{0,1\}^*}$$
$$\{\mathcal{S}_2(x_2, \mathcal{F}_2(x_1, x_2))\}_{x_1,x_2\in\{0,1\}^*} \approx^{\mathrm{c}} \{\mathcal{V}_2^{\Pi}(x_1, x_2)\}_{x_1,x_2\in\{0,1\}^*},$$

where $\approx^{\mathrm{c}}$ denotes computational indistinguishability by classes of polynomial-size circuits.

Informally, a protocol Π privately implements a functionality $\mathcal{F}$ if for any polynomial-size circuit, it can not distinguish between the real and ideal world executions.

Chapter 3

Efficient Inner Product of Encrypted Long Vectors

The inner product of vectors is a basic and important computation. For example, the studies of the independence between human genome and disease (e.g., χ^2 statistic tests) involve inner product of long vectors. Also, descriptive statistics such as Fisher discriminant analysis and principal component analysis use inner products. In the context of the secure computation, we can get a direct method of computing inner products using FHE, since what we need to evaluate the inner product are just additions and multiplications. Indeed, the FHE inner product method from Lauter et al. [2014a] is a such direct method, i.e., encrypting the entries of the vectors separately and applying the multiply-then-add computation to achieve the FHE inner product. Thus the complexity of their method is $\mathcal{O}(d)$ aspect to the vector size of d. However, their method would take a long computation time when long vectors are involved. For instance, in genome-wide association studies (GWAS), e.g., Linkage Disequilibrium and Hardy-Weinberg Equilibrium, the vector size equals to the number of patients (or objects), and thus d would be a few thousands [Bos et al., 2014a]. From the experiments in the following section, we will show that Lauter et al. [2014a]'s approach might take days to conduct such genome-wide association studies.

To overcome the high computation overhead of computing inner product of long vectors, in this chapter, we present to pre-preocess the long vectors into polynomials before encryption by expoliting the algebra struture of the plaintext space $\mathbb{A}_t$. As a result, we reduce the complexity of computing the inner product from $\mathcal{O}(d)$ to $\mathcal{O}(d/N)$ where N is usually set as $N > 4000$, which leads to a significant speed up comparing to the previous method. In addition, as the application, we present an FHE protocol for two common genome-wide association studies, i.e., χ^2 test for independence and Linkage Disequilibrium. The basic strategy is to compute allelic frequency tables and genotype frequency tables privately from encrypted genetic data (i.e., in the form of vectors).

Protocol Π_{IP}

Input from encryptor: Vector $\boldsymbol{u} \in \mathbb{Z}_t^d$.
Input from evaluator: Vector $\boldsymbol{v} \in \mathbb{Z}_t^d$.
Output of decryptor: Inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle \bmod t$. **Protocol:**

Encryptor:

1. Encryptor partitions $\boldsymbol{u}$ into N-sized blocks, i.e., $\boldsymbol{u} = \boldsymbol{u}_1 \| \boldsymbol{u}_2 \| \cdots \| \boldsymbol{u}_{d'}$ where $d' = \lceil d/N \rceil$. (The last block might be less than N elements). Encryptor processes each block with the forward packing and then apply encryption. Then encryptor sends the ciphertexts $\{\mathsf{Enc}\left(\pi_{\mathsf{fwd}}(\boldsymbol{u}_i)\right)\}_{i=1}^{d'}$ to evaluator.

Evaluator:

2. Similarly, evaluator also partitions $\boldsymbol{v}$ into N-sized blocks, $\boldsymbol{v} = \boldsymbol{v}_1 \| \boldsymbol{v}_2 \| \cdots \| \boldsymbol{v}_{d'}$. However, evaluator processes those blocks with the backward packing obtain ciphertexts $\{\mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\boldsymbol{v}_i)\right)\}_{i=1}^{d'}$.
3. Evaluator computes and sends the ciphertext $\mathfrak{c}_{\mathsf{final}}$ to decryptor

   $R \oplus (\bigoplus_{i=1}^{d'} \mathsf{Enc}\left(\pi_{\mathsf{fwd}}(\boldsymbol{u}_i)\right) \otimes \mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\boldsymbol{v}_i)\right))$ where $R \xleftarrow{\mathcal{U}} \mathbb{A}_t$ except that the N-th coeff. $R[N-1] = 0$.

Decrytpor:

4. Decrytpor decrypts $\mathfrak{c}_{\mathsf{final}}$ to R' and outputs the N-th coefficient, i.e. $R'[N-1]$.

Figure 3.1: Inner Product Protocol of Encrypted Vectors

With these tables, GWAS-related statistics including D' measure of Linkage Disequilibrium, the Pearson Goodness-of-Fit, HWT, and the χ^2 test can be conducted. We defer the details of these studies in the following subsection. The experimental results show that using our efficient inner product, these two GWAS computations can be 4000× faster than that uses the inner product of Lauter et al. [2014a].

3.1 Efficient Inner Product of Encrypted Long Vectors

We now present our FHE inner product Π_{IP} in Figure 3.1. The protocol Π_{IP} takes as input of ciphertexts of two length-d vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_t^d$, and outputs the ciphertext of their inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle \bmod t$.

Theorem 1. *Protocol Π_{IP} of Figure 3.1 privately implements the inner product functionality, i.e., $\langle \boldsymbol{u}, \boldsymbol{v} \rangle \bmod t$.*

Proof (Correctness). The vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are partitioned into blocks of the same size. We directly have $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_i \langle \boldsymbol{u}_i, \boldsymbol{u}_i \rangle$. For the correctness, it suffices to prove that $\mathsf{Enc}\left(\pi_{\mathsf{fwd}}(\boldsymbol{u}_i)\right) \otimes \mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\boldsymbol{v}_i)\right)$ gives the ciphertext of $\langle \boldsymbol{u}_i, \boldsymbol{u}_i \rangle$ because the summation can be performed by homomorphic additions.

Table 3.1: Timing of fully homomorphic scheme with parameters $N = 8192, t = 640007, L = 6$.

Operation	Encrypt	Mult	Add	Add with Plaintext
Time (ms)	3.08	7.57	0.032	0.789

From the construction of the forward-backward packing, the N-th coefficient of the resulting polynomial $\pi_{\mathsf{fwd}}(\boldsymbol{u}_i) \times \pi_{\mathsf{bwd}}(\boldsymbol{v}_i)$ is $\sum_{j=0}^{N-1} \boldsymbol{u}_i[j] \cdot \boldsymbol{v}_i[j]$ which is exactly the inner product $\langle \boldsymbol{u}_i, \boldsymbol{v}_i \rangle$. □

Proof (Privacy). The privacy against semi-hones encryptor and evaluator is directly given by the semantic security of the underlying homomorphic encryption since encryptor and evaluator can only see encrypted values.

On the other hand, the view of decryptor consists of the polynomial R'. The coefficients of R', except the N-th coefficient, are distributed uniformly at random over $\mathbb{Z}_t$ due to the random polynomial R in Step 3. Thereby, we can simply construct a simulator to simulate the view of decryptor by sampling a uniform random polynomial from $\mathbb{A}_t$. □

Complexity and Parameter Selection. The computation complexity of the evaluator in Π_{IP} is $\mathcal{O}(d/N)$ where d is the size of vectors and N is one of the FHE parameter. Also, according to the security analysis of Gentry et al. [2012a], we need to set $N \geq 8192$ to achieve at least 128-bit security level. In other words, we can conduct the inner product of vectors with more than 8000 entries via just a single homomorphic multiplication, which is a significant boost comparing to the naive solution (see the next subsection).

Limitation. The N-th coefficient of the resulting polynomial $\pi_{\mathsf{fwd}}(\boldsymbol{u}_i) \times \pi_{\mathsf{bwd}}(\boldsymbol{v}_i)$ does give the inner product of $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. On the other hand, the other $N-1$ coefficients are randomized, and thus we barely re-use these $N-1$ coefficients in any further computation. These $N-1$ randomized coefficients are considered as *noisy terms*.

3.1.1 Comparison to the Naive (non-packing) Approach

Table 3.1 summarizes the computing time of each homomorphic operation under a specific FHE parameter. By using an appropriate packing, we can see that the inner product of two encrypted vectors can be much more efficient than the naive non-packing approach, according to the numbers of Table 3.1. For example, the naive unpacking method to compute the inner product of vectors of length 8192 would need 8192 homomorphic multiplications and additions, which would take about 62 seconds. On the other hand, using the forward packing packing, it would only take one homomorphic multiplication and addition, and thus taking less than 8.0 ms, to compute the inner product of 8192 elements. It is more than a 8000× boost under this FHE parameter.

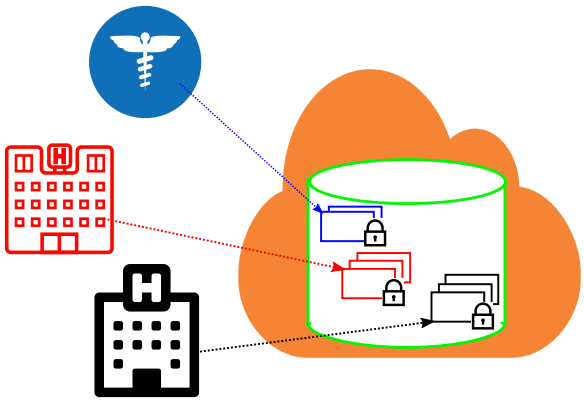


Figure 3.2: Joining encrypted genotype and phenotype data cross multiple data contributors.

3.2 Application: Secure Genome-wide Association Studies on Cloud

The proposed protocol Π_{IP} can be used in many cloud-aid computation scenario. As an example, we give an application of privacy-preserving χ^2 statistic test and linkage disequilibrium (LD) independence test, which are two commonly used statistics in genome-wide association studies. By using our inner product protocol, we can efficiently compute χ^2 and LD from a large (encrypted) genomic and clinical data on the cloud. Empirical results show that, our approach is about $2000\times$ faster than the previous cryptographic solution [Lauter et al., 2014b]. In this section, we give an application of Π_{IP}: secure genome-wide association studies (GWAS). More specifically, we target on two commonly used statistical tests, i.e., χ^2 test and linkage disequilibrium test.

Because of recent advances in DNA sequencing technologies, the cost of DNA sequencers is dropping rapidly. As a result, the scale of genomic data used by researchers is becoming larger and larger. To conduct computations on a large-scale genomic dataset, a cloud server that provides computational resources at low cost is regarded as a promising option. Genomic and clinical data are highly sensitive. Outsourcing these data to an external server raises concerns about the privacy of sensitive data. Consequently, for outsourcing of computation with genomic data, privacy should be rigorously preserved.

Related Work. FHE has been used to protect to genomic and clinical data in the cloud. Bos et al. [2014b] proposed a working implementation of cloud service for private computation of encrypted health data using FHE. Lauter et al. [2014a] demonstrated an approach to conducting private computation using encrypted genomic data with FHE (Figure 3.2). Unfortunately, these cryptographic solutions are not sufficiently time and space efficient to conduct a GWAS-scale computation, which can involve about 3×10^6 SNPs for thousands or more subjects.

Our contribution includes a much more efficient cryptographic protocol for χ^2 and linkage disequilibrium test. Empirical results show that, our approach is about 2000× faster than the previous cryptographic solution [Lauter et al., 2014b].

3.2.1 Problem Statements: χ^2 Test and Linkage Disequilibrium Test

Table 3.2: Examples of raw genome data and raw phenotype data.

(a) Raw genome data D^g

ID	Genomic Data
1	CC CG CT GG AA
2	AG CT CT AG CT
3	CT GG CC AG AA
4	AA GG GG AG CC

(b) Raw phenotype data D^p

ID′	Disease Status
1	Case
2	Control
3	Control
4	Case

Our basic strategy is to compute allelic frequency tables and genotype frequency tables privately from encrypted genetic data. With these tables, GWAS-related statistics including D' measure of LD, the Pearson Goodness-of-Fit, HWE, and the χ^2 test are conducted. In this work particularly, we apply our method to the χ^2 test and LD to demonstrate the effectiveness of our protocol.

We review an allelic frequency table and a genotype frequency table with two markers. Table 3.2a gives a view of a genomic dataset D^g. Each record contains an explicit identifier ID and SNPs. Similarly, Table 3.2b gives a view of a phenotype dataset D^p. Each record contains an explicit identifier ID′ to identify each subject and an attribute to indicate the disease status of the subject. Presuming that M subjects and N SNPs are involved, then the dataset D^g contains N rows, with each row containing M data points; the dataset D^p includes M rows.

Table 3.3: Observed allele frequency in a case–control study of M subjects.

	Allele Type		total
	A	a	
case	o_1	o_2	N_1
control	o_3	o_4	N_2
total	N_1'	N_2'	$2M$

Presuming that A, a are possible alleles. An allelic frequency table (Table 3.3) consists of 2×2

Table 3.4: Genotype frequencies at markers M_1 and M_2 of M subjects.

		Marker M_1			Total
		AA	Aa	aa	
Marker M_2	BB	o_{11}	o_{12}	o_{13}	N_1
	Bb	o_{21}	o_{22}	o_{23}	N_2
	bb	o_{31}	o_{32}	o_{33}	N_3
	Total	N_1'	N_2'	N_3'	$2M$

counts

$$o_1 = 2N_{\mathrm{AA}}^{\mathrm{case}} + N_{\mathrm{Aa}}^{\mathrm{case}} \qquad o_2 = 2N_{\mathrm{aa}}^{\mathrm{case}} + N_{\mathrm{Aa}}^{\mathrm{case}}$$
$$o_3 = 2N_{\mathrm{AA}}^{\mathrm{control}} + N_{\mathrm{Aa}}^{\mathrm{control}} \qquad o_4 = 2N_{\mathrm{aa}}^{\mathrm{control}} + N_{\mathrm{Aa}}^{\mathrm{control}},$$

where $N_{\mathrm{AA}}^{\mathrm{case}}$ and $N_{\mathrm{Aa}}^{\mathrm{case}}$ are the observed population counts for genotype AA and Aa in the case group: $N_{\mathrm{AA}}^{\mathrm{control}}$ and $N_{\mathrm{Aa}}^{\mathrm{control}}$ are the observed counts for the control group.

The χ^2 test for the additive model is equivalent to the χ^2 test based on Table 3.3. The one degree of freedom (d.f.) test statistic is written as

$$\chi_a^2 = \frac{2M(o_2(o_3+o_4) - o_4(o_1+o_2))^2}{N_1 N_2 N_1' N_2'}.$$

In addition to a χ^2 test, we can evaluate the Hardy–Weinberg Equilibrium directly from an allelic frequency table similarly.

Given alleles (A/a and B/b) at two markers, a genotype frequency table (Table 3.4) with two markers is obtained that consists of 3×3 counts

$$o_{11} = N_{\mathrm{AABB}} \qquad o_{12} = N_{\mathrm{AaBB}} \qquad o_{13} = N_{\mathrm{aaBB}}$$
$$o_{21} = N_{\mathrm{AABb}} \qquad o_{22} = N_{\mathrm{AaBb}} \qquad o_{23} = N_{\mathrm{aaBb}}$$
$$o_{31} = N_{\mathrm{AAbb}} \qquad o_{32} = N_{\mathrm{Aabb}} \qquad o_{33} = N_{\mathrm{aabb}}.$$

The value $N_{ii'jj'}$ denotes the observed population counts for genotype ii' and jj' where $i, i' \in \{A, a\}$, and $j, j' \in \{B, b\}$.

We evaluate LD from Table 3.4. The linkage disequilibrium is calculated as $D = \Pr(\mathrm{AB}) - \Pr(\mathrm{A})\Pr(\mathrm{B})$, where probabilities Pr(AB), Pr(A) and Pr(B) are computed, respectively, as $(2o_{11} + o_{12} + o_{21})/2M$, $(2N_1' + N_2' - o_{22})/2M$ and $(2N_1 + N_2 - o_{22})/2M$. We omit the frequency o_{22} to avoid the problem of haplotype ambiguity, especially when only genotypes are measured. See [Ziegler and König, 2010] for more details.

We remark that several measures for measuring linkage disequilibrium were proposed, including Pearson's correlation, Lewontin's D', frequency difference and Yule's Q. Our proposal works for all these measures. However, we applied our method to Lewontin's D' measure in the experimentation because of space limitations. Additional details related to these measurements are explained in an earlier report of the literature [Ziegler and König, 2010].

3.2.2 Data Encoding

Let A and a be the alleles of the biallelic locus. Consequently, the genomic data at the locus is either $AA, Aa,$ or aa. We represent each row of the genomic dataset D^g as two integer vectors $\boldsymbol{x}^{AA}, \boldsymbol{x}^{Aa}$. Here, $\boldsymbol{x}^{AA}[i]$, the i-th element of $\boldsymbol{x}^{AA}$, represents the frequency of genotype AA at the marker locus: $\boldsymbol{x}^{AA}[i] = 2$ for AA and $\boldsymbol{x}^{AA}[i] = 0$ for other genotypes. $\boldsymbol{x}^{Aa}[i]$ is similar to $\boldsymbol{x}^{AA}[i]$ except that $\boldsymbol{x}^{Aa}[i] = 1$ for Aa.

We presume that the disease status of each subject is represented by a binary variable, then "disease" is represented by 1 (case); "non-disease" is represented by 0 (control). The phenotype dataset D^p for all subjects is therefore represented by a binary vector $\boldsymbol{y}^{\text{case}}$.

Presume in addition to the following that dataset D^g consists of N SNPs with M subjects. Q data contributors are involved in the procedure. Therefore, they separately hold the phenotype vector $\boldsymbol{y}^{\text{case}}$ and $2N$ genotype vectors $\boldsymbol{x}^{\text{AA}}_{(i)}$ and $\boldsymbol{x}^{\text{Aa}}_{(i)}$, where (i) is the ID of the genotype data. Let $\pi : \{0,1,2\}^M \times \{1,2,\cdots,Q\} \mapsto \{0,1,2\}^M$ be an assignment function that represents the partition of genotype/phenotype held by the q-th data contributor. For example, the vertical partition of a vector $\boldsymbol{x}$ for the q-th data contributor is represented as shown below.

$$\pi(\boldsymbol{x}, q)[j] = \begin{cases} \boldsymbol{x}[j] & \text{if } q\text{-th data contributor holds the } j\text{-th element of } \boldsymbol{x} \\ 0 & \text{o.w.} \end{cases}.$$

We assume that each element of vectors is contributed from only one data contributor, i.e. $\sum_q \pi(\boldsymbol{x}, q)[j] = \boldsymbol{x}[j]$ holds for every j.

3.2.3 Evaluate the Allelic Frequency Table

With the encoding described, we evaluate Table 3.3 through *scalar products* of the representing vectors. More specifically, frequencies o_1, N'_2, and N_1 in Table 3.3 are evaluated respectively through three scalar products as

$$o_1 = \left\langle \boldsymbol{x}^{AA} + \boldsymbol{x}^{Aa}, \boldsymbol{y}^{\text{case}} \right\rangle, \; N'_1 = \left\langle \boldsymbol{x}^{AA} + \boldsymbol{x}^{Aa}, \mathbf{1} \right\rangle, \; N_1 = \left\langle \boldsymbol{y}^{case}, \mathbf{1} \right\rangle,$$

where $\mathbf{1}$ is a vector of which the elements are 1. Because Table 3.3 is freedom-1 and the number of objects M is assumed to be known,

3.2.4 Evaluate the Genotype Frequency Table

Similarly, we compute the genotype frequency table described by Table 3.4 with two markers by scalar products of the represented vectors as well. In particular, to calculate a D'-measure for the LD, the following six scalar products are needed.

$$\begin{aligned} 4 \cdot o_{11} &= \left\langle \boldsymbol{x}^{AA}, \boldsymbol{x}^{BB} \right\rangle, & 2 \cdot o_{12} &= \left\langle \boldsymbol{x}^{Aa}, \boldsymbol{x}^{BB} \right\rangle, \\ 2 \cdot o_{21} &= \left\langle \boldsymbol{x}^{AA}, \boldsymbol{x}^{Bb} \right\rangle, & o_{22} &= \left\langle \boldsymbol{x}^{Aa}, \boldsymbol{x}^{Bb} \right\rangle, \\ 2N_1' + N_2' &= \left\langle \boldsymbol{x}^{AA} + \boldsymbol{x}^{Aa}, \mathbf{1} \right\rangle, & 2N_1 + N_2 &= \left\langle \boldsymbol{x}^{BB} + \boldsymbol{x}^{Bb}, \mathbf{1} \right\rangle. \end{aligned}$$

3.2.5 Full Protocol

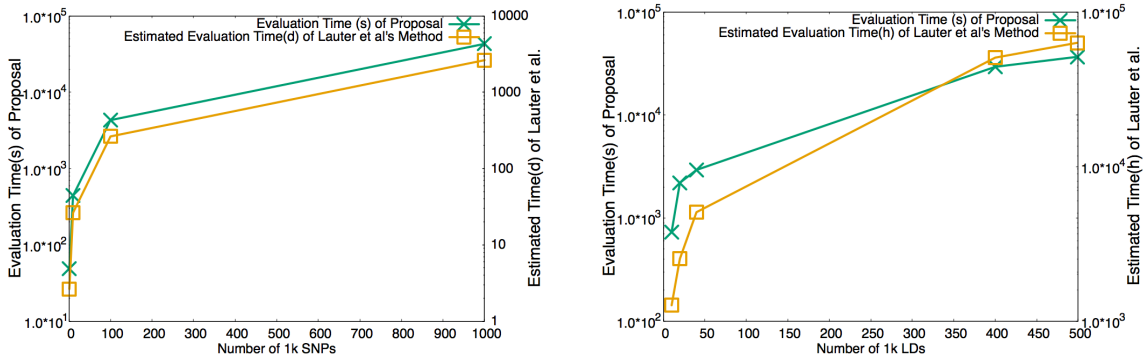
The procedure of secure outsourcing χ^2 and LD to the cloud is shown in Figure 3.4. Recall that the evaluation of scalar product requires a forward-packed vector and a backward-packed vector. Consequently, at Step 1 and Step 2, data contributors upload four copies for one genotype data in the form of the forward-packed and backward-packed vectors. The cloud aggregates the collected ciphertexts at Step 3, which only involves homomorphic additions. Then the cloud computes the allelic frequency table and the genotype frequency table respectively at Step 4 and Step 5.

The correctness of Figure 3.4 is directly given by the description in § 3.2.3 and § 3.2.4. Moreover, the privacy of Figure 3.4 can be reduced trivially to the privacy of our inner product protocol of Figure 3.1. Intuitively, the view of the cloud during the protocol execution of Figure 3.4 consists of only ciphertexts. Thus, we omit the form security proof of Figure 3.4.

3.2.6 Evaluations

We benchmarked the computational costs of our method and compared it with a method proposed by Lauter et al. [2014a], in which a genetic data point and a clinical data point are encoded respectively into three bits and two bits. All experiments were conducted on computers with a 2.60 GHz CPU (Xeon; Intel Corp.) and 32 GB RAM. We measured the computation time separately for Step 1.1 and 1.2 as the preparation time and for Steps 3.1 and 3.2 as the evaluation time. Details of the experiment settings are presented following. 1) An artificial dataset includes 1.0×10^4 subjects. 2) $Q = 5$ data contributors are sharing same quantity of data points. 3) We used 8 threads for computation in parallel. 4) Parameters of the encryption scheme were set as $N = 8192, t = 640007$, and $L = 6$.

Performance of Homomorphic Encryption and Implementation Hints. The implementation of Lauter et al. was done on an algebraic computation system, i.e. Magma, whereas our implementation was developed on native codes. To compare our method with their method fairly, we measured the computation time of operations in HElib and re-estimated the computation time method of Lauter et al. Table 3.1 shows the computation time of the operations of homomorphic



(a) Benchmark for outsourcing χ^2 test with 1.0×10^4 subjects. (b) Benchmark for outsourcing LD with 1.0×10^4 subjects.

Figure 3.3: Comparison of the proposed secure outsourcing χ^2 test and LD with Lauter et al. [2014a].

encryption scheme. Values are the mean of 1000 runs of each operation with 8-threads. We used parameter $N = 8192$, which is not sufficiently large to conduct more than 8192 subjects. Indeed, we partitioned vectors into smaller parts and encrypted each part as a ciphertext. In doing so, we were able to conduct a large-scale dataset while maintaining smaller N. We remark that as the number of the partition increases, more communication time must be used during the upload phase.

Evaluation Results: χ^2 Test. We benchmarked our proposed protocol of evaluating χ^2 test on an artificial dataset that contains $M = 1.0\times10^4$ subjects. The results are presented in Figure 3.3a. The number of the total genotype data was varied from 1.0×10^3 to 1.0×10^6. Recalling that parameter $N = 8192$, one can thereby maximally pack genotype/phenotype data of 8192 subjects into a single ciphertext. Consequently, to conduct the experiment with 1.0×10^4 subjects, we partitioned a vector into two parts having equal length. Figure 3.3a depicts the performance of our proposed method and the estimated computation time of the method of Lauter et al. [2014a]. As shown in Figure 3.3a, for evaluation of χ^2 test statistics of 1.0×10^6 genotype data with $M = 1.0 \times 10^4$ subjects, our method took about 12 hours (about 43ms per test). Compared to the method of Lauter et al., the evaluation is expected to cost more than 2000 days. Our proposal is therefore considerably efficient.

Evaluation Results: Linkage Disequilibrium. The benchmark of the evaluation of LD is presented in Figure 3.3b. In this experiment, we considered a smaller synthesis data containing 1.0×10^3 genotype data of $M = 1.0 \times 10^4$ subjects. The number of LD to be evaluated was about 5.0×10^5 LDs in this experiment. With this settings, our method costs less than 11 hours (about 80 ms per LD). We consider that result as great increase in speed compared with the method of Lauter et al. that might cost more than 2600 days to finish the evaluation in estimation.

Protocol Full Protocol for Secure Outsourcing χ^2 Test and LD Test.

Public known: All data contributor can access to the encryption key pk. The number of subjects M is also publicly known.
Private inputs: The q-th data contributor has input its clinical data $\pi(\boldsymbol{y}^{\text{case}}, q)$. For genotype data with ID(i), the q-th data contributor hash input $\pi(\boldsymbol{x}^{\mathcal{A}}_{(i)}, q)$ for $\mathcal{A} \in \{AA, Aa, aa\}$. The analyst has input the decryption key sk.
Outputs: The analyst obtains the χ^2 statistics or LD.
Protocol:

1. *Upload Phenotype Data:* The q-th data contributor computes and sends ciphertext(s)
$$\mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\pi(\boldsymbol{y}^{\text{case}}, q))\right)$$
to the cloud.
2. *Upload Genotype Data:* For genotype data with ID(i), the q-th data contributor computes and sends ciphertexts
$$\mathsf{Enc}\left(\pi_{\mathsf{fwd}}(\pi(\boldsymbol{x}^{\mathcal{A}}_{(i)}, q))\right) \quad \mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\pi(\boldsymbol{x}^{\mathcal{A}}_{(i)}, q))\right)$$
for $\mathcal{A} \in \{AA, Aa\}$.
3. *Join:* For genotype data with ID(i), the cloud joins the collected ciphertexts
$$\hat{\mathfrak{e}}_{\boldsymbol{x}^{\mathcal{A}}_{(i)}} := \bigoplus_{q=1}^{Q} \mathsf{Enc}\left(\pi_{\mathsf{fwd}}(\pi(\boldsymbol{x}^{\mathcal{A}}_{(i)}, q))\right) \quad \mathfrak{e}_{\boldsymbol{x}^{\mathcal{A}}_{(i)}} := \bigoplus_{q=1}^{Q} \mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\pi(\boldsymbol{x}^{\mathcal{A}}_{(i)}, q))\right).$$
The cloud also joins the ciphertext of phenotype data as $\mathfrak{e}_{\boldsymbol{y}^{\text{case}}} := \oplus_{q=1}^{Q} \mathsf{Enc}\left(\pi_{\mathsf{bwd}}(\pi(\boldsymbol{y}^{\text{case}}, q))\right)$ and prepares two plain values: $\pi_{\mathsf{fwd}}(\mathbf{1})$ and $\pi_{\mathsf{bwd}}(\mathbf{1})$.
4. *Evaluate χ^2 Statistics:* Frequency N_1 in Table 3.3 is homomorphically computed as $\mathfrak{e}_{N_1} = \mathfrak{e}_{\boldsymbol{y}^{\text{case}}} \otimes \pi_{\mathsf{fwd}}(\mathbf{1})$. For genotype data with ID(i), the cloud homomorphically compute sufficient statistics in Table 3.3 as
$$\mathfrak{e}_{o_1} = (\hat{\mathfrak{e}}_{\boldsymbol{x}^{AA}_{(i)}} \oplus \hat{\mathfrak{e}}_{\boldsymbol{x}^{Aa}_{(i)}}) \otimes \mathfrak{e}_{\boldsymbol{y}^{\text{case}}} \quad \mathfrak{e}_{N'_1} = (\hat{\mathfrak{e}}_{\boldsymbol{x}^{AA}_{(i)}} \oplus \hat{\mathfrak{e}}_{\boldsymbol{x}^{Aa}_{(i)}}) \otimes \pi_{\mathsf{bwd}}(\mathbf{1})$$

5.1. *Evaluate LD:* Given two genotype ID(i) and ID(j), the cloud computes six frequencies in Table 3.4.
$$\begin{aligned} \mathfrak{e}_{4o_{11}} &= \hat{\mathfrak{e}}_{\boldsymbol{x}^{AA}_{(i)}} \otimes \mathfrak{e}_{\boldsymbol{x}^{AA}_{(j)}} \quad \mathfrak{e}_{2o_{12}} = \hat{\mathfrak{e}}_{\boldsymbol{x}^{Aa}_{(i)}} \otimes \mathfrak{e}_{\boldsymbol{x}^{AA}_{(j)}} \\ \mathfrak{e}_{2o_{21}} &= \hat{\mathfrak{e}}_{\boldsymbol{x}^{AA}_{(i)}} \otimes \mathfrak{e}_{\boldsymbol{x}^{Aa}_{(j)}} \quad \mathfrak{e}_{o_{22}} = \hat{\mathfrak{e}}_{\boldsymbol{x}^{Aa}_{(i)}} \otimes \mathfrak{e}_{\boldsymbol{x}^{Aa}_{(j)}} \\ \mathfrak{e}_{2N'_1+N'_2} &= (\mathfrak{e}_{\boldsymbol{x}^{AA}_{(i)}} \oplus \mathfrak{e}_{\boldsymbol{x}^{Aa}_{(i)}}) \otimes \pi_{\mathsf{fwd}}(\mathbf{1}) \\ \mathfrak{e}_{2N_1+N_2} &= (\mathfrak{e}_{\boldsymbol{x}^{AA}_{(j)}} \oplus \mathfrak{e}_{\boldsymbol{x}^{Aa}_{(j)}}) \otimes \pi_{\mathsf{fwd}}(\mathbf{1}) \end{aligned}$$

5.2. From these six frequencies, the cloud can further compute the necessary values for D'-measure.
$$\mathfrak{e}_{\Pr(A)} := \mathfrak{e}_{2N'_1+N_2} - \mathfrak{e}_{o_{22}} \quad \mathfrak{e}_{\Pr(B)} := \mathfrak{e}_{2N_1+N_2} - \mathfrak{e}_{o_{22}}$$
$$\mathfrak{e}_{2\Pr(AB)} := \mathfrak{e}_{4o_{11}} \oplus \mathfrak{e}_{2o_{12}} \oplus \mathfrak{e}_{2o_{21}}$$

6. *Query χ^2 Test:* The cloud answers the χ^2 query from the analyst and sends ciphertexts $\mathfrak{e}_{N_1}$, $\mathfrak{e}_{o_1}$ and $\mathfrak{e}_{N'_1}$ to the analyst. Then the analyst can reconstruct the allelic frequency table, i.e., Table 3.3.
7. *Query LD:* The cloud answers the LD query from the analyst and sends ciphertexts $\mathfrak{e}_{2\Pr(AB)}$, $\mathfrak{e}_{\Pr(A)}$ and $\mathfrak{e}_{\Pr(B)}$ to the analyst. Then the analyst can reconstruct the allelic frequency table, i.e., Table 3.4.

Figure 3.4: Full Protocol for Secure Outsourcing χ^2 Test and LD Test.

Chapter 4

Iterative Computation on Encrypted Matrices

Given a matrix $\boldsymbol{A}$, we can use the protocol Π_{IP} described in the previous chapter to compute the multiplication $\boldsymbol{A}^2$ privately since the matrix multiplication can be reduced to inner products. However, we can not use Π_{IP} for a higher degree, i.e., $\boldsymbol{A}^k$ for $k > 2$ due to the error terms introduced by the forward backward packing. On the other hand, the functionality $\boldsymbol{A} \mapsto \boldsymbol{A}^k$ for $k > 2$ is a very important operations for many applications such as statistical analysis (e.g., linear regression) and machine learning (e.g. collaborative filtering). If each entry of the matrix is encrypted separately as Wu and Haven [2012], it is trivial to perform the iterative multiplications on the encrypted matrices. Obviously, the method of Wu and Haven [2012] requires $\mathcal{O}(d^3)$ number of homomorphic operations aspect to the matrix dimension d.

In this chapter, we present a protocol Π_{MP} to compute iterative multiplications of encrypted matrices. The key point of our iterative matrix multiplication protocool is to use the CRT packing in a layout consistent way. Take the row-major layout as the example, that is each row of the matrix is separately CRT-packed. The method of Halevi and Shoup [2014] uses CRT packing, however, it outputs the resulting matrix in a column-major layout. To perform iterative multiplications, time comsuming layout adjustments are needed. On the other hand, our iterative matrix multiplication protocol takes as input of row-majorly encrypted matrices, and outputs the resulting matrix in the row-major layout directly without any further layout adjustment. Overall, our iterative matrix multiplication protocol needs $\mathcal{O}(d^3/\ell)$ homomorphic operations, which is ℓ times faster than the baseline method of Wu and Haven [2012].

4.1 Primitives: Homomorphic Rotation and Homomorphic Replication

We leverage the CRT-packing with a well-tuned homomorphic rotation operation in Π_{MP}. In addition to the 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.

$$\pi_{\mathsf{crt}}^{-1}(\mathsf{Rotate}(\pi_{\mathsf{crt}}(\boldsymbol{x}), k)) = \boldsymbol{u} \in \mathbb{Z}_t^{\ell}$$
$$\forall 1 \leq j \leq \ell \ \boldsymbol{u}[j] = \boldsymbol{x}[j + k \bmod \ell].$$

In other words, we can homomorphically rotate the encryted vector by the offset of k.

Also, from the rotation operation, we can derive a replicate operation $\mathsf{Replicate} : \mathbb{A}_t \times \mathbb{Z} \mapsto \mathbb{A}_t$ directly using $\mathcal{O}(\log \ell)$ rotations.

$$\pi_{\mathsf{crt}}^{-1}(\mathsf{Replicate}(\pi_{\mathsf{crt}}(\boldsymbol{x}), k)) = \boldsymbol{v} \in \mathbb{Z}_t^{\ell},$$
$$\forall 1 \leq j \leq \ell \ \boldsymbol{v}[j] = \boldsymbol{x}[k].$$

That is, we can pick a specific element (i.e., the k-th entry) from the encryted vector, and homomorphically propagate it to the other positions.

4.2 Proposal: Iterative Multiplication of Encrypted Matrices

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 $\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{A}[i,:])\right)\}_{i=1}^{d}$ and $\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{B}[i,:])\right)\}_{i=1}^{d}$ to denote the ciphertexts of each row of $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{Z}_t^{d \times d}$, respectively. The ciphertext of a vector $\boldsymbol{u} \in \mathbb{Z}_t^{d}$ is written as $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{u})\right)$.

Now, we present our iterative matrix multiplication protocol Π_{MP} in Figure 4.1.. To keep the layout consistent, we use the $\mathsf{Replicate}$ function. The following example demonstrates the idea

Protocol Π_{MP}

Inputs: Ciphertext of the matrix $\boldsymbol{A} \in \mathbb{Z}_t^{\ell \times \ell}$, i.e., $\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{A}[i,:])\right)\}_i$ for $1 \leq i \leq \ell$. The exponent $k > 1$.
Outputs: Ciphertext of the matrix exponent $\boldsymbol{A}^k$, i.e., $\left\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{A}^k[i,:])\right)\right\}_i$.
Protocol:

1. We replicate the slots of each row ciphertexts of $\boldsymbol{A}$ and obtain ℓ^2 ciphertexts, i.e.,

$$\mathfrak{a}_{i,j} = \mathsf{Replicate}(\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{A}[i,:])\right), j) \text{ for } 1 \leq i, j \leq \ell.$$

2. Let designate $\mathfrak{c}_i^{(1)} := \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{A}[i,:])\right)$ for $1 \leq i \leq \ell$.
3. Iterate in range $2 \leq k' \leq k$ and compute

$$\mathfrak{c}_i^{(k')} = \bigoplus_{j=1}^{\ell} \left(\mathfrak{c}_i^{(k'-1)} \otimes \mathfrak{a}_{j,i}\right) \text{ for } 1 \leq i \leq \ell.$$

4. Output ℓ ciphertexts $\{\mathfrak{c}_i^{(k)}\}_{i=1}^{\ell}$ which decrypts to $\boldsymbol{A}^k[i,:]$, respectively.

Figure 4.1: Iterative Matrix Multiplication of Encrypted Matrices.

behinds Π_{MP}.

$$\underbrace{\begin{bmatrix}[1,2]\\ [3,4]\end{bmatrix}}_{\boldsymbol{A}} \cdot \underbrace{\begin{bmatrix}[e,f]\\ [g,h]\end{bmatrix}}_{\boldsymbol{B}} = \begin{bmatrix}\overbrace{[1,1]\circ[e,f] \dot{+} [2,2]\circ[g,h]}^{j=1}\\ \underbrace{[3,3]\circ[e,f] \dot{+} [4,4]\circ[g,h]}_{j=2}\end{bmatrix}.$$

Basically, we replicate each element of the left-hand-side matrix $\boldsymbol{A}$ and perform the element-wise additions and multiplications. The resulting matrix is also in the row-major layout.

Theorem 2. *(Correctness.) The protocol of Figure 4.1 correctly implements the functionality of iterative matrix multiplication, i.e., $\boldsymbol{A}^k$.*

Proof. We prove via mathematical induction. It is clear that for $k = 1$, Figure 4.1 works correctly. Suppose that $\mathfrak{c}_i^{(k')}$ decrypts to the i-th row of $\boldsymbol{A}^{k'}$. We prove that $\mathfrak{c}_i^{(k'+1)}$ decrypts to the i-th row of $\boldsymbol{A}^{k'+1}$. The ciphertext $\mathfrak{a}_{i,j}$ in Step 1 of Figure 4.1 decrypts to the (i,j) entry of $\boldsymbol{A}$. Thereby, the Step 3 is homomorphically computing $\sum_j \boldsymbol{A}^{k'}[i,:] \cdot \boldsymbol{A}[j,i]$, which is exactly the i-th row of $\boldsymbol{A}^{k'+1}$. □

Matrix–vector Multiplication. Halevi and Shoup [2014] 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 $\boldsymbol{Au}$ as follows

$$\bigoplus_{i=1}^{d} \mathsf{Enc}\left(\boldsymbol{A}[i,:]\right) \otimes \mathsf{Replicate}(\mathsf{Enc}\left(\boldsymbol{u}\right), i). \tag{4.1}$$

Notice that, this results at a ciphertext of CRT packed vector. Thereby, this matrix–vector multiplication can be used to compute iterative multiplications.

The security analysis of Figure 4.1 is deferred to the last section of this chapter.

4.3 Proposal: Batch Greater-than

We describe the batch greater-than protocol in Figure 4.2 which are used in the next application section. Given integers $0 \leq a, b < D$ for some positive D, we know that $a > b$ if and only if $\exists 1 \leq w < D$ 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 where the random value r is used to hide $|a - b|$. This straw-man protocol, thus, requires $\mathcal{O}(D)$ homomorphic operations and generates $\mathcal{O}(D)$ ciphertexts. 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 π_{crt} 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 $\mathcal{O}(\lceil D/\ell \rceil)$ homomorphic operations and generates $\mathcal{O}(\lceil D/\ell \rceil)$ ciphertexts which is a considerable improvement for a large ℓ.

Theorem 3. *(Correctness.) The protocol of Figure 4.2 correctly implements the batch greater than functionality $\mathcal{I}\{a_i > b_i\}$ for $1 \leq i \leq \theta$ under the semi-honest setting.*

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 the next section, we use the bGT as an intermediate step to obliviously zero-out some rare values in a contingency table.

Summary of Complexity. The complexity of the proposed primitives and basic routines are summarized in Table 4.1. We count the number of homomorphic operations used in each primitive.

4.3.1 Comparison with the Garbled Circuit-based Solutions

We empirically compared the performance of the matrix multiplication protocol Π_{MP} and batch greater-than protocol Π_{bGT} with their garbled circuit counterpart implementations. The comparison

Protocol Batch Greater-than Protocol Π_{bGT}

Inputs: Ciphertext of the integer vector $\boldsymbol{a}, \boldsymbol{b} \in \{0, 1, \cdots, D-1\}^{\theta}$ for $D, \theta \in \mathbb{Z}^{+}$.
Output: Ciphertext of the CRT-packed vector $\boldsymbol{\gamma}$ where the length of $\boldsymbol{\gamma}$ is $\theta \cdot D$.
Remark: One can learn $\mathcal{I}\{\boldsymbol{a}[j] > \boldsymbol{b}[j]\} = \mathcal{I}\{0 \in \{\boldsymbol{\gamma}[k \cdot \theta + j]\}_{k=0}^{D-1}\}$ from $\boldsymbol{\gamma}$.

1. Compute $\tilde{\mathfrak{a}} = \mathsf{Repeat}(\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{a})\right), \theta, D)$; $\tilde{\mathfrak{b}} = \mathsf{Repeat}(\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{b})\right), \theta, D)$.
2. Generate random permutations $\pi_j : \{0, 1, \cdots, D-1\} \rightarrow \{0, 1, \cdots, D-1\}$ for $0 \leq j < \theta$.
3. Compute a $\theta \cdot D$ dimension vector $\boldsymbol{w}$ in which $\boldsymbol{w}[\alpha(j)] = \pi_j(\alpha)$. Here $\alpha(j) := \theta \cdot \alpha + j$, for $\alpha \in \{0, 1, \cdots, D-1\}$ and $0 \leq j < \theta$.
4. Compute $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\beta})\right) = \tilde{\mathfrak{a}} - \tilde{\mathfrak{b}} - \pi_{\mathsf{crt}}(\boldsymbol{w})$.
5. Compute $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\gamma})\right) = \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\beta})\right) \otimes \pi_{\mathsf{crt}}(\boldsymbol{r})$ where $\boldsymbol{r} \xleftarrow{\mathcal{U}} (\mathbb{Z}_t/\{0\})^{\theta \cdot D}$.
6. Output $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\gamma})\right)$.

Figure 4.2: Batch Greater-than Protocol

Sub-Protocol Repeat

Inputs: Ciphertext of CRT packed vector $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{u})\right)$ and $\theta, R > 0$.
Output: Ciphertext of the repeated vectors $\tilde{\boldsymbol{u}}$ which repeats the first θ elements of $\boldsymbol{u}$ for R times.

1. $\tilde{\mathfrak{u}} = \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{u})\right) \otimes \pi_{\mathsf{crt}}([\underbrace{1 \ldots 1}_{\theta} 00 \ldots])$; $\mathfrak{u} = \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\tilde{\boldsymbol{u}})\right)$.
2. View R in the binary format $R = (b_\rho \cdots b_1 b_0)_2$ where b_ρ is the most significant bit.
3. For $0 \leq i \leq \rho$ iterate as follows
   a) If b_i is 1 then $\tilde{\mathfrak{u}} = \mathsf{Rotate}(\tilde{\mathfrak{u}}, k)$.
   b) $\tilde{\mathfrak{u}} = \tilde{\mathfrak{u}} \oplus \mathfrak{u}$.
   c) $\mathfrak{u} = \mathfrak{u} \oplus \mathsf{Rotate}(\mathfrak{u}, k)$
   d) $k = k \times 2$.
4. return $\tilde{\mathfrak{u}}$

Figure 4.3: Repeat Sub-Protocol.

results are shown in Figure 4.4. We first describe our experiment setup.

GC Setting. For GC, we used a state-of-the-art framework, i.e., ObliVM [Liu et al., 2015] 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.

Table 4.1: Complexity of our primitives. d is the matrix dimension. ℓ denotes the number of slots of the CRT packing. θ is the batch-size of the greater than protocol. We write "–" to indicate that the homomorphic operation is not used.

	addition	multiplication	rotation
$\boldsymbol{X} \cdot \boldsymbol{u}$ (Figure 4.1)	$\mathcal{O}(d^2/\ell)$	$\mathcal{O}(d^2/\ell)$	$\mathcal{O}(\frac{d^2 \log \ell}{\ell})$
$\boldsymbol{X} + \boldsymbol{Y}$	$\mathcal{O}(d^2\ell)$	–	–
$\boldsymbol{X} \cdot \boldsymbol{Y}$ (Figure 4.1)	$\mathcal{O}(d^3/\ell)$	$\mathcal{O}(d^3/\ell)$	$\mathcal{O}(\frac{d^3 \log \ell}{\ell})$
bGT (Figure 4.2)	$\mathcal{O}(\lceil(\theta D)/\ell\rceil)$	$\mathcal{O}(\lceil(\theta D)/\ell\rceil)$	$\mathcal{O}(\log D)$

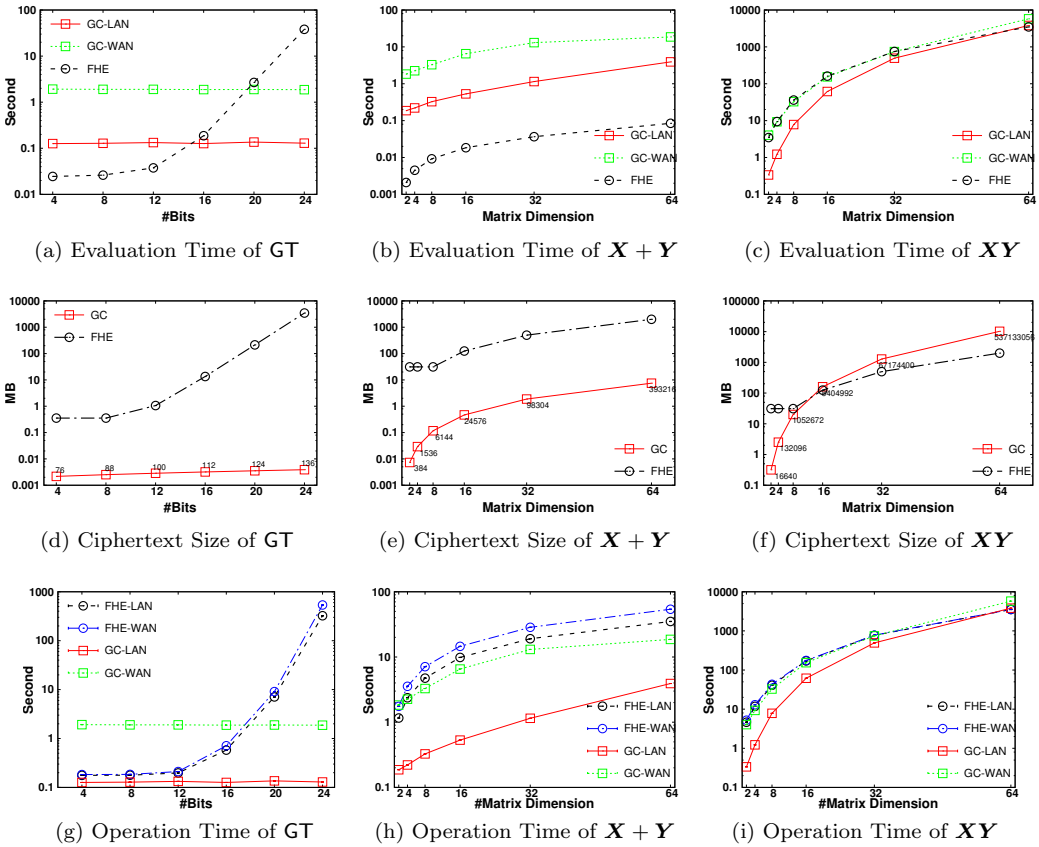
In ObliVM, we used the real-mode which provides the garbled-row-reduction [Naor et al., 1999] and free-XOR [Kolesnikov and Schneider, 2008] optimizations.

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 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



(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 $\boldsymbol{XY}$

Figure 4.4: 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.

network packets exchanged during the execution of the GC-based primitives [1].

Evaluation Result: Greater-than. Figure 4.4a, Figure 4.4d, and Figure 4.4g 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 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., $0 \leq age \leq 150$), we consider $12 - 16$-bits to be sufficient to meet regular requirements in many cases.

Evaluation Result: Matrix Addition. Figure 4.4b, Figure 4.4e, and Figure 4.4h show the

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

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 (Figure 4.4b). 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.

Evaluation Result: Matrix Multiplication. Figure 4.4c, Figure 4.4f, and Figure 4.4i 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. 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.

4.4 Application: Secure Outsourcing Statistical Analysis

At a high level, FHE enables us to perform addition and multiplication on ciphertexts. Thus it allows us to evaluate any function $\mathcal{F}$ 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 $\mathcal{F}$ 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 Naehrig et al. [2011], Yasuda et al. [2013], and Lu et al. [2015], 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.

In this chapter, 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, using the proposed primitives in the previous section. 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.

Related Work. 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 [Naehrig et al., 2011]. In [Wu and Haven, 2012], they also show how to compute the co-variance using FHE. Notice that these statistics involve numerical attributes

only, while in the statistical analysis we also have categorical and ordinal data. For categorical and ordinal statistics, such as histogram and k-percentile, we can use the private database query system of [Boneh et al., 2013]. However, this method requires to evaluate a circuit with $\mathcal{O}(M)$ multiplicative depths where M is the number of data points. Thereby, it would be practical only for a very small M, e.g., $M < 20$.

For predictive statistics, the earlier study [Graepel et al., 2012] 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 [Bost et al., 2015] 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 [Bost et al., 2015]. In [Wu and Haven, 2012], they present a protocol to train a linear regression model from FHE encrypted data using Cramer's rule for matrix inversion. Thereby, the computational complexity of this method blows up factorially with the data dimension. In other words, their method is only suitable for data with small dimensions. Indeed, only six dimension data are used in [Wu and Haven, 2012]. To the best of our knowledge, no practical FHE solution that trains the linear regression model from data with high dimension has been established.

4.4.1 Data Representations: Numerical, Categorical and Ordinal Data

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

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

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 $\boldsymbol{c}_i \in \mathcal{C}$ be a vector of the categorical data. Then $\boldsymbol{c}_i[j] \in \mathcal{C}_j$ 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

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

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 $\boldsymbol{o}_i \in \mathcal{O}$ be the i-th ordinal data. Then $\boldsymbol{o}_i[j]$ 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 $\Delta \in \mathbb{Z}$, we have $\lfloor \Delta x \rceil \in \mathbb{Z}$ 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 M categorical, ordinal, and numerical data points respectively as follows.

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

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 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.

4.4.2 Application Scenario and Stakeholders

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 4.2). The encryptor sends the ciphertext $\mathsf{Enc}(x)$ to the cloud for the computation of a particular functionality $\mathcal{F}$. The cloud operates specified homomorphic operations on $\mathsf{Enc}(x)$ and sends the resulting ciphertext $\mathsf{Enc}(z = \mathcal{F}(x))$ 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 4.3. In the following protocol descriptions, we thus omit the encryption phase of the encryptor.

4.4.3 Problem Statements: Descriptive Statistics and Predictive Statistics

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

Table 4.2: Input-output relationships for the stakeholders. We write "–" to indicate no input or output. Input and output are viewed as bits stream $x, z \in \{0,1\}^*$.

Stakeholder	Possess	Input	Output
encryptor	$\mathsf{pk}, \mathsf{evk}$	x	–
cloud	$\mathsf{pk}, \mathsf{evk}$	–	$\mathsf{Enc}(z)$
decryptor	$\mathsf{pk}, \mathsf{evk}, \mathsf{sk}$	–	z

Table 4.3: A summary of the form of ciphertexts and statistics

Data Type	Ciphertext Form	Statistics
$\boldsymbol{c}_i[q] \in \mathcal{C}_q$	$\mathsf{Enc}(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q]))$	histogram, count, histogram order and contingency table
$\boldsymbol{o}_i[p] \in \mathcal{O}_p$	$\mathsf{Enc}(\mathcal{E}_{\mathrm{st}}(\boldsymbol{o}_i[p]))$	k-percentile
$\boldsymbol{x}_i \in \mathbb{Z}^{d_c}$	$\mathsf{Enc}(\pi_{\mathsf{crt}}(\boldsymbol{x}_i))$	PCA and linear regression

attributes; and the principal component analysis and linear regression for numerical attributes. We present these statistics in turn.

4.4.4 Descriptive Statistics

Single Categorical Attribute. Let $\{\boldsymbol{c}_1[j], \ldots, \boldsymbol{c}_M[j]\}$ be the j-th categorical attribute values of M data points. If c_{ij}s are encoded by the indicator encoding, then the summation of vectors yields the histogram.

$$\mathsf{Hist}(\{\boldsymbol{c}_1[j], \ldots, \boldsymbol{c}_M[j]\}) = \boldsymbol{h} \text{ where } \boldsymbol{h} = \sum_{i=1}^{M} \mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[j]). \tag{4.2}$$

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

$$\mathsf{Count}(\{\boldsymbol{c}_1[j], \ldots, \boldsymbol{c}_M[j]\}, p) = \langle \mathbf{1}_p, \boldsymbol{h} \rangle, \tag{4.3}$$

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

$$\mathsf{HistOrder}(\{c_{1j}, \ldots, c_{Mj}\}) = \boldsymbol{k}, \tag{4.4}$$

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

Table 4.4: A contingency table of two categorical attributes $\mathcal{C}_p$ and $\mathcal{C}_q$ of M data points.

state	s^q_1	$\cdots$	$s^q_{\lvert\mathcal{C}_q\rvert}$	Total
s^p_1	$\mu_{1,1}$	$\cdots$	$\mu_{1,\lvert\mathcal{C}_q\rvert}$	μ'_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$s^p_{\lvert\mathcal{C}_p\rvert}$	$\mu_{1,\lvert\mathcal{C}_p\rvert}$	$\cdots$	$\mu_{\lvert\mathcal{C}_p\rvert,\lvert\mathcal{C}_q\rvert}$	$\mu'_{\lvert\mathcal{C}_p\rvert}$
Total	μ_1	$\cdots$	$\mu_{\lvert\mathcal{C}_q\rvert}$	M

(s^p_u, s^q_v). For instance, one categorical data point $\boldsymbol{c}_i = [\cdots, s^p_2, \cdots, s^q_3, \cdots]$ 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 Table 4.4. We define the functionality of contingency table evaluation as

$$\mathsf{ContingencyTable}(\{\boldsymbol{c}_i[p], \boldsymbol{c}_i[q]\}_{i=1}^M) = \boldsymbol{\mu}. \tag{4.5}$$

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* [Nabar and Mishra, 2009, Kirkendall and Sande, 1998] 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

$$\mathsf{CT\text{-}Suppression}(\{\boldsymbol{c}_i[p], \boldsymbol{c}_i[q]\}_{i=1}^M, \mathcal{T}) = \bar{\boldsymbol{\mu}}, \tag{4.6}$$

where $\bar{\mu}_s = \mu_s \cdot \mathcal{I}\{\mu_s > \mathcal{T}\}$ for $1 \le s \le |\mathcal{C}_p||\mathcal{C}_q|$. Notice that $\boldsymbol{\mu}$ is the output of $\mathsf{ContingencyTable}$.

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 $\{\boldsymbol{o}_1[j], \ldots, \boldsymbol{o}_M[j]\}$ be the j-th ordinal attribute values of M data points, we can sort the ordinal values in ascending order as $\boldsymbol{o}_{\pi(1)}[j] \preceq \cdots \preceq \boldsymbol{o}_{\pi(M)}[j]$. 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

$$k\text{-}\mathsf{Percentile}(\boldsymbol{o}_1[j], \ldots, \boldsymbol{o}_M[j]) = \boldsymbol{o}_{M^*}[j], \tag{4.7}$$

where $M^* := \pi(\lceil (k \cdot M)/100 \rceil)$ and $\boldsymbol{o}_{\pi(i)}[j] \preceq \boldsymbol{o}_{\pi(i+1)}[j]$ holds for all $1 \le i < M$.

4.4.5 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

$$\boldsymbol{\Sigma} = \frac{1}{M}\boldsymbol{X}^\top \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{\mu}^\top \text{ where } \boldsymbol{\mu} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{x}_i^\top. \tag{4.8}$$

Then we compute the eigenvalues and eigenvectors of $\boldsymbol{\Sigma}$. Let the eigenvalues of $\boldsymbol{\Sigma}$ be $\lambda_1 \geq \cdots \geq \lambda_{d_n}$, and denote the corresponding eigenvectors as $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{d_n}$. An iterative algorithm (i.e., Power-Method) 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 $\boldsymbol{v}^{(0)} \xleftarrow{\mathcal{U}} \mathbb{Z}_t^{d_n}$.
3. For $0 \leq \tau < T$, compute
$$\boldsymbol{v}^{(\tau+1)} = \boldsymbol{\Sigma}_k \boldsymbol{v}^{(\tau)}. \tag{4.9}$$
4. Output $\boldsymbol{u}_k = \dfrac{\boldsymbol{v}^{(T)}}{\|\boldsymbol{v}^{(T)}\|}$ and $\lambda_k = \dfrac{\|\boldsymbol{v}^{(T)}\|}{\|\boldsymbol{v}^{(T-1)}\|}$.

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 $\{(\boldsymbol{x}_i^\top, y_i)\}_{i=1}^{M}$ 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:

$$\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}} \frac{1}{M}\sum_{i=1}^{M} \|y_i - \boldsymbol{x}_i^\top \boldsymbol{w}\|_2^2.$$

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

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{4.10}$$

where the matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$ are the collections of numerical data. The Equation 4.10 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 [Guo and Higham, 2006] 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 $0 \leq \tau < T$, compute

$$\begin{aligned} \boldsymbol{R}^{(\tau+1)} &= 2\alpha^{(\tau)}\boldsymbol{R}^{(\tau)} - \boldsymbol{R}^{(\tau)}\boldsymbol{A}^{(\tau)}, \\ \boldsymbol{A}^{(\tau+1)} &= 2\alpha^{(\tau)}\boldsymbol{A}^{(\tau)} - \boldsymbol{A}^{(\tau)}\boldsymbol{A}^{(\tau)}, \\ \alpha^{(\tau+1)} &= \alpha^{(\tau)}\alpha^{(\tau)}. \end{aligned} \tag{4.11}$$

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 Guo and Higham [2006], $\boldsymbol{R}^{(\tau)}$ converges to $\lambda^{2^\tau}\boldsymbol{M}^{-1}$ quadratically if λ is close to the largest eigenvalue of $\boldsymbol{M}$.

4.4.6 Computing Descriptive Statistics from Ciphertexts

This section presents the details of evaluating the statistics described in Section 4.4.3 on FHE encrypted data.

Histogram and Count. The evaluations of Equation 4.2 (histogram) and Equation 4.3 (count) on FHE encrypted categorical data are straightforward using the CRT-packing and indicator encoding. For the collection of categorical data $\boldsymbol{C} \in \mathcal{C}^M$, we can compute the histogram of $\mathcal{C}_p$, i.e. the p-th attribute, as $\oplus_{i=1}^{M} \mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])\right)$. Also, we can compute the histograms of multiple attributes simultaneously. For instance $\oplus_{i=1}^{M} \mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p]) \| \mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q])\right)$ 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, $\left(\oplus_{i=1}^{M} \mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])\right)\right) \otimes \pi_{\mathsf{crt}}(\mathbf{1}_3)$ 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 Equation 4.4 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. In the protocol, the matrix $\boldsymbol{\Delta}$ just acts as a handy helper for us to calculate the histogram order.

Theorem 4. *(Correctness.) The* $\Pi_{\mathsf{HistOrder}}$ *protocol correctly implements the histogram order functionality of Equation 4.4 under the semi-honest setting.*

Proof. The protocol $\Pi_{\mathsf{HistOrder}}$ of Figure 4.5 calls the bGT primitive $\mathcal{O}(|\mathcal{C}_j|^2)$ times. By operating these comparisons, we have obtained the order of the values of the histogram. According to bGT, if

Protocol $\Pi_{\mathsf{HistOrder}}$

Input from encryptors: Ciphertexts of categorical data of the j-th attribute $\{\mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[j])\right)\}_{i=1}^{M}$
Output of the cloud: Ciphertexts $\{\mathsf{Enc}\left(\boldsymbol{\gamma}_{uv}\right)\}_{1\le u<v\le|\mathcal{C}_j|}$
Remark: Decryptor can learn the histogram order Equation 4.2 from $\boldsymbol{\gamma}_{uv}$.

The cloud:

1. Computes the histogram $\mathsf{Enc}\left(\boldsymbol{h}\right) = \bigoplus_{i=1}^{M} \mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[j])\right)$.
2. Computes $\mathsf{Enc}\left(h_p\right) = \mathsf{Replicate}(\mathsf{Enc}\left(\boldsymbol{h}\right), p)$ for $1 \le p \le |\mathcal{C}_j|$.
3. For all $1 \le u < v \le |\mathcal{C}_j|$ pairs, invokes the routine of Figure 4.2 with $D = M$ and $\theta = 1$, and obtains
$$\mathsf{Enc}\left(\boldsymbol{\gamma}_{uv}\right) = \mathsf{bGT}(\mathsf{Enc}\left(\boldsymbol{h}[u]\right), \mathsf{Enc}\left(\boldsymbol{h}[v]\right)).$$
4. Outputs ciphertexts $\{\mathsf{Enc}\left(\boldsymbol{\gamma}_{uv}\right)\}_{1\le u<v\le|\mathcal{C}_j|}$.

The decryptor:

5. Constructs a matrix $\boldsymbol{\Delta} \in \{0,1\}^{|\mathcal{C}_j|\times|\mathcal{C}_j|}$ according to the *decryption* of $\{\mathsf{Enc}\left(\boldsymbol{\gamma}_{uv}\right)\}_{1\le u<v\le|\mathcal{C}_j|}$.
   a) The diagonal of $\boldsymbol{\Delta}$ is set to 0, that is $\boldsymbol{\Delta}[u,u] = 0$.
   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 $\boldsymbol{\Delta}[v,u] = 1 - \boldsymbol{\Delta}[u,v]$.
6. Outputs a vector $\boldsymbol{k}$ with the value $\boldsymbol{k}[l]$ set as the row-index of $\boldsymbol{\Delta}$ which contains exactly $|\mathcal{C}_j| - l$ of 1s for $1 \le l \le |\mathcal{C}_j|$.

Figure 4.5: Protocol $\Pi_{\mathsf{HistOrder}}$ for Private Histogram Order.

$0 \in \boldsymbol{\gamma}_{uv}$ holds then we know that the count of state s_u^j is larger than that of state s_v^j, which follows exactly the histogram order functionality. □

K-percentile. We conduct the k-percentile of the attribute $\mathcal{O}_j$ on FHE ciphertexts as Π_{kP} of Figure 4.6. The decryptor can derive the k-percentile of the attribute $\mathcal{O}_j$ from $\boldsymbol{\gamma}$. Indeed, we obtain the *cumulative frequencies* of the j-th ordinal data $\{\boldsymbol{o}_1[j], \ldots, \boldsymbol{o}_M[j]\}$ in Step 1 due to the use of staircase encoding. For instance, let us consider the ordinal data $\{\hat{s}_1^j, \hat{s}_2^j, \hat{s}_3^j, \hat{s}_3^j, \hat{s}_1^j, \hat{s}_2^j\}$ for $M = 6$. 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 $kM/100$. In the previous example, we know $\hat{s}_2^j$ is the 50-percentile point because $f_1 < 3 \wedge f_2 \ge 3$.

Theorem 5. *(Correctness.) The protocol Π_{kP} of Figure 4.6 correctly implements the k-percentile functionality of Equation 4.7 under the semi-honest setting.*

Proof. The Step 1 of Figure 4.6 gives the ciphertext of cumulative frequencies of the j-th ordinal attribute. If $\hat{s}_{n^*}^j$ is the k-percentile, Step 2 gives a ciphertext of a vector $\boldsymbol{\gamma}$ such that $0 \notin \{\gamma_{k|\mathcal{O}_j|+n^*-1}\}_{k=0}^{M-1}$ while $0 \in \{\gamma_{k|\mathcal{O}_j|+n^*}\}_{k=0}^{M-1}$, since the input to the bGT in Step 2 is the (encrypted) cumulative frequencies of the j-th ordinal attribute. 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 the last attribute state $\hat{s}_{|\mathcal{O}_j|}^j$ is the k-percentile of the population. □

Protocol Π_{kP}

Input from encryptors: Ciphertexts of j-th ordinal attribute. $\{\mathsf{Enc}\left(\mathcal{E}_{\mathrm{st}}(\boldsymbol{o}_i[j])\right)\}_{i=1}^{M}$. Positive integer $0 \leq k \leq 100$.
Output of the cloud: Ciphertexts of a vector $\boldsymbol{\gamma}$ where $|\boldsymbol{\gamma}| = M \cdot |\mathcal{O}_j|$.
Remark: The decryptor can learn k-percentile of the j-th ordinal attribute from $\boldsymbol{\gamma}$.

The cloud:

1. Aggregates the ciphertexts via additions $\mathfrak{c} = \bigoplus_{i=1}^{M} \mathsf{Enc}\left(\mathcal{E}_{\mathrm{st}}(\boldsymbol{o}_i[j])\right)$.
2. Invokes bGT on the aggregated ciphertext:

$$\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\gamma})\right) = \mathsf{bGT}(\mathfrak{c}, M \cdot k/100)$$

   with $D = M$ and $\theta = |\mathcal{O}_j|$ as the parameters of the bGT protocol.
3. Outputs $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\gamma})\right)$ to the decryptor.

The decryptor:

4. Decrypts $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{\gamma})\right)$ as $\boldsymbol{\gamma}$.
5. Finds out an index $1 \leq n^* \leq |\mathcal{O}_j|$ 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. The k-percentile of the j-th ordinal attribute is output as $\hat{s}^j_{n^*}$.

Figure 4.6: Protocol Π_{kP} for Private k-percentile.

	$\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])$		$\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])$		$\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])$	
element-wise multi.	$\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q])$			$\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q])$		
contribute to	$\mu_{1,1}$	$\mu_{2,2}$	–	$\mu_{2,1}$	$\mu_{1,2}$	–

Figure 4.7: One multiplication gives 2×2 combinations of attributes of $\boldsymbol{c}_i[p] \in \mathcal{C}_p$ and $\boldsymbol{c}_i[q] \in \mathcal{C}_q$ where $|\mathcal{C}_p| = 2$ and $|\mathcal{C}_q| = 2$.

Contingency Table with Cell Suppression. The count $\mu_{u,v}$ in the contingency table (i.e., Table 4.4) 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 Figure 4.7, in which the domain sizes are $|\mathcal{C}_p| = |\mathcal{C}_q| = 2$ and $k_1 = 2, k_2 = 3$. In Figure 4.7, the white cells 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. The co-prime duplication plays a major role in the above evaluation.

Theorem 6. *(Correctness.) The protocol of Figure 4.8 correctly implements the functionality of Equation 4.6 (contingency table with suppression) under the semi-honest model.*

[2]Indices start from 1.

Protocol Π_{CTS}

Inputs: Ciphertexts of the p-th and q-th categorical data $\{\mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])\right), \mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q])\right)\}_{i=1}^{M}$. The cell suppression threshold $\mathcal{T} > 0$. Let $\Sigma := |\mathcal{C}_p| \cdot |\mathcal{C}_q|$ as the product of the size of the categorical attributes.
Outputs: Ciphertexts $\mathfrak{a}', \mathfrak{b}'$ and $\mathfrak{b}^*$.
Remarks: One can learn the cell suppressed contingency table defined in Equation 4.6 from the decryptions of $\mathfrak{a}', \mathfrak{b}'$ and $\mathfrak{b}^*$.

The cloud:

1. Finds the smallest co-prime integers k_1 and k_2 such that $k_1 \geq |\mathcal{C}_p|$ and $k_2 \geq |\mathcal{C}_q|$.
2. For $1 \leq i \leq M$, computes
$$\mathfrak{p}_i = \mathsf{Repeat}\left(\mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[p])\right), k_1, k_2\right)$$
$$\mathfrak{q}_i = \mathsf{Repeat}(\mathsf{Enc}\left(\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[q])\right), k_2, k_1).$$
3. Computes $\mathfrak{a} = \bigoplus_{i=1}^{M} \mathfrak{p}_i \otimes \mathfrak{q}_i$.
4. Invokes bGT: $\mathfrak{b} = \mathsf{bGT}(\mathfrak{a}, \mathcal{T})$ with $D = M$ and $\theta = \Sigma$ for the bGT protocol.
5. Computes $\mathfrak{b}' = \mathfrak{b} \oplus \pi_{\mathsf{crt}}(\boldsymbol{\delta})$ where $\boldsymbol{\delta} \xleftarrow{\mathcal{U}} \mathbb{Z}_t^{\Sigma}$.
6. Computes $\mathfrak{a}' = \mathfrak{a} \oplus \pi_{\mathsf{crt}}(\boldsymbol{r})$ where the length of the vector $\boldsymbol{r}$ is $M \cdot \Sigma$, $\boldsymbol{r}[k \cdot \Sigma + x] = \boldsymbol{\delta}[x]$ for $0 \leq k < N$, and $1 \leq x \leq \Sigma$.
7. Samples $\boldsymbol{r}^* \xleftarrow{\mathcal{U}} (\mathbb{Z}_t/\{0\})^{M \cdot \Sigma}$ and computes $\mathfrak{b}^* = \mathfrak{b} \otimes \pi_{\mathsf{crt}}(\boldsymbol{r}^*)$.
8. Outputs ciphertexts $\mathfrak{a}', \mathfrak{b}'$ and $\mathfrak{b}^*$.

The decryptor:

9. Decrypts $\mathfrak{a}', \mathfrak{b}'$ and $\mathfrak{b}^*$ to $\boldsymbol{a}', \boldsymbol{b}'$ and $\boldsymbol{b}^* \in \mathbb{Z}_t^{M \cdot \Sigma}$, respectively.
10. Finds out the set $\mathcal{SZ} := \{(s, z = k \cdot \Sigma + s) | \boldsymbol{b}^*[k \cdot \Sigma + s] = 0, 1 \leq s \leq \Sigma, 0 \leq k < N\}$.
11. Outputs $\hat{\boldsymbol{\mu}}$ where $\hat{\boldsymbol{\mu}}[s] = \boldsymbol{b}'[z] - \boldsymbol{a}'[z]$ for $(s, z) \in \mathcal{SZ}$.

Figure 4.8: Protocol Π_{CTS} for Private Contingency Table with Cell Suppression.

Proof. (Correctness.) We now describe the idea of our cell suppression protocol Π_{CTS} of Figure 4.8. Without loss of generality, we presume that $\boldsymbol{\mu}[s] > \mathcal{T}$ for some specific $1 \leq s \leq \Sigma$. According to the bGT protocol of Figure 4.2, we have *one and only one* 0 in the set $\Gamma_s := \{\boldsymbol{b}[k \cdot \Sigma + s]\}_{k=0}^{M-1}$ (where $\boldsymbol{b}$ is the decryption of $\mathfrak{b}$ in Step 4 of Figure 4.8). This enables us to hide numbers. If we have only one 0 in Γ_s, we can recover the value of $\boldsymbol{\mu}[s]$ from the tuple $\{\boldsymbol{\mu}[s] + \delta, \delta \dot{+} \Gamma_s, r^* \cdot \Gamma_s\}$ with some non-zero random value r^*. Here the mathematical operations are carried out on each element of Γ_s.

On the other hand, if $\boldsymbol{\mu}[s] \leq \mathcal{T}$, after the execution of the bGT protocol we have $0 \notin \Gamma_s$. We can not recover the value of $\boldsymbol{\mu}_s$ from the tuple. Thereby, the suppression is achieved. □

4.4.7 Computing Predictive Statistics from Ciphertexts

Principal Component Analysis. For the evaluation of PCA, we can perform the computation of Equation 4.8 and Equation 4.9 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 the protocol Π_{PCA} of Figure 4.9. Step 1 and Step 2 follow Equation 4.8 except we can not perform the division on ciphertexts. Notice that, in Step 2, the operation generates ciphertexts of a matrix, i.e., $M^2\boldsymbol{\Sigma}$. The evaluation in Step 3 is also straightforward using our matrix–vector multiplication primitives of Equation 4.1.

Protocol Π_{PCA}

Inputs: Ciphertexts of the numerical data $\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{x}_i^\top)\right), \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{x}_i\boldsymbol{x}_i^\top)\right)\}_{i=1}^M$. Number of iterations T.
Outputs: Ciphertexts $\mathfrak{v}^{(T)}$ and $\mathfrak{v}^{(T-1)}$.
Remarks: One can learn the (approximated) largest eigenvalue and the associated eigenvector from the decryptions of $\mathfrak{v}^{(T)}$ and $\mathfrak{v}^{(T-1)}$.

The cloud:

1. Computes $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(M \cdot \boldsymbol{\mu})\right) = \bigoplus_{i=1}^M \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{x}_i^\top)\right)$.
2. Computes $\mathsf{Enc}\left(\pi_{\mathsf{crt}}(M^2\boldsymbol{\Sigma})\right) = \left(M \otimes \bigoplus_{i=1}^M \mathsf{Enc}\left(\boldsymbol{x}_i\boldsymbol{x}_i^\top\right)\right) \ominus \mathsf{Enc}\left(\pi_{\mathsf{crt}}(M \cdot \boldsymbol{\mu})\right) \otimes \mathsf{Enc}\left(\pi_{\mathsf{crt}}(M \cdot \boldsymbol{\mu}^\top)\right)$.
3. From $0 \le \tau < T$, iterates
$$\mathfrak{v}^{(\tau+1)} = \mathsf{Enc}\left(\pi_{\mathsf{crt}}(M^2\boldsymbol{\Sigma})\right) \otimes \mathfrak{v}^{(\tau)},$$
where $\mathfrak{v}^{(0)}$ is initialized as any non-zero vector.
4. Outputs $\mathfrak{v}^{(T)}$ and $\mathfrak{v}^{(T-1)}$.

The decryptor:

5. Decrypts $\mathfrak{v}^{(T)}$ and $\mathfrak{v}^{(T-1)}$ to vectors $\boldsymbol{v}^{(T)}$ and $\boldsymbol{v}^{(T-1)}$, respectively. Outputs the largest eigenvalue as $\lambda_1 = \|\boldsymbol{v}^{(T)}\|/\|\boldsymbol{v}^{(T-1)}\|$ and the associated eigenvector as $\boldsymbol{u}_1 = \boldsymbol{v}^{(T)}/\|\boldsymbol{v}^{(T)}\|$.

Figure 4.9: Protocol Π_{PCA} for Private Principal Component Analysis.

Theorem 7. *(Correctness.) The protocol Π_{PCA} of Figure 4.9 correctly implements the functionality of Equation 4.9 with approximations under the semi-honest model.*

Linear Regression. To conduct the linear regression of Equation 4.10, 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 Equation 4.11. The evaluation of Equation 4.11 on ciphertexts are straightforward using our matrix multiplication primitive of Figure 4.1. Given the collection of numerical data $\{(\boldsymbol{x}_i^\top, y_i)\}_{i=1}^M$ and the largest eigenvalue λ_1 of the design matrix, we can evaluate Equation 4.10 with T iterations as the protocol of Figure 4.10.

The DF-MatrixInversion in Step 3 computes the matrix inversion with a known factor $\lambda_1^{2^T}$. Thereby, the output ciphertext $\mathfrak{w}$ is the linear regression model $\boldsymbol{w}^*$ argumened by the factor $\lambda_1^{2^T}$.

Protocol Π_{LR}

Inputs: Ciphertexts of the numerical data $\{\mathsf{Enc}\left(\pi_{\mathsf{crt}}(y_i\boldsymbol{x}_i^\top)\right), \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{x}_i\boldsymbol{x}_i^\top)\right)\}_{i=1}^{M}$. The ciphertext of the largest eigenvalue $\mathsf{Enc}\left(\lambda_1\right)$ and the number of iterations T.
Outputs: One encrypted vector $\mathfrak{w}$.
Remarks: One can learn the (approximated) linear regression model $\boldsymbol{w}^*$ with the largest eigenvalue λ_1 and the decryption of $\mathfrak{w}$.

The cloud:

1. Aggregates the ciphertexts

$$\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{X}^\top\boldsymbol{y})\right) = \bigoplus_{i=1}^{N} \mathsf{Enc}\left(\pi_{\mathsf{crt}}(y_i\boldsymbol{x}_i^\top)\right)$$
$$\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{X}^\top\boldsymbol{X})\right) = \bigoplus_{i=1}^{N} \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{x}_i\boldsymbol{x}_i^\top)\right).$$

2. Invokes the DF-MatrixInversion procedure of Equation 4.11

$$\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\lambda_1^{2^T}(\boldsymbol{X}^\top\boldsymbol{X})^{-1})\right) = \mathsf{DF\text{-}MatrixInversion}\left(\mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{X}^\top\boldsymbol{X})\right), \mathsf{Enc}\left(\lambda_1\right), T\right).$$

3. Outputs $\mathfrak{w} = \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\lambda_1^{2^T}(\boldsymbol{X}^\top\boldsymbol{X})^{-1})\right) \otimes \mathsf{Enc}\left(\pi_{\mathsf{crt}}(\boldsymbol{X}^\top\boldsymbol{y})\right)$.

The decryptor:

4. The decryptor decrypts $\mathfrak{w}$ to the vector $\boldsymbol{w}$ and outputs $\boldsymbol{w}^*$ as $\boldsymbol{w}/\lambda_1^{2^T}$.

Figure 4.10: Protocol Π_{LR} for Private Linear Regression.

Theorem 8. *(Correctness.) The protocol of Figure 4.10 correctly implements the linear regression functionality of Equation 4.10 with approximation under the semi-honest model.*

4.4.8 Evaluation

We implemented our building blocks and all the procedures that is described in Section 4.4.6. Our implementations were written in C++, and we used the HElib library [Halevi and Shoup, 2017] 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 4.5. 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 [Lichman, 2013]. For detailed discussions, we focus on one of them, the Adult dataset, which includes $M = 32561$ records with 6 numerical attributes, 7 categorical attributes, and 1 ordinal attribute. Specifically, to show the scalability of the Π_{PCA} and Π_{LR} protocols, we also gave the benchmarks on other four datasets.

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 [Halevi and Shoup, 2017], 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 $\mathbf{\Sigma}$ (as Equation 4.8) is B-bounded, i.e. $|\sigma_{ij}| \leq B$ for all $\sigma_{ij} \in \mathbf{\Sigma}$. After T iterations, the output from Equation 4.9 is bounded by $d_n^T \Delta^{T+1} B^{T+1}$. Recall that we need to introduce a fixed magnifier Δ to convert the real values to integers. Presuming that we use $B = 10^2$, $\Delta = 10^3$, and $d_n = 5$, then the estimation above reveals that $T = 3$ iterations are not allowed because $d_n^3 \Delta^4 B^4 \approx 2^{73}$ 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.

We introduce PPE to achieve a higher plaintext precision with the application of the Chinese-Remainder-Theorem (CRT). 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.

Table 4.5: Parameter sets of the BGV scheme. t_k denotes the coefficient modulo. ℓ is the number of slots of the CRT packing. L is the number of primes in the ciphertext moduli. K is the expansion factor used in for PPE. κ is the security level for that parameter set.

	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

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 4.5. These parameter sets offer at least 80-bit security level and provide the number of slots up to several thousand.

Error Ratio. Our private PCA and LR 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.

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

This error ratio definition enables us to estimate the loss of accuracy.

4.4.9 Evaluation

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

Evaluation Results: Histogram Order & K-percentile. Table 4.6 shows the experimental results of the $\Pi_{\mathsf{HistOrder}}$ and Π_{kP} protocols. 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 became the largest part as M was increased. That was because we needed to decrypt $\lceil(|\mathcal{C}_j|^2 \cdot M)/\ell\rceil$ ciphertexts. The decryption is totally parallelizable so it can be easily reduced by using more cores.

Table 4.6: Benchmark (adult dataset) of the $\Pi_{\mathsf{HistOrder}}$ (Figure 4.5) and Π_{kP} (Figure 4.6). Values are averaged over 10 runs.

Protocol	Domain	# records M	Evaluation	Decryption
$\Pi_{\mathsf{HistOrder}}$	$\lvert\mathcal{C}_j\rvert = 8$	500	1.26 ± 0.145s	1.26s
		1k	1.31 ± 0.157s	1.23s
		10k	2.72 ± 0.289s	4.80s
		32k	6.28 ± 0.484s	13.2s
	$\lvert\mathcal{C}_j\rvert = 16$	500	2.42 ± 0.439s	3.27s
		1k	2.53 ± 0.336s	3.30s
		10k	6.24 ± 0.448s	13.6s
		32k	13.8 ± 1.38s	41.2s
Π_{kP}	$\lvert\mathcal{O}_j\rvert = 100$	500	4.768 ± 0.12s	3.27s
		1k	9.487 ± 0.92s	3.11s
		10k	97.515± 1.60s	18.6s
		32k	321.285 ± 21.7s	48.8s

Table 4.7: Benchmark of the protocol Π_{CTS} (Figure 4.8). Values are averaged over 10 runs.

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

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

Evaluation Results: Contingency Table. Table 4.7 shows the benchmarks of the Π_{CTS}. 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 M, but this computation is entirely parallelizable in our Π_{CTS} protocol. We can easily accelerate this procedure with a higher level of parallelism.

Most of the decryption time in the Π_{CTS} protocol is the time of decrypting the output of the bGT protocol due to the suppression functionality.

Table 4.8: Benchmarks of the PCA and LR protocol (adult dataset): Δ stands for magnification constant; T denotes the number of iterations. K is the expansion factor. Values are averaged over 10 runs.

(a) PCA (for the 1-st principal component)

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

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

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

Evaluation Results: PCA & Linear Regression. We used three different magnification constants Δ and three different iteration numbers T to benchmark the PCA protocol (only for the 1-st principal component). The results are shown in Table 4.8a.

By applying the CRT-packing, the number of ciphertexts to transfer and decrypt during the post-processing phase are $\mathcal{O}(\lceil d_n/\ell \rceil)$, which is independent of the number of records M. As shown in Table 4.8a, the download and decryption time were steady. It took less than 3 minutes to evaluate one principal component with a low error ratio $\text{Error}_{\lambda^*} < 0.1$.

The experimental results of the LR protocol Π_{LR} are shown in Table 4.8b. We omit here the computation time for obtaining the largest eigenvalue λ_1. Similarly, we benchmarked the protocol in 9 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. For the pre-processing time, it took about 17 minutes to achieve the error guarantee.

Extra Experiments for the Predictive Statistics. The extra experimental results of the PCA protocol (the 1-st principal component only) and the LR protocol are shown in Table 4.9. Here, we used $\Delta = 1000$ and $T = 3$, and the running time for the evaluation and decryption were listed. We can see that the running time of the PCA procedure increases linearly with the input dimension d_n, while the running time of the LR procedure increases quadratically with d_n.

Table 4.9: Experimental results of the PCA and LR protocol using UCI datasets. d_n stands for the number of numerical attributes.

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

Table 4.10: Model classification

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	$\boldsymbol{X}, \boldsymbol{Y}$	$\boldsymbol{XY}$	$\boldsymbol{XY}$
	Π_{oGT} (see Chapter 6)	a, b	$\mathcal{I}\{a > b\}$	$\mathcal{I}\{a > b\}$
	Π_{PDT} (see Chapter 6)	$\boldsymbol{a}, \mathcal{T}$	$\mathcal{T}(\boldsymbol{a})$	$\mathcal{T}(\boldsymbol{a})$
model-II	Π_{bGT}	a, b	γ	$\mathcal{I}\{a > b\}$
	Π_{CTS}	$\{\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_p[j]), \mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_q[j])\}_{i=1}^{M}$	$\boldsymbol{\mu}', \boldsymbol{\gamma}', \boldsymbol{\gamma}^*$	$\hat{\boldsymbol{\mu}}$ (Eq. 4.6)
	Π_{kP}	$\{\mathcal{E}_{\mathrm{st}}(\boldsymbol{o}_i[j])\}_{i=1}^{M}$	γ	$\hat{s}_{n^*}^j$ (Eq. 4.7)
	$\Pi_{\mathsf{HistOrder}}$	$\{\mathcal{E}_{\mathrm{id}}(\boldsymbol{c}_i[j])\}_{i=1}^{M}$	$\{\gamma_{uv}\}_{1 \le u < v \le \lvert\mathcal{C}_j\rvert}$	$\boldsymbol{k}$ (Eq. 4.4)
	Π_{PCA}	$\{\boldsymbol{x}_i^\top, \boldsymbol{x}_i \boldsymbol{x}_i^\top\}_{i=1}^{M}$	$\boldsymbol{v}^{(T)}, \boldsymbol{v}^{(T-1)}$	$\boldsymbol{u}_1, \lambda_1$ (Eq. 4.9)
	Π_{LR}	$\{y_i \boldsymbol{x}_i^\top, \boldsymbol{x}_i \boldsymbol{x}_i^\top, \lambda_1\}_{i=1}^{M}$	$\lambda_1^{2^T} \boldsymbol{w}^*$	$\boldsymbol{w}^*$ (Eq. 4.10)

4.5 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 $\mathsf{Enc}(x)$ to the cloud. The cloud homomorphically evaluates f on $\mathsf{Enc}(x)$ and sends $\mathsf{Enc}(f(x))$ to the decryptor. The decryptor decrypts $\mathsf{Enc}(f(x))$ and obtains $f(x)$.

Model-II ($z \neq f(x)$)**.** The encryptor encrypts his private input x and sends $\mathsf{Enc}(x)$ to the cloud. The cloud performs specified homomorphic operations on $\mathsf{Enc}(x)$ and sends the resulting ciphertext $\mathsf{Enc}(z)$ to the decryptor. The decryptor decrypts $\mathsf{Enc}(z)$ 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 4.10. We give the security statements about the protocols.

Theorem 9. *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 9 in the next paragraph. If $z = f(x)$, Theorem 9 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. $\Pi_{\mathsf{bGT}}, \Pi_{\mathsf{CTS}}, \Pi_{\mathsf{kP}}$ and $\Pi_{\mathsf{HistOrder}}$), we show that z does not reveal any information except $f(x)$. However, in our construction, we allow the protocol of Π_{PCA} and Π_{LR} 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 9 and defer the full argument to the full version of our paper. Our proof follows the simulation-based paradigm Goldreich [2009]. 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 9 (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 $\mathsf{bGT}(\mathsf{Enc}(a), \mathsf{Enc}(b))$ (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.

Contingency Table with Cell 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.

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 $\lceil kM/100 \rceil$ and cumulative frequencies of $\hat{s}^j_{n'}$ with $n' > n^*$ are larger than $\lceil kM/100 \rceil$. 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.

Histogram Order. 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 PrivateHistOrder protocol reveals to the decryptor no more than the order of counts in the histogram.

Principal Component Analysis. 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.

Linear Regression. 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.

Chapter 5

Communication Efficient Batch Inner Products

5.1 Target Functionality and the Basic Protocol

In the previous chapters, we have presented cryptographic approaches for computing multiplication of encrypted matrices via inner product of encrypted vectors. These methods have their advantages, i.e., low computation overhead (Π_{IP} of Figure 3.1) or supporting iterative computation (Π_{MP} of Figure 4.1). However, these methods do not count the communication overhead as a requirement. In some cases, such as client–server applications, the communication overhead between the client and the server might be the most important concern, since increasing the outgoing bandwidth of the server is more difficult than increasing the computing power.

The matrix multiplication can be represented as a batch inner products functionality of Figure 5.1. The method of Mohassel and Zhang [2017], Demmler et al. [2015] for this functionality has a relative small communication overhead, which are $\mathcal{O}(d^2)$ ciphertexts, however the computation complexity of this method is $\mathcal{O}(d^3)$ which would take a long time when the matrix dimension d is large. The optimization from Damgård et al. [2012] and Liu et al. [2017] reduces the computation complexity to $\mathcal{O}(d^3/N)$ by using the CRT packing, at the cost of increasing the communication head to $\mathcal{O}(d^3)$ ciphertexts. In other words, the existing FHE methods for the functionality of Figure 5.1 stay in the two opposing extremes aspect to computation efficiency and communication efficiency.

In this chapter, we present a new protocol Π_{SMP} for computing multiplication of encrypted matrices with a smaller communication and computation overhead. We achieve this via a brand new packing which addresses the functionality of batch inner products of Figure 5.1. Specifically, the computation complexity of Π_{SMP} is $\mathcal{O}(d^3/\ell)$ homomorphic operations and the communication complexity is $\mathcal{O}(d^2/\ell)$ aspect to the matrix size d, which achieves a better balance between the

Functionality Batch Inner Products Functionality $\mathcal{F}_{\mathsf{bIP}}$

Inputs: $\ell > 0$ pairs of vector $(\boldsymbol{a}_i, \boldsymbol{b}_i)_{i=1}^{\ell}$, where $\boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{Z}_t^d$ for length $d > 0$.
Outputs: Inner products $\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle \in \mathbb{Z}_t$ for $1 \leq i \leq \ell$.

Figure 5.1: Batch Inner Product Functionality.

communication efficiency and the computation efficiency than the previous methods. As one of the application of Π_{SMP}, we show how to improve the performance of two existing privacy-preserving machine learning frameworks, i.e., SecureML [Mohassel and Zhang, 2017] and MiniONN [Liu et al., 2017]. From empirical results, it shows that Π_{SMP} reduces about 92% communication bandwidth of SecureML and about 95% communication bandwidth of MiniONN. More details on the privacy-preserving machine learning are given in § 5.4.

5.2 Double Packing: New Message Packing for Batching Inner Products

From a high-level explanation, the double packing employs the forward-back packing (Equation 2.1) to each plaintext slot of the CRT packing (Equation 2.2). One should recall that the CRT packing introduces an isomorphic map

$$\mathbb{Z}_t[X]/(X^N + 1) \cong \prod_{k=1}^{\ell} \mathbb{Z}_t[X]/(F_k),$$

where the factor polynomial F_k is a degree-d polynomial and $N = d \cdot \ell$. In other words, the polynomial multiplication in each sub-field is over the modulo F_k. Without assuming the arrangement of F_ks' coefficients, we have the following theorem which is a natural extension of Theorem 1 to the CRT packing.

Theorem 10. *Presume integer vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_t^{\delta}$. Let F_k be one of the degree-d factor polynomial from the CRT-packing. If $0 < \delta \leq d/2$, then the following holds.*

$$\textit{Let } P = \pi_{\mathsf{fwd}}(\boldsymbol{u}) \times \pi_{\mathsf{bwd}}(\boldsymbol{v}) \mod F_k. \textit{ We have } P[\delta - 1] = \langle \boldsymbol{u}, \boldsymbol{v} \rangle .$$

Proof. If the vector length $\delta \leq d/2$, then the degree of the product polynomial $\pi_{\mathsf{fwd}}(\boldsymbol{u}) \times \pi_{\mathsf{bwd}}(\boldsymbol{v})$ is less than d. As a result, taking modulo of the degree-d polynomial F_k does not change the correctness of Theorem 1. □

Theorem 10 simply implies that a batch of ℓ inner products of vectors of length $d/2$ can be

Protocol (Basic) Batch Inner Product Protocol

Input of encryptor : Private vectors $\boldsymbol{a}_i \in \mathbb{Z}_t^d$ for $1 \le i \le \ell$.
Input of evaluator : Private vectors $\boldsymbol{b}_i \in \mathbb{Z}_t^d$ for $1 \le i \le \ell$.
Output of decryptor: : Inner products $\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle \bmod t$ for $1 \le i \le \ell$.

1. **Encryption.** The encryptor encrypts the elements of $\boldsymbol{a}_i$ and sends $d \cdot \ell$ ciphertexts to the evaluator
$$\{\mathsf{Enc}\left(\boldsymbol{a}_i[k]\right)\}_{1 \le i < \ell, 1 \le k < d}.$$
2. **Evaluation.** The evaluator computes as follows.
   (1) Computes the inner product via homomorphic operations For example, the j-th inner product $\langle \boldsymbol{a}_j, \boldsymbol{b}_j \rangle$ is computed by
$$\mathfrak{c}_j = \left(\bigoplus_{k=1}^{d} \mathsf{Enc}\left(\boldsymbol{a}_i[k]\right) \otimes \boldsymbol{b}_j[k] \right)$$
   (2) Outputs ciphertexts $\{\mathfrak{c}_j\}_{j=1}^{\ell}$.
3. **Extraction.** The decryptor decrypts $\mathfrak{c}_j$ and obtain $\langle \boldsymbol{a}_j, \boldsymbol{b}_j \rangle$ for $1 \le j \le \ell$.

Figure 5.2: Basic Secure Batch Inner Products Protocol.

homomorphically computed via one homomorphic multiplication, resulting in one single ciphertext. When using the double packing for secure matrix multiplication, say $\boldsymbol{AB}$ for $|\boldsymbol{A}| = n_1 \times n_2$ and $|\boldsymbol{B}| = n_2 \times n_3$, the encryptor will break down each row of $\boldsymbol{A}$ into vectors with at most $d/2$ entries, and encrypt ℓ vectors that picked from distinct row as one ciphertext. In total, the encryptor will send $n_1/\ell \cdot 2n_2/d \approx 2n_1n_2/N$ ciphertexts to the evaluator. To compute the matrix product, the evaluator first breaks down each column vector of $\boldsymbol{B}$ into vectors with at most $d/2$ entries, and encodes ℓ copies of each shorter vector as one polynomial. Then, the evaluator can homomorphically compute the matrix product via about $2n_1n_2n_3/N$ operations, resulting in n_1n_3/ℓ ciphertexts. In other words, by applying Theorem 10, we can reduce the communication overhead of the basic protocol of Figure 5.2 by a factor of $N/2$, and reduce the computation overhead by a factor of $N/2$, too.

5.2.1 Double the Capacity of Double Packing

Using the optimization in the previous section, we can compute ℓ inner products of vectors with at most $d/2$ entries. We now present a way to double this length from $d/2$ to d which helps reducing half of the computation time and communication cost of our secure matrix product protocol. To do so, we must find a such prime t that "shapes" all polynomials F_k into a specific form

$$F_k = X^d + \beta_k \text{ s.t. } \beta_k \neq 0. \tag{5.1}$$

Table 5.1: Prime t that satisfies Equation 5.1 when $N = 4096$.

t	84961	82241	82561	70913	84481	87041
ℓ	16	32	64	128	256	512

For example, polynomial $X^{1024}+1$ can be decomposed as $(X^{256}+10)(X^{256}+41)(X^{256}+96)(X^{256}+127) \mod 137$, that is $m = 1024$, $d = 256$, and $t = 137$. With such F_k's, we can apply the forward-backward trick to the plaintext slots with vectors of at most d entries. Empirically, we have found many primes that satisfy this requirement. In Table 5.1, we present some examples of practical N and ℓ for the matrix product functionality. Specifically, the underlying encryption scheme should provide at least 80-bit security level when $N = 4096$, according to the security analysis from [Halevi and Shoup, 2014].

Theorem 11. *Suppose integer vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_t^d$ and $\beta_k \neq 0$. Then the following holds.*

$$\textit{Let } P = \pi_{\mathsf{fwd}}(\boldsymbol{u}) \times \pi_{\mathsf{bwd}}(\boldsymbol{v}) \mod X^d + \beta_k. \textit{We have } P[d-1] = \langle \boldsymbol{u}, \boldsymbol{v} \rangle .$$

Proof. Before taking the modulo $X^d + \beta_k$, the $(d-1)$-th coefficient of the product $\pi_{\mathsf{fwd}}(\boldsymbol{u}) \times \pi_{\mathsf{bwd}}(\boldsymbol{v})$ equals to $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, according to the forward-backward trick described in Theorem 1. In addition, the degree of this polynomial is $2d-2$. Therefore, the $(d-1)$-th coefficient remains unchanged, even taking the modulo $X^d + \beta_k$. □

Theorem 11 implies that a batch of ℓ inner products of vectors of length d can be homomorphically computed via one homomorphic multiplication, resulting in one single ciphertext if parameters N and t are chosen properly. The number of encryptions performed by the encryptor is reduced from $2n_1n_2/N$ to n_1n_2/N, and the number of homomorphic operations operated by the evaluator is reduced from $2n_1n_2n_3/N$ to $n_1n_2n_3/N$.

Finally, we write $\overrightarrow{\pi}_{\mathsf{w}}$ and $\overleftarrow{\pi}_{\mathsf{w}} : (\mathbb{Z}_t^d)^\ell \mapsto \mathbb{A}_t$ respectively denoting the double packing functions

$$\overrightarrow{\pi}_{\mathsf{w}}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1}) := \pi_{\mathsf{crt}}(\pi_{\mathsf{fwd}}(\boldsymbol{a}_0), \ldots, \pi_{\mathsf{fwd}}(\boldsymbol{a}_{\ell-1}))$$
$$\overleftarrow{\pi}_{\mathsf{w}}(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{\ell-1}) := \pi_{\mathsf{crt}}(\pi_{\mathsf{bwd}}(\boldsymbol{b}_0), \ldots, \pi_{\mathsf{bwd}}(\boldsymbol{b}_{\ell-1})).$$

The subscript 'w' means double packing. When vectors are packed in this way, we can compute the functionality of Figure 5.1 via one single polynomial multiplication, i.e.,

$$\pi_{\mathsf{w}}^{-1}\left(\overrightarrow{\pi}_{\mathsf{w}}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{\ell-1}) \times \overleftarrow{\pi}_{\mathsf{w}}(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{\ell-1})\right) \text{ gives } \{\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle\}_{i=1}^{\ell}$$

In the next section, we introduce how to efficiently unpack (i.e., π_{w}^{-1}) the double packed vectors.

5.2.2 Efficient Double Unpacking

The double unpacking function is defined as $\pi_{\mathsf{w}}^{-1} : \mathbb{A}_t \mapsto \mathbb{Z}_q^{\ell}$, which is used by the decryptor to extract ℓ inner products after decrypting the ciphertexts received from the evaluator. Suppose the polynomial $A = \sum_{i=0}^{N-1} a_i X^i$ is decrypted from one of the ciphertext that computed by the evaluator. To extract ℓ inner products from A, mathematically, the decryptor must compute the $(d-1)$-th coefficient of $A \mod F_k$ for each modulo $F_k = X^d + \beta_k$. This can be accomplished by using the unpacking function of the CRT packing π_{crt}^{-1}: take the modulo $A \mod F_k$; then keep only the $(d-1)$-th coefficient of the resulting polynomial and discard the remains. But the effort of computing the discarded coefficients becomes meaningless.

We now present a faster implementation of π_{w}^{-1} by examining the algebraic property

$$(-\beta_k)^{t-1} X^{d-1} = X^{td-1} \mod X^d + \beta_k$$

for positive $t > 0$. This gives us a way to compute the $(d-1)$-th coefficient of $A \mod X^d + \beta_k$ directly. Specifically, we compute the $(d-1)$-th coefficient of $A \mod X^d + \beta_k$ as follows

$$a_{d-1} + a_{2d-1}(-\beta_k) + \cdots + a_{\ell d-1}(-\beta_k)^{\ell-1}. \tag{5.2}$$

Because β_ks are known during the key generation, the values $(-\beta_k)^2, \cdots, (-\beta_k)^{\ell-1}$ can be computed once and reused for many unpackings. For the case of secure matrix product, thousands of unpackings is usually required, and thus Equation 5.2 can reduce the computation cost at the decryptor's side significantly. From our empirical results, π_{w}^{-1} that uses Equation 5.2 was about $25 - 1000$ times faster than using π_{crt}^{-1} directly.

5.3 Application: Communication Efficient Secure Matrix Multiplication

We now present our secure matrix product protocol Π_{SMP} in Figure 5.3 using the double packing. It is noteworthy that we fix the matrix size of $\boldsymbol{A}$ to $\ell \times d$ and the size of $\boldsymbol{B}$ to $d \times \ell$ for the sake of simplicity. The protocol in Figure 5.3 can be easily extended for general size matrices, as described later.

The matrix product $\boldsymbol{AB}$ is computed through inner products between the row vectors of $\boldsymbol{A}$ and the column vectors of $\boldsymbol{B}$. Specifically, the encryptor processes rows of $\boldsymbol{A}$ with $\overrightarrow{\pi}_{\mathsf{w}}$ before doing encryption in Step 2. This step produces one ciphertext ctx which is sent to the evaluator. In Step 2, the evaluator applies $\overleftarrow{\pi}_{\mathsf{w}}$ to ℓ-copies of each column of $\boldsymbol{B}$, and then multiplies the packed copies to ctx, resulting a ciphertext of one row of the product matrix. In Step 3, the decryptor decrypts all the ciphertexts and uses π_{w}^{-1} to obtain the result $\boldsymbol{C}$. We now show that $\boldsymbol{C} = \boldsymbol{AB} \mod t$.

Protocol Secure Matrix Product Protocol Π_{SMP}

Remarks: Let sk be a private key and evk be the corresponding evaluation key of the underlying homomorphic encryption scheme. The decryptor holds the private key sk.
Input of encryptor: private matrix $\boldsymbol{A} \in \mathbb{Z}_t^{\ell \times d}$
Input of evaluator: private matrix $\boldsymbol{B} \in \mathbb{Z}_t^{d \times \ell}$
Output of decryptor: the matrix product $\boldsymbol{AB} \in \mathbb{Z}_t^{\ell \times \ell}$

Encryptor:

1. The encryptor processes rows of $\boldsymbol{A}$ with the double packing and sends the ciphertext

$$\mathsf{ctx} = \mathsf{Enc}\left(\vec{\pi}_{\mathsf{w}}\left(\boldsymbol{A}[0,:], \ldots, \boldsymbol{A}[\ell-1,:]\right)\right)$$

to the decryptor, attached with its evaluation key evk.

Evaluator:

2. For each column $\boldsymbol{B}[:,j]$, the evaluator computes a ciphertext

$$\mathsf{ctx}_j = \mathsf{ctx} \otimes \overleftarrow{\pi}_{\mathsf{w}}(\underbrace{\boldsymbol{B}[:,j], \boldsymbol{B}[:,j], \ldots, \boldsymbol{B}[:,j]}_{\ell\text{-copies}}) \oplus \pi_{\mathsf{crt}}([R_{j,1}, R_{j,2}, \cdots, R_{j,\ell}])$$

where polynomials $R_{j,k} \xleftarrow{\mathcal{U}} \mathbb{Z}_t[X]/(F_k)$ and the d-th coefficients are set as zero, i.e., $R_{j,k}[d-1] = 0$ for $1 \le k \le \ell$. The evaluator sends back ℓ ciphertexts $\{\mathsf{ctx}_j\}_{j\in[\ell]}$ to the decryptor.

Decryptor:

3. The decryptor firstly sets $\boldsymbol{C} \in \mathbb{Z}_t^{\ell \times \ell}$ as a zero matrix. For each ctx_j, the decryptor decrypts it and uses π_{w}^{-1} presented in § 5.2.2 to extract ℓ values, which are placed in the j-th row of $\boldsymbol{C}$.
4. The decryptor outputs $\boldsymbol{C}$.

Figure 5.3: Communication Efficient Secure Matrix Product Protocol Π_{SMP}.

Theorem 12. *The protocol of Figure 5.3 computes privately the matrix product functionality, i.e,.* $\boldsymbol{AB}$ *under the semi-honest setting.*

Proof. (Correctness) In Step 1, the encryptor processes the rows of $\boldsymbol{A}$ with $\vec{\pi}_{\mathsf{w}}$. According to Theorem 11, ctx_j of Step 2 gives the inner products between the rows of $\boldsymbol{A}$ and j-th column of $\boldsymbol{B}$, which forms the j-th row of $\boldsymbol{AB}$. By iterating all columns of $\boldsymbol{B}$, the matrix product $\boldsymbol{AB}$ is then encrypted in ciphertexts $\{\mathsf{ctx}_j\}_{j\in[\ell]}$. Thus we have $\boldsymbol{C} = \boldsymbol{AB} \mod t$. □

Proof. (Privacy.) **Security Against a Semi-Honest Evaluator.** We first prove security against a semi-honest evaluator. Given the fact that, the evaluator's view during the protocol execution consists only of ciphertexts, and thus the security against a semi-honest evaluator can be reduced to the semantic security of the underlying encryption scheme.
Security Against a Semi-Honest Decryptor. Next, we prove security against a semi-honest decryptor. The view of the decryptor during the real execution consists of ℓ^2 independent polynomials with coefficients (except the d-th coefficient which is the inner product) distributed uniformly

Table 5.2: Comparing the computation complexity of three HE-based methods. n indicates the matrix dimension. N and ℓ are parameters of the underlying RLWE-based encryption scheme.

Method	Encryptor	Evaluator	Decryptor
Mohassel and Zhang [2017] (AHE-based)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
Liu et al. [2017] (RLWE-based)	$\mathcal{O}(n^2/N)$	$\mathcal{O}(n^3/N)$	$\mathcal{O}(n^3/N)$
Ours (RLWE-basd)	$\mathcal{O}(n^2/\ell)$	$\mathcal{O}(n^3/N)$	$\mathcal{O}(n^2/\ell)$

over $\mathbb{Z}_t$ due to the random polynomials $R_{j,k}$ used in Step 2 of Figure 5.3. Thereby, we can simply construct a simulator for the view of the decryptor during the real execution by sampling uniform random polynomials from $\mathbb{A}_t$. □

General Case and Complexity Analysis. For the case of general matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$ can be partitioned into block matrices $\{\boldsymbol{A}_{ik}\}$ and $\{\boldsymbol{B}_{kj}\}$ where the size of $\boldsymbol{A}_{ik}$ is $\ell \times d$ and the size of $\boldsymbol{B}_{kj}$ is $d \times \ell$. Zero-padding might be used to align the size. Now, the encryptor must process each block $\boldsymbol{A}_{ik}$ in Step 1. Then $\boldsymbol{AB}$ is computed through a summation of products of the block matrices, i.e., $\sum_k \boldsymbol{A}_{ik}\boldsymbol{B}_{kj}$. The block-matrix product $\boldsymbol{A}_{ik}\boldsymbol{B}_{kj}$ is computed in Step 2 of Figure 5.3, and the summation can be accomplished from homomorphic additions. Thus, the correctness of Theorem 12 follows. Furthermore, no extra interaction between the protocol players is introduced. Therefore, the privacy of Theorem 12 follows, too.

In total, the encryptor processes $\mathcal{O}(n_1 n_2/N)$ blocks. The evaluator operates $\mathcal{O}(n_1 n_2 n_3/N)$ homomorphic multiplications and additions, resulting $\mathcal{O}(n_1 n_3/\ell)$ ciphertexts which will be transferred to the decryptor. In Table 5.2, we summarize and compare the complexity of our method with other homomorphic encryption-based methods. Although the computational complexity of our method are in the same order with the AHE-based method of SecureML [Mohassel and Zhang, 2017] our method can provide a considerable acceleration given the fact that m is usually a large value, e.g., $N \geq 2^{12}$. Suppose n_1, n_2, and $n_3 = 128$ and $N = 2^{12}$. For our method, the server only operate 512 homomorphic operations as opposed to the 2.0×10^6 homomorphic operations of the AHE-based method.

Special Case: Matrix–Vector Product. When $n_1 = 1$ (i.e., $\boldsymbol{A}$ becomes a single row matrix), the matrix product $\boldsymbol{AB}$ can be specially regarded as the matrix–vector product. The algorithm of Figure 5.3 does cover this special case by zero-padding $\boldsymbol{A}$, although with a small modification, the complexity of the algorithm can be improved in this setting. That is, in Step 1 of Figure 5.3, the encryptor sends a ciphertext of ℓ-copies of $\boldsymbol{A}$, $\mathsf{ctx} = \mathsf{Enc}\left(\left(\vec{\pi}_{\mathsf{w}}(\boldsymbol{A}[0,:], \ldots, \boldsymbol{A}[0,:])\right)\right)$. In Step 2, the evaluator operates the homomorphic multiplication with ℓ columns of $\boldsymbol{B}$ instead of just one, i.e.,

$$\mathsf{ctx}_0 = \mathsf{ctx} \otimes \pi_{\mathsf{w}}(\boldsymbol{B}[:,0], \boldsymbol{B}[:,1], \ldots, \boldsymbol{B}[:,\ell-1]).$$

Table 5.3: Speedup of the unpacking π_{w}^{-1} due to the pre-computation from Equation 5.2. The RLWE parameters N and t follow Table 5.1.

#slots ℓ	c4.x8large		Raspberry Pi	
	with pre-comp.	w/o pre-comp.	with pre-comp.	w/o pre-comp.
16	0.008 ms	31.1 ms	0.360 ms	549 ms
64	0.032 ms	33.2 ms	1.33 ms	573 ms
128	0.120 ms	33.3 ms	5.03 ms	612 ms
256	0.437 ms	36.3 ms	21.9 ms	694 ms
512	1.68 ms	41.6 ms	99.9 ms	853 ms

Table 5.4: Micro-benchmarks of Π_{SMP} using one single access. The performance numbers were averaged from 50 runs. m and ℓ are parameters of the underlying encryption scheme, and κ denotes the security level.

(m, ℓ, κ)	$n_1 \times n_3$	$\vec{\pi}_{\mathsf{w}}$	ENC	EVA	DEC	π_{w}^{-1}	Total Time (sec)		Communication	
							LAN	WAN	encryptor	decryptor
$(4096, 128, 80)$	128^2	27.9 ms	16.1 ms	205 ms	233 ms	28.9 ms	2.45	3.30	0.25 MB	8 MB
	256^2	43.8 ms	26.2 ms	670 ms	744 ms	83.8 ms	7.25	8.22	0.5 MB	32 MB
	512^2	81.1 ms	44.9 ms	2.63 sec	2.82 sec	299 ms	26.4	27.7	1.0 MB	128 MB
$(8192, 256, 300)$	256^2	67.9 ms	31.8 ms	614 ms	770 ms	197 ms	9.37	10.7	0.5 MB	32 MB
	512^2	108 ms	48.4 ms	2.46 sec	2.88 sec	720 ms	29.1	30.4	1.0 MB	128 MB
	1024^2	140 ms	75.9 ms	9.80 sec	11.0 sec	2.81 sec	65.4	110	2.0 MB	512 MB

The remaining steps of the algorithm follows. In this case, the evaluator performs $\mathcal{O}(n_2 n_3/N)$ homomorphic operations, and transfers $\mathcal{O}(n_3/\ell)$ ciphertexts to the decryptor.

5.3.1 Evaluations

Implementations. We implemented Π_{SMP} using HElib [Halevi and Shoup, 2017]. The parameters $N = 4096$ and $t = 70913$ were used to provide $\ell = 128$ slots. The other parameters of HElib were set properly to provide at least $\kappa = 80$-bit security level. A single ciphertext under this setting was about 64 kilobytes. Additionally, $N = 8192$ and $t = 84481$, which can provide at least $\kappa = 300$-bit security level, were also used in the micro-benchmarks of Π_{SMP} (Table 5.4).

We compared Π_{SMP} with three existing methods, including the OT-based method, the AHE-based method from SecureML and the RLWE-based method from MiniONN. Specifically, we used the EMP-toolkit [Wang et al., 2016a] to implement the OT-based method of SecureML, using 16-bit inputs for a fair comparison because $\log_2 t \approx 16$. For the AHE-based method, we instantiated the AHE with the DGK scheme [Damgård et al., 2009] with a 1024-bit RSA modulus using the implementation from [Demmler et al., 2015]. For the method of MiniONN, we used the HElib and parameters were set as $N = 4096$ and $t = 65537$, aiming to provide $\ell = 4096$ plaintext slots.

Computation and Communication Environments. We conducted extensive experiments using a various types of computing machines and under two network settings.

- *Evaluator specifications.* We used one AWS c5.18xlarge instance of 72 virtual CPUs (vCPUs) @3.00 GHz and 25 Gbps outgoing bandwidth as the evaluator node.
- *Encrytor & Decryptor specifications.* The encryptor' requests were equally launched from five AWS c4.8xlarge instances of 36 vCPUs @2.90 GHz for each instance. Also, we ran experiments on a weak device, i.e., Raspberry Pi model B v1.2 of one CPU @900 MHz.
- *LAN setting.* For the LAN setting, all the AWS instances were launched inside the same region. The ping delay was about 2.5 ms.
- *WAN setting.* For the WAN setting, the evaluator instance was launched in west US and the encryptor instances were launched in east Asia. The bandwidth was about 114 Mbps with about 110 ms ping delay.

Measurements. We measured the end-to-end running time of these matrix product protocols using a high resolution clock (i.e., the standard chrono library). For the OT-based method, the time for computing the OT-extension [Asharov et al., 2013] is included. For Π_{SMP}, the computing time of Equation 5.2 is included while the key generation time is excluded. We also measured the total amount of data transferred by each method. Moreover, we provide five more micro-benchmarks of Π_{SMP}. That is, the computation time of packing $\vec{\pi}_{\mathsf{w}}$, encryption (ENC), decryption (DEC), and unpacking π_{w}^{-1} on the encryptor and decryptor's side, and the evaluation (EVA) time on the evaluator's side.

5.3.2 Evaluation Results: Faster Unpacking Optimization

We can instantiate the double unpacking function π_{w}^{-1} using the unpacking function π_{crt}^{-1}, but this will introduce an expensive computation at the client's side. In § 5.2.2, we present to use a pre-computed table to eliminate the needs of π_{crt}^{-1}. Experiment results (Table 5.3) show that this optimization can significantly reduce the computation burden of the client. This experiment was conducted on two devices, the powerful AWS instance and the much weaker Raspberry Pi. We can see that the unpacking π_{w}^{-1} from using Equation 5.2 was much faster than using π_{crt}^{-1}.

Table 5.5: Comparing the performances of Π_{SMP} with the AHE-based method and the OT-based method from SecureML [Mohassel and Zhang, 2017], and the RLWE-based method from MiniONN [Liu et al., 2017], under LAN and WAN, respectively.

Dimension n_1, n_2, n_3	Method	MT Generation Time (LAN/WAN)			Communication
		1 encryptor	10 encryptors	50 encryptors	
128, 128, 128	AHE	26.3 sec/26.9 sec	26.1 sec/26.7 sec	33.2 sec/33.0 sec	3.98 MB
	OT	4.07 sec/60.8 sec	140 sec/168 sec	389 sec/485 sec	432 MB
	RLWE	123 sec/124 sec	123 sec/127 sec	146 sec/151 sec	32.3 MB
	Ours	2.49 sec/3.26 sec	2.54 sec/3.35 sec	3.53 sec/3.36 sec	8.25 MB
256, 128, 256	AHE	95.1 sec/95.3 sec	94.6 sec/95.0 sec	122 sec/123 sec	11.9 MB
	OT	16.1 sec/235 sec	588 sec/656 sec	0.426 hr/0.472 hr	1664 MB
	RLWE	486 sec/489 sec	495 sec/498 sec	545 sec/545 sec	129 MB
	Ours	7.38 sec/8.21 sec	7.41 sec/8.63 sec	10.8 sec/11.0 sec	32.5 MB
512, 128, 512	AHE	361 sec/361 sec	356 sec/359 sec	507 sec/510 sec	39.8 MB
	OT	57.5 sec/917 sec	0.723 hr/0.732 hr	1.59 hr/1.82 hr	6520 MB
	RLWE	0.522 hr/0.538 hr	0.556 hr/0.574 hr	0.560 hr/0.570 hr	513 MB
	Ours	26.6 sec/27.5 sec	26.8 sec/29.1 sec	41.2 sec/41.4 sec	129 MB

5.3.3 Evaluation Results: Micro-benchmarks

The micro-benchmarks of Π_{SMP} is given in Table 5.4. Additionally, we used a higher security level $\kappa = 160$ in this experiment to demonstrate its performance growth with respect to κ. By the virtue of our new packing method and its extension, the computation on the evaluator' side (i.e., the EVA column) was very fast.

5.3.4 Evaluation Results: Comparison to Other Matrix Product Methods

The comparison results to the existing secure matrix product methods are given in Table 6.1. From Table 6.1, we know that Π_{SMP} outperformed these methods in terms of computation time, especially when many encryptors access to the evaluator concurrently.

For the AHE-based method, the evaluator needs to operate $n_1 n_2 n_3$ homomorphic operations while our method requires only $n_1 n_2 n_3/m$ homomorphic operations. Since the parameter m of the underlying encryption scheme is usually large, e.g., $m \geq 2^{12}$, our method can provide a considerable boost, e.g., it was about $8-12$ times faster than the AHE-based method according to our benchmarks. Although the consumed network traffic of Π_{SMP} was $2-3$ times larger than that of the AHE-based method, the absolute amount of the network traffic was still small enough to be transferred through a narrow bandwidth within a reasonable time.

Comparing to the OT-based method, Π_{SMP} only exchanged about 1.9% of data of the OT-based

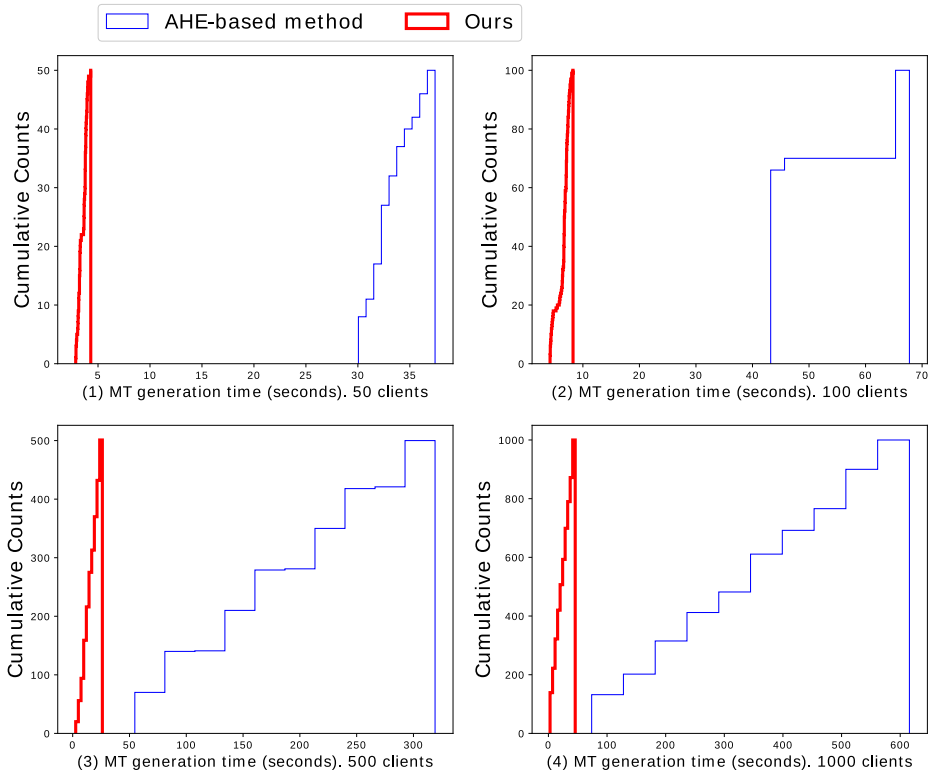


Figure 5.4: Cumulative histograms of the performances of Π_{SMP} and the AHE-based method of [Mohassel and Zhang, 2017] under 50, 100, 500, and 1000 concurrent accesses. The size of matrices was 128×128.

method. As a result, when many clients access to the evaluator concurrently, the running time of Π_{SMP} was much faster than the OT-based method, e.g., about $66-97$ times faster when the number of concurrent accesses was 10. Moreover, the performance of our method was less sensitive to the network latency than the OT-based method. That is because our method is a single-round protocol while the OT-based method requires multiple rounds of communication between the encryptor and the evaluator.

5.3.5 Evaluation Results: One Thousand Concurrent Accesses and Constrained Devices

To show the feasibility of using the proposed method under the high concurrency setting, we benchmarked Π_{SMP} with more concurrent accesses under the LAN setting. Specifically, for each of the five AWS c4.8xlarge instances, we launched 10, 20, 100, and 200 MT generation requests to the server instance, and recorded the total running time for each request. In the other words, the server instance handled concurrent accesses of 50, 100, 500 and 1000 using its 72 vCPUs. We only compare

with the AHE-based method of Mohassel and Zhang [2017] because of the OT-based method and the SwHE-based method would take too much time under the high concurrency setting. The results are shown in Figure 5.4 The server handled 100 concurrent accesses within 10 seconds (that is, the 2-rd plot of Figure 5.4). Therefore, if we want to handle 1000 concurrent accesses within 10 seconds, we need about 10 server nodes.

We also ran the secure matrix multiplication protocols on a much more weaker device, i.e., the Raspberry Pi. However we failed to compile the OT library [Wang et al., 2016a] on the Raspberry Pi. Therefore, only the homomorphic encryption-based methods were considered in this experiment. Matrices of size 128×128 were used. The total running times are as follows.

- Ours: 16.2 seconds.
- AHE-based method of Mohassel and Zhang [2017]: 389 seconds.
- RLWE-based method of Liu et al. [2017]: more than 23 minutes.

We can see that our method can perform better than the existing HE-based methods on resource constrained devices.

5.4 Application: Privacy-preserving Machine Learning

Machine learning is becoming ubiquitous. More and more machine learning-based online services, such as shopping recommendation [Nikolaenko et al., 2013], traffic-aware navigation [Wu et al., 2016], medical diagnosis [Singh and Guttag, 2011], and face recognition [Erkin et al., 2009], are reachable by thousands of clients. One important feature of these online services is *high concurrency*. That is, there might be thousands of clients using the services simultaneously. Suppose a traffic-aware navigation service is running in a metropolitan area, e.g., Tokyo and New York. The number of clients using the navigation service can be numerous during the peak time. Thus, the machine learning-based service should be scalable under this highly concurrent setting. Moreover, clients of these online services are usually equipped with limited computing resources. Take the navigation service as the example again; the client might connect to the navigation service via his/her cell phone or through on-vehicle devices. Therefore, the machine learning-based service should avoid introducing heavy computations at the client's side.

To use a machine learning-based online service, a client must reveal his/her data to the server. When the data involves sensitive information of the client, such as locations, shopping logs and medical records, revealing these data to the server might raise potential risks of compromising client's privacy, such as data breaches. A natural question here is whether one can use the machine learning-based online services and still maintain the privacy of client's data?

One solution is to use secure two-party computation (2PC) techniques [Goldwasser and Micali, 1982, Yao, 1982]. 2PC allows two parties (e.g., the client and the server) to jointly compute a

function on their private inputs, learning only the output of the function. In the context of the privacy-preserving machine learning, one of the most fundamental components is a practical secure matrix multiplication protocol, since computing the product of matrices is the essential operation of many popularly used machine learning algorithms such as linear regression [Lu et al., 2017, Gascón et al., 2017], logistic regression [Wang et al., 2016b] and neural networks [Liu et al., 2017, Riazi et al., 2018].

Based on the discussions above, the main objective of this work is to develop a practical secure matrix multiplication protocol that can run fast under the following (realistic) situations.

1. **High Concurrency.** There can be thousands of clients continuously and concurrently access to the server while the outgoing and incoming bandwidth of the server and clients are bounded, e.g., 20 Gbps and 100 Mbps, respectively. In other words, the server can only allocate a small ratio of its bandwidth for each client.

2. **Weak Client.** The computing resources at the client's side might be constrained. For example, the computing power of the autonomous vehicles and the Internet of Things (IoT) devices are usually limited, e.g., one 1.0 - 2.0 GHz CPU chip.

5.4.1 Related Work and Challenges

Recent improvements and optimizations to 2PC, such as Brakerski et al. [2012], Damgård et al. [2012], Asharov et al. [2013], Demmler et al. [2015], Mohassel and Zhang [2017], Huang et al. [August 8-12, 2011], Wang et al. [2016a], Liu et al. [2015], Songhori et al. [2015] to name a few, enable efficient secure matrix multiplication protocols. Specifically, we can separate a secure multiplication protocol into two stages: an *offline stage* and an *online stage*. The computation during the offline stage is input-independent, that is the client and the server do not need to provide any private data in this stage. They jointly compute some (one-time use) auxiliary data, i.e., Beaver's multiplication triples (MTs) Beaver [1995], so that the evaluation of the online stage can be significantly accelerated using MTs.

Beaver's MT is one of the most efficient way to perform secure matrix multiplication which has been applied to many 2PC frameworks such as SPDZ [Damgård et al., 2012], ABY [Demmler et al., 2015], SecureML [Mohassel and Zhang, 2017] and MiniONN [Liu et al., 2017]. Many optimizations are proposed to improve the MT generation (i.e., the performance in the offline stage) in these frameworks. Specifically, ABY and SecureML suggest to use a vectorization technique and propose a variant type of MT which is specialized for secure dot product. Then, secure matrix multiplication can be computed by the iterations of secure dot products. They present two concrete solutions from oblivious transfer (OT) and additively homomorphic encryption (AHE), respectively. On the other hand, SPDZ and MiniONN propose another optimization from a RLWE-based homomorphic encryption scheme [Brakerski et al., 2012, Chen et al., 2017] and the message packing technique [Smart

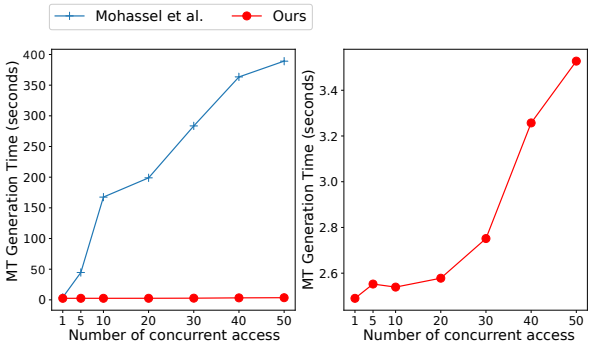


Figure 5.5: Left: Running time for generating MTs for multiplying matrices of 128×128 entries using the OT-based method of Mohassel and Zhang [2017] under a 25 Gbps outgoing bandwidth. Right: Running time of our method under the same setting.

and Vercauteren, 2014]. Specifically, a batch of $\ell > 0$ integers are encrypted as a single ciphertext where ℓ is a parameter related to the RLWE-based scheme. Then, the homomorphic multiplication will be carried out to the packed ℓ integers simultaneously.

The secure matrix multiplication protocol can also be constructed from tools such as Yao's garbled circuit [Yao, 1982, Kolesnikov and Schneider, 2008, Zahur et al., 2015, Nikolaenko et al., 2013] and garbled arithmetic circuit [Applebaum et al., 2011]. However, such generic tools also require a wide bandwidth as the OT-based methods. Other ad hoc methods [Mishra et al., 2017, Duong et al., 2016] are efficient regarding computation time but the matrix size is constrained, e.g., smaller than 16×16, which might not be sufficient for modern machine learning algorithms like deep neural networks.

Challenges. Suppose the client and the server evaluate a secure matrix multiplication with private matrices of n dimension of t-bits integers using the existing methods [Demmler et al., 2015, Mohassel and Zhang, 2017, Damgård et al., 2012, Liu et al., 2017]. We now show that these methods might not scale well under the high concurrency and weak client situations.

For the OT-based method of [Demmler et al., 2015, Mohassel and Zhang, 2017], the client and the server need to perform $\mathcal{O}(n^3 t)$ instances of correlated OT [Asharov et al., 2013], which is lightweight in terms of computation. However, when many clients access to the server concurrently, these OT-based methods will take a considerably longer time, since the bandwidth allocated for each user is eventually bounded. Figure 5.5 gives an example of using the OT-based method of [Mohassel and Zhang, 2017] to generate MTs for secure matrix multiplication of two 128×128 matrices. In this experiment, the outgoing bandwidth on the server's side was about 25 Gbps. When the number of concurrent accesses was small, e.g., $1 - 2$, clients only need to wait less than 5 seconds. As the number of concurrent accesses increases, the waiting time becomes significantly longer. In other words, if the server wants to handle 1000 concurrent accesses within 5 seconds, it might need to prepare more than 25000 Gbps outgoing bandwidth, which seems unrealistic for the current Internet.

Table 5.6: Using Π_{SMP} to improve the pre-computation stage of SecureML. The mini-batch size B was fixed as 128 as SecureML. The performance numbers of the methods of SecureML are taken from their paper Mohassel and Zhang [2017].

N, D, t	Method	Time (LAN/WAN)	Commu.
$10^4, 100, 156$	AHE	248.4 sec/252.9 sec	20 MB
	OT	7.9 sec/420.2 sec	1.9 GB
	Ours	23.4 sec/30.2 sec	159 MB
$10^5, 100, 1563$	AHE	2437.1 sec/2478.1 sec	200 MB
	OT	88.0 sec/4125.1 sec	19 GB
	Ours	168 sec/187 sec	785 MB

For the RLWE-based method of [Liu et al., 2017], the client and the server exchange $\mathcal{O}(n^3/\ell)$ RLWE ciphertexts through the network. This method can consume less bandwidth than the OT-based solutions because of the packing technique allows embedding a large number of plaintext values into a single ciphertext, such as $\ell = 2^{12}$. However, this method requires the client to perform $\mathcal{O}(n^3/\ell)$ unpackings, which can be extremely expensive for a weak client. For instance, in our benchmarks, it took a Raspberry Pi more than 23 minutes to perform the unpacking when $n = 128$ and $\ell = 2^{12}$. Even for a powerful AWS instance, this computation still took more than 2 minutes.

For the AHE-based method of [Demmler et al., 2015, Mohassel and Zhang, 2017], the client and the server exchange $\mathcal{O}(n^2)$ AHE ciphertexts. The client needs to perform $\mathcal{O}(n^2)$ decryptions, and no unpacking is needed. It seems that the AHE-based method is the best under our situations. However, the server needs to operate $\mathcal{O}(n^3)$ public key operations which can become the performance bottleneck when n is large. For instance, when $n = 512$, the client would need to wait more than 6 minutes before the server completes the secure matrix multiplication.

5.4.2 Application to Private Machine Learning Model Training

SecureML is a secure computation framework that is originally designed for training machine learning models from a dataset that is already shared additively between two collusion-free servers. The private model training is performed between the two collusion-free servers using OT-based secure matrix multiplication protocol, and other cryptographic tools such as Yao's garbled circuit Yao [1982], Wang et al. [2016a].

Specifically, SecureML uses the mini-batch stochastic gradient descent (SGD) of a batch size $B > 0$ to train their models from a dataset $\boldsymbol{X} \in \mathbb{Z}^{N \times D}$ within t steps. Here, N indicates the number of data points in the dataset and D indicates the number of features. As suggested by SecureML, when using SGD for some classes of machine learning algorithms, e.g., linear regression

Table 5.7: Using Π_{SMP} to improve the pre-computation stage of MiniONN. The performance numbers of the method of MiniONN are taken from their paper Liu et al. [2017].

(a) The matrix products involved in NN-CIFAR.

Layer	1	3	6	8
$\boldsymbol{A}$	1024×27	1024×576	256×576	256×576
$\boldsymbol{B}$	27×64	576×64	576×64	576×64
Layer	11	13	15	17
$\boldsymbol{A}$	64×576	64×64	64×64	1×1024
$\boldsymbol{B}$	576×64	64×64	64×16	1024×10

(b) Performances under a single access.

Method	Time (LAN/WAN)	Communication
MiniONN	472 sec/ –	3046 MB
Ours	24.5 sec/26.6 sec	137 MB

and logistic regression, the SGD computation involves a matrix product $\boldsymbol{UV}$, where the size of $\boldsymbol{U}$ is $B \times D$ and the size of $\boldsymbol{V}$ is $D \times t$. The value of B is usually a few hundreds, e.g., $B = 128$ as in SecureML, and the value of t is usually set such that $Bt > N$.

We now show that Π_{SMP} can be a better alternative for performing secure matrix multiplication in SecureML, especially under the high concurrency setting. Specifically, we compared the computation time and communication cost with the performance numbers presented in their paper Mohassel and Zhang [2017]. Notice that, 64-bit inputs were used in SecureML while the plaintext precision of our encryption scheme was $\log_2 t \approx 16$. To have a fair comparison, we use the techniques of [Lu et al., 2017] to achieve the same level of 64-bits precision. The comparison details are given in Table 5.6. It is apparent that our method is more efficient, especially when the matrix size is large and the number of concurrent accesses is more than one. Specifically, our method was about $3 - 21$ times faster than the AHE-based method when the matrix size was 100×1563. Also, it was $5 - 35$ times faster than the OT-based method and consumed only about $4.0\% - 5.3\%$ of the OT-based method.

5.4.3 Application to Private Deep Neural Networks Evaluation

The recent explosive evolution of neural network research has led to breakthroughs in many machine learning tasks. The application area covers not only image and speech recognition but also diverse types of predictive and cognitive modeling. The unprecedented accuracy of deep learning models enables various novel services that might have marked effects on our society, for example, human virus detection [Brion et al., 2005], and drug discovery [Baskin et al., 2016].

The private neural network evaluation framework, i.e., MiniONN, is proposed by Liu et al. [2017], which apply the CRT packing technique to speed up the server's computation time, at the

Table 5.8: Neural Network Description. c' indicates the number of channels, h is the size of filters and s is the stride size. ρ is the pooling size.

Layer	Output Size	Activation	Note
Input	$32 \times 32 \times 3$	-	RGB image
Conv-1	$30 \times 30 \times 32$	ReLU	$c' = 32, h = 3, s = 1$
Conv-2	$28 \times 28 \times 32$	ReLU	$c' = 32, h = 3, s = 1$
MaxPool-1	$14 \times 14 \times 32$	-	$\rho = 2$
Conv-3	$13 \times 13 \times 64$	ReLU	$c' = 64, h = 2, s = 1$
Conv-4	$12 \times 12 \times 64$	ReLU	$c' = 64, h = 2, s = 1$
MaxPool-2	$6 \times 6 \times 64$	-	$\rho = 2$
FC-1	512×1	ReLU	2304×512
FC-2	10×1	Softmax	512×10

cost increasing the communication overhead and the computation overhead of the client. More specifically, in the protocol of Liu et al. [2017], the server transfers a cubic number of ciphertexts to the client. For example, the protocol of Liu et al. [2017] requires the client to transfer more than 9 GB data to evaluate a middle-sized convolution network (i.e., achieving about 81% test accuracy on the CIFAR-10 dataset). Also, the client might need to perform many decryption to obtain the final result, which is expensive especially for clients of constrained bandwidth and computing power.

We experimentally show that Π_{SMP} is a more practical option for MiniONN to perform secure matrix multiplication. Specifically, we take the 17-layer neural network from MiniONN as an example. This network was originally designed for classifying the CIFAR-10 dataset [Krizhevsky and Hinton, 2009], and thus we designate it as NN-CIFAR. In Table 5.7, we list up all the matrix products involved in NN-CIFAR where the matrix $\boldsymbol{A}$ is the private input from the client and $\boldsymbol{B}$ is the private input from the serverr. We compared the computation time and communication cost of our protocol with MiniONN. The comparison details are shown in 5.7b. It is apparent that Π_{SMP} considerably reduced the computation time and the communication cost of the pre-computation stage of MiniONN for evaluating NN-CIFAR, i.e., saving more than 95% of the computation time and communication cost.

5.4.4 Concret Example of Private Deep Neural Networks Evaluation

To demonstrate the practicality of our Π_{SMP}, we conduct experiments with a ten-layer nonlinear CNN model (Table 5.8) which can provide about 82.8% accuracy for the CIFAR-10 dataset. The linear transforms (i.e., convolution and fully connected layers) are computed via our Π_{SMP} protocol, and the non-linear parts are turned to garbled circuit.

Some Optimizations for Nonlinear Operations. In the some typical deep neural network architecture, max pooling follows after an activation, that is, ReLU in the network of Table 5.8. We note that the computation result is unchanged no matter we do the ReLU first or do the max pooling

Table 5.9: Experimental results of our private CNN evaluation protocol. The evaluation time includes the computation time and the communication time.

Layer	Evaluation (ms)	Communication (MB)	
		client → server	server → client
Conv-1	339 ± 2.88	15.08	2.23
ReLU	13678 ± 335.64	23.52	78.96
Conv-2	3464 ± 38.34	160.48	2.23
ReLUPool	9218 ± 32.29	12.12	58.07
Conv-3	2355 ± 14.75	71.47	1.12
ReLU	6015 ± 175.55	9.60	32.22
Conv-4	4672 ± 34.05	142.95	1.12
ReLUPool	3395 ± 100.86	3.88	18.59
FC-1	4152 ± 2.77	4.47	10.05
ReLU	524 ± 13.12	0.41	1.37
FC-2	763 ± 0.53	0.56	2.23
Total	48575 ± 750.78	444.54	208.19

first, that is, $\mathsf{Pool}(\mathsf{ReLU}(\boldsymbol{T}), \rho) = \mathsf{ReLU}(\mathsf{Pool}(\boldsymbol{T}, \rho))$ holds. However, the computation complexity of the first one is larger than that of the second one. From a simple calculation, we can know the computation complexity is $2n'^2$ and $n'^2 + n'^2/\rho^2$, respectively. Moreover, both ReLU and max pooling require the max operation only. This motivates us to evaluate $\mathsf{ReLU}(\mathsf{Pool}(\cdot))$ by combining the ReLU operation with the following max pooling operation (if exist).

The experimental results are shown in Table 5.9. From the results, we can see that we can privately evaluate a deep neural network within a reasonable time and communication overhead.

5.5 Conclusion

In this chapter, we presented Π_{SMP} for computing the multiplication of two matrices efficiently. Π_{SMP} is built from ring-based homomorphic encryption with three algorithmic and implementation optimizations. Our optimizations significantly reduce the computation time, both at the severs side and at the client's side. Moreover, the communication cost of our method is considerably less than the existing OT-based methods. According to our experimental results, Π_{SMP} outperformed the existing methods under the high concurrency setting. We also applied Π_{SMP} to two frameworks of privacy-preserving machine learning, i.e., SecureML and MiniONN. The experimental results showed that Π_{SMP} can reduce MT generation time of these frameworks by more than $74\% - 97\%$. With the combination of Π_{SMP} and garbled circuit, we also show that we can privately evaluate a ten-layers

CNN within one minute with a reasonable communication overhead. In concluding, we consider that Π_{SMP} can help forwarding the deployment of more practical and usable secure two-party computation to machine learning-based online applications.

Chapter 6

Non-interactive and Expressive Comparison

We have presented protocols for the inner product which are a linear function of the input while for some algorithms, e.g., decision tree and support vector machine, the comparison (which is a non-linear function) are also used. The protocol Π_{bGT} described in the previous chapter is efficient for comparing two encrypted integers, but it provides a very limited ability to perform further computation on the comparison bits. For example, given the ciphertexts of two encrypted vectors, we can not use Π_{bGT} to compute how many elements in the first vector is larger than the elements in the second vector. For this purpose, we present another comparison protocol oGT (Figure 6.1) which can provide more expressiveness. As one of the possible application of oGT, we present the first non-interactive privacy-preserving decision tree evaluation protocol in the following section.

We first describe the basic idea behind Figure 6.1. The message space of FHE is a polynomial ring, and thus we can put integers in the coefficient *or the degree of the polynomials*. In oGT, we encode $0 \leq a, b < N$ in the degree of the polynomials. Remember that N is the parameter of the underlying FHE scheme which is usually set as a few thousands, e.g., $N \leq 2^{13}$. Specifically, we use the encoding $\pi : \mathbb{Z} \mapsto \mathbb{A}_t$, given by $\pi(a) = X^a$. The core idea behind oGT is to use the polynomial

$$C_0 = T_0 \times \pi(a) \times \pi(-b) \bmod (X^N + 1), \tag{6.1}$$

where $T_0 = 1 + X + \cdots + X^{N-1}$. Notice that the polynomial with a negative degree X^{-b} is equivalent to $-X^{N-b}$ modulo $X^N + 1$. We argue that the 0-th coefficient $C_0[0] \in \{1, -1\}$. When $a \leq b$, $C_0[0]$ comes from $T_0[b-a]$, and thus $C_0[0] = 1$. On the other hand, when $a > b$, $C_0[0]$ comes from the $(m - (a-b))$-th coefficient of T_0, but in this case $C_0[0] = -1$ due to the degree wrap around, i.e., $X^{N-(a-b)} \times X^{a-b} = -1 \mod X^N + 1$. In other words, $C_0[0] = 1$ if $a \leq b$, else $C_0[0] = -1$.

Protocol Output Expressive Greater-than oGT

Input: Ciphertexts $\mathsf{Enc}\left(X^a\right)$, $\mathsf{Enc}\left(X^b\right)$ where $a, b \in \mathbb{Z}_N$ where N is one of the FHE parameter.
Output: $\mathsf{Enc}\left(C\right)$, i.e., the ciphertext of a polynomial $C \in \mathbb{A}_t$.
Remark: One can learn $\mathcal{I}\{a > b\}$ from the first coefficient of C, i.e., $\mathcal{I}\{a > b\} = C[0]$.

1. Compute $\bar{\mu} = 2^{-1} \bmod t$ and $\mu = -\bar{\mu} \bmod t$
2. Set the polynomial $T = \mu + \mu X + \cdots + \mu X^{m-1}$
3. Sample polynomial $R \xleftarrow{\mathcal{U}} \mathbb{A}_t$ and set $R[0] = \bar{\mu}$
4. Negate the degree homomorphically $\mathsf{Enc}\left(X^{-b}\right) \leftarrow \mathsf{Enc}\left(X^b\right)$
5. Compute and output $\mathsf{Enc}\left(C\right) = \mathsf{Enc}\left(X^a\right) \otimes \mathsf{Enc}\left(X^{-b}\right) \otimes T \oplus R$

Figure 6.1: Output Expressive Greater-than

Theorem 13. *(Correctness.) The protocol of Figure 6.1 correctly implements the greater-than functionality* $\mathcal{I}\{a > b\}$.

Proof. We consider when $a \leq b$. In this case, we have exactly one term, i.e., μX^{b-a} of T which leads the 0-th coefficient of the polynomial $(X^a \times X^{-b} \times T)$ being μ. As a result, $C[0] = \mu + R[0] = \mu + \bar{\mu} = 0$. On the opposite side, we consider the case $a > b$. In this case, this coefficient becomes $-\mu$ because $\mu X^m = -\mu \mod X^m + 1$. As a result $C[0] = -\mu + R[0]$ which equals to $2\bar{\mu} = 1$. This completes our proof that $C[0] = \mathcal{I}\{a > b\}$. □

Indeed, Step 4 of Figure 6.1 can be avoided if encryptor sends four ciphertexts $\{\mathsf{Enc}\left(X^a\right), \mathsf{Enc}\left(X^{-a}\right)\}$ and $\{\mathsf{Enc}\left(X^b\right), \mathsf{Enc}\left(X^{-b}\right)\}$ to the evaluator. However, this doubles encryptor's computation and communication overhead. We now present a way that allows the evaluator to homomorphically compute $\mathsf{Enc}\left(X^{-b}\right)$ from $\mathsf{Enc}\left(X^b\right)$. We achieve this negate step by applying homomorphically the automorphism map $\mathcal{M} : X \mapsto X^{2m-1}$

$$\mathcal{M}(X^b) = X^{2mb-b} = \underbrace{(-1)^{2b}}_{=1} \cdot X^{-b} \mod X^m + 1.$$

This automorphism is possible on the FHE ciphertext since $2m - 1 \in \mathbb{Z}_{2m}^*$ which introduces one extra key switching matrix into the public key. We refer to [Halevi and Shoup, 2014, Gentry et al., 2012c] for more details about the automorphism on FHE ciphertexts.

6.1 Preserving Homomorphism

Arithmetic addition and multiplication are still possible on the output ciphertexts of oGT. To be precise, the resulting ciphertext from oGT encrypts a polynomial of the form $C = \mathcal{I}\{a >$

$b\} + \sum_{j=1}^{N-1} r_j X^j$ where r_js' are randomly generated integers. We can homomorphically sum the outputs from several calls of oGT. For example, from the decryption of $\mathsf{oGT}(\mathsf{Enc}\left(X^a\right), \mathsf{Enc}\left(X^{b_1}\right)) \oplus \mathsf{oGT}(\mathsf{Enc}\left(X^a\right), \mathsf{Enc}\left(X^{b_2}\right))$, we can know $\mathcal{I}\{a > b_1\} + \mathcal{I}\{a > b_2\}$.

For the multiplication, we require some constraints. If we operate $\mathsf{oGT}(\mathsf{Enc}\left(X^a\right), \mathsf{Enc}\left(X^{b_1}\right)) \otimes \mathsf{oGT}(\mathsf{Enc}\left(X^a\right), \mathsf{Enc}\left(X^{b_2}\right))$, we can not obtain any meaningful result due to the random coefficients introduced in oGT. In other words, we can not multiply the outputs from several calls of oGT directly. On the other hand, we can only multiply the encryption of an integer $e \in \mathbb{Z}_p$ to the encrypted result of oGT, giving a ciphertext $\mathsf{Enc}\left(e \cdot C\right)$. Thereby, after the decryption, we can obtain a meaningful result $e \cdot \mathcal{I}\{a > b\}$ from the 0-th coefficient.

6.2 Feasible Domain Extension

By exploiting the polynomial structure, our protocol can efficiently compare two encrypted integers. However, the feasible input domain of oGT is relatively small, that is $\mathbb{Z}_N$, and this is the major limitation of oGT. We now present a method to expand the feasible domain of oGT to $\mathbb{Z}_{N^2}$ while the non-interactive and output expressive properties of oGT remain unchanged.

Our idea is very simple, that is to separately compare each digit of the inputs, from the most significant digit down to the least significant one. More precisely, an integer $a' \in \mathbb{Z}_{N^2}$ is partitioned into two digits $0 \leq a'_1, a'_0 < N$ such that $a' = a'_1 \cdot N + a'_0$. We can see that $\mathcal{I}\{a' > b'\}$ is equivalent to

$$\mathcal{I}\{a'_1 \neq b'_1\} \cdot (\mathcal{I}\{a'_1 > b'_1\} - \mathcal{I}\{a'_0 > b'_0\}) + \mathcal{I}\{a'_0 > b'_0\}. \tag{6.2}$$

The two comparisons in Equation 6.2 can be done through oGT, and the preserved additive homomorphism of oGT allows us to perform the subtraction and addition. The main challenge for evaluating this equation lies in the computation of the bit $\mathcal{I}\{a'_1 \neq b'_1\}$, and to multiply this bit to the output of oGT.

We now show how to compute $\mathsf{Enc}\left(\mathcal{I}\{a'_1 \neq b'_1\}\right)$ from the ciphertexts $\mathsf{Enc}\left(X^{a'_1}\right)$ and $\mathsf{Enc}\left(X^{b'_1}\right)$. We also present a method to convert this ciphertext to a oGT-compatible form so that we can homomorphically multiply $\mathsf{Enc}\left(\mathcal{I}\{a'_1 \neq b'_1\}\right)$ to the private comparison result, i.e.,

$$\mathsf{oGT}\left(\mathsf{Enc}\left(X^{a'_1}\right), \mathsf{Enc}\left(X^{b'_1}\right)\right) \ominus \mathsf{oGT}\left(\mathsf{Enc}\left(X^{a'_0}\right), \mathsf{Enc}\left(X^{b'_0}\right)\right).$$

We can simply compute inequality test through $\mathsf{Enc}\left(A\right) = 1 \ominus \mathsf{Enc}\left(X^{a'_1}\right) \otimes \mathsf{Enc}\left(X^{-b'_1}\right)$. Specifically, when $a'_1 \neq b'_1$, A is a polynomial of $1 - X^{a'_1 - b'_1} \neq 0$. If $a'_1 = b'_1$, A downgrades to zero. In other words, the 0-th coefficient of A gives the inequality bit $\mathcal{I}\{a'_1 \neq b'_1\}$.

An encrypted integer value can be multiplied to the output of oGT, and thus is oGT-compatible. However, the ciphertext $\mathsf{Enc}\left(A\right)$ above is not oGT-compatible, that is because $\mathsf{Enc}\left(A\right)$ might encrypt

a polynomial more than an integer. Thus, we need to convert $\mathsf{Enc}(A)$ to an encrypted integer value if $A \neq 0$ (i.e., when $a'_1 \neq b'_1$), otherwise we should obtain a ciphertext of zero. We now show how to do this conversion.

The core idea is to employ Fermat's little theorem under the polynomial modulo $X^N + 1$. That is $A^{p^d-1} = 1 \mod X^N + 1$ if $A \neq 0$, where $p^d = 1 \mod m$. When $A = 0$, A^{p^d-1} is still zero. The naive way needs a multiplicative depth of $d \log p$ for computing $\mathsf{Enc}\left(A^{p^d-1}\right)$ from $\mathsf{Enc}(A)$. According to Halevi and Shoup [2014, 2017], we can use automorphisms to reduce this depth. That is, we first compute the exponent $\mathsf{Enc}\left(A^{p-1}\right)$ from $\mathsf{Enc}(A)$ which needs a multiplicative depth of $\log_2 p$. Then we use d automorphism maps $\mathcal{K}_j : X \mapsto X^{p^{j-1}}$ for $1 \leq j \leq d$. Finally, the product of d ciphertexts gives $\mathsf{Enc}\left(A^{p^d-1}\right)$

$$\begin{aligned}\prod_{1\leq j\leq d} \mathcal{K}_j(\mathsf{Enc}\left(A^{p-1}\right)) &= \prod_{1\leq j\leq d} \mathsf{Enc}\left(A^{(p-1)p^{j-1}}\right) \\ &= \mathsf{Enc}\left(A^{(p-1)\sum_{j\in[d]}(p^{j-1})}\right) = \mathsf{Enc}\left(A^{p^d-1}\right)\end{aligned}$$

In total, we need a multiplicative depth of $\log_2 p + \log_2 d$ and d automorphisms for this conversion step.

The FHE parameter p defines the message space which is usually determined by the application scenario. In order to limit the overhead of this conversion step, we tend choosing such p that d is a small value, e.g., $d = 2$.

6.3 Comparison with Other HE-based Solutions

Implementation Details. We instantiated oGT using HElib [Halevi and Shoup, 2017] and SEAL [Chen et al., 2017] separately. SEAL and HElib use a different representation for representing the plain polynomials from $\mathbb{Z}_t[X]/(X^N+1)$. HElib's representation for plain polynomials needs more computation effort than SEAL. As a result, for the ciphertext-plaintext comparison case, oGT instantiated from SEAL can provide a better evaluation performance than the HElib based one. However, HElib allows automorphisms which are not supported in SEAL yet. The automorphism allows the "degree-negating" operation described in Section 3.2.1.

All the programs were written in C++ and compiled with g++-6.3. We ran experiments on a machine equipped with an Intel Xeon E5-2640 v3@2.60 GHz CPU and 32GB RAM running Ubuntu 14.04

Additive HE Setting. We implement the HE based comparison protocols of Fischlin [2001], Blake and Kolesnikov [2004], Damgård et al. [2009] using the GMP library. The method in Fischlin [2001] uses the GM encryption scheme with the AND-gate extension technique [Sander et al., 1999]. The protocols of Blake and Kolesnikov [2004], Damgård et al. [2009] use an additively HE which was instantiated as the Paillier encryption [Paillier, 1999]. We also instantiated Damgård et al. [2009]

with lifted ElGamal over elliptic curves [Shigeo, 2017].

We used two security parameters $\kappa \in \{80, 128\}$. We set the public key sizes of $\{1024, 3072\}$-bits for the GM encryption and Paillier encryption, and $\{160, 256\}$-bits for the elliptic curve. In addition, we set $\lambda \in \{20, 40\}$ in the probabilistic method of Fischlin [2001]. For the FHE scheme, we set the polynomial degree $N = 2^{12}$ and $N = 2^{13}$.

TFHE Setting. TFHE [Chillotti et al., 2016] also supports output expressive private comparison by encrypting each bit of a and b and performing gate-level bootstrapping. However, for a relative small domain, e.g., less than 16 bits, the TFHE-based solution might take a longer evaluation time than oGT due to the bootstrapping operations.

Measurements. We measured the computation time of three parts: 1) time for encrypting the inputs, 2) time for evaluating comparison, and 3) time for decrypting the resulting ciphertext(s).

Evaluation Results: Additive Homomorphic Encryption. The experimental results are reported in Table 6.1. The probabilistic method of Fischlin [2001] is able to compare two encrypted integers within an error rate of $2^{-\lambda}$. The complexity of this method is linear in λ. As a result, the method of Fischlin [2001] has high computation and communication complexity for a negligible error rate, e.g., $\lambda = 40$.

For the FF-based implementations, our method outperformed Blake and Kolesnikov [2004], Fischlin [2001], Damgård et al. [2009] in terms of evaluation time. We have two settings, that is two-ciphertexts case, and single-ciphertext case. In the first case, oGT was about $45 - 90$ times faster than Fischlin [2001]. On the other hand, in the second case, oGT can be $54 - 175$ times faster than Blake and Kolesnikov [2004], Damgård et al. [2009] in terms of evaluation time. Notice that, when only one input was encrypted, oGT itself was even about $5 - 6$ times faster than with two encrypted inputs.

Our method took more evaluation time than the ECC-based implementations of Damgård et al. [2009], but the performance gap was less than one order of magnitude. Note that the elliptic curve library (i.e., Shigeo [2017]) we used is better optimized than HElib. By using the more recent FHE library SEAL, we reduced this performance gap. We thus conclude that our comparison algorithm that exploits the structure of the polynomial ring $X^m + 1$ is efficient. Remember that the output of oGT still offers additive homomorphism, and multiplicative homomorphism under a certain condition. This property is absent in all previous HE-based solutions and is helpful for constructing a higher level protocol beyond integer comparison.

Evaluation Results: TFHE. We compared the evaluation time of oGT with the TFHE-based approach from [Chillotti et al., 2016]. The results are shown in Figure 6.2. For bit length $\delta \in \{8, 9, 10\}$, we used the same parameter $N = 2^{10}$ for oGT due to the 80-bit security level requirement. We can see that the performance of the TFHE method increased linearly with the bit length while our method increased exponentially. Also, the current TFHE implementation is a symmetric encryption scheme while oGT was instantiated with an asymmetric scheme. Even though, for small domains

Table 6.1: Experimentally comparing oGT with previous HE-based private comparison protocols. δ denotes the input bit length. κ is the security parameter. FF and ECC mean that the protocols were instantiated with a finite filed scheme and an elliptic curve scheme, respectively. Timing numbers were measured as the mean of a thousand runs.

δ, κ	Method	ENC (ms)	EVAL (ms)		DEC (ms)
			ctxt–ctxt	ctxt–plain	
$12, 80$	Fischlin [2001] ($\lambda = 20$, FF)	18.63	574.44	-	2.93
	Fischlin [2001] ($\lambda = 40$, FF)	18.87	1164.23	-	3.291
	Blake and Kolesnikov [2004] (FF)	22.27	-	115.98	20.54
	Damgård et al. [2009] (FF)	26.50	-	54.24	16.99
	Damgård et al. [2009] (ECC)	0.83	-	**0.799**	0.45
	Ours (HElib)	2.93	**13.07**	2.28	1.68
	Ours (SEAL)	6.97	-	1.00	0.60
$13, 128$	Fischlin [2001] ($\lambda = 20$, FF)	21.77	672.96	-	63.80
	Fischlin [2001] ($\lambda = 40$, FF)	21.78	1346.97	-	64.97
	Blake and Kolesnikov [2004] (FF)	437.77	-	2224.54	410.32
	Damgård et al. [2009] (FF)	352.24	-	705.61	340.58
	Damgård et al. [2009] (ECC)	2.11	-	**1.98**	1.10
	Ours (HElib)	6.17	**28.22**	4.68	3.65
	Ours (SEAL)	16.54	-	4.04	2.20

(i.e., less than 16-bits), our method was more than 5 – 500 times faster than TFHE. We admit that for a larger input bit length, the TFHE method will outperform oGT, while for applications that require only a relatively small domain, oGT can be more efficient.

6.4 Application: Privacy-preserving Outsourcing Decision Tree Evaluation

6.4.1 Problem Statements

Decision tree is a fundamental and popularly used classification algorithm, in which a number of comparisons are evaluated between the elements of an input vector. A decision tree $\mathcal{T} : \mathbb{Z}^\gamma \to \{1, 2, \cdots, \zeta\}$ can be viewed as a function that maps from a γ-dimensional feature space to ζ classes. The feature space is typically $\mathbb{R}^\gamma$, and we use fixed-point representation to handle real numbers in FHE. Then client's query $\boldsymbol{a} \in \mathbb{Z}^\gamma$ is designated as feature vector. The decision tree $\mathcal{T}$ is a binary-tree consists of *internal nodes* and *leaf nodes*. Let the number of internal nodes be M', then the number of leaf nodes is $M' + 1$. Each internal node v_k is attached with a threshold $\tau_k \in \mathbb{Z}$ and a predicate $f_k(\boldsymbol{a}) = \mathcal{I}\{\boldsymbol{a}[i_k] > \tau_k\}$, where $1 \le i_k \le \gamma$ is an index of the feature vector (different internal nodes can have the same index). Each leaf node u_l is attached with an output label $z_l \in \{1, 2, \cdots, \zeta\}$.

To evaluate the decision tree on an query $\boldsymbol{x} \in \mathbb{Z}^\gamma$, we begin from the root node (i.e., v_1), and at

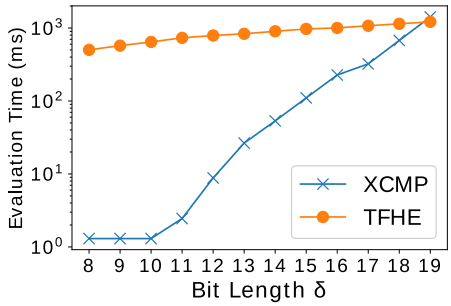


Figure 6.2: Evaluation time of oGT and the bit-wise solution from TFHE [Chillotti et al., 2016] for comparing two encrypted values aspect to various bit length.

each internal node v_k, we compute the predicate $f_k(\boldsymbol{a})$. According to the result of $f_k(\boldsymbol{a})$, we choose either the left (i.e., $f_k(\boldsymbol{a}) = 0$) or right childnode of v_k. This process is repeated until a leaf node u_{l^*} is reached. The output $\mathcal{T}(\boldsymbol{a})$ is the label z_{l^*} of the leaf node u_{l^*}. The depth of tree $\mathcal{T}$ is defined as the length of the longest path from the root node to a leaf node.

The entity that owns the decision tree model (i.e., $\{\tau_k \in \mathbb{Z}_p\}_{k\in[M']}$) is called model holder. Privacy of the decision tree model requires the values of τ_k to be kept secret.

6.4.2 Existing Private Decision Tree Evaluation Protocols

Existing protocols such as [Bost et al., 2015, Tai et al., 2017] are usually designed as a two-party computation protocol, in which a client sends its input $\boldsymbol{a}$ to a model holder, who holds the thresholds $\{\tau_k\}$, for the classification label of $\boldsymbol{a}$. These protocols offer privacy to both the client and the model holder. That is, after the execution of the protocol, the model holder does not learn about the client's input $\boldsymbol{a}$, except the number of features (i.e., input privacy), while the client learns nothing about the thresholds $\{\tau_k\}$, except the result label (i.e., model privacy). However, the existing private decision tree evaluation protocols require multiple rounds of interaction between the model holder and the client.

6.4.3 New Non-interactive Private Decision Tree Evaluation Protocol

We can obtain a non-interactive variant protocol of Tai et al. [2017] by instantiating the private comparison with oGT, as opposed to Tai et al. [2017] which used the DGK's private comparison protocol [Damgård et al., 2009]. The other parts of our protocol are identical to the semi-honest protocol of [Tai et al., 2017].

By replacing the DGK's private comparison with oGT, our protocol is superior to its origin [Tai et al., 2017] in two points. The first is that our protocol allows the client to be offline after he has sent the ciphertexts of $\boldsymbol{a}$ to the evaluator. The second is that our protocol allows the model holder to

Protocol Π_{PDT}

Input of client: feature vector $\boldsymbol{a} \in \mathbb{Z}_m^\gamma$, key pairs sk, pk
Input of server: decision tree $\mathcal{T}$
Output of client: classification result $\mathcal{T}(\boldsymbol{a})$

Client:

1. Client sends ciphertexts $\{\mathsf{Enc}\left(X^{a_i}\right)\}_{i\in[\gamma]}$ to the server attached with his public key pk.

Server:

2. For each internal node v_k, the server calls oGT of Figure 6.1

$$\mathsf{Enc}\left(A_k\right) = \mathsf{oGT}(\mathsf{Enc}\left(X^{a_{i_k}}\right), X^{\tau_k}).$$

3. For each path $\mathsf{path}(u_l)$, the server aggregates ciphertexts along the path

$$\begin{aligned}
\mathsf{Enc}\left(A'_l\right) &= \sum_{w_k^b \in \mathsf{path}(u_l)} \mathsf{Enc}\left(A_k\right) \ominus \sum_{w_k^b \in \mathsf{path}(u_l)} \mathcal{I}\{b=1\} \\
\mathsf{Enc}\left(A_l\right) &= \mathsf{Enc}\left(A'_l\right) \otimes r \oplus z_l,\ r \xleftarrow{\mathcal{U}} \mathbb{Z}_t/\{0\} \\
\mathsf{Enc}\left(\hat{A}_l\right) &= \mathsf{Enc}\left(A'_l\right) \otimes r',\ r' \xleftarrow{\mathcal{U}} \mathbb{Z}_t/\{0\}
\end{aligned}$$

4. The server sends ciphertexts $\{(\mathsf{Enc}\left(A_l\right), \mathsf{Enc}(\hat{A}_l))\}_{1\le l\le M'+1}$ to the client,

Client:

5. Client decrypts the ciphertexts received from the server to $M'+1$ pairs of polynomials i.e., $(A_l, \hat{A}_l)$ for $1 \le l \le M'+1$. Client outputs the predication result as the first coefficient of $A_{l^*}[0]$ for a index l^* such that $\hat{A}_{l^*}[0] = 0$.

Figure 6.3: Private Decision Tree Evaluation Protocol Π_{PDT}.

delegate the evaluation to a third-party evaluator, e.g., a public cloud server, without compromising the model privacy. That is, the model holder can encrypt the thresholds $\{\tau_k\}$, and place the ciphertexts on the public cloud. Then the cloud can process classification upon the (encrypted) request from the client using the ciphertexts of $\{\tau_k\}$ for the model holder. Also, during the classification process, no interaction between the cloud, the model holder and the clients is needed.

Path. We assign an *indicator* to each branch of the decision tree. Specifically, for an internal node v_k, we write w_k^b ($b \in \{0,1\}$) to denote its left branch and right branch, respectively. Then we have a unique path $\mathsf{path}(u_l)$ from the root node to each leaf node. We perform homomorphic summations along the path of each leaf node in a way that the summation along the predication path $\mathsf{path}(u_{l^*})$ evaluates to zero, while that of other paths are non-zero. This allows the client to learn $\mathcal{T}(\boldsymbol{a})$. Our Π_{PDT} protocol of Figure 6.3 requires only one round-trip communication between the client and the server, as opposed to two round-trips of Tai et al. [2017].

Theorem 14. *The protocol Π_{PDT} of Figure 6.3 privately implements the decision tree functionality*

Table 6.2: Comparing total computation time of Π_{PDT} with Tai et al. [2017]. γ and M' respectively denotes the number of features and the number of internal nodes of the decision tree. δ is the input bit length.

Data set	(γ, M')	Ours ($\delta = 13$)	Tai et al. [2017] ($\delta = 64$)
Heart-disease	$(13, 5)$	0.59s	0.25s
Housing	(13, 92)	10.27s	1.98s
Spambase	(57, 58)	6.88s	1.80s
Artificial	(16, 500)	56.37s	10.42s

Table 6.3: Performance of our Π_{PDT} protocol on the trained decision tree model. The feasible domain of oGT was $N = 2^{13}$.

Data set	ENC	EVAL	DEC
Heart-disease	44.18 ± 2.59 ms	0.32 ± 0.06 s	0.32 ± 0.06 s
Housing	46.35 ± 3.71 ms	5.11 ± 0.08 s	5.12 ± 0.08 s
Spambase	193.58 ± 7.27 ms	3.34 ± 0.07 s	3.35 ± 0.07 s
Artificial	54.62 ± 1.96 ms	28.16 ± 0.61 s	28.15 ± 0.61 s

$\mathcal{T}$ under the semi-honest model.

Proof. (Correctness.) Since oGT correctly implements the greater than functionality, and additive homomorphism is preserved, we have $A'_l[0] = \alpha_l \mod p$ by using $\mu_0 = 0$ and $\mu_1 = 1$. Thereby $A_l[0] = \alpha_l \cdot r + z_l \mod p$ and $\hat{A}_l[0] = \alpha_l \cdot r' \mod p$ with non-zero random values r and r'. The client, thus can learn z_{l^*} by finding $1 \leq l^* \leq M' + 1$ such that $\hat{A}_{l^*}[0] = 0$. It suffices to prove that $\alpha_{l^*} = 0$ and $\alpha_l \neq 0$ for all $l \neq l^*$. From the definition of decision tree and path, it is easy to see $b = f_k(\boldsymbol{a})$ for $w_k^b \in \mathsf{path}(u_{l^*})$. We can see that $\alpha_{l^*} = \sum_{w_k^b \in \mathsf{path}(u_{l^*})} b = 0$. On the other hand, for $l \neq l^*$, we have $b = 1 - f_k(\boldsymbol{a})$ for $w_b^k \in \mathsf{path}(u_l)/\mathsf{path}(u_{l^*})$. In this case, $0 < |\alpha_l| \leq |\mathsf{path}(u_l)/\mathsf{path}(u_{l^*})|$, and thus $\alpha_l \neq 0$ for all other paths that $l \neq l^*$. □

6.4.4 Evaluation

We trained decision tree models through the scikit-learn library on three real data sets from the UCI repository Lichman [2013]. We also used an artificial data to show the scalability of our Π_{PDT} protocol.

- *Heart-disease*: $\gamma = 13$ and $M' = 5$.
- *Housing*: $\gamma = 13$ and $M' = 92$.
- *Spambase*: $\gamma = 57$ and $M' = 58$.
- Artificial: $\gamma = 16$ and $M' = 500$.

Here γ and M' respectively denotes the number of features and the number of internal nodes of the decision tree. Note that in the training phase was done on plain data without any crypto.

Measurements. We measured the performance of our Π_{PDT} protocol for the case that both the client's input and the tree model are encrypted. In particular, we measured the encryption and decryption time on the client's side, and the evaluation time on the cloud's side.

Empirical Results. The benchmark performance of our Π_{PDT} protocol on the trained models are given in Table 6.3. The evaluation time on the server's side was linear with the number of internal nodes N, which can be accelerated easily through multi-threads. However, the client needs to decrypt $\mathcal{O}(2N)$ ciphertexts which still takes high computation effort.

Comparison with the-State-of-the-Art. We also compared our protocol to its origin [Tai et al., 2017], which is the most efficient HE-based private decision tree protocol. Remind that, in [Tai et al., 2017], the model thresholds $\{\tau_k\}$ were plaintext due to the DGK's private comparison, while in our protocol both the client's input and the model thresholds were encrypted. The performance details are shown in Table 6.2. We can see that the performance gap between our protocol and [Tai et al., 2017], was smaller than than 25-fold, counting the differences of bit length δ. Our protocol can be hosted on a public cloud. Thus we can take advantage of the abundant computing resources of the cloud to easily accelerate the protocol execution, e.g., by parallelism.

Chapter 7

Conclusion

In this dissertation, we tackle the problem of secure computation on the cloud. We have shown that with appropriate packing methods, FHE-based protocols can be feasible for cloud applications. Our contributions in this dissertation are summarized as follows.

Private Inner Product of Long Vectors. In Chapter 3, we deal with a problem of computing the inner product of two encrypted large size vectors. As we have shown, the naive approach would be too expensive to use. To overcome this difficulty, we propose forward backward packing, which encodes long vectors as polynomials before encryption, and it enables us to compute the inner product of the vectors with a much smaller number of homomorphic multiplications. Specifically, the number of homomorphic multiplications are reduced from $\mathcal{O}(n)$ to $\mathcal{O}(n/N)$ where n is the size of vectors and N is the FHE parameter. As a concrete application, we show how to use the proposed inner product protocol to conduct secure χ^2 test and linkage equilibrium (Figure 3.4) with a large clinical and genomic data, on the cloud. Empirical results show that our approach is about $2000\times$ faster than the previous cryptographic solution for χ^2 and linkage equilibrium.

Iterative Computation on Encrypted Vectors. In Chapter 4, we present three feasible FHE primitives, i.e., Π_{MP}, Π_{bGT} and Π_{oGT}, which enable us to perform iterative computation on encrypted vectors, including matrix multiplication and greater than. Using these primitives, we develop a framework of secure outsourcing statistical analysis to the cloud. Specifically, we show how to evaluate descriptive and predictive statistics from encrypted values, including mean, covariance, histogram, contingency table with cell suppression, k-percentile, decision tree, principal component analysis and linear regression. The experiment results show that FHE is usable for non-trivial cloud applications when appropriate packings and encoding methods are used.

Communication Efficient Secure Matrix Multiplication. In Chapter 5, we present a communication efficient secure inner product protocol via a newly proposed double packing. The proposed protocol enables a communication efficient matrix multiplication protocol (Π_{SMP} of Figure 5.3),

which reduces the cubic communication overhead of the previous method to a quadratic communication overhead. Empirically, the amount of data transferred by Figure 5.3 was only 1.9% of the existing OT-based methods. As a concrete application, we employ the protocol Π_{SMP} to two existing privacy-preserving machine learning frameworks, i.e., MiniONN and SecureML. Experiment results show that our protocol can save more than 90% communication cost of these frameworks while keeping a comparable computation time. We consider that Π_{SMP} can help forwarding the development of more practical and useable privacy-preserving machine learning cloud-based applications.

Appendix A

Techniques to Implement Fast Homomorphic Encryption

A good programming implementation of the underlying FHE library is also critical. In this chapter, we introduce some techniques that help us to write efficient codes that implement BGV's scheme.

A.1 Polynomial Multiplication

The most important operation of constructing the BGV scheme (and other ring-based homomorphic encryption) is *multiplication of polynomials* modulo $X^N + 1$ and some prime t, i.e.,

$$A \cdot B \bmod (X^N + 1, t)$$

for polynomials $A, B \in \mathbb{A}_t$. The naive polynomial multiplication takes $\mathcal{O}(N^2)$ operations which might be impractical, since the parameter N of an FHE scheme is usually large, e.g., $N = 2^{13}$. The discrete Fourier transform (also named as number theoretic transform) is the classic algorithm to perform fast polynomial multiplications. Let $\mathsf{NTT} : \mathbb{A}_t \mapsto \mathbb{Z}_t^N$ as the forward transform, and denote NTT^{-1} as the backward transform. NTT has the following property.

$$\mathsf{NTT}^{-1}(\mathsf{NTT}(A) \circ \mathsf{NTT}(B)) = A \cdot B \bmod (X^N + 1, t),$$

where $\circ$ is the element-wise multiplication of vectors. By using fast Fourier transform, such as Cooley-Tukey algorithm [Cooley and Stern, 1965], the forward and backward transform take $\mathcal{O}(N \log_2 N)$ operations. Since the element-wise multiplication is linear, the polynomial multiplication can be done within $\mathcal{O}(N \log_2 N)$ operations.

A classic NTT algorithm that uses Cooley-Tukey needs to bit-reversed reorder the coefficients

Protocol NTT on the Cooley-Tukey Butterfly

Inputs: Polynomial $A := \sum_{i=0}^{N-1} a_i X^i$ where N is power of 2 and $t = 1 \bmod 2N$. The $2N$-th primitive root of unit ψ aspect to t. A pre-computed table $\Psi \in \mathbb{Z}_t^N$ stores powers of ψ in bit-reversed order.

1. **For** $0 \leq m < \log_2 N$ **do**
   2. $n = 2^{\log_2 N - m - 1}$
   3. **For** $0 \leq i < 2^m$ **do**
      3.1. $d = 2 \cdot i \cdot n$
      3.2. $\eta = \Psi[2^m + i]$
      3.3. **For** $0 \leq k < n$ **do**

$$\begin{bmatrix} a_{d+k} \\ a_{d+k+n} \end{bmatrix} = \eta \cdot \begin{bmatrix} a_{d+k} + a_{d+k+n} \\ a_{d+k} - a_{d+k+n} \end{bmatrix} \bmod t \tag{A.1}$$

4. Output $\{a_i\}$ for $0 \leq i < N$.

Figure A.1: Forward Number Theoretic Transform

$\{a_i\}$ and multiplying the coefficients with the $2N$-th primitive root of unit. Indeed we can save these operations by mixing up two kinds of butterflies, i.e., using Cooley-Tukey butterfly in NTT and using Gentleman-Sande butterfly in INTT [Longa and Naehrig, 2016].

We can see from Figure A.1 and Figure A.2 that the most basic operation of the NTT is the so-called butterfly operation (Equation A.1), i.e., $(x, y) \rightarrow (x + y, w \cdot (x - y))$. Notice that, the additions and multiplications are performed over the finite field $\mathbb{Z}_t$. We use the optimization from Harvey [2014] for the butterfly operation.

A.1.1 Faster Butterfly for Number Theoretic Transforms

The naive implementation of Equation A.1 would need three correction steps to make sure the results are inside the range $[0, t)$. However, these correction steps would introduce a large overhead due to branch misprediction. Harvey [2014] proposes a method to eliminate two of them. Figure A.3 follows the optimized butterfly proposed by Harvey [2014] which also leverages Shoup's trick for multiplication reduction. That is, we tend to choose β as power of 2 so that the division and modulo by β can be done with cheap bit-wise shift and mask operations, respectively.

A.2 Residue Number System

Multi-word values are necessary for constructing BGV's scheme. That is, the value in the moduli chain $q_0 < q_1 < \cdots < q_L$ is usually large, e.g., $q_L > 2^{300}$. From the view of programming, we can use multi-precision library such as the GMP library for this. However, these multi-word

Protocol NTT^{-1} on the Gentleman Sande Butterfly

Inputs: Vector $\boldsymbol{a} := [a_0, a_1, \cdots a_{N-1}] \in \mathbb{Z}_t^N$ in bit-reversed ordering, where N is power of 2 and $t = 1 \bmod 2N$. The inverse of the $2N$-th primitive root of unit $\psi^{-1} \bmod t$. A pre-computed table $\Psi^{-1} \in \mathbb{Z}_t^N$ stores the power of ψ^{-1} in the bit-reversed order.
Output: $\mathsf{NTT}^{-1}(\boldsymbol{a})$ in standard ordering.

1. **For** $0 \leq m < \log_2 N$ **do**
   2. $h = 2^{\log_2 N - m - 1}$
   3. **For** $0 \leq i < h$ **do**
      3.1. $\eta = \Psi^{-1}[h + i]$
      3.2. $d = 2 \cdot i \cdot 2^m$
      – **For** $0 \leq k < 2^m$ **do**

$$\begin{bmatrix} a_{d+k} \\ a_{d+k+2^m} \end{bmatrix} = \begin{bmatrix} a_{d+k} + a_{d+k+2^m} \\ \eta \cdot (a_{d+k} - a_{d+k+2^m}) \end{bmatrix} \bmod t$$

4. Output $\{a_i \cdot N^{-1} \bmod t\}$ for $0 \leq i < N$.

Figure A.2: Backward Number Theoretic Transform

implementations are usually a few magnitudes slower than the naive machine word operations. We introduce an alternative method to achieve multi-word precision using the naive machine word only. Before describing the alternative method, we need some notations.

- Let $p_1, p_2, \cdots, p_k$ be k prime numbers that fix into a machine word.
  - Let p be the product of these k primes, i.e., $p = \prod_i p_i$.
  - Let $p_i^* = p/p_i \in \mathbb{Z}$ and $\tilde{p}_i = (p_i^*)^{-1} \bmod p_i$, i.e., $\tilde{p}_i \cdot p_i^* = 1 \bmod p_i$.
- For an integer $0 \leq x < p$, we write $[x]_p$ to represent the vector

$$[x_1, x_2, \cdots, x_k] \text{ s.t. } x_i = x \bmod p_i.$$

- For a polynomial $A \in \mathbb{Z}_p[X]/(X^N + 1)$, we write $[A]_{p_i}$ to denote the residue of coefficients of A aspect to p_i, i.e., $[A]_{p_i} = [A[0] \bmod p_i, A[1] \bmod p_i, \cdots, A[N-1] \bmod p_i]$.

Indeed, $[x]_p$ is so-called residue number system (RNS) of x. For the RNS representation of values $0 \leq x, y < p$, we have the following properties.

$$[x]_p \dot{+} [y]_p = [x + y \bmod p]_p$$
$$[x]_p \circ [y]_p = [x \times y \bmod p]_p,$$

Protocol Harvey's Butterfly

Inputs: $0 < \eta < t$, $\eta' = \lfloor \eta \cdot \beta / t \rfloor$ for some $\beta > 4t$. $0 \leq x, y < 2t$.
Outputs: $x' = x + y \bmod t$ and $y' = \eta \cdot (x - y) \bmod t$.

1. $x' = x + y$
   - if $x' \geq 2t$ then $x' = x' - 2t$.
2. $q = \lfloor \eta' \cdot T / \beta \rfloor$ where $T = x - y + 2t$.
3. $y' = (\eta \cdot T - q \cdot t) \bmod \beta$.

Figure A.3: Faster Butterfly for Number Theoretic Transforms

where $\dot{+}$ and $\circ$ is the element-wise addition and multiplication of vectors, respectively. In other words, by using RNS representation, we can achieve multi-word precision for addition and multiplication over the finite field. Most importantly, the RNS representation is compatible with the NTT algorithm, since NTT algorithm involves only additions and multiplications over the finite field. In more details, we represent the coefficients of polynomials in the RNS format. At the result, we need to repeat the NTT algorithm through each residue of the polynomial. Since each residue would not interleave with each other during the NTT algorithm, we tend to store the RNS representation of polynomials in the residue-order

$$[[A]_{p_1}, [A]_{p_2}, \cdots, [A]_{p_k}].$$

Notice that, parallelism can be used to accelerate the NTT algorithm on the RNS formatted polynomials, since each residue is totally independent to each other.

A.2.1 Division on RNS

The homomorphic multiplication of BGV' scheme involves some kind of division operations, which is tricky if we use the RNS format, since it preserves only the additive and multiplicative properties. For a general division operation, we might need to convert back from the RNS format to a multi-precision format such as using GMP, and then perform the division and then convert back to the RNS format. Fortunately, the division needed in the BGV's scheme is not that general. It has some constructive properties. We now introduce the *rescale* routine from Halevi et al. [2018], which allows us to perform the needed division operation in the homomorphic multiplication without using multi-precision values.

Let $x \in \mathbb{Z}_p$ be given in the RNS format $[x_1, x_2, \cdots, x_k]$, and let $p' \in \mathbb{Z}$ be an integer modulus $p' > 2$. We want to "scale-down" x by a factor p'/p, namely to compute the integer $y = \lceil p'/p \cdot x \rfloor \in$

$\mathbb{Z}_{p'}$.

$$y := \lceil \frac{p'}{p} x \rfloor = \lceil (\sum_i x_i \cdot \tilde{p}_i \cdot p_i^* \cdot \frac{p'}{p}) - v' \cdot p \cdot \frac{p'}{p} \rfloor$$
$$= \lceil \sum_i x_i \cdot (\tilde{p}_i \cdot \frac{p'}{p_i}) \rfloor - v' \cdot p' = \lceil \sum_i x_i \cdot (\tilde{p}_i \cdot \frac{p'}{p_i}) \rfloor \bmod p'$$

The first equation comes from the fact that $x = \sum_i x_i \cdot \tilde{p}_i \cdot p_i^* - v' \cdot p$ for some $v' \in \mathbb{Z}$. In the context of homomorphic multiplication, we can pre-process the rational numbers $p' \cdot \tilde{q}_i / q_i \in [0, p')$, and separate into their integer and fractional parts:

$$p' \cdot \tilde{q}_i / q_i = \alpha_i + \beta_i, \text{ with } \alpha_i \in \mathbb{Z}_{p'} \text{ and } \beta_i \in [0, 1).$$

With (α_i, β_i) pre-computed, we can rescale x to y via two sums: $\alpha = \sum_i x_i \cdot \alpha_i \bmod p'$ and $\beta = \lceil \sum_i x_i \cdot \beta \rfloor$, and then output $y = \alpha + \beta \bmod p'$.

A.2.2 Barrett Reduction

Barrett reduction [Barrett, 1986] is a fast method used to perform $x \bmod t$, assuming $0 \le x < t^2$ and t is constant. The general idea behinds Barrett reduction is to compute $x - \lfloor x \cdot s \rfloor \cdot t$ where $s \approx \frac{1}{t}$. As long as s is computed with sufficient accuracy, this result is exact. However division can be expensive. On the other hand, Barrett reduction approximates $1/t$ with a value $t'/2^k$ with an approximation parameter k. Division by 2^k is just a right-shift and thus is cheap. The value t' is generally set as $t' = \lfloor 2^k / t \rfloor$. Given t' and k, Barrett reduction on $0 \le x < t^2$ is working as follows.

1. $q = (x \cdot t') \gg k$.
2. $y = x - q \cdot t$.
3. output y if $y < t$, otherwise output $y - t$.

For a larger k, we have a better approximation, and we need to make sure that the approximation is good enough for reducing values in the range $[0, t^2)$, which is required by the lazy reduction technique in the next section. Suppose the machine word size is w and the limb size of t is w', i.e., $t < 2^{w'} \le 2^w$. We use a large approximation parameter, i.e, $k = 2w$. Notice that, using a such large k, the multiplication $x \cdot t'$ will result at a multi-word value. To avoid using multi-precision library, we need one more constraint on the limb size w, i.e., $w' \le w - 2$, but now we can only store the lowest w-bit of $\lfloor 2^k / t \rfloor$ because $2^k / t > 2^w$. The Barrett reduction we used is given as follows.

Setup: Suppose that $k = 2w$ and $t < 2^{w'}$. Let $\tilde{t}$ the lowest w-bit of $\lfloor 2^k / t \rfloor$ and $\delta = w - w'$.

1. $q = (\tilde{t} \cdot (x \gg w) + (x \ll \delta)) \gg w$.

2. $y = x - q \cdot t$.

3. output y if $y < t$, otherwise output $y - t$.

The correctness of Step 1 can be seen as follows. In the origin Barrett reduction, the step $(x \cdot t') \gg k$ is computing the highest w-bit of $(x \cdot t')$, since $k = 2^{2w}$ is used. Also, we have $t' = 2^{\delta} \cdot 2^{w} + \tilde{t}$ with $\tilde{t} < 2^{w}$. In other words, t' is represented in its w-radix format. We can rewrite $0 \leq x < 2^{2w'}$ in its w-radix format, i.e., $x = 2^{w} \cdot x_1 + x_0$ for $0 \leq x_0, x_1 < 2^{w}$. Thereby the multiplication $x \cdot t'$ is decomposed as

$$\begin{aligned} x \cdot t' &= (x_1 \cdot 2^{w} + x_0) \cdot (2^{\delta} \cdot 2^{w} + \tilde{t}) \\ &= (2^{\delta} \cdot x_1) \cdot 2^{2w} + (2^{\delta} \cdot x_0 + \tilde{t} \cdot x_1) \cdot 2^{w} + x_0 \cdot \tilde{t} \end{aligned}$$

The highest w-bit is $2^{\delta} \cdot x_1 + (2^{\delta} \cdot x_0 + x_1 \cdot \tilde{t}) \gg w$ which equals to $(2^{\delta} \cdot 2^{w} \cdot x_1 + 2^{\delta} \cdot x_0 + \tilde{t} \cdot x_1) \gg w$, and which is exactly $((x \ll \delta) + \tilde{t} \cdot (x \gg w)) \gg w$.

A.2.3 Lazy Reduction

Lazy reduction is used to speed up "inner product" computation over the finite filed, such as $\sum_i x_i \cdot y_i \bmod t$ for $x_i, y_i \in \mathbb{Z}_t$. We tend to defer the modulo reduction for each multiplication, and perform just a single modulo reduction after the summation. To do so, we need to present double machine-word. For example, if $x_i, y_i < 2^{64}$ then we need a data structure to represent 128-bit integers. Indeed, during the computation of inner product, we do need some modulo reduction, since we need to make sure no overflow is incurred from the summations. The number of summations we can skip before the modulo reduction depends on the limb size of t. Suppose the machine word size is w and the limb size of t is w', i.e., $t < 2^{w'} \leq 2^{w}$. The number of summations we can skip is $2^{2w'-2w} - 1$.

Bibliography

Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In *Progress in Cryptology - LATINCRYPT 2014 - International Conference on Cryptology and Information Security in Latin America.*, pages 3–27, Florianópolis, Brazil, 2014a. Springer.

Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 619–631, Dallas, Texas, USA, 2017.

Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 19–38, San Jose, California, USA, 2017.

Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *Computer Security - ESORICS 2017 - European Symposium on Research in Computer Security, Oslo, Norway, Proceedings, Part II*, pages 494–512, 2017.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - International Conference on the Theory and Application of Cryptology and Information Security. Proceedings, Part I*, pages 3–33, Hanoi, Vietnam, 2016.

Gahin. http://www.gahin.org/http://www.gahin.org/, 2015.

Nationwide health information network. https://www.healthit.gov/policy-researchers-implementers/nationwide-health-information-network-nwhin., 2015.

e-estonia. https://e-estonia.com/solutions/interoperability-services/x-road/, 2018.

Microsoft. Hosting and cloud study 2016. https://www.microsoft.com/en-us/download/details.aspx?id=52045, 2016.

Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - Annual Cryptology Conference. Proceedings*, pages 850–867, Santa Barbara, CA, USA, 2012a.

Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012 -Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings*, pages 465–482, Cambridge, UK, 2012b.

David Wu and Jacob Haven. Using homomorphic encryption for large scale statistical analysis. http://cs.stanford.edu/people/dwu4/FHE-SI_Report.pdf, 2012.

Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812, 2014.

Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. *IACR Cryptology ePrint Archive*, 2018:465, 2018.

Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings*, pages 1–23, French Riviera, 2010.

Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

Shai Halevi and Victor Shoup. HELib. http://shaih.github.io/HElib, 2017. Accessed: 2017-04-10.

Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library - SEAL v2.1. *IACR Cryptology ePrint Archive*, 2017:224, 2017.

Shai Halevi and Victor Shoup. Bootstrapping for HElib. In *Advances in Cryptology - EUROCRYPT 2015 - Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings, Part I*, pages 641–670, Sofia, Bulgaria, 2015.

Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013.*, pages 65–76, Berlin, Germany, 2013.

Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. *BMC medical informatics and decision making*, 15(5):S1, 2015.

Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM Journal on Computing*, 40(1): 122–141, 2011.

Joppe W. Bos, Kristin E. Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014a.

Kristin E. Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil*, pages 3–27, 2014b.

Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014b.

Andreas Ziegler and Inke R König. *A Statistical Approach to Genetic Epidemiology: Concepts and Applications*. John Wiley & Sons, Berlin, 2nd edition, 2010.

Shai Halevi and Victor Shoup. Algorithms in HElib. In *Advances in Cryptology - CRYPTO 2014 - Annual Cryptology Conference, Proceedings, Part I*, pages 554–571, Santa Barbara, CA, USA, 2014.

Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 359–376, San Jose, CA, USA, 2015.

Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139, 1999.

Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium - ICALP 2008 - Proceedings, Part II*, pages 486–498, Reykjavik, Iceland, 2008.

Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW'11, Proceedings of the 2011 ACM Cloud Computing Security Workshop, Co-located with CCS 2011.*, pages 113–124, Chicago, IL, USA, 2011.

Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In *Applied Cryptography and Network Security - ACNS 2013 - International Conference. Proceedings*, pages 102–118, Banff, AB, Canada, 2013.

Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Revised Selected Papers*, pages 1–21, Seoul, Korea, 2012.

Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS 2015 - Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 2015.

Shubha U Nabar and Nina Mishra. Releasing private contingency tables. *Journal of Privacy and Confidentiality*, 2(1):9, 2009.

Nancy Kirkendall and Gordon Sande. Comparison of systems implementing automated cell suppression for economic statistics. *Journal of Official Statistics*, 14(4):513, 1998.

Chun-Hua Guo and Nicholas J. Higham. A schur-newton method for the matrix pth root and its inverse. *SIAM J. Matrix Analysis Applications*, 28(3):788–804, 2006.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015 - Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 2015.

Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012, Annual International Cryptology Conference, Proceedings*, pages 643–662, Santa Barbara, CA, USA, 2012.

Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016a.

Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. *International Journal of Advancements in Computing Technology*, 1(4): 323–324, 2009.

Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pages 535–548, Berlin, Germany, 2013.

Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pages 801–812, Berlin, Germany, 2013.

David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. Privacy-preserving shortest path computation. In *NDSS 2016 - Annual Network and Distributed System Security Symposium*, 2016.

Anima Singh and John V. Guttag. A comparison of non-symmetric entropy-based classification trees and support vector machine for cardiovascular risk stratification. In *EMBC 2011*, pages 79–82, Boston, MA, USA, 2011.

Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Proceedings on Privacy Enhancing Techniques (PoPETs) 2009*, pages 235–253, Seattle, WA, USA, 2009.

Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing STOC 1982*, pages 365–377, San Francisco, CA, USA, 1982.

Andrew C Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science, FOCS '82*, pages 160–164, Chicago, IO, USA, 1982.

Wen-jie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In *NDSS 2017 - Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 2017.

Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PoPETs*, 2017(4):345–364, 2017.

Shuang Wang, Yuchen Zhang, Wenrui Dai, Kristin E. Lauter, Miran Kim, Yuzhe Tang, Hongkai Xiong, and Xiaoqian Jiang. HEALER: homomorphic computation of exact logistic regression for secure rare disease variants analysis in GWAS. *Bioinformatics*, 32(2):211–218, 2016b.

M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. *CoRR*, abs/1801.03239, 2018.

Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, Proceedings*, San Francisco, CA, USA, August 8-12, 2011.

Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 411–428, San Jose, CA, USA, 2015.

Donald Beaver. Precomputing oblivious transfer. In *Advances in Cryptology - CRYPTO '95, Annual International Cryptology Conference, Proceedings*, pages 97–109, Santa Barbara, CA, USA, 1995.

Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015, Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings*, pages 220–250, Sofia, Bulgaria, 2015.

Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *IEEE Annual Symposium on Foundations of Computer Science, FOCS 2011. Proceedings*, pages 120–129, Palm Springs, CA, USA, 2011.

Pradeep Kumar Mishra, Dung Hoang Duong, and Masaya Yasuda. Enhancement for secure multiple matrix multiplications over ring-lwe homomorphic encryption. In *Information Security Practice and Experience - ISPEC 2017 - International Conference. Proceedings*, pages 320–330, Melbourne, VIC, Australia, 2017.

Dung Hoang Duong, Pradeep Kumar Mishra, and Masaya Yasuda. Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra Mountains Mathematical Publications*, 67 (1):69–83, 2016.

Gail Brion, Chandramouli Viswanathan, TR Neelakantan, Srinivasa Lingireddy, Rosina Girones, David Lees, Annika Allard, and Apostolos Vantarakis. Artificial neural network prediction of viruses in shellfish. *Applied and environmental microbiology*, 71(9):5244–5253, 2005.

Igor I Baskin, David Winkler, and Igor V Tetko. A renaissance of neural networks in drug discovery. *Expert opinion on drug discovery*, 11(8):785–795, 2016.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf, 2009.

Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012 - Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–482, Cambridge, UK, 2012c.

Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001 , Proceedings*, pages 457–472, San Francisco, CA, USA, 2001.

Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, Proceedings*, pages 515–529, 2004.

Tomas Sander, Adam L. Young, and Moti Yung. Non-interactive cryptocomputing for nc^1. In *Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 554–567, New York, NY, USA, 1999.

Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238, 1999.

Misunari Shigeo. mcl: a generic and fast pairing-based cryptography library. https://github.com/herumi/mcl, 2017. Accessed: 2017-11-10.

James W. Cooley and Frank Stern. Solution of the equation for wave propagation in layered slabs with complex dielectric constants. *IBM Journal of Research and Development*, 9(5):405–411, 1965.

Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *Proceedings of - CANS 2016- the 15th International Conference of Cryptology and Network Security.*, pages 124–139, Milan, Italy, 2016.

David Harvey. Faster arithmetic for number-theoretic transforms. *Journal of Symbolic Computation*, 60:113–119, 2014.

Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. *IACR Cryptology ePrint Archive*, 2018:117, 2018.

Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology - CRYPTO '86 - Annual Cryptology Conference. Proceedings*, pages 311–323, Santa Barbara, CA, USE, 1986.

Publications List

- Wen-jie Lu and Jun Sakuma. More Practical Privacy-Preserving Machine Learning as A Service via Efficient Secure Matrix Multiplication. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '18, pages 25 – 36, Toronto, Canada, October 15 – 19, 2018.

- Wen-jie Lu, Jun-jie Zhou, and Jun Sakuma. Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, AsiaCCS '18, pages 67 – 74, Incheon, Republic of Korea, June 4 – 8, 2018.

- Wen-jie Lu, Shohei Kawasaki, Jun Sakuma. Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data. In *Proceedings of the 2017 on Network and Distributed System Security Symposium*, NDSS '17, San Diego, USA, February 26 – March 1, 2017.

- Wen-jie Lu, Yoshimi Yamada, Jun Sakuma. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. In *Proceedings of BMC Medical Informatics and Decision Making 2015, 15(Suppl 5).*

- Wen-jie Lu, Yoshimi Yamada, Jun Sakuma. Efficient Secure Outsourcing of Genome-Wide Association Studies. In *Proceedings of the 2nd International Workshop on Genome Privacy and Security*, GenoPri15, pages 3 – 6, San Jose, CA, USA. May 21, 2015.