11-2016

An efficient privacy-preserving outsourced calculation toolkit with multiple keys

Ximeng LIU
Singapore Management University, xmliu@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Kim-Kwang Raymond CHOO
University of Texas at San Antonio

Jian WENG
Jinan University

DOI: https://doi.org/10.1109/TIFS.2016.2573770

Published in IEEE Transactions on Information Forensics and Security, 2016 Nov, Volume 11, Issue 11, 2401-2414.
http://doi.org/10.1109/TIFS.2016.2573770

An Efficient Privacy-Preserving Outsourced Calculation Toolkit With Multiple Keys

Ximeng Liu, *Member, IEEE*, Robert H. Deng, *Fellow, IEEE*,
Kim-Kwang Raymond Choo, *Senior Member, IEEE*, and Jian Weng

***Abstract*— In this paper, we propose a toolkit for efficient and privacy-preserving outsourced calculation under multiple encrypted keys (EPOM). Using EPOM, a large scale of users can securely outsource their data to a cloud server for storage. Moreover, encrypted data belonging to multiple users can be processed without compromising on the security of the individual user's (original) data and the final computed results. To reduce the associated key management cost and private key exposure risk in EPOM, we present a distributed two-trapdoor public-key cryptosystem, the core cryptographic primitive. We also present the toolkit to ensure that the commonly used integer operations can be securely handled across different encrypted domains. We then prove that the proposed EPOM achieves the goal of secure integer number processing without resulting in privacy leakage of data to unauthorized parties. Last, we demonstrate the utility and the efficiency of EPOM using simulations.**

***Index Terms*— Privacy-preserving, homomorphic encryption, outsourced computation, multiple keys.**

I. Introduction

CLOUD computing due to its capability to support real-time and massive storing and processing of data, is increasingly used in domains such as Internet of Things (IoT) [1], e-commerce [2], and scientific research [3]–[5]. It is, therefore, unsurprising that cloud computing is considered a viable solution to address the demands due to a significant increase in storage media and the number of digital and Internet-connected devices (e.g. Internet of Things and medical devices). For example, in 2011, the U.S Federal Government adopted a 'Cloud First' policy which requires government agency's Chief Information Officers to implement a cloud-based service whenever there was a secure, reliable, and cost-effective option [6], [7]. Despite the benefits afforded by the use of cloud computing, data security and privacy remain areas of ongoing focus. For example, in the final US Government Cloud Computing Technology Roadmap published by the National Institute of Standards and Technology (NIST), security and privacy are considered one of the high-priority requirements [8], and a number of dedicated cloud computing research labs, such as [9] and [10], have been established in recent years.

In attempts to conserve resources, reduce operational costs, and maintain efficiency, cloud service providers often store data belonging to multiple users on the same server (i.e. multi-tenancy) [11]. Therefore, different users should be distributed with an individual key (i.e. multiple keys [12], a.k.a, multi-key), to avoid multi-tenancy related attacks (e.g. a user's private data viewed by other unauthorized users). One application of the multi-key setting is e-healthcare cloud [13], where patients can transmit and store their health related information (e.g. patient's heart rate, blood pressure and glucose levels) on the hospital's cloud servers. This will facilitate diagnosis of the patients' physical condition based on the information. It is, however, important to ensure the security and privacy of patient's health and other personally identifiable information (PII), such as health status. The privacy of decision making model used is also considered by the e-health service provider as a trade secret. One way to achieve the security and privacy of the data is to issue all users (e.g. patients and service provider) different (unique) keys. In addition, an e-health service provider uses patients' health and PII (encrypted under different keys) in their training decision model. For example, historical medical data are used to train Naïve Bayesian classifier in Clinical Decision Support System (CDSS) [14]. However, achieving secure calculation over the data under multiple keys without comprising the privacy of individual data remains a hard problem.

In this paper, we propose an **E**fficient **P**rivacy-preserving **O**utsourced calculation framework with **M**ultiple keys (EPOM) to address the above-mentioned challenge. We regard the contributions of this paper to be four-fold, namely:

- Our proposed EPOM is designed to allow different data providers to outsource their data (e.g. data belonging to users from different data providers) to the cloud server for secure storage and processing.

Manuscript received December 19, 2015; revised April 6, 2016; accepted May 4, 2016. Date of publication May 25, 2016; date of current version August 26, 2016. This work was supported in part by the Singapore National Research Foundation through the National Cybersecurity Research and Development Program under Grant NRF2014NCR-NCR001-012 and in part by the National Natural Science Foundation of China under Grant 61370078, Grant 61502400, Grant 61502248, Grant 61402112, and Grant 61402109. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Giuseppe Persiano.

X. Liu and R. H. Deng are with the School of Information Systems, Singapore Management University, Singapore 188065 (e-mail: snbnix@gmail.com; robertdeng@smu.edu.sg).

K.-K. R. Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA, also with the School of Information Technology and Mathematical Sciences, University of South Australia, Adelaide, SA 5001, Australia, and also with the School of Computer Science, China University of Geosciences, Wuhan 430074, China (e-mail: raymond.choo@fulbrightmail.org).

J. Weng is with the School of Information Technology, Jinan University, Guangzhou 100044, China (e-mail: cryptjweng@gmail.com).

- We construct a new cryptographic primitive, Distributed Two Trapdoors Public-Key Cryptosystem (DT-PKC), which is deployed in EPOM to split a strong private key into different shares. This will allow us to reduce the risk of private key leakage and private key management cost in a multi-key setting.
- We build a privacy-preserving outsourced calculation toolkit of integer numbers with multiple keys. The toolkit consists of commonly used elementary operations, such as multiplication, division, comparison, sorting, sign bit acquisition, equivalence testing and greatest common divisor. The extension of the toolkit can also securely store and process real numbers.
- We then demonstrate the utility of EPOM using a purposefully built simulator in Java, which demonstrates that our proposal can effectively and securely outsources the storage and process of data in a multiple keys setting.

The remainder of this paper is organized as follows. In Section II, we describe the preliminaries required in the understanding of our proposed EPOM. In Section III, we formalize the system model, as well as outlining the problem statement and the attacker model. Then, we present DT-PKC and secure multi-key calculation toolkit of integer in Sections IV and V, respectively. The security analysis and performance evaluation are presented in Sections VI and VII, respectively. Related work is discussed in Section VIII. Section IX concludes this paper.

II. Preliminary

In this section, we outline the notations used in the paper. We also define the Additive Homomorphic Cryptosystem (AHC) and Secure Bit-Decomposition (**SBD**) Protocol, which are the building blocks in the proposed EPOM.

Throughout the paper, we use pk_i and sk_i to denote the public key and weak private key of party i, respectively. pk_Σ denotes the joint public key (see Section IV for the construction), SK denotes the system strong private key, and $SK^{(1)}$ and $SK^{(2)}$ denote the partial strong private keys. Furthermore, we denote $[x]_{pk_i}$ as the encrypted data x under pk_i, $D_{sk}(\cdot)$ as the decryption algorithm using sk, $\mathcal{L}(x)$ as the bit-length of x, and $|x|$ to represent the absolute value of x.

A. Additive Homomorphic Cryptosystem

Suppose $[m_1]_{pk}$ and $[m_2]_{pk}$ are two additive homomorphic ciphertexts under the same public key pk in an additive homomorphic cryptosystem. The additive homomorphic cryptosystem (e.g. Paillier cryptosystem [15] and Benaloh cryptosystem [16]) has the **additive homomorphism** property:

$$D_{sk}([m_1]_{pk} \cdot [m_2]_{pk}) = m_1 + m_2.$$

B. Secure Bit-Decomposition Protocol (SBD)

Suppose that there are two parties in the protocol, Alice and Bob. Bob holds the AHC encrypted value $[x]_{pk}$, where $0 \leq x < 2^\mu$ and μ is the domain size of x in bits. We also remark that x is known to neither Alice nor Bob.

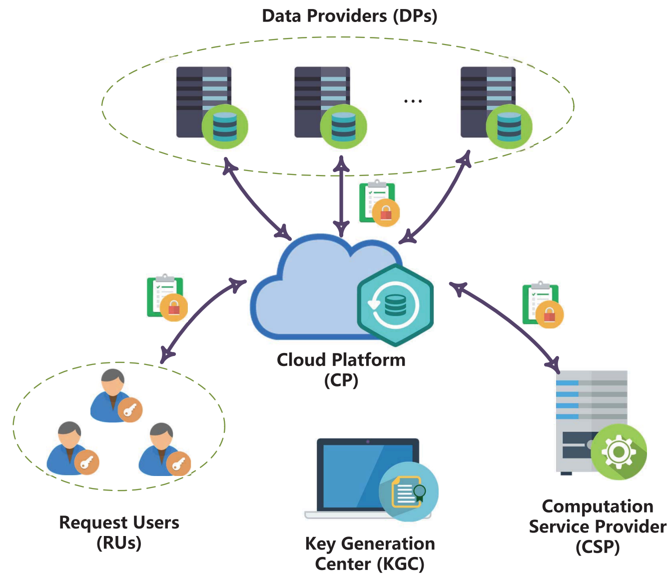

Fig. 1. System model under consideration.

Let $(x_{\mu-1}, \cdots, x_0)$ denotes the binary representation of x, where x_0 and $x_{\mu-1}$ are the least and most significant bits, respectively. The goal of **SBD** is to convert the encryption of x into the encryption of the individual bits of x, without disclosing any information regarding x to both parties. More formally, we define the **SBD** protocol as follows:

$$([x_{\mu-1}]_{pk}, \cdots, [x_0]_{pk}) \leftarrow \mathbf{SBD}([x]_{pk}).$$

We refer the interested reader to [17] for the detailed construction of the **SBD** protocol.

III. System Model & Privacy Requirement

In this section, we formalize the EPOM system model, outline the problem statement, and define the attack model.

A. System Model

In our system, we mainly focus on how the cloud server responds to user request in a privacy-preserving manner. The system comprises a Key Generation Center (KGC), a Cloud Platform (CP), a Computation Service Provider (CSP), Data Providers (DPs) and Request Users (RUs) – see Fig. 1.

1. KGC: The trusted KGC is tasked with the distribution and management of both public and private keys in the system.

2. DP: Generally, a DP will use its public key to encrypt some data, before storing the encrypted data with a CP.

3. CP: A CP has 'unlimited' data storage space, and stores and manages data outsourced from all registered RUs. A CP also stores all intermediate and final results in encrypted form. Furthermore, a CP is able to perform certain calculations over encrypted data.

4. CSP: A CSP provides online computation services to users. The CSP is also able to partial decrypt ciphertexts sent by the CP, perform certain calculations over the partial decrypted data, and then re-encrypt the calculated results.

5. RUs: The goal of a RU is to request a CP to perform some calculations over the encrypted data under multiple keys.

After the calculation has been performed, the result can be decrypted by RU upon successful authentication.

B. Problem Statement

Consider a database T that contains α records with β dimensions $x_{i,j}$ $(1 \leq i \leq \alpha; 1 \leq j \leq \beta)$, where $x_{i,j}$ is a integer number and belongs to DP k. Such data need to be encrypted prior to being outsourced to a CP for storage and maintenance. A RU can issue a query to the CP in order to obtain some statistic information about T. For example, the RU can query for the mean and variance over some dimension j (i.e. calculates the mean $\bar{x}_j = \sum_{i=1}^{\alpha} x_{i,j}/\alpha$ and the variance $d_j = \sum_{i=1}^{\alpha}(x_{i,j} - \bar{x}_j)^2/\alpha$). The RU can also perform some self-defined calculations (e.g. calculates the sum $X = \sum_{j=1}^{\beta} x_{i,j}$, or multiplication $X' = \prod_{j=1}^{\beta} x_{i,j}$). Since $x_{i,j}$ is required to be outsourced to the cloud for storage, we have the following challenges:

1) Secure Outsourced Storage: As the cloud storage service is often provided by third-party servers who may be untrusted or semi-trusted, it is important for DP to outsource the data to the cloud without compromising its own privacy.

2) Secure Processing Toolkit for Integer: In order to achieve data processing on-the-fly, the encrypted integer calculation toolkit needs to be built to support commonly used integer number operations over the plaintext. For example, additions, multiplications and divisions should be achievable by operating on two encrypted numbers.

3) Secure Processing under Multiple Keys: In order to support outsourced data processing across different parties, a multi-key data calculation mechanism (e.g. comparison of encrypted numbers under different public keys) needs to be constructed. Moreover, as the final result contains information belonging to different parties, fine-grained authentication mechanisms should be designed to guarantee the privacy of individual DP.

C. Attack Model

In our attack model, we consider the KGC to be a trusted entity, which generates the public and private keys for the system. On the other hand, RUs, DPs, CP and CSP are *curious-but-honest* parties, which strictly follow the protocol. However, RUs, DPs, CP and CSP are also interested to learn data belonging to other parties. Therefore, we introduce an active adversary $\mathcal{A}^*$ in our model. The goal of $\mathcal{A}^*$ is to decrypt the challenge DP's original ciphertext and the challenge RU's encrypted final results with the following capabilities:

1) $\mathcal{A}^*$ may *eavesdrop* all communications to obtain the encrypted data.

2) $\mathcal{A}^*$ may *compromise* the CP to guess the plaintext value of all ciphertexts outsourced from the DPs (including the challenge DPs), and all ciphertext sent from the CSP by executing an interactive protocol.

3) $\mathcal{A}^*$ may *compromise* the CSP to guess the plaintext value of all ciphertexts sent from the CP by executing an interactive protocol.

4) $\mathcal{A}^*$ may *compromise* one or more RUs and DPs, with the exception of the challenge RU or challenge DP, to obtain access to their decryption capabilities, and guess all ciphertexts belonging to the challenge RU or challenge DP.

The adversary $\mathcal{A}^*$ is, however, restricted from compromising (1) both the CSP and the CP concurrently, (2) the challenge DP, and (3) the challenge RU. We remark that such restrictions are typical in adversary models used in cryptographic protocols (see the review of adversary models in [18] and [19]).

IV. Basic Crypto–Distributed Two Trapdoors Public-Key Cryptosystem (DT-PKC)

In order to realize EPOM, the public-key cryptosystem with a double trapdoor decryption cryptosystem introduced by Bresson et al. [20] could be a suitable solution for key management in the multi-key setting at first glance. However, the strong trapdoor leakage is a risk to the system, since encrypted data in Bresson et al.'s cryptosystem can be decrypted by the strong trapdoor. Therefore, we design a new cryptosystem – Distributed Two Trapdoors Public-Key Cryptosystem (DT-PKC) – to split a strong private key into different shares. In addition, the weak decryption algorithm should support distributed decryption to solve the authorization problem in the multi-key environment (see Section V-I). Our DT-PKC is based on Bresson et al.'s cryptosystem [20], follows the idea in [21], and works as follows:

KeyGen: Given a security parameter k and two large prime numbers p, q, where $\mathcal{L}(p) = \mathcal{L}(q) = k$, we have two strong primes p', q', s.t., $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ (due to the property of the strong primes). We then compute $N = pq$ and $\lambda = lcm(p-1, q-1)/2$, define a function $L(x) = \frac{x-1}{N}$, and choose a generator g of order $(p-1)(q-1)/2$ (this can be achieved by selecting a random number $a \in \mathbb{Z}_{N^2}^*$ and computing g as $g = -a^{2N}$ [22]). We also randomly select $\theta_i \in [1, N/4]$ and compute $h_i = g^{\theta_i} \mod N^2$ for party i. The public key for i is $pk_i = (N, g, h_i)$, and the corresponding weak private key is $sk_i = \theta_i$. The system's strong private key is $SK = \lambda$.

Encryption (Enc): Given a message $m \in \mathbb{Z}_N$, we choose a random number $r \in [1, N/4]$. The ciphertext under pk_i can be generated as $[m]_{pk_i} = \{T_{i,1}, T_{i,2}\}$, where $T_{i,1} = g^{r\theta_i}(1 + mN) \mod N^2$; $T_{i,2} = g^r \mod N^2$.

Decryption With Weak Private Key (WDec): $[m]_{pk_i}$ can be decrypted using decryption algorithm $D_{sk_i}(\cdot)$ with weak private key $sk_i = \theta_i$:

$$m = L(\frac{T_{i,1}}{T_{i,2}^{\theta_i}} \mod N^2).$$

Decryption With Strong Private Key (SDec): Any ciphertext $[m]_{pk_i}$ can be decrypted using decryption algorithm $D_{SK}(\cdot)$ with strong private key $SK = \lambda$ by first calculating:

$$T_{i,1}^{\lambda} \mod N^2 = g^{\lambda \cdot \theta_i r}(1 + mN\lambda) \mod N^2 = (1 + mN\lambda).$$

Then, due to $gcd(\lambda, N) = 1$, m can be recovered as follows:

$$m = L(T_{i,1}^{\lambda} \mod N^2)\lambda^{-1} \mod N.$$

Strong Private Key Splitting (SkeyS): The strong private key $SK = \lambda$ can be randomly split into two parts. The partial strong private keys are denoted as $SK^{(i)} = \lambda_j$ $(j = 1, 2)$, s.t., $\lambda_1 + \lambda_2 \equiv 0 \mod \lambda$ and $\lambda_1 + \lambda_2 \equiv 1 \mod N^2$ hold at the same time (the existence of the strong private key splitting can be found in Section VI-A1).

Partial Decryption With Partial Strong Private Key Step One (PSDec1): Once $[m]_{pk_i} = \{T_{i,1}, T_{i,2}\}$ is received, the **PSDec1** algorithm $PDO_{SK^{(1)}}(\cdot)$ can be run as follows:

Using partial strong private key $SK^{(1)} = \lambda_1$, the partial decrypted ciphertext $CT_i^{(1)}$ can be calculated as:

$$CT_i^{(1)} = (T_{i,1})^{\lambda_1} = g^{r\theta_i\lambda_1}(1 + mN\lambda_1) \mod N^2.$$

Partial Decryption With Partial Strong Private Key Step Two (PSDec2): Once $CT_i^{(1)}$ and $[m]_{pk_i}$ are received, the **PSDec2** algorithm $PDT_{SK^{(2)}}(\cdot, \cdot)$ can be run to obtain the original message m, i.e., the PSDec2 first executes

$$CT_i^{(2)} = (T_{i,1})^{\lambda_2} = g^{r\theta_i\lambda_2}(1 + mN\lambda_2) \mod N^2.$$

Then, the algorithm computes $T'' = CT_i^{(1)} \cdot CT_i^{(2)}$, and calculates

$$m = L(T'').$$

Partial Decryption With Partial Weak Private Key Step One (PWDec1): Once $[m]_{pk_{\Sigma_\rho}} = \{T_{\Sigma_\rho,1}, T_{\Sigma_\rho,2}\}$[1] is received, the **PWDec1** algorithm can be run with partial private key $sk_i = \theta_i$. The partial weak decrypted ciphertext $WT^{(i)}$ can be calculated as:

$$WT^{(i)} = (T_{\Sigma_\rho,2})^{\theta_i} = g^{r\theta_i} \mod N^2.$$

Partial Decryption With Partial Weak Private Key Step Two (PWDec2): Once $[m]_{pk_{\Sigma_\rho}}, WT^{(1)}, \cdots, WT^{(\kappa)}$ are received, the **PWDec2** algorithm can be run as follows:

Using partial private key $sk_\rho = \theta_\rho$, the partial weak decrypted ciphertext $WT^{(\rho)}$ can be calculated as:

$$WT^{(\rho)} = (T_{\Sigma_\rho,2})^{\theta_\rho} = g^{r\theta_\rho} \mod N^2.$$

We then calculate $WT = \prod_{i=1}^{\kappa} WT^{(i)} \cdot WT^{(\rho)}$ and

$$m = L(\frac{T_{\Sigma_\rho,1}}{WT} \mod N^2).$$

Ciphertext Refresh (CR): Once $[m]_{pk_i}$ is received, the *CR* algorithm can refresh the ciphertext without changing the original message m, by randomly choosing $r' \in \mathbb{Z}_N$ and refreshing the ciphertext as $[m]'_{pk_i} = \{T'_{i,1}, T'_{i,2}\}$, where

$$T'_{i,1} = T_{i,1} \cdot h_i^{r'} \mod N^2; \quad T'_{i,2} = T_{i,2} \cdot g^{r'} \mod N^2$$

Note that for given $m_1, m_2 \in \mathbb{Z}_N$ under the same pk, we have

$$\begin{aligned} [m_1]_{pk} \cdot [m_2]_{pk} &= \{(1 + (m_1 + m_2) \cdot N) \cdot h^{r_1+r_2} \mod N^2, \\ &\quad g^{r_1+r_2} \mod N^2\} = [m_1 + m_2]_{pk}. \\ ([m]_{pk})^{N-1} &= \{(1 + (N-1)m \cdot N) \cdot h^{(N-1)r_1} \mod N^2, \\ &\quad g^{(N-1)r_1} \mod N^2\} = [-m]_{pk}. \end{aligned}$$

[1]The joint public key is constructed as $pk_{\Sigma_\rho} = (N, g, h_{\Sigma_\rho} = g^{\theta_\rho + \sum_{j=1,\cdots,\kappa}\theta_j})$ which associates with DP $j(j = 1, \cdots, \kappa)$ and RU ρ.

In the system, we have η RU and κ DP. The KGC first generates $pk_i = (N, g, h_i = g^{\theta_i})$ and $sk_i = \theta_i$ $(i = 1, \cdots, \eta + \kappa)$ under the same N and g, and sends the individual public-private key pair to each RU and DP. Moreover, the SK should be randomly split into $SK^{(1)}$ and $SK^{(2)}$ using **SkeyS** algorithm, prior sending to CP and CSP for storage respectively. In addition, DP i can encrypt data with their own public key pk_i, and outsource the ciphertexts to the CP for storage. Moreover, the DP's public key $pk_j(j = 1, \cdots, \kappa)$ and joint public key $pk_{\Sigma_k}(k = 1, \cdots, \eta)$ should be sent to CP and CSP. For simplicity, if all the ciphertexts with joint key are associated with the RU ρ, we will simply omit the subscript ρ from the symbols (e.g., use pk_Σ instead of pk_{Σ_ρ}) for simplicity / readability.

V. Privacy Preserving Integer Calculation Toolkit for Multiple Keys

After introducing the underlying algorithms in DT-PKC, we will now present the secure sub-protocols as the toolkit for processing integers, namely: Secure Addition Protocol across Domains (**SAD**), Secure Multiplication Protocol across Domains (**SMD**), Secure Sign Bit Acquisition Protocol (**SSBA**) Secure Less Than Protocol (**SLT**), Secure Maximum and Minimum Sorting Protocol (**SMMS**), Secure Equivalent Testing Protocol (**SEQ**), Secure Division Protocol (**SDIV**) and Secure Greatest Common Divisor Protocol (**SGCD**). We assume that both CP and CSP will be involved in the sub-protocol, as the CP holds a partial strong private key $SK^{(1)}$, and the CSP has the remaining partial strong private key $SK^{(2)}$ and public key pk_Σ. Note that both x, y involved in the above sub-protocols are integer (i.e. x, y can be positive, negative or zero); therefore, we restrict $|x|$ and $|y|$ to be in the range of $[0, R_1]$, where $\mathcal{L}(R_1) < \mathcal{L}(N)/8$. If a larger plaintext range is needed, we can simply use a larger N. A larger N implies a broader plaintext range, and therefore, a higher level of security. However, this will affect the efficiency of DT-PKC (See Fig. 2(a)).

A. Secure Addition Protocol Across Domains (SAD)

Our DT-PKC cryptosystem can support additive homomorphism; however, it can only be achieved under the same public key (i.e. $[m_1+m_2]_{pk} = [m_1]_{pk} \cdot [m_2]_{pk}$). Our **SAD** is designed for plaintext addition over encrypted data with different keys. In other words, given two encrypted data $[x]_{pk_a}$ and $[y]_{pk_b}$ under different keys, the goal of **SAD** protocol is to calculate $[x + y]_{pk_\Sigma}$. The description of the **SAD** protocol is as follows:

Step-1 (@CP): Chooses a random number $r_a, r_b \in \mathbb{Z}_N$, calculates

$$\begin{aligned} X &= [x]_{pk_a} \cdot [r_a]_{pk_a} = [x + r_a]_{pk_a}, \\ Y &= [y]_{pk_b} \cdot [r_b]_{pk_b} = [y + r_b]_{pk_b}, \end{aligned}$$

calculates $X' = PDO_{SK^{(1)}}(X)$ and $Y' = PDO_{SK^{(1)}}(Y)$, and sends X, Y, X' and Y' to CSP.

Step-2 (@CSP): Calculates $X'' = PDT_{SK^{(2)}}(X'; X)$ and $Y'' = PDT_{SK^{(2)}}(Y'; Y)$, calculates $S = X'' + Y''$, encrypts S as $[S]_{pk_\Sigma}$, and sends the encrypted data to CP.

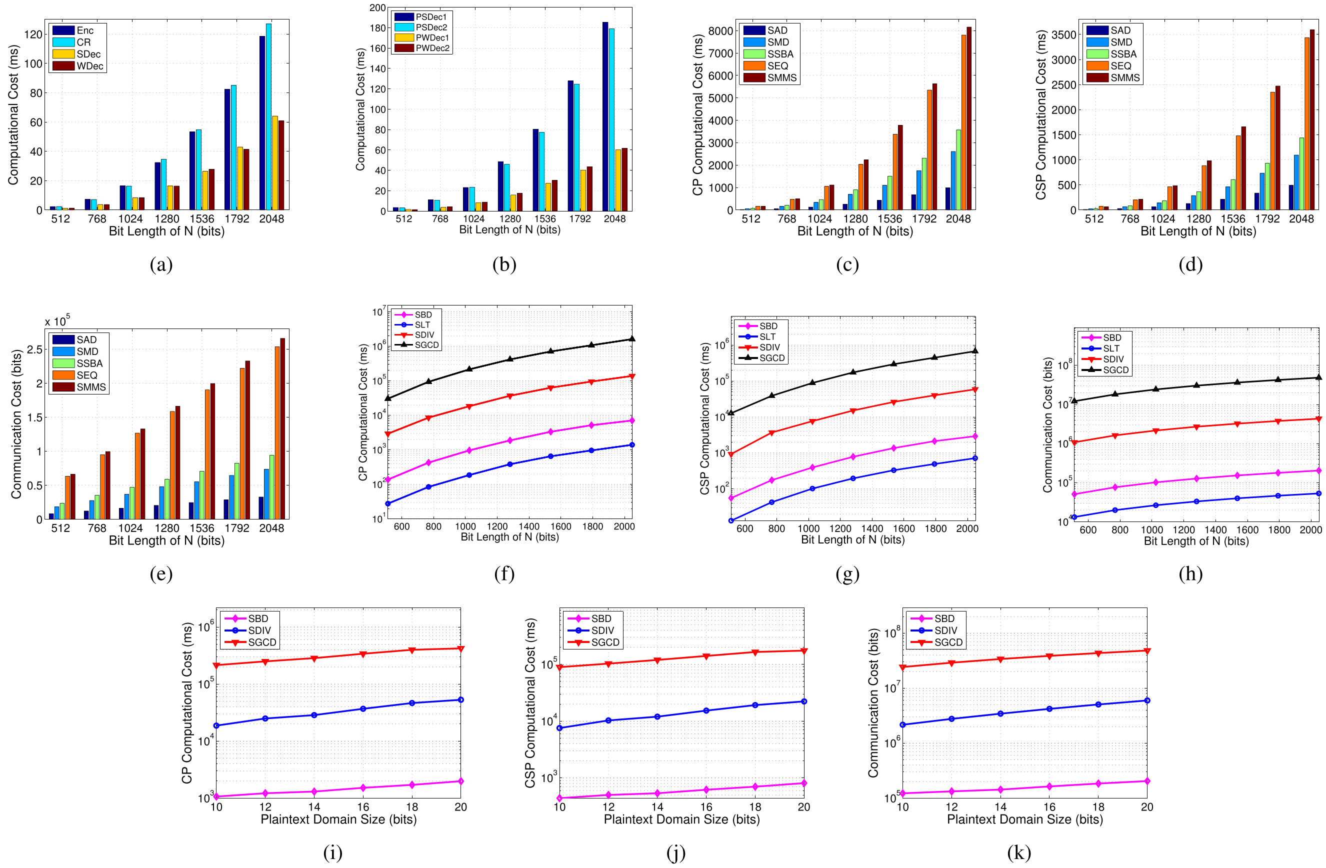

Fig. 2. Evaluation findings. (a) Run time of DT-PKC (vary with bit length of N). (b) Run time of DT-PKC (vary with bit length of N). (c) Run time on CP (vary with bit length of N). (d) Run time on CSP (vary with bit length of N). (e) Communication cost (vary with bit length of N). (f) Run time on CP (vary with bit length of N). (g) Run time on CSP (vary with bit length of N). (h) Communication costs between CP and CSP (vary with bit length of N). (i) Run time on CP (vary with plaintext domain). (j) Run time on CSP (vary with plaintext domain). (k) Communication cost between CP and CSP (vary with plaintext domain).

Step-3 (@CP): CP calculates $R = r_a + r_b$, uses pk_Σ to encrypt R as $[R]_{pk_\Sigma}$, and calculates

$$[S]_{pk_\Sigma} \cdot ([R]_{pk_\Sigma})^{N-1} = [S - R]_{pk_\Sigma} = [x + y]_{pk_\Sigma}$$

B. Secure Multiplication Protocol Across Domains (SMD)

Given two encrypted data $[x]_{pk_a}$ and $[y]_{pk_b}$ under two different public keys pk_a and pk_b, respectively, the goal of **SMD** is to calculate $[x \cdot y]_{pk_\Sigma}$ under pk_Σ. The description of **SMD** is as follows:

Step-1 (@CP): CP selects four random numbers $r_x, r_y, R_x, R_y \in \mathbb{Z}_N$, calculates

$$X = [x]_{pk_a} \cdot [r_x]_{pk_a}, Y = [y]_{pk_b} \cdot [r_y]_{pk_b},$$
$$S = [R_x]_{pk_a} \cdot ([x]_{pk_a})^{N-r_y} = [R_x - r_y \cdot x]_{pk_a},$$
$$T = [R_y]_{pk_b} \cdot ([y]_{pk_b})^{N-r_x} = [R_y - r_x \cdot y]_{pk_b},$$

calculates $X_1 = PDO_{SK^{(1)}}(X)$, $Y_1 = PDO_{SK^{(1)}}(Y)$, $S_1 = PDO_{SK^{(1)}}(S)$, and $T_1 = PDO_{SK^{(1)}}(T)$, and sends X_1, Y_1, S_1, T_1, X, Y, S, T to CSP.

Step-2 (@CSP): Using the other partial strong private key $SK^{(2)}$, CSP calculates

$$h = PDT_{SK^{(2)}}(X_1; X) \cdot PDT_{SK^{(2)}}(Y_1; X),$$
$$S_2 = PDT_{SK^{(2)}}(S_1; S), T_2 = PDT_{SK^{(2)}}(T_1; T).$$

CSP encrypts h, S_2, T_2 using pk_Σ, denoted as $H = [h]_{pk_\Sigma}$, $S_3 = [S_2]_{pk_\Sigma}$, $T_3 = [T_2]_{pk_\Sigma}$, and sends H, S_3 and T_3 to CP. It is trivial to verify that $h = (x + r_x)(y + r_y)$.

Step-3 (@CP): Once H, S_3 and T_3 are received, CP computes $S_4 = ([r_x \cdot r_y]_{pk_\Sigma})^{N-1}$, $S_5 = ([R_x]_{pk_\Sigma})^{N-1}$ and $S_6 = ([R_y]_{pk_\Sigma})^{N-1}$, and calculates the following to recover the encrypted $x \cdot y$:

$$\begin{aligned} H \cdot T_3 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6 &= [(h + (R_x - r_y \cdot x) + (R_y - r_x \cdot y) \\ &\quad - r_x \cdot r_y - R_x - R_y)]_{pk_\Sigma} \\ &= [x \cdot y]_{pk_\Sigma}. \end{aligned}$$

C. Secure Sign Bit Acquisition Protocol (SSBA)

Given an encrypted integer number $[x]_{pk_a}$, the goal of **SSBA** protocol is to obtain the encrypted sign bit $[s^*]_{pk_\Sigma}$ and the transformed number $[x^*]_{pk_\Sigma}$, s.t., $x^* = x$ and $s^* = 1$ when $x \geq 0$, while $x^* = N - x$ and $s^* = 0$ when $x < 0$. The description of the **SSBA** protocol is as follows:

Step-1 (@CP): CP flips a coin s, and chooses a random number r, s.t. $\mathcal{L}(r) < \mathcal{L}(N)/4$. If $s = 1$, CP calculates

$$[l]_{pk_a} = (([x]_{pk_a})^2 \cdot [1]_{pk_a})^r = [r(2x+1)]_{pk_a}.^2$$

[2]We transform x into $2x + 1$ in order to prevent CSP to know the value x when $x = 0$.

If $s = 0$, CP calculates

$$[l]_{pk_a} = (([x]_{pk_a})^2 \cdot [1]_{pk_a})^{N-r} = [-r(2x+1)]_{pk_a}.$$

Then, CP calculates $L = PDO_{SK^{(1)}}([l]_{pk_a})$ and sends L and $[l]_{pk_a}$ to CSP.

Step-2 (@CSP): CSP calculates $PDT_{SK^{(2)}}(L; [l]_{pk_a})$ to obtain l. If $\mathcal{L}(l) < 3/8 \cdot \mathcal{L}(N)$, denotes $u = 1$; otherwise, $u = 0$.

Then, u is encrypted using pk_Σ, and $[u]_{pk_\Sigma}$ is sent to CSP.

Step-3 (@CP): (1) If $s = 1$, CP calculates

$$[s^*]_{pk_\Sigma} = \mathbf{CR}([u]_{pk_\Sigma});$$

otherwise, calculates $[s^*]_{pk_\Sigma} = \mathbf{CR}([1]_{pk_\Sigma} \cdot [u]^{N-1}_{pk_\Sigma})$.

(2) CP then calculates

$$[x^*]_{pk_\Sigma} \leftarrow \texttt{SMD}([x]_{pk_\Sigma}; [s^*]^2_{pk_\Sigma} \cdot ([1]_{pk_\Sigma})^{N-1}).$$

D. Secure Less Than Protocol (SLT)

Given two encrypted numbers $[x]_{pk_a}$ and $[y]_{pk_b}$, the goal of **SLT** protocol is to obtain the encrypted data $[u^*]_{pk_\Sigma}$ to show the relationship between the plaintext of the two encrypted data (i.e. $x \geq y$ or $x < y$). The description of the **SLT** protocol is as follows:

Step-1: (1) CP calculates

$$[x_1]_{pk_a} = ([x]_{pk_a})^2 \cdot [1]_{pk_a} = [2x+1]_{pk_a};$$
$$[y_1]_{pk_b} = ([y]_{pk_b})^2 = [2y]_{pk_b}.$$[3]

(2) CP flips a coin s randomly. If $s = 1$, CP calculates

$$[l]_{pk_\Sigma} \leftarrow \texttt{SAD}([x_1]_{pk_a}; ([y_1]_{pk_b})^{N-1}).$$

If $s = 0$, CP calculates

$$[l]_{pk_\Sigma} \leftarrow \texttt{SAD}([y_1]_{pk_b}; ([x_1]_{pk_a})^{N-1}).$$

(3) CP chooses a random number r, s.t., s.t. $\mathcal{L}(r) < \mathcal{L}(N)/4$, and calculates $[l_1]_{pk_\Sigma} = ([l]_{pk_\Sigma})^r$. Then, CP uses $SK^{(1)}$ to calculate $K = PDO_{SK^{(1)}}([l_1]_{pk_\Sigma})$, and sends the result to CSP.

Step-2 (@CSP): Uses $SK^{(2)}$ to decrypt K, and obtains l.

If $\mathcal{L}(l) > \mathcal{L}(N)/2$, CSP denotes $u' = 1$ and $u' = 0$ otherwise. Then, CSP uses pk_Σ to encrypt u', and sends $[u']_{pk_\Sigma}$ to CP.

Step-3 (@CP): Once $[u']_{pk_\Sigma}$ is received, CP computes as follows: if $s = 1$, CP denotes $[u^*]_{pk_\Sigma} = \mathbf{CR}([u']_{pk_\Sigma})$; otherwise, CP computes

$$[u^*]_{pk_\Sigma} = [1]_{pk_\Sigma} \cdot ([u']_{pk_\Sigma})^{N-1} = [1-u']_{pk_\Sigma}.$$

If $u^* = 0$, it shows $x \geq y$; and if $u^* = 1$, it shows $x < y$.

E. Secure Maximum and Minimum Sorting Protocol (SMMS)

Given two encrypted numbers $[x]_{pk_a}$ and $[y]_{pk_b}$, the goal of **SMMS** protocol is to obtain the encrypted sorting results $[A]_{pk_\Sigma}$ and $[I]_{pk_\Sigma}$, s.t., $A \geq I$. The description of the **SMMS** protocol is as follows:

(1) CP and CSP jointly calculate

$$[x]_{pk_\Sigma} \leftarrow \texttt{SAD}([x]_{pk_a}; [0]_{pk_b});$$
$$[y]_{pk_\Sigma} \leftarrow \texttt{SAD}([0]_{pk_a}; [y]_{pk_b});$$
$$[u^*]_{pk_\Sigma} \leftarrow \texttt{SLT}([x]_{pk_a}; [y]_{pk_b});$$
$$[X]_{pk_\Sigma} \leftarrow \texttt{SMD}([u^*]_{pk_\Sigma}; [x]_{pk_a});$$
$$[Y]_{pk_\Sigma} \leftarrow \texttt{SMD}([u^*]_{pk_\Sigma}; [y]_{pk_b}).$$

(2) Once $[u^*]_{pk_\Sigma}$ is received, CP computes

$$[A]_{pk_\Sigma} = [x]_{pk_\Sigma} \cdot [X]^{N-1}_{pk_\Sigma} \cdot [Y]_{pk_\Sigma} = [(1-u^*)x + u^*y]_{pk_\Sigma};$$
$$[I]_{pk_\Sigma} = [y]_{pk_\Sigma} \cdot [Y]^{N-1}_{pk_\Sigma} \cdot [X]_{pk_\Sigma} = [(1-u^*)y + u^*x]_{pk_\Sigma}.$$

F. Secure Equivalent Testing Protocol (SEQ)

Given two encrypted data $[x]_{pk_a}$ and $[y]_{pk_b}$, the goal of **SEQ** protocol is to obtain the encrypted result $[f]_{pk_\Sigma}$ to determine whether the plaintext of the two encrypted data are equal (i.e. $x = y$). The description of the **SEQ** protocol is as follows:

(1) CP and CSP jointly calculate

$$[u_1]_{pk_\Sigma} \leftarrow \texttt{SLT}([x]_{pk_a}, [y]_{pk_b});$$
$$[u_2]_{pk_\Sigma} \leftarrow \texttt{SLT}([y]_{pk_b}, [x]_{pk_a});$$
$$[f_1^*]_{pk_\Sigma} \leftarrow \texttt{SMD}([1]_{pk_\Sigma} \cdot ([u_1]_{pk_\Sigma})^{N-1}; [u_2]_{pk_\Sigma});$$
$$[f_2^*]_{pk_\Sigma} \leftarrow \texttt{SMD}([u_1]_{pk_\Sigma}; [1]_{pk_\Sigma} \cdot ([u_2]_{pk_\Sigma})^{N-1}).$$

(2) CP calculates and outputs $[f]_{pk_\Sigma}$ as follows:

$$[f]_{pk_\Sigma} = [u_1 \oplus u_2]_{pk_\Sigma} = [f_1^*]_{pk_\Sigma} \cdot [f_2^*]_{pk_\Sigma}.$$

If $f = 0$, then $x = y$; otherwise, $x \neq y$.

G. Secure Division Protocol (SDIV)

Given an encrypted numerator $[y]_{pk_b}$ and an encrypted denominator $[x]_{pk_a}$, the **SDIV** will provide the encrypted quotient $[q^*]_{pk_\Sigma}$ and encrypted remainder $[r^*]_{pk_\Sigma}$, without compromising the privacy of data, s.t., $y = q^* \cdot x + r^*$ ($|y| \geq |x|$). The **SDIV** is explained in **Algorithm 1**, and a brief description is given below.

In the event that the value of the denominator is 0, we will mark $x = 1$ and $y = 0$ as we cannot simply abort **SDIV**. Otherwise, CP will know that $x = 0$ once **SDIV** is aborted (lines 1-5). We will then use **SSBA** to obtain $[x^*]_{pk_\Sigma}$ and $[y^*]_{pk_\Sigma}$ (x^* and y^* are the absolute value of x and y, line 6), and **SBD** to expand $[y^*]_{pk_\Sigma}$ into encrypted bits, denoted as $([q_{\mu-1}]_{pk_\Sigma}, \cdots, [q_0]_{pk_\Sigma})$ (line 7). Also, we use $([0]_{pk_\Sigma}, \cdots, [0]_{pk_\Sigma})$ to initialize $([a_{\mu-1}]_{pk_\Sigma}, \cdots, [a_0]_{pk_\Sigma})$ (line 8). Next, the following procedures will be executed μ-times: move $[a_{\mu-1}]_{pk_\Sigma}, \cdots [a_0]_{pk_\Sigma}, [q_{\mu-1}]_{pk_\Sigma}, \cdots, [q_0]_{pk_\Sigma}$ by one position to the left (i.e. mark $[a_i]_{pk_\Sigma} = [a_{i-1}]_{pk_\Sigma}$ for $i = \mu - 1$ to 1, and mark both $[a_0]_{pk_\Sigma} = [q_{\mu-1}]_{pk_\Sigma}$ and $[q_i]_{pk_\Sigma} = [q_{i-1}]_{pk_\Sigma}$ for $i = \mu - 1$ to 1). Then, the CP calculates $[a_{\mu-1}]_{pk_\Sigma}, \cdots, [a_0]_{pk_\Sigma}$ and converts from binary to integer $[A]_{pk_\Sigma}$ before comparing A with x^* using **SLT**. If $A < x^*$, **SDIV** will mark $q_0 = 0$; otherwise, **SDIV** will mark $q_0 = 1$ and compute $A = A - x^*$ (lines 9-15).

After calculating μ times, the remainder r is the integer value of $(a_{\mu-1}, \cdots, a_0)$ while the value quotient q is the

Algorithm 1 Secure Division Protocol (SDIV)

Input: Encrypted numerator $[y]_{pk_b}$ and encrypted denominator $[x]_{pk_a}$.
Output: Encrypted quotient $[q^*]_{pk_\Sigma}$ and encrypted remainder $[r^*]_{pk_\Sigma}$

1 Both CP and CSP jointly calculate
2 $[f]_{pk_\Sigma} \leftarrow \texttt{SEQ}([x]_{pk_a}, [0]_{pk_\Sigma})$.
3 CP calculates $[1]_{pk_\Sigma} \cdot ([f]_{pk_\Sigma})^{N-1} = [1-f]_{pk_\Sigma}$.
4 Then, both CP and CSP jointly calculate $[f \cdot x]_{pk_\Sigma} \leftarrow \texttt{SMD}([f]_{pk_\Sigma}, [x]_{pk_a})$ and $[y']_{pk_\Sigma} = [f \cdot y]_{pk_\Sigma} \leftarrow \texttt{SMD}([f]_{pk_\Sigma}, [y]_{pk_\Sigma})$.
5 CP calculates

$$[x']_{pk_\Sigma} = [f \cdot x + (1-f) \cdot 1]_{pk_\Sigma} = [f \cdot x]_{pk_\Sigma} \cdot [1-f]_{pk_\Sigma}.$$

6 CP and CSP jointly execute $([x^*]_{pk_\Sigma}, [s_x]_{pk_\Sigma}) \leftarrow \texttt{SSBA}([x']_{pk_a})$, $([y^*]_{pk_\Sigma}, [s_y]_{pk_\Sigma}) \leftarrow \texttt{SSBA}([y']_{pk_b})$.
7 Both CP and CSP jointly execute **SBD**, s.t., $([y_{\mu-1}]_{pk_\Sigma}, \cdots, [y_0]_{pk_\Sigma}) \leftarrow \texttt{SBD}([y^*]_{pk_\Sigma})$ and mark $([q_{\mu-1}]_{pk_\Sigma}, \cdots, [q_0]_{pk_\Sigma}) \leftarrow ([y_{\mu-1}]_{pk_\Sigma}, \cdots, [y_0]_{pk_\Sigma})$.
8 CP also initializes

$$([a_{\mu-1}]_{pk_\Sigma}, \cdots, [a_0]_{pk_\Sigma}) \leftarrow (\underbrace{[0]_{pk_\Sigma}, \cdots, [0]_{pk_\Sigma}}_{\mu-\text{elements}}).$$

9 **for** *executing μ times* **do**
10 denote $[a_i]_{pk_\Sigma} = [a_{i-1}]_{pk_\Sigma}$ (for $i = \mu$ to 1); then denote $[a_0]_{pk_\Sigma} = [q_{\mu-1}]_{pk_\Sigma}$; finally, denote $[q_i]_{pk_\Sigma} = [q_{i-1}]_{pk_\Sigma}$(for $i = \mu$ to 1);
11 calculate $[A]_{pk_\Sigma} = [a_0]_{pk_\Sigma} \cdot [a_1]^2_{pk_\Sigma} \cdots [a_{\mu-1}]^{2^{\mu-1}}_{pk_\Sigma}$;
12 calculate $[Q]_{pk_\Sigma} \leftarrow \texttt{SLT}([A]_{pk_\Sigma}; [x^*]_{pk_\Sigma})$;
13 calculate $[q_0]_{pk_\Sigma} = [1]_{pk_\Sigma} \cdot [Q]^{N-1}_{pk_\Sigma} = [1-Q]_{pk_\Sigma}$;
14 execute $[B]_{pk_\Sigma} \leftarrow \texttt{SMD}([x^*]^{N-1}_{pk_\Sigma}, [q_0]_{pk_\Sigma})$;
15 calculate $[A]_{pk_\Sigma} = [A]_{pk_\Sigma} \cdot [B]_{pk_\Sigma}$ and then execute **SBD** protocol as:

$$([a_{\mu-1}]_{pk_\Sigma}, \cdots [a_0]_{pk_\Sigma}) \leftarrow \texttt{SBD}([A]_{pk_\Sigma});$$

16 Finally, calculates
$[r]_{pk_\Sigma} = [a_0]_{pk_\Sigma} \cdot ([a_1]_{pk_\Sigma})^2 \cdots ([a_{\mu-1}]_{pk_\Sigma})^{2^{\mu-1}}$;
$[q]_{pk_\Sigma} = [q_0]_{pk_\Sigma} \cdot ([q_1]_{pk_\Sigma})^2 \cdots ([q_{\mu-1}]_{pk_\Sigma})^{2^{\mu-1}}$;
17 Computes $[K_1]_{pk_\Sigma} \leftarrow \texttt{SMD}([s_x]_{pk_\Sigma}; [s_y]_{pk_\Sigma})$;
$[r^*]_{pk_\Sigma} \leftarrow \texttt{SMD}([r]_{pk_\Sigma}; [s_y]_{pk_\Sigma})$;
$[q^*]_{pk_\Sigma} \leftarrow \texttt{SMD}([q]_{pk_\Sigma}; [K_1]_{pk_\Sigma})$.

integer value of $(q_{\mu-1}, \cdots, q_0)$ (line 16). Moreover, we should decide the sign of r^* and q^*: the sign of remainder r^* is the same to that of the numerator y, while the sign of quotient q^* is denoted as the multiplication of the sign of numerator and denominator (line 17). A short example can be demonstrated the correctness of line 17: if $x = 3$ and $y = 5$, we have $q^* = 1$ and $r^* = 2$, s.t., $5 = 1 \times 3 + 2$. If $x = 3$ and $y = -5$, we have $q^* = -1$ and $r^* = -2$, s.t., $-5 = (-1) \times 3 + (-2)$. If $x = -3$ and $y = 5$, we have $q^* = -1$ and $r^* = 2$, s.t., $5 = (-1) \times (-3) + 2$. Finally, if $x = -3$ and $y = -5$, we have $q^* = -1$ and $r^* = -2$, s.t., $-5 = 1 \times (-3) + (-2)$.

Here, we give a short example to show how the **SDIV** works. Given a numerator $y = -5$ and a denominator $x = 3$ as input, the algorithm first gets the absolute value $y^* = 5$ and $x^* = 3$. The bit formation of y^* is denoted as q. Next, the **SDIV** initializes $a = 0000$, executes line 9-15, and gets

Algorithm 2 Secure Greatest Common Divisor Protocol (SGCD)

Input: Two encrypted numbers, $[x]_{pk_a}$ and $[y]_{pk_b}$.
Output: Encrypted greatest common divisor $[C]_{pk_\Sigma}$.

Both CP and CSP jointly execute **SMMS**, s.t., $([y']_{pk_\Sigma}, [x']_{pk_\Sigma}) \leftarrow \texttt{SMMS}([x]_{pk_a}, [y]_{pk_b})$;
for $i = 1$ *to* μ **do**
  calculate $([q_i]_{pk_\Sigma}, [r_i]_{pk_\Sigma}) \leftarrow \texttt{SDIV}([y']_{pk_\Sigma}, [x']_{pk_\Sigma})$;
  denote $[y']_{pk_\Sigma} = [x']_{pk_\Sigma}$ and $[x']_{pk_\Sigma} = [r_i]_{pk_\Sigma}$;
denote $[r_0]_{pk_\Sigma} = [q_1]_{pk_\Sigma}$;
for $i = 0$ *to* μ **do**
  calculate $[u_i]_{pk_\Sigma} \leftarrow \texttt{SEQ}([r_i]_{pk_\Sigma}, [0]_{pk_\Sigma})$;
for $i = 1$ *to* μ **do**
  execute $[f^*_{i-1,i}]_{pk_\Sigma} \leftarrow \texttt{SMD}([1]_{pk_\Sigma} \cdot [u_{i-1}]^{N-1}_{pk_\Sigma}; [u_i]_{pk_\Sigma})$;
  execute $[f^*_{i,i-1}]_{pk_\Sigma} \leftarrow \texttt{SMD}([u_{i-1}]_{pk_\Sigma}; [1]_{pk_\Sigma} \cdot [u_i]^{N-1}_{pk_\Sigma})$;
  calculate $[f_{i-1,i}]_{pk_\Sigma} = [u_{i-1} \oplus u_i]_{pk_\Sigma} = [f^*_{i,i-1}] \cdot [f^*_{i-1,i}]_{pk_\Sigma}$;

$$[C_i]_{pk_\Sigma} \leftarrow \texttt{SMD}([r_{i-1}]_{pk_\Sigma}; [f_{i-1,i}]_{pk_\Sigma});$$

calculate and return $[C]_{pk_\Sigma} = \prod_{j=1}^{m} [C_i]_{pk_\Sigma}$.

TABLE I
EXAMPLE OF SDIV

Round	a	q	x^*	Operations
1	0000	0101	3	
	0000	1010	3	Shift Left a & q together. $A < x^*, Q \leftarrow 1, q_0 \leftarrow 0,$
	$\underline{0000}$	$101\underline{0}$	3	$B \leftarrow 0, A \leftarrow A + B.$
2	0001	0100	3	Shift Left a & q together. $A < x^*, Q \leftarrow 1, q_0 \leftarrow 0,$
	$\underline{0001}$	$010\underline{0}$	3	$B \leftarrow 0, A \leftarrow A + B.$
3	0010	1000	3	Shift Left a & q together. $A < x^*, Q \leftarrow 1, q_0 \leftarrow 0,$
	$\underline{0010}$	$100\underline{0}$	3	$B \leftarrow 0, A \leftarrow A + B.$
4	0101	0000	3	Shift Left a & q together. $A > x^*, Q \leftarrow 0, q_0 \leftarrow 1,$
	$\underline{0010}$	$000\underline{1}$	3	$B \leftarrow -x^*, A \leftarrow A + B.$

– In the table, A is integer format of a.

$q = 1$ and $r = 2$ as shown in the Table I. Finally, the algorithm should decide the sign of quotient and remainder, and output $q^* = -1$ and $r^* = -2$.

H. Secure Greatest Common Divisor Protocol (SGCD)

Given two encrypted numbers $[x]_{pk_a}$ and $[y]_{pk_b}$ ($x > 0$, $y > 0$)[4], **SGCD** will provide the encrypted GCD $[C]_{pk_\Sigma}$, without compromising the privacy of data. **SGCD** is explained in **Algorithm 2**, and a brief description is given below.

Prior to calculating the GCD, CP needs to determine which of the two plaintext values (i.e. x and y) is larger, as the larger value will be chosen as the numerator and the smaller value as the denominator to run **SDIV**. Next, in order to calculate GCD privately, we revisit the Euclidean algorithm: the GCD of two numbers does not change if the larger number is substituted with the difference between the two numbers. Since this substitution reduces the larger of the two numbers, repeating

[4]Mathematically, we only consider the greatest common divisor (GCD) between both positive integers.

this process gives successively smaller pairs of numbers until one of the two numbers reaches zero. However, we are not able to use the Euclidean algorithm as it is, without leaking information since the adversary will know how many protocol rounds have been executed (e.g. the adversary knows the two integers are coprime, as only two rounds of calculation have been performed). Therefore, we will run the Euclidean algorithm for fixed μ rounds (related to the domain size of the integer, line 2-4). Unfortunately, the denominator will be equal to zero if the number of calculation rounds is fixed. This issue is resolved by **SDIV** (as explained in Section V-G). After running the fixed calculation rounds, CP obtains $\mu + 1$ encrypted remainders. The GCD between x and y is the last non-zero remainder. We only need to determine the first zero value remainder, and use the zero remainder to find the GCD. The idea is easy to follow: we denote the non-zero remainder as 1 and the zero remainder as 0 (line 6-7). We use XOR operations to find the position of the last non-zero remainder (line 8-12) – see **Algorithm 2**.

I. Decryption With Fine-Grained Authentication

Once a RU wishes to retrieve some data from DP a, the RU must get the authorization from DP a (as the owner of the data is DP a). If a RU wants to perform some calculations over different DPs (i.e. different encrypted domains), the calculated results will contain information about the original data. Without deploying any specific authorization mechanism, the final result may leak some information about the data from the individual DP. Such attacks are simple and efficient. For example, a RU constructs a query and CP will compute accordingly to the protocol. After the calculation is completed, DP will request both CP and CSP to transform the results into RU's domain. Then, RU can easily obtain the results after the decryption. In other words, if a RU ρ is interested in DP i's encrypted data $[x]_{pk_i}$ stored in the cloud, ρ sends $[1]_{pk_\rho}$ to the cloud and CP will compute $[x]_{pk_\rho} \leftarrow \mathbf{SMD}([x]_{pk_i}; [1]_{pk_\rho})$, and $[x]_{pk_\rho}$ will be sent to DP ρ for decryption to obtain x.

In order to solve this problem, we present a simple yet elegant solution. The final results will be encrypted using the joint public key associated with different DPs and the RU. If the RU wishes to obtain the final plaintext, the RU needs to obtain partial decrypted results (authorization) from the involved DPs. For example, a RU ρ's public key is $pk_\rho = (N, g, h_a = g^{\theta_\rho})$, DP a and DP b public keys are $pk_a = (N, g, h_a = g^{\theta_a})$ and $pk_b = (N, g, h_b = g^{\theta_b})$, respectively. The final result x is encrypted with $pk_\Sigma = (N, g, h_\Sigma = g^{\theta_\rho+\theta_a+\theta_b})$ (i.e. $[x]_{pk_\Sigma}$). For the RU to decrypt the result, $[x]_{pk_\Sigma}$ should be first sent to both DP a and DP b for decryption authorization. If DP a (and DP b) allows the RU to access the finally results, DP a (DP b) executes **PWDec1** to generate partial decrypted ciphertext WT_a (WT_b) to the RU ρ. Once both partial decrypted ciphertext and original ciphertext are received, the RU should execute **PWDec2** using $sk_\rho = \theta_\rho$ to obtain x. We regard our solution to be fine-grained as it is related to each encrypted result, at the cost of a communication round between CP and all DPs.[5] If the system does not need this fine-grained authorization, the system can simply use traditional authentication method to authorize DPs and RUs [23], [24]. In this situation, DPs can be offline after outsourcing the data.

J. The Extension to Handle Encrypted Rational Number

If DP i wishes to outsource the rational numbers to CP for storage, a key challenge is ensuring secure encryption of the rational numbers prior to outsourcing.

As any rational number can be presented as a fraction (i.e. x can be stored as $x^\uparrow/x^\downarrow$), the storage challenge can be easily solved by encrypting the numerator $x^\uparrow$ and denominator $x^\downarrow$, and storing $([x^\uparrow]_{pk_i}, [x^\downarrow]_{pk_i})$, where pk_i is DP i's public key. For example, -0.2 can be represented as $-1/5 = x^\uparrow/x^\downarrow$. Using the DT-PKC scheme, we encrypt $x^\uparrow$ and $x^\downarrow$ as $[x^\uparrow]_{pk_i} = ([1]_{pk_i})^{N-1} = [-1]_{pk_i}$ and $[x^\downarrow]_{pk_i} = [5]_{pk_i}$. After the encryption, $([x^\uparrow]_{pk_i}, [x^\downarrow]_{pk_i})$ are outsourced to the CP.

Another challenge is performing encrypted rational number calculations in a multi-key setting. This challenge can be easily solved using combination of operations with secure integer calculation toolkit for multi-key. For instance, given two encrypted rational numbers $([x^\uparrow]_{pk_a}, [x^\downarrow]_{pk_a})$ and $([y^\uparrow]_{pk_b}, [y^\downarrow]_{pk_b})$, the rational number multiplication result is $([z^\uparrow]_{pk_\Sigma}, [z^\downarrow]_{pk_\Sigma})$, where

$$[z^\uparrow]_{pk_\Sigma} \leftarrow \mathbf{SMD}([x^\uparrow]_{pk_a}; [y^\uparrow]_{pk_b});$$
$$[z^\downarrow]_{pk_\Sigma} \leftarrow \mathbf{SMD}([x^\downarrow]_{pk_a}; [y^\downarrow]_{pk_b}).$$

The constructions of other rational calculation are trivial, and due to space constraints, we will not discuss the constructions in this paper, and refer reader to read [25].

1) The Necessity of CSP: To ensure efficiency, we use AHC for data encryption before outsourcing to CP for storage. Since we use AHC (or other partial homomorphic cryptosystem), we will need CSP to perform plaintext multiplication, as the CP is not able to perform both addition and multiplication homomorphic calculations over encrypted data at the same time (unlike, a fully homomorphic cryptosystem). Unfortunately, both single key and multiple keys fully homomorphic cryptosystem in the existing scheme are rather inefficient, in terms of computation and storage [26], [27], [43], [44]. In the near future, if an efficient multi-key fully homomorphic cryptosystem exists, we can remove the CSP from the system which will also result in a more elegant system.

2) The Extension to Handle Real Number: In our system, we use the nearest rational number to simulate the real number, at the cost of some accuracy. For example, we represent $\sqrt{2}$ with 1.414 (i.e. $\frac{707}{500}$). If we want a higher level of accuracy, we can use 1.41421 (i.e. $\frac{141421}{100000}$) to represent $\sqrt{2}$. In other words, a higher level of accuracy will require a longer plaintext length.

[5] A RU can obtain the data from some specific DPs for calculation. Such information can be protected from the adversary by involving all DPs in the execution of **PWdec1**. If the information does not need to be protected, only the necessary DPs are involved in the partial decryption.

VI. Security Analysis

In this section, we will analyze the security of the basic encryption primitive and the sub-protocols, before demonstrating the security of our EPOM framework.

A. Analysis of DT-PKC

1) The Existence of Strong Private Key Splitting: We randomly split the strong private key $SK = \lambda$ into two parts, denoted as λ_1 and λ_2, s.t., both $\lambda_1 + \lambda_2 \equiv 0 \mod \lambda$ and $\lambda_1 + \lambda_2 \equiv 1 \mod N^2$ hold. Since $gcd(\lambda, N^2) = 1$, thus $\exists s$, s.t. both $s \equiv 0 \mod \lambda$ and $s \equiv 1 \mod N^2$ hold (according to the Chinese remainder theorem [28]; $s = \lambda \cdot (\lambda^{-1} \mod N^2) \mod \lambda N^2$). Therefore, we only need to randomly choose λ_1 and λ_2, s.t., $\lambda_1 + \lambda_2 = s$.

2) Security of DT-PKC: The security of our DT-PKC is given by the following theorem.

*Theorem 1: The **DT-PKC** scheme described in Section IV is semantically secure, based on the assumed intractability of the DDH assumption over $\mathbb{Z}^*_{N^2}$ [20].*

Proof: The security of **DT-PKC** follows directly from that of the public-key cryptosystem with a double trapdoor decryption, which has been proven to be semantically secure in the standard model assuming the intractability of the DDH assumption over $\mathbb{Z}^*_{N^2}$ [20] (the hardness of DDH assumption over $\mathbb{Z}^*_{N^2}$ can be found in [20]).

The privacy of divided private key is guaranteed by Shamir secret sharing scheme [29] which is information-theoretic secure. The strong private key SK is randomly split into two shares in a way that any less than two shares cannot recover the original SK (i.e., $(2, 2)$-Shamir secret sharing technique is used). It further implies that the adversary cannot cover the original plaintext with less than two shares of partial decrypted ciphertexts (as the adversary can select a share all by himself). $\square$

B. Security Model Definition

Here, we recall the security model for securely realizing an ideal functionality in the presence of non-colluding semi-honest adversaries [30]. For simplicity, we use the scenario involving two parties, DP a (i.e. "D_a") and DP b (i.e. "D_b"), and two servers CP (i.e. "S_1") and CSP (i.e. "S_2"). We refer the reader to [31] for the general case definition.

Let $\mathcal{P} = (D_a, D_b, S_1, S_2)$ be the set of all protocol parties. We consider four kinds of adversaries $(\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ that corrupt D_a, D_a, S_1 and S_2, respectively. In the real world, D_a and D_b run with inputs x and y (with additional auxiliary inputs z_x and z_y), respectively, while S_1 and S_2 receive auxiliary inputs z_1 and z_2. Let $H \subset \mathcal{P}$ be the set of honest parties. Then, for every $P \in H$, let out_P be the output of party P. If P is corrupted (i.e. $P \in \mathcal{P} \backslash H$), then out_P denotes the view of P during the protocol Π.

For every $P^* \in \mathcal{P}$, the partial view of P^* in a real-world execution of protocol Π in the presence of adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ is defined as

$$REAL^{P^*}_{\Pi,\mathcal{A},H,\mathbf{z}}(x, y) = \{out_P : P \in H\} \cup out_{P^*}.$$

In the ideal world, there is an ideal functionality $\mathbf{f}$ for a function f and the parties interact only with $\mathbf{f}$. Here, the challenge DP a and DP b send x and y to $\mathbf{f}$, respectively. If either x or y is $\perp$, then $\mathbf{f}$ returns $\perp$. Finally, $\mathbf{f}$ returns $f(x, y)$ to the challenge RU. As before, let $H \subset \mathcal{P}$ be the set of honest parties. Then, for every $P \in H$, let out_P be the output returned by $\mathbf{f}$ to party P. If P is corrupted, then out_P is the same value returned by P.

For every $P^* \in \mathcal{P}$, the partial view of P^* in an ideal-world execution in the presence of independent simulators $\mathsf{Sim}= (\mathsf{Sim}_{D_a}, \mathsf{Sim}_{D_b}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2})$ is defined as

$$IDEAL^{P^*}_{\mathbf{f},\mathsf{Sim},H,\mathbf{z}}(x, y) = \{out_P : P \in H\} \cup out_{P^*}.$$

Informally, a protocol Π is considered secure against non-colluding semi-honest adversaries if it partial emulates, in the real world, an execution of $\mathbf{f}$ in the ideal world. More formally,

Definition 1: Let $\mathbf{f}$ be a deterministic functionality among parties in $\mathcal{P}$. Let $H \subset \mathcal{P}$ be the subset of honest parties in $\mathcal{P}$. We say that Π securely realizes $\mathbf{f}$ if there exists a set $\mathsf{Sim}= (\mathsf{Sim}_{D_a}, \mathsf{Sim}_{D_b}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2})$ of PPT transformations (where $Sim_{D_a} = Sim_{D_a}(\mathcal{A}_{D_a})$ and so on) such that for all semi-honest PPT adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$, for all inputs x, y and auxiliary inputs z, and for all parties $P \in \mathcal{P}$ it holds

$$\{REAL^{P^*}_{\Pi,\mathcal{A},H,\mathbf{z}}(\lambda, x, y)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{IDEAL^{P^*}_{\mathbf{f},\mathsf{Sim},H,\mathbf{z}}(\lambda, x, y)\}_{\lambda\in\mathbb{N}}$$

where $\overset{c}{\approx}$ denotes computational indistinguishability.

C. The Security of Sub-Protocols

Here, we prove the security of the sub-protocols based on the security model defined in Section VI-B.[6]

*Theorem 2: The **SAD** protocol described in Section V-A securely computes addition over ciphertext across domains in the presence of semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.*

Proof: Here, we show how to construct four independent simulators, namely $\mathsf{Sim}_{D_a}, \mathsf{Sim}_{D_b}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2}$.

Sim_{D_a} receives x as input and simulates $\mathcal{A}_{D_a}$ as follows: It generates encryption $[x]_{pk_a} \leftarrow \mathbf{Enc}(pk_a, x)$ of x, returns $[x]_{pk_a}$ to $\mathcal{A}_{D_a}$, and outputs $\mathcal{A}_{D_a}$'s entire view. The view of $\mathcal{A}_{D_a}$ consists of the encrypted data. The views of $\mathcal{A}_{D_a}$ in both real and ideal executions are indistinguishable due to the semantic security of DT-PKC.

Sim_{D_b} works analogously to Sim_{D_a}.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: It generates (fictitious) encryptions of the inputs $[\hat{x}]_{pk_a}$ and $[\hat{y}]_{pk_b}$ by running $\mathbf{Enc}(\cdot, \cdot)$ on randomly chosen $\hat{x}, \hat{y}$, randomly generates $r_a, r_b \in \mathbb{Z}_N$, and calculates $\hat{X}$ and $\hat{Y}$. Then, it calculates $\hat{X}'$ and $\hat{Y}'$ using $\mathbf{PWDec1}(\cdot, \cdot)$. After that, Sim_{S_1} sends the encryption $\hat{X}$, $\hat{Y}$, $\hat{X}'$ and $\hat{Y}'$ to $\mathcal{A}_{S_1}$. If $\mathcal{A}_{S_1}$ replies with $\perp$, then Sim_{S_1}

[6] Although the model described in Section VI-B can be employed to protect the content of the data (including the input data and its final output), this model does not capture information leakage due to data access pattern. The latter can be solved using oblivious RAM for secure two-party computation, which is beyond the scope of this paper. We refer interested reader to [32] for the construction.

returns $\perp$. The view of $\mathcal{A}_{S_1}$ consists of the encrypted data it creates. In both real and the ideal executions, it receives the output of the encryptions $\hat{X}$, $\hat{Y}$, $\hat{X}'$, and $\hat{Y}'$. In the real world, it is guaranteed by the fact that the DPs are honest and the semantic security of DT-PKC. The views of $\mathcal{A}_{S_1}$ in the real and the ideal executions are indistinguishable.

Sim_{S_2} simulates $\mathcal{A}_{S_2}$ as follows: It randomly chooses $\hat{S}$, uses the $\mathbf{Enc}(\cdot,\cdot)$ to obtain $[\hat{S}]_{pk_\Sigma}$, and then sends the encryption to $\mathcal{A}_{S_2}$. If $\mathcal{A}_{S_2}$ replies with $\perp$, then Sim_{S_2} returns $\perp$. The view of $\mathcal{A}_{S_2}$ consists of the encrypted data it creates. In both real and the ideal executions, it receives the output of the encryptions $[S]_{pk_\Sigma}$. In the real world, it is guaranteed by the semantic security of DT-PKC. The views of $\mathcal{A}_{S_1}$ in the real and the ideal executions are indistinguishable. □

The security proof of **SMD** is similar to that of **SAD** protocol under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. In the following section, we prove the security of **SLT**.

*Theorem 3: The **SLT** protocol described in Section V-D is to securely evaluate the comparison result of plaintext over ciphertext in the presence of semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.*

Proof: We now demonstrate how to construct four independent simulators, namely: $\mathsf{Sim}_{D_a}, \mathsf{Sim}_{D_b}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2}$.

Sim_{D_a} receives x as input and then simulates $\mathcal{A}_{D_a}$ as follows: It generates encryption $[x]_{pk_a} \leftarrow \mathbf{Enc}(pk_a, x)$ of x, prior to returning $[x]_{pk_a}$ to $\mathcal{A}_{D_a}$ and producing $\mathcal{A}_{D_a}$'s entire view. The view of $\mathcal{A}_{D_a}$ consists of the encrypted data. The views of $\mathcal{A}_{D_a}$ in both real and the ideal executions are indistinguishable due to the semantic security of DT-PKC.

Sim_{D_b} works analogously to Sim_{D_a}.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: it generates (fictitious) encryptions of the inputs $[\hat{x}]_{pk_a}$ and $[\hat{y}]_{pk_b}$ by running $\mathbf{Enc}(\cdot,\cdot)$ on randomly chosen $\hat{x}$, $\hat{y}$. Then, it calculates $[\hat{x}_1]_{pk_a}$ and $[\hat{y}_1]_{pk_b}$, which are used as inputs to $\mathsf{Sim}_{S_1}^{(\mathbf{SAD})}(\cdot)$ and generate $[\hat{l}]_{pk_\Sigma}$ according to the randomly tossed coin $\hat{s}$. It calculates $[\hat{l}_1]_{pk_\Sigma}$ and $\hat{K}$ using $[\hat{l}]_{pk_\Sigma}$, randomly tosses a coin $\hat{u^*}$, and generates $[\hat{u^*}]_{pk_\Sigma}$ by running $\mathbf{Enc}(\cdot,\cdot)$. Finally, the encryption $[\hat{l}_1]_{pk_\Sigma}$, $\hat{K}$, and the middle encrypted data executed by $\mathsf{Sim}_{S_1}^{(\mathbf{SAD})}(\cdot,\cdot)$ are sent to $\mathcal{A}_{S_1}$. If $\mathcal{A}_{S_1}$ replies with $\perp$, then Sim_{S_1} returns $\perp$. In the real world, this is guaranteed by the fact that the DPs are honest and the semantic security of DT-PKC. The views of $\mathcal{A}_{S_1}$ in both real and ideal executions are indistinguishable.

Sim_{S_2} is analogous to Sim_{S_1}. □

The security proofs of **SEQ**, **SSBA**, **SDIV**, **SMMS**, and **SGCD** are similar to that of **SLT** under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_a}, \mathcal{A}_{D_b}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. Next, we will demonstrate that our EPOM is secure under an active adversary $\mathcal{A}^*$ defined in III-C.

D. Security of EPOM

If $\mathcal{A}^*$ eavesdrops on the transmission between the challenge RU and the CP, the original encrypted data and the final results will be obtained by $\mathcal{A}^*$. Moreover, ciphertext results (obtained by executing **SAD**, **SMD**, **SLT**, **SSBA**, **SMMS**, **SEQ**, **SDIV**, and **SGCD**) transmitted between CP and CSP may also be made available to $\mathcal{A}^*$ due to the eavesdropping. As these data are encrypted during transmission, $\mathcal{A}^*$ will not be able to decrypt the ciphertext without knowing the challenge DP's private key due to the semantic security of the DT-PKC cryptosystem. Next, suppose $\mathcal{A}^*$ has compromised CP or CSP to obtain the partial strong private key. However, $\mathcal{A}^*$ is unable to recover the strong private key to decrypt the ciphertext, as the private key is randomly split by executing **SKeyS** algorithm of DT-PKC. Even when $\mathcal{A}^*$ obtains all plaintext value from the sub-protocols by compromising CSP, it is still unable for $\mathcal{A}^*$ to obtain useful information as our protocols use the known technique of "blinding" the plaintext [33]: given an encryption of a message, we use the additive homomorphic property of the DT-PKC cryptosystem to add a random message to it. Therefore, original plaintext is "blinded". In the event that $\mathcal{A}^*$ gets hold of private keys belonging to other DPs/RUs (i.e. not the challenge DPs/RUs), $\mathcal{A}^*$ is still unable to decrypt the challenge DP's ciphertext or the challenge RU's final result due to the unrelated property of different DP and RU's weak private keys in our system (recall private keys in the system are selected randomly and independently).

VII. Performance Analysis

In this section, we evaluate the performance of EPOM.

A. Experiment Analysis

The computation cost and communication overhead of the proposed EPOM were evaluated using a custom simulator built in Java, and the evaluations were performed on a personal computer (PC) with 3.6 GHz eight-cores processor and 12 GB RAM memory.

1) Basic Crypto Primitive & Protocols' Performance: We first evaluate the performance of our basic cryptographic primitive and toolkit for integer on our PC testbed – see Tables II and III, respectively. We denote N as 1024 bits to achieve 80-bit security levels [34]. We then use a smartphone with eight-core processor (4×Cortex-A17 + 4×Cortex-A7) and 2 GB RAM memory to evaluate the performance of the basic crypto primitive – see Table II. The evaluations demonstrated that the algorithms in DT-PKC are suitable for both PC and smartphone deployments. Note that the toolkit for integer number calculations is designed for outsourced computation; therefore, they are only evaluated in the PC testbed.

2) Factors Affecting Protocols' Performance: For our proposed DT-PKC, the length of N will affect the running time of the proposed cryptosystem. From Fig. 2(a) and Fig. 2(b), we observe that both run time and communication overhead of the basic algorithms increase with N. This is because run time of the basic operations (modular multiplication and exponent) increases as N increases, resulting in the transmission of more bits. For the toolkit of integer protocols, two factors will affect the performance, namely: (i) length of N (for all protocols), and (ii) domain size of the plaintext (for **SBD**, **SDIV**, and **SGCD**). From Fig. 2(c) - Fig. 2(h), we observe that both computational and communication costs of all protocols increase with N, as the protocols rely on the basic DT-PKC

TABLE II

THE PERFORMANCE OF DT-PKC (1000-TIME ON AVERAGE, 80-BIT SECURITY LEVEL)

Algorithm	Enc	CR	PDec	WDec	PSDec1	PSDec2	PWDec1	PWDec2
PC Run Time	16.408 ms	16.275 ms	8.361 ms	8.432 ms	23.135 ms	23.248 ms	8.257 ms	8.799 ms
Smartphone Run Time	89.671 ms	90.643 ms	47.043 ms	50.651 ms	135.130 ms	130.712 ms	45.675 ms	57.240 ms

TABLE III

PERFORMANCE OF SUB-PROTOCOL (1000-TIMES FOR AVERAGE, 80-BITS SECURITY LEVEL)

Protocol	CP compute.	CSP compute.	Commu.
SAD	124.913 ms	61.420 ms	1.998 KB
SMD	340.479 ms	141.226 ms	4.491 KB
SBD (10-bits)	0.969 s	0.396 s	14.997 KB
SSBA	459.936 ms	185.559 ms	5.741 KB
SLT	192.226 ms	96.237 ms	3.244 KB
SEQ	1.006 s	0.439 s	15.485 KB
SMMS	1.054 s	0.459 s	16.219 KB
SDIV (10-bits)	9.263 s	3.742 s	132.039 KB
SGCD (10-bits)	102.675 s	42.899 s	1.446 MB

and basic operations. From Fig. 2(i) - Fig. 2(k), we observe that **SBD**, and the computational cost and the communication overhead in both **SDIV** and **SGCD** increase with the plaintext bit length. This is due to the increase in encrypted data which consumes more computation and communication resources. Next, we present the theoretical analysis for EPOM.

a) Optimization: From Fig. 2 and Table III, we note that both computational overhead and communication cost at the server's side are relatively high. Thus, it is important to find solutions to speed up server-side computation and reduce communication rounds. First, some protocol steps can be processed in parallel. For example, in the **SEQ** protocol, two **SLT** protocols can execute simultaneously, following by simultaneous execution of two **SMD** protocols; thus, reducing four communication rounds into two. In addition, privacy-preserving calculation for each tuple can be processed independently. Therefore, we can use parallel computing [35], [36] to solve the problem (i.e. all tuples can be processed in parallel and simultaneously). We can also use GPU [37], [38] to accelerate the computation. Specifically, using GPU allows us to execute individual protocols simultaneously, whilst in our experiment, we use CPU-based calculation and serial computing to execute the protocol one at a time. For individual protocol acceleration, we can also choose to use a server with higher performance specification or a cloud.

B. Theoretical Analysis

1) Computational Overhead: Let us assume that one regular exponentiation operation with an exponent of length $|N|$ requires $1.5|N|$ multiplications [39] (i.e. length of r is $|N|$ and that computing g^r requires $1.5|N|$ multiplications). As an exponentiation operation is more costly than an addition operation or a multiplication operation, we ignore the fixed numbers of addition and multiplication operations in our analysis. For the DT-PKC scheme, **Enc** and **CR** algorithm require $3|N|$ multiplications to encrypt the message, **WDec** algorithm costs $1.5|N|$ multiplications to decrypt the message, **PDecW1** needs $1.5|N|$ multiplications to process, and **PDecW2** needs $1.5|N| + \kappa$ multiplications. **SDec** needs $1.5|N|$ multiplications to decrypt the ciphertextï¼Œ **PDecS1** needs $4.5|N|$ multiplications to process, and **PDecS2** needs $4.5|N|$ multiplications.

For the basic sub-protocols, it costs $21|N|$ multiplications for CP and $12|N|$ for CSP by executing the **SAD** protocol. For the **SMD** protocol, it costs $45|N|$ multiplications for CP and $27|N|$ multiplications for CSP to run. For the **SMMS** protocol, it costs $172.5|N|$ multiplications for CP and $97.5|N|$ multiplications for CSP to run. For the **SBD** protocol, it costs between $13.5\mu|N|$ multiplications (best case) and $16.5\mu|N|$ multiplications (worst case) for CP, and takes $7.5\mu|N|$ multiplications for CSP to run. For the **SSBA** protocol, it costs $58.5|N|$ multiplications for CP and $34.5|N|$ multiplications for CSP to run. For the **SLT** protocol, it costs $34.5|N|$ multiplications for CP and $19.5|N|$ multiplications for CSP to run. For the **SEQ** protocol, it costs $165|N|$ multiplications for CP and $93|N|$ multiplications for CSP to run. For the **SDIV** protocol, it costs $O(\mu^2|N| + \mu^3)$ multiplications for CP and costs $O(\mu^2|N|)$ multiplications for CSP to run. For the **SGCD** protocol, it costs $O(\mu^3|N| + \mu^4)$ multiplications for CP and $O(\mu^3|N|)$ multiplications for CSP.

2) Communication Overhead: In the DT-PKC scheme, each T_1, T_2, CT and WT require $2|N|$ bits to represent. Thus, the ciphertext $[x]_{pk}$ needs $4|N|$ bits to transmit. For the basic sub-protocols, it takes $16|N|$ bits between CP and CSP to run the **SAD** protocol. Also, it takes $36|N|$ bits between CP and CSP to run the **SMD** protocol, $46|N|$ bits to run the **SSBA** protocol, $174|N|$ bits to run the **SLT** protocol, $278|N|$ bits to run the **SMMS** protocol, $420|N|$ bits for the **SEQ** protocol, $10\mu|N|$ bits to run the **SBD** protocol, $O(\mu^2|N|)$ bits to run the **SDIV** protocol, and $O(\mu^3|N|)$ bits to run the **SGCD** protocol.

C. Comparative Summary

Our EPOM is closely related to the work in [33], where two servers ($\mathcal{C}$ and $\mathcal{S}$) are used to process the encrypted data under multiple keys. The multi-key ciphertexts are stored in server $\mathcal{C}$, while server $\mathcal{S}$ directly holds the strong private key. However, as we pointed out in Section IV, the decryption ability of strong trapdoor is too powerful (i.e. capability to decrypt all ciphertexts in the system). Consequently, any leakage or (insider) abuse would result in a major compromise of the system (i.e. single point of attack). For example, a compromised server $\mathcal{S}$ can be used to decrypt all ciphertexts in the transmission link. Our approach differs from [33] in the sense that EPOM randomly separates the strong trapdoor into two shares,[7] and distributes the shares to two different servers. Only when both servers work together can the ciphertext be successfully decrypted. This decreases the risk.

[7]For enhanced security (protection of key leaking), the strong trapdoor can be further separated into n shares, s.t., $\sum_i \lambda_i \equiv 0 \mod \lambda$ and $\sum_i \lambda_i \equiv 1 \mod N^2$ hold at the same time. The shares are then distributed to $n-1$ CSP and CP for storage respectively. This will require an additional $n-2$ servers in the system.

TABLE IV

COMPARATIVE SUMMARY

Algorithm	Addition Round Trips	Multiplication Round Trips	Additive Homomophic Cryptosystem	Support Multi-key	Reduce Key Leaking Risk	Complex Operations	Process non-integer number
[33]	One	Two	✓	✓	×	×	×
Ours	One	One	✓	✓	✓	✓	✓

Moreover, in order to achieve multiplication of the plaintexts in [40], the **KeyProd** protocol should be first used to transfer the ciphertexts with different public keys into the ciphertexts with a same joint public key, without changing the corresponding plaintext value. Then, the **Mult** protocol needs to be used to achieve the plaintext multiplication using the transferred ciphertexts. In all, two rounds of communication are necessary to achieve multiplication of the plaintexts in [33]. In EPOM, only one round of communication is required (i.e. **SMD**). In addition, two protocols are constructed to achieve secure addition and multiplication under multi-key in [33], whilst EPOM realizes commonly used secure operations under multi-key, such as comparison, division, etc. We also remark that our EPOM can be extended to store and process data beyond integer numbers. A comparative summary between the two schemes is shown in Table IV.

VIII. RELATED WORK

With the constant evolution of cloud and related technologies, more users choose to encrypt before outsource their own data to cloud servers for storage. However, it is important to ensure the security and privacy of outsourced data. While homomorphic encryption technique allows searching of encrypted data, it is not yet practical to do so. More specifically, Gentry [40] constructed the first fully homomorphic encryption scheme based on lattice-based cryptography to support an arbitrary number of addition and multiplication operations. Since the seminal work of Gentry in 2009, a number of single-key fully homomorphic encryption schemes (see [41], [42]) and multi-key fully homomorphic encryption schemes (see [12], [43], [44]) had been proposed. However, one of the biggest drawbacks of fully homomorphic cryptosystems is complexity in both computation (including encryption and decryption) and storage (including both public/private key size and ciphertext size). It is not yet practical to implement fully homomorphic cryptosystem in the real-world [26], [27].

Partial homomorphic encryptions (including additive and multiplicative homomorphic encryption) are often considered the next best solution. However, partial homomorphic encryptions can only handle one kind of homomorphic operation with arbitrary times. Additive homomorphic encryption scheme, such as Paillier cryptosystem [15] and Benaloh cryptosystem [16], allows other parties to securely perform some additive homomorphic calculations over the ciphertext. Multiplicative homomorphic encryption scheme, such as unpadded RSA cryptosystem [45] and El-Gamal cryptosystem [46], allows some multiplication over the plaintext. In recent years, some cryptosystems attempt to provide for both additive and multiplicative operations. However, these systems generally achieve only limited numbers of homomorphic operations. For example, the BGN cryptosystem [47] can only support limited numbers of additive homomorphic operations and only one multiplicative homomorphic operation.

A number of privacy-preserving protocols have also been constructed using partial homomorphic encryption, and examples include secure sum protocol [48], [49], secure comparison protocol [50], [51], secure set intersection protocol [52], [53], secure scalar product protocol [54], [55], secure division protocol [25], and secure top-k protocol [14]. Moreover, many real-world applications use these privacy-preserving protocols for system design. For instance, Li et al. [56] used the secure set intersection protocol to construct the profile matching framework. Although these privacy-preserving protocols are promising, these protocols are designed for a single-key setting which is not scalable for a real-world outsourced environment. Peter et al. [33] designed an efficient outsourcing multiparty computation framework for a multi-key setting. However, the scheme does not support complex operations, such as securely perform integer division operation. This is the gap that this paper contributed to.

IX. CONCLUSION

In this paper, we proposed a new efficient and privacy-preserving outsourced calculation framework with multiple keys. The framework is designed to allow different data providers to securely outsource their data with their own public key, and for a cloud server to process the multi-key encryption data on-the-fly. To ensure that the scheme can be deployed in a real-world application, we proposed a new cryptographic primitive, Distributed Two Trapdoors Public-Key Cryptosystem (DT-PKC), to reduce both key management cost and private key exposure risk. We also built toolkit to perform privacy preserving calculations to handle commonly used integer operations in a privacy preserving way. Our evaluations demonstrated that our framework (and the underlying building blocks) are sufficiently efficient for a real-world deployment.

ACKNOWLEDGMENT

The authors thank the associate editor and the anonymous reviewers for their constructive and generous feedback.

REFERENCES

[1] B. Chamberlin. *IoT (Internet of Things) Will go Nowhere Without Cloud Computing and Big Data Analytics*, accessed on 2014. [Online]. Available: http://ibmcai.com/2014/11/20/iot-internet-of-things-will-go-nowhere-without-cloud-computing-and-big-data-analytics/

[2] H. Wang. *Cloud Computing in Ecommerce*, accessed on 2011. [Online]. Available: http://www.comp.leeds.ac.uk/mscproj/reports/1011/wang.pdf

[3] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud computing: Distributed Internet computing for IT and scientific research," *IEEE Internet Comput.*, vol. 13, no. 5, pp. 10–13, Sep./Oct. 2009.

[4] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *Proc. 10th IEEE Int. Conf. High Perform. Comput. Commun. (HPCC)*, Dalian, China, Sep. 2008, pp. 825–830.

[5] D. Quick, B. Martini, and K.-K. R. Choo, *Cloud Storage Forensics*. Amsterdam, The Netherlands: Elsevier, 2013.

[6] V. Kundra. *Federal Cloud Computing Strategy*, accessed on 2011. [Online]. Available: http://www.whitehouse.gov/sites/default/files/omb/\assets/egov_docs/federal-cloud-computing-strategy.pdf

[7] P. M. Figliola and E. A. Fischer. *Overview and Issues for Implementation of the Federal Cloud Computing Initiative: Implications for Federal Information Technology Reform Management*, accessed on 2015. [Online]. Available: https://www.fas.org/sgp/crs/misc/R42887.pdf

[8] National Institute of Standards and Technology. (2014). *U.S. Government Cloud Computing Technology. Roadmap Volume I*. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-293.pdf

[9] University of Melbourne. *The Cloud Computing and Distributed Systems (CLOUDS) Laboratory*, accessed on 2015. [Online]. Available: http://www.cloudbus.org/

[10] *Mobile & Cloud Computing Laboratory (Mobile Cloud Lab)*, accessed on 2016. [Online]. Available: http://mc.cs.ut.ee/

[11] *Cloud Computing*, accessed on 2016. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing

[12] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, 2012, pp. 1219–1234.

[13] E. AbuKhousa, N. Mohamed, and J. Al-Jaroodi, "e-Health cloud: Opportunities and challenges," *Future Internet*, vol. 4, no. 3, pp. 621–645, 2012.

[14] X. Liu, R. Lu, J. Ma, L. Chen, and B. Qin, "Privacy-preserving patient-centric clinical decision support system on Naïve Bayesian classification," *IEEE J. Biomed. Health Informat.*, vol. 20, no. 2, pp. 655–668, Mar. 2016.

[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol.—EUROCRYPT*, Prague, Czech Republic, May 1999, pp. 223–238.

[16] J. Benaloh, "Dense probabilistic encryption," in *Proc. Workshop Sel. Areas cryptogr.*, 1994, pp. 120–128.

[17] B. K. Samanthula, H. Chun, and W. Jiang, "An efficient and probabilistic secure bit-decomposition," in *Proc. 8th ACM Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, Hangzhou, China, May 2013, pp. 541–546.

[18] X. Liu, B. Qin, R. Deng, and Y. Li, "An efficient privacy-preserving outsourced computation over public data," *IEEE Trans. Service Comput.*, to be published.

[19] Q. Do, B. Martini, and K.-K. R. Choo, "A forensically sound adversary model for mobile devices," *PLoS ONE*, vol. 10, no. 9, p. e0138449, 2015.

[20] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Proc. 9th Int. Conf. Theory Appl. Cryptol. Inf. Secur. Adv. Cryptol.—ASIACRYPT*, Taipei, Taiwan, Nov./Dec. 2003, pp. 37–54.

[21] P.-A. Fouque and D. Pointcheval, "Threshold cryptosystems secure against chosen-ciphertext attacks," in *Advances in Cryptology—ASIACRYPT*. Gold Coast, Australia: Springer, 2001, pp. 351–368.

[22] R. Cramer and V. Shoup, "Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol.—EUROCRYPT*, Amsterdam, The Netherlands, Apr./May 2002, pp. 45–64.

[23] L. J. Hoffman, K. Lawson-Jenkins, and J. J. Blum, "Trust beyond security: An expanded trust model," *Commun. ACM*, vol. 49, no. 7, pp. 94–101, 2006.

[24] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981.

[25] X. Liu, R. Choo, R. Deng, R. Lu, and J. Weng, "Efficient and privacy-preserving outsourced calculation of rational numbers," *IEEE Trans. Dependable Secure Comput.*, to be published.

[26] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography—PKC*. Tokyo, Japan: Springer, 2010, pp. 420–443.

[27] L. Morris. (2013). *Analysis of Partially and Fully Homomorphic Encryption*. [Online]. Available: http://www.liammorris.com/crypto2/\Homomorphic%20Encryption%20\Paper.pdf

[28] C. Ding, *Chinese Remainder Theorem*. Singapore: World Scientific, 1996.

[29] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[30] D. Catalano and D. Fiore, "Boosting linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data," *IACR Cryptol. ePrint Archive*, Oct. 2014. [Online]. Available: http://eprint.iacr.org/2014/813

[31] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *IACR Cryptol. ePrint Archive*, Oct. 2011. [Online]. Available: http://eprint.iacr.org/2011/272

[32] S. Lu and R. Ostrovsky, "Distributed oblivious RAM for secure two-party computation," in *Theory of Cryptography*. Tokyo, Japan: Springer, 2013, pp. 377–396.

[33] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 2046–2058, Dec. 2013.

[34] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, """Recommendation for key management part 1: General (revised)," *NIST Special Publication*, vol. 800, no. 57, pp. 1–142, 2007.

[35] A. Geist, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA, USA: MIT Press, 1994.

[36] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency, Pract. Exper.*, vol. 2, no. 4, pp. 315–339, Dec. 1990.

[37] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[38] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010.

[39] D. E. Knuth, *Art of Computer Programming: Seminumerical Algorithms*, vol. 2. Reading, MA, USA: Addison-Wesley, 1981.

[40] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput. (STOC)*, Bethesda, MD, USA, May/Jun. 2009, 2009, pp. 169–178.

[41] C. Gentry and S. Halevi, "Fully homomorphic encryption without squashing using depth-3 arithmetic circuits," in *Proc. IEEE 52nd Annu. Symp. Found. Comput. Sci. (FOCS)* Palm Springs, CA, USA, Oct. 2011, pp. 107–109.

[42] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Comput.*, vol. 43, no. 2, pp. 831–871, 2014.

[43] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled FHE from learning with errors," in *Proc. 35th Annu. Cryptol. Conf. Adv. Cryptol.—CRYPTO*, Santa Barbara, CA, USA, Aug. 2015, pp. 630–656.

[44] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," *IACR Cryptol. ePrint Archive*, Apr. 2015. [Online]. Available: http://eprint.iacr.org/2015/345

[45] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[46] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[47] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Theory of Cryptography*. Cambridge, MA, USA: Springer, 2005, pp. 325–341.

[48] R. Sheikh, B. Kumar, and D. K. Mishra, "Privacy preserving k secure sum protocol," *CoRR*, vol. abs/0912.0956, 2009. [Online]. Available: http://arxiv.org/abs/0912.0956

[49] R. Sheikh, B. Kumar, and D. K. Mishra, "A distributed k-secure sum protocol for secure multi-party computations," *CoRR*, vol. abs/1003.4071, 2010. [Online]. Available: http://arxiv.org/abs/1003.4071

[50] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the Millionaires' problem based on homomorphic encryption," in *Applied Cryptography and Network Security*. New York, NY, USA: Springer, 2005, pp. 456–466.

[51] X. Liu, R. Lu, J. Ma, L. Chen, and H. Bao, "Efficient and privacy-preserving skyline computation framework across domains," *Future Generat. Comput. Syst.*, vol. 62, pp. 161–174, Sep. 2016.

[52] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Berlin, Germany, Nov. 2013, pp. 789–800.

[53] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, Seoul, South Korea, May 2012, 2012, pp. 85–86.

[54] A. Amirbekyan and V. Estivill-Castro, "A new efficient privacy-preserving scalar product protocol," in *Proc. 6th Australasian Conf. Data Mining Anal.*, vol. 70. 2007, pp. 209–214.

[55] R. Lu, H. Zhu, X. Liu, J. K. Liu, and J. Shao, “Toward efficient and privacy-preserving computing in big data era,” *IEEE Netw.*, vol. 28, no. 4, pp. 46–50, Jul./Aug. 2014.

[56] M. Li, N. Cao, S. Yu, and W. Lou, “FindU: Privacy-preserving personal profile matching in mobile social networks,” in *Proc. 30th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Shanghai, China, Apr. 2011, pp. 2435–2443.

Ximeng Liu (S’13–M’16) received the B.Sc. degree in electronics engineering and the Ph.D. degree in cryptography from Xidian University, Xi’an, China, in 2010 and 2015, respectively. He was a Research Assistant with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, from 2013 to 2014. He is currently a Research Fellow with the School of Information System, Singapore Management University, Singapore. His research interests include cloud security, applied cryptography, and big data security.

Robert H. Deng (F’16) has been a Professor with the School of Information Systems, Singapore Management University, since 2004. His research interests include data security and privacy, multimedia security, and network and system security. He was an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY from 2009 to 2012. He is currently an Associate Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and *Security and Communication Networks* (John Wiley). He is the cochair of the Steering Committee of the ACM Symposium on Information, Computer and Communications Security. He received the University Outstanding Researcher Award from the National University of Singapore in 1999 and the Lee Kuan Yew Fellow for Research Excellence from Singapore Management University in 2006. He was named Community Service Star and Showcased Senior Information Security Professional by (ISC)^2 under its Asia-Pacific Information Security Leadership Achievements Program in 2010.

Kim-Kwang Raymond Choo received the Ph.D. degree in information security from Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship at the University of Texas at San Antonio, and is also an Associate Professor with the University of South Australia, and a Guest Professor with the China University of Geosciences, Wuhan, China. He was a recipient of various awards, including the ESORICS 2015 Best Paper Award, the 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, the Fulbright Scholarship in 2009, the 2008 Australia Day Achievement Medallion, and the British Computer Society’s Wilkes Award in 2008.

Jian Weng received the B.S. and M.S. degrees from the South China University of Technology, in 2000 and 2004, respectively, and the Ph.D. degree from Shanghai Jiao Tong University, in 2008, all in computer science and engineering. From 2008 to 2010, he held a postdoctoral position with the School of Information Systems, Singapore Management University. He is currently a Professor and Vice Dean of the School of Information Technology, Jinan University. He has authored over 60 papers in cryptography conferences and journals, such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, CT-RSA, IEEE TDSC, and IEEE TIFS. He served as a PC cochair or PC member for more than 20 international conferences. He received the 2014 Cryptographic Innovation Award from the Chinese Association for Cryptographic Research, the best paper award from the 28th Symposium on Cryptography and Information Security (2011), and the Best Student Award from the 8th International Conference on Provable Security (2014).