11-2016

Privacy-preserving outsourced calculation on floating point numbers

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

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

Wenxiu DING
University of New Brunswick

Rongxing LU
Southwest University of Science and Technology

DOI: https://doi.org/10.1109/TIFS.2016.2585121

Published in IEEE Transactions on Information Forensics and Security, 2016 Nov, Volume 11, Issue 11, Pages 2513-2527.
http://doi.org/10.1109/TIFS.2016.2585121

Privacy-Preserving Outsourced Calculation of Floating Point Numbers

Ximeng Liu, *Member, IEEE,* Robert H. Deng, *Fellow, IEEE,* Wenxiu Ding, Rongxing Lu, *Senior Member, IEEE,* Baodong Qin

***Abstract*—In this paper, we propose a framework for privacy-preserving outsourced calculation of floating point numbers, which we refer to as POCF. Using POCF, a user can securely outsource the storing and processing of floating point numbers to a cloud server while preserving the privacy of the (original) data and the computed results. Specifically, we first present privacy-preserving integer processing protocols for common integer operations. We then present an approach to outsourcing floating point numbers for storage in privacy-preserving way, and securely processing commonly used floating point number operations on-the-fly. We prove that the proposed POCF achieves the goal of floating point number processing without privacy leakage to unauthorized parties, and demonstrate the utility and the efficiency of POCF using simulations.**

***Index Terms*—Privacy-Preserving; Homomorphic Encryption; Outsourced Computation; Floating Point Numbers.**

I. Introduction

CLOUD computing, with its immense computing and storage power, is a promising solution to store and process data for individuals as well as organizations. Cloud computing has been increasingly used in various domains such as Internet of Things (IoT) [1], e-commerce [2], and e-healthcare [3]. The U.S. Federal Government cloud computing market enters into double-digit growth phase - at about 16.2% compounded annual growth rate over the period of 2015-2020, with annual federal cloud computing market to hit $10 billion landmark by 2020 [4]. Forrester forecasts that the global market for cloud computing will grow from $40.7 billion in 2011 to more than $241 billion in 2020 [5]. It is, therefore, unsurprising that research labs, such as [6], [7], dedicated to cloud computing research have also been set up.

Real numbers are the most common numerical data used in measuring quantities that we use every day. In an e-heathcare cloud system, body sensors are used to monitor patients' conditions, such as heart rates, blood glucoses, insulin, and C-peptide levels. These data (always contain real numbers) are sent to the cloud for storage and processing because sensors are invariably resource-constrained. However, patients' data are highly sensitive and must be protected in accordance with regulations such as Health Insurance Portability and Accountability Act (HIPPA). Thus, the security and privacy of personal health information in cloud computing is considered an important issue. Moreover, computation on the outsourced data is required in many applications. In the e-heathcare cloud example, some decision making model can be used to automatically check a patients' health status. The parameters of the decision making model frequently contain real numbers (e.g., the probability used in the naïve Bayes model [8]), so do the outsourced personal health data (e. g., blood glucose, insulin, and C-peptide levels are often constructed as non-integer, see the recent study involving 36,745 subjects aged between 40 and 69 years old in the Japan Public Health Center-based prospective study [9]).

However, cryptosystems are generally designed to protect only integer values and computations on integers envivnately affect the accuracy of data and decision making results, and may even lead to wrong diagnosis of a patient's illness. Floating point numbers (FPNs), ubiquitous in computer systems and languages, are the formulaic representations that approximate real numbers so as to support a trade-off between range and precision. How to achieve floating point number calculation without compromising the privacy of the outsourced data becomes a challenging issue.

In this paper, we seek to address the above-mentioned challenge by presenting a framework for **P**rivacy-preserving **O**utsourced **C**alculation of **F**loating Point Numbers (POCF). Different from the exiting secure FPN calculation frameworks [10], [11], [12], [13], [14], [15], we regard the challenge issues of our POCF to be five-fold, namely:

- *Secure Centralized Floating Point Number Storage:* Floating point numbers are encrypted and outsourced to central public cloud servers without unnecessary plaintext expansion. Data splitting method, which is used to generate extra dummy messages [10], [11], [12], [13], [14], is not required in POCF. Instead, all the FPNs are centrally stored in the cloud platform with specific format and constant-length.
- *Secure Floating Point Number Computation:* Our POCF can securely achieve the commonly used outsourced FPN operations on-the-fly. Both the original FPNs and the final computed results are not leaked to any unauthorized parties, including curious-but-honest insiders, during the secure FPN processing.
- *Efficient Exception Handling:* Our framework is able to handle FPN exceptions (such as overflow and underflow) from storage to processing. No complex pre-computation

X. Liu, R.H. Deng, and W. Ding are with the School of Information Systems, Singapore Management University, Singapore. E-mail:snbnix@gmail.com, robertdeng@smu.edu.sg, wenxiuding_1989@126.com.

R. Lu is with the School of Electrical and Electronics Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore. E-mail: rxlu@ntu.edu.sg.

B. Qin is with School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang, China. E-mail: qinbaodong@swust.edu.cn.

is needed in order to handle exceptions and no extra component is required to store any exception information for encrypted FPNs in our POCF.
- *Supporting Iterative Calculations:* In order to support unlimited iterative calculations, both the original and the computed FPNs in POFC maintain the same format such that the output of the previous secure FPN computations can be directly used as the input for the next secure FPN computation.
- *Ease of Use:* POFC does not require its clients to perform any complex pre-processing to outsource FPNs to the cloud - a client only needs to encrypt-and-outsource his FPNs. Moreover, the interaction between clients and the cloud server is kept to the minimal - a client only needs to send a computation query to the cloud server, and waits for the cloud to reply with the result of the computation in a single round.

The remainder of this paper is organized as follows: In Section II, we describe some preliminaries required for understanding of our proposed POCF. In Section III, we formalize the system model, state the problem, and show the attacker model. Then, we present Paillier cryptosystem with partial decryption (PCPD) and privacy-preserving integer processing protocols in Section IV, followed by secure floating point number storage and calculation protocols in Section V. 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 review the definitions of Additive Homomorphic Cryptosystem (AHC) and Floating Point Numbers (FPNs), which serve as the basic conception for constructing the proposed POCF.

A. Notations

Throughout the paper, we use $\|x\|$ to denote bit length of x, and $|x|$ to represent the absolute value of x. Moreover, we use pk_a and sk_a to denote the public and private keys of a Request User (RU) a, $sk_a^{(1)}, sk_a^{(2)}$ to denote the partial private keys that form sk_a, $[x]_{pk_a}$ to denote the encrypted data of x using pk_a, and $D_{sk_a}(\cdot)$ to denote the decryption algorithm using sk_a. For simplicity, if all ciphertexts belong to a specific RU, say a, we will simply use $[x]$ instead of $[x]_{pk_a}$.

B. Additive Homomorphic Cryptosystem (AHC)

Suppose that $[m_1]$ and $[m_2]$ are two additive homomorphic ciphertexts under the same public key pk in an additive homomorphic cryptosystem. The additive homomorphic cryptosystem (e.g. Paillier cryptosystem [16] and Benaloh cryptosystem [17]) has the **additive homomorphism** property:

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

C. Floating Point Numbers (FPNs)

Here, we first define the floating-point format used in our paper. We say a floating-point format is characterized by four integers: 1) a radix (or base) $\beta \geq 2$; 2) a precision $\eta \geq 2$ (roughly speaking, η is the number of "significant digits" of the representation); 3) two extremal exponents e_{min} and e_{max} such that $e_{min} < 0 < e_{max}$. A finite floating-point number in such a format is a number for which there exists at least one representation triplet (s, m, e), such that,

$$x = (-1)^s \cdot m \cdot \beta^{e-\eta+1}$$

- $s \in \{0, 1\}$ is the sign of x.
- m is an integer of absolute value less than or equal to $\beta^\eta - 1$. It is called the integral *significand* of the representation of x;
- e is an integer such that $e_{min} \leq e \leq e_{max}$, called the *exponent* of the representation of x.

Note that the result of an operation (or function) on floating-point numbers is not exactly representable in the floating-point system being used, so it has to be rounded. For instance, the four rounding modes that appear in the IEEE 754-2008 standard [18] are:

Round toward $-\infty$: $\mathrm{RD}(x)$ the largest floating-point number (possibly $-\infty$) less than or equal to x;

Round toward $+\infty$: $\mathrm{RU}(x)$ is the smallest floating-point number (possibly $+\infty$) greater than or equal to x;

Round toward zero: $\mathrm{RZ}(x)$ is the closest floating-point number to x that is no greater in magnitude than x.

Round to nearest: $\mathrm{RN}(x)$ is the floating-point number that is the closest to x.

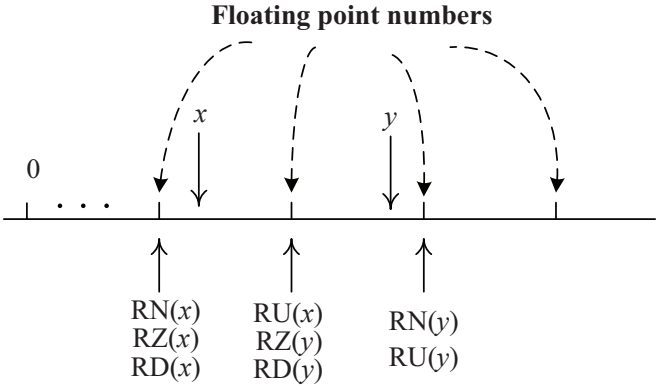


Fig. 1. The four rounding modes

We illustrate these four rounding modes in Fig. 1. Without loss of generality, we use *round toward zero* mode in our POCF.

III. System Model & Privacy Requirement

In this section, we formalize the POCF 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), and Request Users (RUs) - see Fig. 2.

- **KGC:** The trusted KGC is tasked with the distribution and management of the private keys in the system.
- **RUs:** Generally, a RU uses its public key to encrypt some data, and then outsources the encrypted data to a CP. The RU can also request a CP to perform some calculations over the outsourced data.
- **CP:** A CP has 'unlimited' data storage space, and stores and manages data outsourced from all registered RUs. A CP also stores all the intermediate and final results in encrypted form. Furthermore, a CP is able to perform certain calculations over encrypted data.
- **CSP:** A CSP provides online computation services to users. For example, a CSP can process encrypted data, such as multiplication of plaintext over the encrypted data. Also, the CSP is able to partially decrypt ciphertexts sent by the CP, perform certain calculations over the partially decrypted data, and then re-encrypt the calculated results.

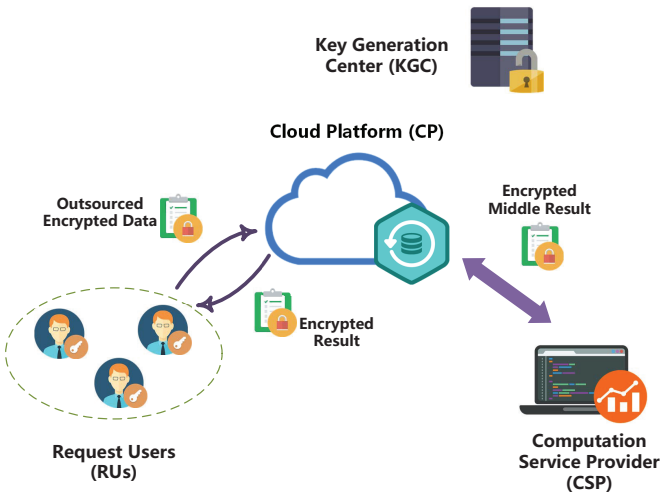


Fig. 2. System model under consideration

B. Problem Statement

Consider a database T which contains α records x_i $(1 \leq i \leq \alpha)$, where x_i is a floating point number[1] belonging to a RU (e.g. insulin level). Data should be encrypted prior to being outsourced to a CP for storage. The RU can launch a query to the CP to obtain some statistic information about T at will. For example, the RU can query for the mean, $\bar{x} = \sum_{i=1}^{\alpha} x_i/\alpha$, and the variance, $d_j = \sum_{i=1}^{\alpha}(x_i - \bar{x})^2/\alpha$, as well as performing self-defined calculations, such as some sums (e.g. $X = \sum_{i=1}^{\alpha} x_i$) and multiplications (e.g. $X' = \prod_{i=1}^{\alpha} x_i$). As x_i is a FPN and needs to be encrypted during the calculation, we have the following challenges:

[1]The non-numeric data can be stored in our POCF. A group of non-numeric data can be encrypted and stored simultaneously in order to reduce the storage overhead (see Section V-C).

1) Traditional encryption method can only encrypt positive integers and zero over a finite field. Therefore, we need to be able to store FPNs without compromising the privacy of the data owner (RU).

2) Before constructing FPN calculation, the encrypted integer calculation protocols need to been inbuilt first to support for commonly used operations. For example, integer number operations, such as multiplications, comparisons, and modular, should be achievable by operating on the encrypted integer numbers.

3) In order to support outsourced FPN processing, the basic operations on FPNs (e.g. addition, multiplication, and comparison between encrypted FPNs) need to be constructed.

C. Attack Model

In our attack model, RUs, CP and CSP are *curious-but-honest* parties, which strictly follow the protocol, but 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 RU's ciphertext with the following capabilities:

1) $\mathcal{A}^*$ may *eavesdrop* on all communication links to obtain encrypted data.

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

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

4) $\mathcal{A}^*$ may *compromise* RUs, with the exception of the challenge RU, to get access to their decryption capabilities, and try to guess all plaintexts belonging to the challenge RU.

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

IV. Crypto Primitive and Basic Privacy Preserving Integer Calculation Protocols

In order to achieve the secure outsourced storage, we will firstly design a new Paillier based Cryptosystem. Then, the basic privacy-preserving integer calculation protocols will be designed as the basis of constructing secure floating point numbers calculation in Section V.

A. Paillier Cryptosystem with Partial Decryption (PCPD)

In order to realize POCF, the Paillier-based cryptosystem [20] appears to be a suitable solution for our system at first glance. However, the RU is not able to directly send the private key to the servers and the servers can use the private key to get the corresponding user's data squarely. Therefore, we adapt the Paillier-based cryptosystem to separate its private key into different shares in order to reduce the leaking risk, and we refer to this new system the Paillier Cryptosystem with

Partial Decryption (PCPD). PCPD consists of the following algorithms:

KeyGen: Let k be the security parameter and p, q be two large prime numbers, where $\|p\| = \|q\| = k$. We then compute $N = pq$ and $\lambda = (p-1)(q-1)/2$,[2] define a function $L(x) = \frac{x-1}{N}$, and choose a generator g of order N. Same to [16], we choose $g = 1 + N$, as $1 + N$ is just an element of order N. The public key is then $pk = N$, and the corresponding private key is $sk = \lambda$.

Encryption (Enc): Given a message $m \in \mathbb{Z}_N$, we choose a random number $r \in \mathbb{Z}_N^*$. The ciphertext is generated as

$$[m] = g^m \cdot r^N \mod N^2 = (1 + mN) \cdot r^N \mod N^2.$$

Decryption (Dec): To decrypt $[m]$ using the decryption algorithm $D_{sk}(\cdot)$ and the corresponding private key $sk = \lambda$, we need to compute

$$[m]^\lambda \mod N^2 = r^{\lambda N}(1+mN\lambda) \mod N^2 = (1+mN\lambda).$$

Since $gcd(\lambda, N) = 1$, m can be recovered as:

$$m = L([m]^\lambda \mod N^2)\lambda^{-1} \mod N.$$

Private Key Splitting (KeyS): The private key $sk = \lambda$ is split into two parts, denoted as $sk^{(1)} = \lambda_1$ and $sk^{(2)} = \lambda_2$, s.t., $\lambda_1 + \lambda_2 \equiv 0 \mod \lambda$ and $\lambda_1 + \lambda_2 \equiv 1 \mod N$ holding at the same time (see Section VI-A1).

Partial decryption Step 1 (PDec1): Once $[m]$ is received, the **PDec1** algorithm $PD_{sk^{(1)}}(\cdot)$ can be run as follows:

Using partial private key $sk^{(1)} = \lambda_1$, the partial decrypted ciphertext $CT^{(1)}$ is calculated as:

$$CT^{(1)} = [m]^{\lambda_1} = r^{\lambda_1 N}(1 + mN\lambda_1) \mod N^2.$$

Partial decryption Step 2 (PDec2): Once $CT^{(1)}$ and $[m]$ are received, the **PDec2** algorithm $PD_{sk^{(2)}}(CT^{(1)}; [m])$ can be run to recover the original message m.

$$CT^{(2)} = [m]^{\lambda_2} = r^{\lambda_2 N}(1 + mN\lambda_2) \mod N^2.$$

The algorithm computes $T'' = CT^{(1)} \cdot CT^{(2)} \mod N^2$, and then calculates $m = L(T'')$.

Ciphertext Refresh (CR): Once $[m]$ is received, the *CR* algorithm can refresh the ciphertext without changing the original message m, by randomly choosing $r' \in \mathbb{Z}_N^*$ and calculating

$$[m]' = [m] \cdot r'^N = (r \cdot r')^N \cdot (1 + mN) \mod N^2.$$

Given $m_1, m_2 \in \mathbb{Z}_N$, our PCPD has the following properties

$$\begin{aligned}[m_1] \cdot [m_2] &= (1 + (m_1 + m_2) \cdot N) \cdot (r_1 r_2)^N \mod N^2 \\ &= [m_1 + m_2].\end{aligned}$$

$$\begin{aligned}[m_1]^{N-1} &= (1 + (N-1)m_1 \cdot N)) \cdot r^{N-1 \cdot N} \mod N^2 \\ &= [-m_1].\end{aligned}$$

[2]In the original Paillier cryptosystem, the decryption key λ is chosen as $\text{lcm}(p-1, q-1)$. We choose $\lambda = (p-1)(q-1)/2$ as we need to sample the partial decryption without the knowledge of λ in the security proof of PCPD scheme (see Theorem 1).

The KGC generates the private key sk and public key pk for a RU, and randomly divides sk into two partial private keys $sk^{(1)}$ and $sk^{(2)}$ before sending $sk^{(1)}$ and pk to the CP, and $sk^{(2)}$ and pk to the CSP. The RU can encrypt data using pk and outsource the ciphertexts to the CP for storage.

After introducing the underlying algorithms in PCPD, we will now present the secure sub-protocols for processing integers. Before constructing, we assume that both CP and CSP will be involved in the following integer processing sub-protocols. Unless otherwise specified, the input integers x, y involved in the following sub-protocols can be positive, zero and negative[3], and restricted in the range of $[-R_1, R_1]$, where $\|R_1\| < \|N\|/4-1$. If we need a larger plaintext range, we can simply use the larger N. A larger N implies a broader plaintext range, and therefore, a higher level of security. However, this will affect the efficiency of the PCPD (See Fig. 3(a)).

B. Revised Secure Multiplication Protocol (RSM)

As our PCPD can only support additive homomorphism, we are unable to achieve multiplication between two plaintexts. In order to allow plaintext multiplication, we revise the original Secure Multiplication (**SM**) protocol [21], and present the revised protocol, Revised Secure Multiplication Protocol (**RSM**). When the CP is given two encrypted data $[x]$ and $[y]$ as input, the **RSM** will securely compute $[x \cdot y]$ as follows:

Step-1(@CP): The CP selects two random numbers $r_x, r_y \in \mathbb{Z}_N^*$, calculates $X = [x] \cdot [r_x]$, $Y = [y] \cdot [r_y]$, $X_1 = PD_{sk^{(1)}}(X)$, and $Y_1 = PD_{sk^{(1)}}(Y)$, sends X,Y, X_1 and Y_1 to the CSP.

Step-2(@CSP): The CSP calculates

$$h = PD_{sk^{(2)}}(X; X_1) \cdot PD_{sk^{(2)}}(Y; Y_1)$$

by using the other partially private key $sk^{(2)}$. Then, the CSP encrypts h using pk, denoted them as $H = [h]$, and sends H to the CP. It can be shown easily that $h = (x + r_x)(y + r_y)$.

Step-3(@CP): Once h' is received, the CP first computes $S_1 = [r_x \cdot r_y]^{N-1}$, $S_2 = [x]^{N-r_y}$ and $S_3 = [y]^{N-r_x}$. Then, the CP uses the following formula:

$$H \cdot S_1 \cdot S_2 \cdot S_3 = [h - r_y \cdot x - r_x \cdot y - r_x \cdot r_y] = [x \cdot y].$$

Therefore, both CP and CSP can jointly compute $[x \cdot y]$.

C. Secure Exclusive OR calculation Protocol (SXOR)

Given two encrypted data $[x]$ and $[y]$ ($x, y \in \{0, 1\}$), the goal of **SXOR** protocol is to get the encrypted bit XOR result $[f]$. The overall steps of **SXOR** protocol are jointly calculated as follows:

$$[f_1^*] \leftarrow \texttt{RSM}([1] \cdot ([x])^{N-1}; [y]); [f_2^*] \leftarrow \texttt{RSM}([x]; [1] \cdot ([y])^{N-1}).$$

$$[f] = [u_1 \oplus u_2] = [f_1^*] \cdot [f_2^*].$$

Notice that if $f = 0$, then $x = y$. Otherwise, $x \neq y$.

[3]In PCPD, a number with interval $(0, N/2]$ represents as positive number, while a number with interval $(N/2, N)$ represents as negative number.

D. Secure Less Than Protocol (SLT)

Given two encrypted numbers $[x]$ and $[y]$, the goal of **SLT** protocol is to get the encrypted data $[u^*]$ to show the relationship between the plaintext of the two encrypted data, i.e., $x \geq y$ or $x < y$. The overall steps of **SLT** protocol are shown as follows:

Step-1: (1) CP and CSP jointly calculates

$$[x_1] = ([x])^2 \cdot [1] = [2x+1]; [y_1] = ([y])^2 = [2y].^4$$

(2) CP flips a coin s and chooses a random number r ($r \neq 0$), s.t. $\|r\| < \|N\|/4$, if $s = 1$, calculates

$$[l] = ([x_1] \cdot ([y_1])^{N-1})^r = [r(x_1 - y_1)].$$

If $s = 0$, calculates

$$[l] = ([y_1] \cdot ([x_1])^{N-1})^r = [r(y_1 - x_1)].$$

(3) CP uses $sk^{(1)}$ to calculate $K = PD_{sk^{(1)}}([l])$, and sends the result to CSP.

Step-2(@CSP): Use $sk^{(2)}$ to decrypt K, and gets l.

If $\|l\| > \|N\|/2$, CSP denotes $u' = 1$ and denotes $u' = 0$ otherwise. Then CSP uses pk to encrypt u', and sends $[u']$ back to CP.

Step-3(@CP): Once $[u']$ is received, CP computes as follows: (1) if $s = 1$, CP denotes $[U] = \mathbf{CR}([u'])$. If $s = 0$, CP computes $[U] = [1] \cdot ([u'])^{N-1} = [1 - u']$.

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

E. Secure Equivalent Testing Protocol (SEQ)

Given two encrypted data $[x]$ and $[y]$, the goal of **SEQ** protocol is to get the encrypted result $[f]$ to show whether the plaintexts of the two encrypted data are equivalent, i.e., $x = y$. The overall steps of **SEQ** protocol are shown as follows:

$$[u_1] \leftarrow \mathbf{SLT}([x],[y]); [u_2] \leftarrow \mathbf{SLT}([y],[x]);$$
$$[f] = [u_1 \oplus u_2] = \mathbf{SXOR}([u_1]; [u_2]).$$

Notice that if $f = 0$, then $x = y$. Otherwise, $x \neq y$.

F. Secure Exponent Calculation (SEXP)

Given a public number x and an encrypted number $[y]$ ($x > 0, y \geq 0$), the **SEXP** protocol will provide the encrypted data $[U]$, s.t., $U = x^y$.[5] The **SEXP** is described as follows:

Step-1(@CP): (1) The CP chooses a random number $r \in \mathbb{Z}_N^*$ and computes

$$[y_1] = [y] \cdot [r] = [y + r];\ S = (x^r)^{-1} \mod N.$$

(2) Since the CP knows $sk^{(1)}$, the CP can compute $Y = PD_{sk^{(1)}}([y_1])$, prior to sending Y and $[y_1]$ to the CSP.

Step-2(@CSP): The CSP will decrypt Y using $sk^{(2)}$, and obtain y_1. Then, the CSP calculates $h = x^{y_1}$, encrypts h using pk, denoted them as $H = [h]$, and sends H to the CP. It can be easily found that $h = x^{y+r} \mod N$.

Step-3(@CP): Once $[h]$ is received, the CP gains $[x^y]$ as follows: $[U] = [h]^S = [x^{y+r} \cdot (x^r)^{-1}] = [x^y]$.

G. Secure Inverse Calculation (SINV)

Given an encrypted number $[x]$ ($x \neq 0$), the **SEXP** protocol outputs an encrypted data $[U]$ such that $U = x^{-1} \mod N$. The **SINV** is described as follows:

Step-1(@CP): (1) The CP chooses a random number $r \in \mathbb{Z}_N^*$ and computes $[x_1] = [x]^r$.

(2) Since the CP knows $sk^{(1)}$, the CP can compute $X = PD_{sk^{(1)}}([x_1])$, prior to send Y and $[x_1]$ to the CSP.

Step-2(@CSP): The CSP will decrypt Y using $sk^{(2)}$, and obtain x_1. Then, the CSP calculates $h = (x_1)^{-1} \mod N$, encrypts h using pk, denoted them as $H = [h]$, and sends H to the CP.

Step-3(@CP): Once $[h]$ is received, the CP gains $[x^y]$ as follows $[U] = [h]^r = [(x \cdot r)^{-1} \cdot (r)] = [x^{-1} \mod N]$.

H. Secure Modular Calculation (SMOD)

Given a public number p ($p \ll N$) and an encrypted number $[x]$, the goal of **SMOD** protocol is to calculate the encrypted data $[x \mod p]$. The **SMOD** is described as follows:

Step-1(@CP): (1) The CP chooses a random number $r \in \mathbb{Z}_N^*$ and computes $[x_1] = [x] \cdot [r] = [x + r]$;

(2) the CP can compute $X = PD_{sk^{(1)}}([x_1])$, prior to sending X and $[x_1]$ to the CSP.

Step-2(@CSP): The CSP will decrypt X using $sk^{(2)}$, and obtain x_1. Then, the CSP calculates $h = x_1 \mod p$, encrypts h using pk, denoted as $H = [h]$, and sends H to the CP.

Step-3 : Once $[h]$ is received, 1) the CP calculates

$$[U] = [h] \cdot [r \mod p]^{N-1} = [(h \mod p) - (r \mod p)].^6$$

2) CP and CSP jointly calculate $[k_u] \leftarrow \mathbf{SLT}([U]; [0])$;

3) CP calculates $[T] = [U] \cdot [k_u]^p = [U + p \cdot k_u]$.

Next, we will use the above protocols to achieve secure outsourced FPN processing.

V. Privacy Preserving Storage and Calculation on Floating Point Numbers

If a RU wants to outsource the Floating Point Numbers (FPNs) to the CP, two technical challenges need to be solved: 1) secure outsourced FPN storage, 2) secure calculation over outsourced FPNs.

A. Storing Encrypted Floating Point Numbers

In our system, the same precision η and β are used in all the data. Also, a FPN $x = (-1)^s \cdot (m_1 m_2 \cdots m_\eta) \cdot \beta^{e-\eta+1}$ can be represented as a triple $(s, m, e - \eta + 1)$, where $1 \leq m_1 < \beta, 0 \leq m_i < \beta$. For example, with $\eta = 3$ and $\beta = 10$, a number -17 can be denoted as $(-1)^1 \cdot 170 \cdot 10^{-1}$ and stored as $(1, 170, -1)$, while 1 can be denoted as $(-1)^0 \cdot 100 \cdot 10^{-2}$

[4] The orignial data x and y needs to be transformed into x_1 and x_2, in order to avoid showing the equivalent relationship to CSP. For example, if $x = y = 6$, we will obtain $x_1 = 13$ and $y_1 = 12$ after the transformation, s.t., $x_1 \neq y_1$.

[5] If we want to calculate $U = x^y$, we must make a restriction, s.t. y is relative small number, i.e., $x^y < N$. If we want to calculate $x^y \mod N$, there is no above restriction.

[6] As $(h - r) \mod p = ((h \mod p) - (r \mod p)) \mod p$ and $-p < U < p$, we determine whether U is greater than 0 or not. If $-p < U < 0$, we will calculate the final result as $U + p$, otherwise, we will calculate $U + 0$.

and stored as $(0, 100, -2)$. In order to securely store the FPN, our POCF will encrypt the FPN as triplet $([s], [m], [t])$, where $s \in \{0, 1\}$, m is η-digital number, $(e_{min} - \eta + 1) \leq t \leq (e_{max} - \eta + 1)$. Moreover, we define some special characters: positive infinity $(+\infty)$, negative infinity $(-\infty)$, and not a number (NaN) in TABLE I.

TABLE I
DEFINITION OF SPECIAL CHARACTERS

Datum	Sign s	Significand m	Exponent e
$+0$	0	0	0
-0	1	0	0
$+\infty$	0	0	$e_{max}+1$
$-\infty$	1	0	$e_{max}+1$
NaN	0	nonzero	$e_{max}+1$

The encrypted data are construct as $\langle x \rangle = ([s_x], [m_x], [t_x])$. We first define two kinds of FPN operations "+" and "×":

$$\begin{aligned}\langle x \rangle + \langle y \rangle &= ([s_x] \cdot [s_y], [m_x] \cdot [m_y], [t_x] \cdot [t_y]) \\ &= ([s_x + s_y], [m_x + m_y], [t_x + t_y]),\end{aligned}$$

$$[A] \times \langle x \rangle = ([s_z], [m_z], [t_z]),$$

where $[s_z] \leftarrow \texttt{RSM}([A], [s_x])$, $[m_z] \leftarrow \texttt{RSM}([A], [m_x])$, $[t_z] \leftarrow \texttt{RSM}([A], [t_x])$. Notice that the FPN operation "×" has a higher computation priority than the FPN operation "+".

B. Secure Outsourced Floating Point Number Computation

Here, we introduce five protocols to achieve five different secure floating point numbers computation in the outsourced environment.

1) Secure Floating Point Numbers Equivalent Protocol (FPEQ): Given two encrypted FPNs $\langle x \rangle = ([s_x], [m_x], [t_x])$ and $\langle y \rangle = ([s_y], [m_y], [t_y])$, the goal of **FPEQ** protocol is to calculate $[f]$, s.t., $f = 0$ when $x = y$, and $f = 1$ otherwise. The idea is easy to follow: we only need to test $s_x \stackrel{?}{=} s_y$, $m_x \stackrel{?}{=} m_y$, and $t_x \stackrel{?}{=} t_y$. The overall steps of **FPEQ** protocol are shown as follows:

$$[u_1] \leftarrow \texttt{SXOR}([s_x]; [s_y]);\ [u_2] \leftarrow \texttt{SEQ}([m_x]; [m_y]);$$

$$[u_3] \leftarrow \texttt{SEQ}([t_x], [t_y]); [u_4] \leftarrow \texttt{RSM}([1][u_1]^{N-1}; [1][u_2]^{N-1});$$

$$[u_5] \leftarrow \texttt{RSM}([1] \cdot [u_3]^{N-1}; [u_4]);\ [f] = [1] \cdot [u_5]^{N-1}.$$

2) Secure Floating Point Numbers Sorting with Absolute Value (SFPS): Given two encrypted FPNs $\langle x \rangle$ and $\langle y \rangle$, the goal of **SFPS** protocol is to generate $\langle A \rangle$ and $\langle I \rangle$, s.t., $A = max(|x|, |y|)$ and $I = min(|x|, |y|)$. The overall steps of **SFPS** protocol are shown as follows:

Step-1: use **FPEQ** and **RSM** protocols

$$[X] \leftarrow \texttt{FPEQ}(\langle x \rangle ; \langle \text{NaN} \rangle); [Y] \leftarrow \texttt{FPEQ}(\langle y \rangle ; \langle \text{NaN} \rangle);$$

$$[N_0] \leftarrow \texttt{RSM}([X]; [Y]);$$

to judge $\langle x \rangle$ or $\langle y \rangle$ is encrypted NaN or not. If x or y is NaN, the protocol will use $\langle \text{NaN} \rangle$ to replace $\langle x \rangle$ and $\langle y \rangle$. Otherwise, the value of x and y are not changed. It can be achieved by constructing $\langle x^* \rangle = [N_0] \times \langle x \rangle + [1 - N_0] \times \langle \text{NaN} \rangle$ and $\langle y^* \rangle = [N_0] \times \langle y \rangle + [1 - N_0] \times \langle \text{NaN} \rangle$.

Step-2: We execute $[S_t] \leftarrow \texttt{SEQ}([t_x], [t_y])$ to test whether the exponents of x^* and y^* are equal. If $t_x \neq t_y$ $(S_t = 1)$, we choose the number with bigger exponent as the larger FPN number. The construction is easy: we first calculate $[L_t] \leftarrow \texttt{SLT}([t_x], [t_y])$; Then, we construct the bigger encrypted value as $[S_t] \times ([1 - L_t] \times \langle x^* \rangle + [L_t] \times \langle y^* \rangle)$. The smaller one is constructed as $[S_t] \times ([L_t] \times \langle x^* \rangle + [1 - L_t] \times \langle y^* \rangle)$. If exponents of x and y are equal, i.e., $t_x = t_y$, we choose the number with bigger significand as the larger FPN number. The construction is as follows: we first calculate $[L_m] \leftarrow \texttt{SLT}([m_x], [m_y])$; Then, we construct the bigger one as $[1 - S_t] \times ([1 - L_m] \times \langle x^* \rangle + [L_m] \times \langle y^* \rangle)$. The smaller one is constructed as $[1 - S_t] \times ([L_m] \times \langle x^* \rangle + [1 - L_m] \times \langle y^* \rangle)$.

After these steps, the **SFPS** will output the maximum encrypted FPN value $\langle A \rangle = [S_t] \times ([1 - L_t] \times \langle x^* \rangle + [L_t] \times \langle y^* \rangle) + ([1 - S_t] \times ([1 - L_m] \times \langle x^* \rangle + [L_m] \times \langle y^* \rangle))$ and the minimum encrypted FPN value $\langle I \rangle = [S_t] \times ([L_t] \times \langle x^* \rangle + [1 - L_t] \times \langle y^* \rangle) + [1 - S_t] \times ([L_m] \times \langle x^* \rangle + [1 - L_m] \times \langle y^* \rangle)$.

3) Secure Floating Point Numbers Addition (SFPA): Given two encrypted FPNs $\langle x \rangle$ and $\langle y \rangle$, the goal of **SFPA** protocol is to calculate the result $\langle f^* \rangle$, s.t., $f^* = x + y$.[7] The overall steps of **SFPA** protocol are shown as follows:

Step-1: We first need to handle the special cases, if $\langle x \rangle$ or $\langle y \rangle$ is equal to $\langle \text{NaN} \rangle$, the final result is $\langle \text{NaN} \rangle$. Also, if one is $+\infty$ and the other is $-\infty$, the result is still $\langle \text{NaN} \rangle$. This can be achieved by following calculations:

$$[X_1] \leftarrow \texttt{FPEQ}(\langle x \rangle ; \langle \text{NaN} \rangle);\ [Y_1] \leftarrow \texttt{FPEQ}(\langle y \rangle ; \langle \text{NaN} \rangle);$$

$$[X_2] \leftarrow \texttt{FPEQ}(\langle x \rangle ; \langle +\infty \rangle);\ [Y_2] \leftarrow \texttt{FPEQ}(\langle y \rangle ; \langle +\infty \rangle);$$

$$[X_3] \leftarrow \texttt{FPEQ}(\langle x \rangle ; \langle -\infty \rangle);\ [Y_3] \leftarrow \texttt{FPEQ}(\langle y \rangle ; \langle -\infty \rangle);$$

$$[U_1] \leftarrow \texttt{RSM}([1] \cdot [X_2]^{N-1};\ [1] \cdot [Y_3]^{N-1}); \quad (1)$$

$$[U_2] \leftarrow \texttt{RSM}([1] \cdot [Y_2]^{N-1};\ [1] \cdot [X_3]^{N-1}); \quad (2)$$

$[u_1^*] = [1] \cdot [X_1]^{N-1} [1] \cdot [Y_1]^{N-1} \cdot [U_1] \cdot [U_2] = [(1 - X_1) + (1 - Y_1) + (1 - X_2)(1 - Y_3) + (1 - X_3)(1 - Y_2)]$.

Then, we execute $[u^*] \leftarrow \texttt{SLT}([u_1^*]; [1])$ and construct $\langle x^* \rangle = [u^*] \times \langle x \rangle + [1 - u^*] \times \langle \text{NaN} \rangle$ and $\langle y^* \rangle = [u^*] \times \langle y \rangle + [1 - u^*] \times \langle \text{NaN} \rangle$.

Step-2: We execute **SFPS** to sort x^* and y^*, i.e., $(\langle A \rangle ; \langle I \rangle) \leftarrow \texttt{SFPS}(\langle x^* \rangle ; \langle y^* \rangle)$. Then compute:

$$[t_1] = [t_A] \cdot [t_I]^{N-1} = [t_A - t_I];\ [B_1] \leftarrow \texttt{SEXP}(10;\ [t_1]);$$

$$[t_2] = [t_A] \cdot [\eta] \cdot [t_I]^{N-1} = [t_A - t_I + \eta];$$

$$[B_2] \leftarrow \texttt{SEXP}(10;\ [t_2]);$$

$$[K_1] = [m_A \cdot 10^{t_A - t_I}] \leftarrow \texttt{RSM}([m_A];\ [B_1]).$$

Furthermore, we need to execute

$$[s_v] \leftarrow \texttt{SXOR}([s_A];\ [s_I]);\ [S_V] \leftarrow \texttt{SEXP}(-1;\ [s_v]);$$

$$[S_V \cdot m_I] \leftarrow \texttt{RSM}([S_V];\ [m_I]), \text{ and}$$

$$[E_1] = [m_A \cdot 10^{t_A - t_I} + (-1)^{s_v} \cdot m_I] = [K_1][S_V \cdot m_I]. \quad (3)$$

Step-3: Initial $[u_{\eta-1}] = [1]$. Next, we judge how many digits are the result significand, i.e., the length of E_1. The construction are is follows:

[7] Note that $\langle x + y \rangle \neq \langle x \rangle + \langle y \rangle$ (defined in Section V-A).

for($j = \eta$ to $2\eta - 1$) {calculate $[E_j^*] \leftarrow$ **SMOD**$([E_1]; 10^j)$;

$$[T_j^*] = [E_1] \cdot [E_j^*]^{N-1};\ [u_j] \leftarrow \textbf{SEQ}([T^*]; [0]).$$

$$[w_{j,j-1}] \leftarrow \textbf{SXOR}([u_j]; [u_{j-1}]);\ [u'_j] \leftarrow \textbf{RSM}([w_{j,j-1}]; [j]);\}$$

Finally, calculate $[U] = \sum_{i=\eta}^{2\eta-1} [u'_j]$.[8]

Step-4: Thereafter, we need to construct the result significand with η digitals as follows: 1) execute

$$[t_3] = [U] \cdot [\eta]^{N-1};\ [B_3] \leftarrow \textbf{SEXP}(10;\ [t_3]);\ [M_f] = [0];$$

2) **for**($j = 0$ to $\eta - 1$) {calculate $[B'_j] \leftarrow$ **SEQ**$([j];\ [t_3])$;

$$[E_3] \leftarrow \textbf{SMOD}([E_1]; 10^j);\ [M'_j] = [E_1] \cdot [E_3]^{N-1};$$

$$[u'_j] \leftarrow \textbf{RSM}([1] \cdot [B'_j]^{N-1}; [M'_j]); [M_f] = [M_f] \cdot [u'_j].\}$$

3) CP calculate $[t_f] = [t_A] \cdot [U];\ [s_f^*] = [s_A]$.

$$[B_4] \leftarrow \textbf{SINV}([B_3]);\ [m_f] \leftarrow \textbf{RSM}([M_f];\ [B_4]);$$

Step-5: Finally, we determine whether the result reaches $\pm\infty$ or not. If the answer is yes, we refresh the result into $\pm\infty$, otherwise, we use the calculated result. The construction is as follows:

$$[u_f] \leftarrow \textbf{SLT}([t_f]; [\eta_{max}]);\ [m_f^*] \leftarrow \textbf{RSM}([u_f]; [m_f]);$$

$$[t_1^*] \leftarrow \textbf{RSM}([u_f]; [t_f]);\ [t_2^*] \leftarrow \textbf{RSM}([1] \cdot [u_f]^{N-1}; [\eta_{max}]);$$

$$[t_f^*] = [t_1^*] \cdot [t_2^*] = [u^f \cdot t_f + (1 - u_f) \cdot \eta_{max}].$$

Step-6: We need to execute $[u_t] \leftarrow$ **SLT**$([t_1]; [\eta])$. If $t_A - T_I \geq \eta$, we directly choose $\langle A \rangle$ as the final result. Otherwise, we will choose $\langle f^* \rangle$ as the result. The FPN addition result is $\langle f' \rangle = ([1 - u_t] \times \langle A \rangle + [u_t] \times \langle f^* \rangle)$, where $\langle f^* \rangle = ([s_f^*], [m_f^*], [t_f^*])$. □

For secure FPN subtraction, we first need to change formula (1)-(3) into (4)-(6), respectively,

$$[U_1] \leftarrow \textbf{RSM}([1] \cdot [X_2]^{N-1};\ [1] \cdot [Y_2]^{N-1}); \tag{4}$$

$$[U_2] \leftarrow \textbf{RSM}([1] \cdot [X_3]^{N-1};\ [1] \cdot [Y_3]^{N-1}); \tag{5}$$

$$[E_1] = [m_A \cdot 10^{t_A - t_I} - (-1)^{s_v} \cdot m_I] = [K_1][S_V \cdot m_I]. \tag{6}$$

Moreover, to handle the special case that minuend is equal to 0, the formula $[s_f^*] = [s_A]$ in **Step-4-3** of **SFPA** is changed into

$$[Z] \leftarrow \textbf{FPEQ}(\langle x \rangle; \langle 0 \rangle);\ [s_A \cdot Z] \leftarrow \textbf{RSM}([s_A]; [Z]);$$

$$[s_f^*] = [1] \cdot [Z]^{N-1} \cdot [s_A \cdot Z] = [1 - Z + s_A \cdot Z].$$

4) Secure Floating Point Numbers Multiplication (SFPM): Given two encrypted FPNs $\langle x \rangle$ and $\langle y \rangle$, the goal of **SFPM** protocol is to calculate the result $\langle f^* \rangle$, s.t., $f^* = x \cdot y$.[9] The overall steps of **SFPM** protocol are shown as follows:

Step-1: it is similar to that of the **Step-1** in **SFPA**. The different is that we need to handle the special case that one FPN is 0 and the other FPN is ∞ (include both $+\infty$ and $-\infty$). After this step, $\langle x^* \rangle$ and $\langle y^* \rangle$ are generated.

[8]Step-3 can be parallel executed which only takes one round communication.

[9]Note that $\langle x \cdot y \rangle \neq \langle x \rangle \times \langle y \rangle$ (defined in Section V-A).

Step-2: Here, we calculate sign, significand and exponents as follows:

$[P_1] \leftarrow \textbf{RSM}([m_x];\ [m_y]);\ [s_f^*] \leftarrow \textbf{SXOR}([s_x];\ [s_y])$.

Step-3: we judge the P_1 is $2\eta - 1$ or 2η digits.

$$[E^*] \leftarrow \textbf{SMOD}([P_1]; 10^{2\eta-1});\ [T^*] = [P_1] \cdot [E^*]^{N-1};$$

$$[U] \leftarrow \textbf{SEQ}([T^*]; [0]).$$

If $U = 1$, it shows that P_1 is 2η digits. Otherwise, it is $2\eta - 1$ digits.

Step-4: Thereafter, we need to construct the result significand with η digitals as follows: 1) execute

$$[t_3] = [U] \cdot [\eta - 1];\ [B_3] \leftarrow \textbf{SEXP}(10;\ [t_3]);\ [M_f] = [0];$$

2) **for**($j = \eta - 1$ to η) {calculate $[B'_j] \leftarrow$ **SEQ**$([j];\ [t_3])$;

$$[E_3] \leftarrow \textbf{SMOD}([E_1]; 10^j);\ [M'_j] = [P_1] \cdot [E_3]^{N-1};$$

$$[u'_j] \leftarrow \textbf{RSM}([1] \cdot [B'_j]^{N-1}; [M'_j]); [M_f] = [M_f] \cdot [u'_j].\}$$

3) CP calculate $[t_f] = [t_x] \cdot [t_y] \cdot [\eta] \cdot [U];\ [s_f^*] = [s_A]$.

$$[B_4] \leftarrow \textbf{SINV}([B_3]);\ [m_f] \leftarrow \textbf{RSM}([M_f];\ [B_4]);$$

Step-5: Finally, we determine whether the result reach $\pm\infty$ or not. This step is same to that of **Step-5** in **SFPA**.

5) Secure Floating Point Numbers Comparison (SFPC): Given two encrypted FPNs $\langle x \rangle$ and $\langle y \rangle$, the goal of **SFPC** protocol is to calculate $[f]$, s.t., f reflects the relationship between x and y. The overall steps of **SFPC** protocol are shown as follows:

Step-1: We execute **SFPS** to sort x and y

$$(\langle A \rangle; \langle I \rangle) \leftarrow \textbf{SFPS}(\langle x \rangle; \langle y \rangle).$$

Step-2: Here, we test whether x or y is equal to NaN or not. If not, we test the sign symbols of x and y are different or not. If the sign symbols of x and y are different, the FPN with positive sign is the larger one. If the sign symbol of x and y are the same, the one with larger absolute value is the larger one if the sign of x and y are positive. Otherwise, the one with larger absolute value is the small one if both sign of x and y are negative. The construction is as follows:

$$[U] \leftarrow \textbf{FPEQ}(\langle A \rangle; \langle \text{NaN} \rangle);\ [U_1] \leftarrow \textbf{FPEQ}(\langle x \rangle; \langle A \rangle);$$

$$[s_v] \leftarrow \textbf{SXOR}([s_x];\ [s_y]).\ [P_1] \leftarrow \textbf{RSM}([s_x];\ [U_1]);$$

$$[P_2] \leftarrow \textbf{RSM}([1] \cdot [U_1]^{N-1}; [1] \cdot [s_x]^{N-1}) = [(1 - U_1)(1 - s_x)];$$

$$[E_1] \leftarrow \textbf{RSM}([P_1] \cdot [P_2]; [1] \cdot [s_v]^{N-1});$$

$$[E_2] \leftarrow \textbf{RSM}([s_v]; [1] \cdot [s_x]^{N-1});$$

$$[U_x] = [E_1] \cdot [E_2] = [((1 - U_1)(1 - s_x) + U_1 s_x)(1 - s_x \oplus s_y) + (s_x \oplus s_y)(1 - s_x)];$$

$$[K_1] \leftarrow \textbf{RSM}([1] \cdot [U_x]^{N-1}; [1]^{N-1});\ [K_2] = [K_1] \cdot [U_x];$$

$$[f] \leftarrow \textbf{RSM}([U]; [K_2]) = [U \cdot (U_x + (1 - U_x) \cdot (-1))];$$

Notice that if $f = 0$, it shows that x or y is equal to NaN. If $f = 1$, it shows that $x \geq y$. If $f = -1$, it shows that $x < y$.

Remark: The format of PCPD ciphertext is used as the output of **SFPC** and **FPEQ** protocol, i.e., output as $[f]$. If the floating pointing ciphertext format is needed, the output of **SFPC** and **FPEQ** protocol can be stored as $\langle f \rangle = ([0], [f], [0])$.

C. The Overview of POCF

Here, we show how to achieve data outsourcing and computation in the real environment with POCF.

1. *Secure Data Outsourcing Phase:* When some data are needed to be outsourced, the RU should first decide whether the data are numerical or not. If the RU wants to store numeric message, he only need to transform the number into floating pointing format, and encrypt the FPN by using the technique introduced in section V-A. If the RU need to store the text message, we need to encode each character into 8-bit or 16-bit integer by using Unicode [22]. As calculation over the text messages is not necessary, we can store a group of text message at same time. For example, if N is 1024-bit length, one ciphertext can handle at most 128 or 64 characters. The RU can encrypt 384 or 192 characters at the same time by adopting floating point format $\langle x \rangle$ (the sign, significand, and exponent can be used for storage). After the encryption, all the ciphertext will be sent to the CP for storage.

2. *Secure Data Processing Phase:* Once the outsourced calculation is needed, the RU will send a query to the CP. Then, the CP and CSP will jointly process the data according the RU's request. As all the data are stored in an encrypted form, secure outsourced FPN computation technique in Section V-B should be used[10]. After calculation, the computed results are also stored as encrypted form in CP. Here, we give an example to show how to use the above secure FPN operations to construct the real-world applications (Single-layer Neural Network, a.k.a., SNN). Suppose two encrypted FPN vectors $\langle \mathbf{x} \rangle = (\langle x_1 \rangle, \cdots, \langle x_n \rangle)$ and $\langle \mathbf{w} \rangle = (\langle w_1 \rangle, \cdots, \langle w_n \rangle)$, and an encrypted FPN element $\langle b \rangle$ are stored in CP, where $x_1 \cdots, x_n$ are called the input value, $w_1, \cdots, w_n$ are called the weight values, and b is called bias. The goal is to get the result $\langle c' \rangle$, where $c' = c$ (c is 0 or 1, without loss of generality, we denote $c = 1$) if $\mathbf{w} \cdot \mathbf{x} \geq b$. Otherwise, denote $c' = \overline{c}$ ($\overline{c}$ is the bitwise NOT of c, i.e., $c' = 0$). The overall steps can be listed as follows:

1) Initial $\langle A \rangle = ([0], [0], [0])$.

2) For $i = 1, \cdots, n$, compute $\langle A_i \rangle \leftarrow \mathtt{SFPS}(\langle x_i \rangle; \langle w_i \rangle)$. Then, calculate $\langle A \rangle \leftarrow \mathtt{SFPA}(\langle A \rangle; \langle A_i \rangle)$.

3) Calculate $[B] \leftarrow \mathtt{SFPC}(\langle A \rangle; \langle b \rangle)$. Finally, it calculate $\langle c' \rangle = [B] \times \left\langle \frac{1}{2} \right\rangle + \left\langle \frac{1}{2} \right\rangle$. Note that the above FPN operations '+' and '×' are defined in Section V-A.

3. *Secure Data Retrieval Phase:* Suppose n encrypted data $\langle x_1 \rangle, \cdots, \langle x_n \rangle$ are stored in the CP. If the k-th floating point ciphertext is needed, the RU can directly send the position information to CP for data retrieval. However, the RU's query information (which ciphertext is needed by RU) will be exposed to CP. Instead, we adopt a different technique to achieve secure floating point ciphertext retrieval: The RU first sends the encrypted query $[a_1], \cdots, [a_n]$ to the cloud, where $a_k = 1$ and $a_i = 0$ $(i = 1, \cdots, n; i \neq k)$, i.e., $\langle x_k \rangle$ is the target ciphertext. Then, the CP will calculate as follows:

1) for $i = 1, \cdots, n$, calculate $\langle A_i \rangle \leftarrow [a_i] \times \langle x_i \rangle$;

2) compute $\langle A \rangle \leftarrow \langle A_1 \rangle + \cdots + \langle A_n \rangle$.

After that, $\langle A \rangle$ is sent back to RU for decryption.

The Necessity of CSP: As our PCPD is additive homomorphic encryption scheme, addition and multiplication homomorphic operations over encrypted data cannot be manipulated in a single server at the same time (different from fully homomorphic encryption scheme). Unfortunately, existing fully homomorphic cryptosystem is rather inefficient, in term of computation and storage overhead [25]. In the near future, we can remove CSP from the system which will also result in a more elegant system if an efficient fully homomorphic cryptosystem exists.

VI. Security Analysis

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

A. Analysis of PCPD

1) The existence of private key splitting: We randomly split the 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 $\lambda_1 + \lambda_2 \equiv 1 \mod N$ hold. Since $gcd(\lambda, N) = 1$, there exists a unique $s \in [1, \lambda N]$, s.t. both $s \equiv 0 \mod \lambda$ and $s \equiv 1 \mod N$ hold According to the Chinese remainder theorem [26], the value of s can be represented as $s = \lambda \cdot (\lambda^{-1} \mod N)$. Thus, we only need to randomly choose $\lambda_1 \in [1, \lambda N]$ and set $\lambda_2 = s - \lambda_1 \mod \lambda N$.

2) The correctness of the partial decryption of PCPD: Once $CT^{(1)} = r^{\lambda_1 N}(1 + mN\lambda_1) \mod N^2$ is generated by PDec1, both $CT^{(1)}$ and $[m]$ are served as input of PDec2 algorithm. Then, $CT^{(2)} = r^{\lambda_2 N}(1 + mN\lambda_2) \mod N^2$ is computed, and calculates T'' as follows:

$$\begin{aligned} T'' &= CT^{(1)} \cdot CT^{(2)} \mod N^2 \\ &= r^{(\lambda_1+\lambda_2)N}(1 + mN(\lambda_1 + \lambda_2)) \mod N^2 \end{aligned} \quad (7)$$

Due to $\lambda_1 + \lambda_2 \equiv 0 \mod \lambda$, then $r^{(\lambda_1+\lambda_2)N} \mod N^2 = 1$. Because $\lambda_1 + \lambda_2 \equiv 1 \mod N$ (i.e., $\exists\ k$, s.t., $\lambda_1 + \lambda_2 = 1 + kN$), then we have $(1 + mN(\lambda_1 + \lambda_2)) \mod N^2 = (1 + mN(1+kN)) \mod N^2 = 1 + mN$. Finally, it calculates $m = L(T'') = L(1 + mN) = \frac{(1+mN)-1}{N} = m$, which guarantees the correctness.

3) Security of PCPD: In this section, we show that the **PCPD** scheme is semantically secure against semi-trusted CSP or CP even if it has one part of the decryption key of the **PCPD** scheme. Here, we naturally assume that the CSP and the CP can not collude with each other. Next, we define the semantic security model for a public-key encryption scheme that supports partial decryption, and then prove the security of the **PCPD** scheme in this model.

Definition 1 (Semantic Security). *Let $\mathscr{E} = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ be a public-key encryption scheme supporting partial decryption. We say that $\mathscr{E}$ is semantically secure if for any polynomial-time adversary $\mathcal{A}$, it has negligible advantage (in the security parameter) in the following experiment (between the challenger and the adversary):*

[10]Our POCF can be employed to protect the content of the data (including the input data and its final output). Protecting the computation procedure is not considered in the paper, i.e., the cloud server can still know which kinds of calculation is needed by the RU. If this type of information needs to be protected, we refer reader to [23], [24].

1) *The challenger runs* $\mathbf{Gen}(1^k)$ *to obtain a public key and secret key pair* (pk, sk)*, and then splits* sk *into two parts* (sk_1, sk_2)*. It sends the public key* pk *as well as one part of the secret key, e.g.,* sk_1 *to the adversary* $\mathcal{A}$*.*
2) *The adversary* $\mathcal{A}$ *chooses two equal-length messages* m_0 *and* m_1 *and sends them to the challenger.*
3) *The challenger chooses a random bit* $b \in \{0, 1\}$ *and sends the ciphertext* $c^* = \mathbf{Enc}(m_b)$ *to* $\mathcal{A}$*.*
4) *The adversary outputs a bit* b' *as a guess of* b*.*

The adversary's advantage in the above experiment is defined as $\mathsf{Adv}_{\mathscr{E}}(k) := |\Pr[b' = b] - 1/2|$.

Here, we briefly review the definition of Paillier's Decisional Composite Residuosity (DCR) assumption [20]. Let $N = pq$ be the product of two safe primes. Then the DCR assumption roughly says that the set of N-th powers modulo N^2 is computationally indistinguishable from the uniform distribution modulo N^2, i.e.,

Definition 2 (The Decisional Composite Residuosity (DCR) Assumption)**.** *The Decisional Composite Residuosity (DCR) assumption states that*

$$\{x^N \mod N^2 : x \in \mathbb{Z}^*_{N^2}\} \stackrel{c}{\approx} \{x : x \in \mathbb{Z}^*_{N^2}\},$$

where $\stackrel{c}{\approx}$ *denotes computational indistinguishability.*

The semantic security of Paillier cryptosystem is proved under the DCR assumption [20]. Next, we show that if the Paillier cryptosystem is semantically secure, then so does the **PCPD** scheme.

Theorem 1. *The* ***PCPD*** *scheme described in Section IV-A is semantically secure, assuming the semantic security of the underlying Paillier cryptosystem.*

Proof. Assume that there exists a PPT adversary $\mathcal{A}$ breaking the semantic security of the **PCPD** scheme with advantage at most ϵ, then we can construct an algorithm $\mathcal{S}$ to break the semantic security of the Paillier cryptosystem with advantage at least $\epsilon - \frac{1}{2^k}$, where k is the half bit-length of the modulus N. The algorithm $\mathcal{S}$ has almost the same time complexity with $\mathcal{A}$.

Given the challenge public key $pk = (N, g)$ of the Paillier cryptosystem, $\mathcal{S}$ first chooses a random element sk_1 from the interval $[1, N(N-1)/2]$ and then sends (pk, sk_1) to $\mathcal{A}$. When $\mathcal{S}$ receives two equal-length messages (m_0, m_1) from $\mathcal{A}$, it passes them on to the challenger of the Paillier cryptosystem. The challenger will randomly choose a message m_b and send the corresponding challenge ciphertext c^* to $\mathcal{S}$. $\mathcal{S}$ sends the same challenge ciphertext c^* to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b' to $\mathcal{S}$, which is just the guess of $\mathcal{S}$. With the exception of the partial decryption key sk_1, the public key and the challenge ciphertext have the same distributions as in the real semantic security experiment from the adversary $\mathcal{A}$'s point of view. As the real partial decryption key comes from $[1, \lambda N]$, it is easy to show that two variables X and Y randomly chosen from $[1, \lambda N]$ and $[1, N(N-1)/2]$ respectively have at most $\frac{1}{2^k}$ statistical distance. So, $\mathcal{S}$ breaks the semantic security of the Paillier cryptosystem is at least $\epsilon - \frac{1}{2^k}$. □

B. The Security of Sub-protocols

Here we recall the security model for securely realizing an ideal functionality in the presence of non-colluding semi-honest adversaries. For simplicity, we do it for the specific scenario of our functionality, which involve three parties, challenge RU (a.k.a. "D_R"), CP (a.k.a. "S_1") and CSP (a.k.a."S_2"). We need to construct three simulators Sim= $(\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2})$ to against three kinds of adversaries $(\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ that corrupt D_R, S_1 and S_2, respectively. We refer the reader to [27], [28] for the general case definitions.

Let $\mathcal{P} = (D_R, S_1, S_2)$ be the set of all protocol parties. We consider three kinds of adversaries $(\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ that corrupt D_R, S_1 and S_2, respectively. In the real world, D_R runs on input x and y (with additional auxiliary inputs z_x and z_y), 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, whereas 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 $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ 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 user sends x and y to $\mathbf{f}$. If any of x or y is $\perp$, then $\mathbf{f}$ returns $\perp$. Finally, $\mathbf{f}$ returns $f(x, y)$ to the challenge user. 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, whereas if P is corrupted, 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 Sim= $(\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2})$ 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 *partially emulates*, in the real world, an execution of $\mathbf{f}$ in the ideal world. More formally,

Definition 3. *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* Sim= $(\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2})$ *of PPT transformations (where* $Sim_{D_1} = Sim_{D_1}(\mathcal{A}_{D_1})$ *and so on) such that for all semi-honest PPT adversaries* $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$*, for all inputs* x*,* y *and auxiliary inputs* $\mathbf{z}$*, and for all parties* $P \in \mathcal{P}$ *it holds*

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

Theorem 2. *The* ***RSM*** *protocol described in Section IV-B can securely compute multiplication of plaintext on ciphertext in the presence of semi-honest (non-colluding) adversaries* $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.

Proof. We only provide a proof to show how to construct three independent simulators $\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2}$.

Sim_{D_R} receives x and y as input and then simulates $\mathcal{A}_{D_R}$ as follows: it generates encryption $[x] = \mathbf{Enc}(x)$ of x and encryption $[y] = \mathbf{Enc}(y)$ of y. Finally, it returns $[x]$ and $[y]$ to $\mathcal{A}_{D_R}$ and outputs $\mathcal{A}_{D_R}$'s entire view.

The view of $\mathcal{A}_{D_R}$ consists of the encrypted data. The views of $\mathcal{A}_{D_R}$ in the real and the ideal executions are indistinguishable due to the semantic security of PCPD.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: First, it generates (fictitious) encryptions of the inputs $[\hat{x}]$ and $[\hat{y}]$ by running $\mathbf{Enc}(\cdot)$ on randomly chosen $\hat{x}, \hat{y}$, randomly generates $r_i \in Z_N$, calculates $\hat{X}$ and $\hat{Y}$, and then calculates $\hat{X}_1$ and $\hat{Y}_1$ by using **PDec1**$(\cdot)$. Sim_{S_1} sends the encryption $\hat{X}, \hat{Y}, \hat{X}_1, \hat{Y}_1$ to $\mathcal{A}_{D_1}$. If $\mathcal{A}_{D_1}$ replies with $\perp$, then Sim_{S_1} returns $\perp$.

The view of $\mathcal{A}_{S_1}$ consists of the encrypted data it creates. In both the real and the ideal executions, he receives the output the encryptions $\hat{X}, \hat{Y}, \hat{X}_1, \hat{Y}_1$. In the real world, this is guaranteed by the fact that the RU is honest and the semantic security of PCPD. 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 choses h, uses the $\mathbf{Enc}(\cdot)$ to get $[h]$, and then sends the encryptions to $\mathcal{A}_{D_2}$. If $\mathcal{A}_{D_2}$ 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 the real and the ideal executions, he receives the output the encryptions $[h]$. In the real world, it is guaranteed by the semantic security of PCPD. The views of $\mathcal{A}_{S_1}$ in the real and the ideal executions are indistinguishable. □

The security proofs of **SEXP**,and **SINV** protocols are similar to that of **RSM** protocol under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. In the following section, we need to prove the security of **SXOR**.

Theorem 3. *The **SXOR** protocol described in Section IV-C is to securely evaluate the Exclusive OR operation of plaintext over ciphertext in the presence of semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.*

Proof. We now demonstrate how to construct three independent simulators $\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2}$.

Sim_{D_R} receives x and y as input and simulates $\mathcal{A}_{D_R}$ as follows: it generates encryption $[x] = \mathbf{Enc}(x)$ of x and encryption $[y] = \mathbf{Enc}(y)$ of y. Finally, it returns $[x]$ and $[y]$ to $\mathcal{A}_{D_R}$ and outputs $\mathcal{A}_{D_R}$'s entire view.

The view of $\mathcal{A}_{D_R}$ consists of the encrypted data. The views of $\mathcal{A}_{D_R}$ in the real and the ideal executions are indistinguishable due to the semantic security of PCPD.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: First, it generates (fictitious) encryptions $[\hat{x}]$ and $[\hat{y}]$ by running $\mathbf{Enc}(\cdot)$ on randomly chosen $\hat{x}, \hat{y}$, Then, it calculates $[1]\cdot[\hat{x}]^{N-1}$ and $[1]\cdot[\hat{y}]^{N-1}$, uses $[1]\cdot[\hat{x}]^{N-1}$ and $[\hat{y}]$ as the inputs of $\mathsf{Sim}_{S_1}^{(\mathbf{RSM})}(\cdot,\cdot)$, uses $[\hat{x}]$ and $[1]\cdot[\hat{y}]^{N-1}$ as the inputs of $\mathsf{Sim}_{S_1}^{(\mathbf{RSM})}(\cdot,\cdot)$, and generates $[\hat{f}_1^*]$ and $[\hat{f}_2^*]$, respectively. Finally, it calculates $[\hat{f}] = [\hat{f}_1^*]\cdot[\hat{f}_2^*]$, sends the encryption $[\hat{f}_1^*], [\hat{f}_2^*], [\hat{f}]$ to $\mathcal{A}_{S_1}$. If $\mathcal{A}_{S_1}$ replies with $\perp$, then Sim_{S_1} returns $\perp$.

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

The security proofs of **SLT**, **SEQ** and **SMOD** are similar to that of the **SXOR** under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. For the encrypted floating point number calculations (include **FPEQ**, **SFPS**, **SFPA**, **SFPM**, and **SFPC**), the security relies on the basic encrypted integer calculation (the prove method is similar to that of the **SXOR**), which has been proven. All the calculations are operated over a ciphertext domain which is secure due to the semantic security of PCPD. Next, we will illustrate our POCF is secure under an active adversary $\mathcal{A}^*$ defined in III-C.

C. Security of POCF framework

The security of POCF can be guaranteed by the following theorem.

Theorem 4. *The POCF framework can securely evaluate the FPN operations of plaintext on ciphertext in the presence of semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_R}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.*

Proof. We now demonstrate how to construct three independent simulators $\mathsf{Sim}_{D_R}, \mathsf{Sim}_{S_1}, \mathsf{Sim}_{S_2}$.

Sim_{D_R} receives x_i $(i = 1, \cdots, \alpha)$ and $a_j (j = 1, \cdots, n)$ as input and simulates $\mathcal{A}_{D_R}$ as follows: it generates encryption $\langle x_i \rangle = ([s_{x_i}], [m_{x_i}], [t_{x_i}])$, where $[s_{x_i}], [m_{x_i}], [t_{x_i}]$ can be generated by executing $\mathbf{Enc}(\cdot)$. Moreover, it generates $[a_j] = \mathbf{Enc}(a_j)$ of a_j. After that, it returns $\langle x_i \rangle$ $(i = 1, \cdots, \alpha)$ and $[a_j] (j = 1, \cdots, n)$ to $\mathcal{A}_{D_R}$ and outputs $\mathcal{A}_{D_R}$'s entire view.

The view of $\mathcal{A}_{D_R}$ consists of the encrypted data. The views of $\mathcal{A}_{D_R}$ in the real and the ideal executions are indistinguishable due to the semantic security of PCPD.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: First, it receive RU's computation query, i.e., which kinds of secure FPN operations are needed. Then, it generates (fictitious) encryptions $\langle \hat{x}_i \rangle$ and $[\hat{a}_j]$ by running $\mathbf{Enc}(\cdot)$ on randomly chosing $s_{x_i}, m_{x_i}, t_{x_i}, a_j$ $(i = 1, \cdots, \alpha, j = 1, \cdots, n)$. Then, it uses $\mathsf{Sim}_{S_1}^{(\star)}(\cdot,\cdot)$ to generate $\langle \hat{x}_k \rangle$, where $k = \alpha + 1, \cdots, n$, and $\star$ can be **FPEQ**, **SFPS**, **SFPA**, **SFPM**, and **SFPC** according to RU's query. After that, $\langle \hat{A}_j \rangle = ([\hat{s}_{A_j}], [\hat{m}_{A_j}], [\hat{t}_{A_j}])$ should be computed, where $[\hat{s}_{A_j}] \leftarrow \mathsf{Sim}_{S_1}^{(\mathbf{RSM})}([\hat{a}_j], [\hat{s}_{x_j}])$, $[\hat{m}_{A_j}] \leftarrow \mathsf{Sim}_{S_1}^{(\mathbf{RSM})}([\hat{a}_j], [\hat{m}_{x_j}])$, and $[\hat{t}_{A_j}] \leftarrow \mathsf{Sim}_{S_1}^{(\mathbf{RSM})}([\hat{a}_j], [\hat{t}_{x_j}])$. Finally, it calculates $[\hat{s}_A] = \prod_j [\hat{s}_{A_j}]$, $[\hat{m}_A] = \prod_j [\hat{m}_{A_j}]$, $[\hat{t}_A] = \prod_j [\hat{t}_{A_j}]$, and sends the encryption $\langle \hat{A} \rangle = ([\hat{s}_A], [\hat{m}_A], [\hat{t}_A])$ to $\mathcal{A}_{S_1}$. If $\mathcal{A}_{S_1}$ replies with $\perp$, then Sim_{S_1} returns $\perp$.

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

Here, we also give an analysis to show that our POCF can resist system attacker defined in Section III-C. The analysis describes as follows: 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 **FPEQ**, **SFPS**, **SFPA**, **SFPM**, and **SFPC**) transmitted between CP and CSP may also be available to $\mathcal{A}^*$ due to the eavesdropping. However, these data are encrypted during transmit, $\mathcal{A}^*$ will not be able to decrypt the ciphertext without knowing the challenge RU's private key due to the semantic security of the PCPD cryptosystem. Next, we suppose $\mathcal{A}^*$ has compromised the CSP (or CP) to obtain the challenge RU's partial private key. However, $\mathcal{A}^*$ is unable to recover the challenge RU's private key to

TABLE II
THE PERFORMANCE OF PCPD (1000-TIME ON AVERAGE, 80-BIT SECURITY LEVEL)

Algorithm	Enc	PDec1	PDec2	CR	Dec
PC Run Time	7.660 ms	14.509 ms	22.168 ms	7.942 ms	8.221 ms
Smartphone Run Time	44.727 ms	89.460 ms	130.860 ms	44.791 ms	45.904 ms

decrypt the ciphertext, as the private key is randomly split by executing **KeyS** algorithm of PCPD. Even when the CSP is compromised, $\mathcal{A}^*$ is unable to obtain useful information as our protocols use the known technique of "blinding" the plaintext [29]: given an encryption of a message, we use the additively homomorphic property of the PCPD 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 RUs (i.e. not the challenge RU), $\mathcal{A}^*$ is still unable to decrypt the challenge RU's ciphertext due to the unrelated property of different RU's private keys in our system (recall private keys in the system are selected randomly and independently).

VII. EVALUATIONS

In this section, we evaluate the performance of POCF.

A. Experiment Analysis

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

1) Basic Crypto Primitive and Protocols' Performance: We first evaluate the performance of our basic cryptographic primitive and toolkits for both integer number and FPNs on our PC testbed – see TABLEs II, III and IV, respectively. We let N be 1024 bits to achieve 80-bit security [30]. We then use a smartphone with a four-core processor (4×Cortex-A53) and 2 GB RAM memory to evaluate the performance of the basic crypto primitive – see TABLE II. The evaluations demonstrate that the algorithms in PCPD are suitable for both PC and smartphone platforms. Note that both the secure integer and FPN calculation protocols are constructed for outsourced computation; therefore, they will be only evaluated in the PC testbed.

2) Factors Affecting Protocols' Performance: For secure integer processing protocols, the length of N affects the running time of the proposed schemes. From Fig. 3(a)-3(f), we can see that both the running time and the communication overhead of the integer processing schemes increase with N. This is because the running time of the basic operations (modular multiplication and exponential) increases as N increases. More bits need to be transmitted due to the increase in N. For the secure FPNs computation, two factors affect the performance: i) the length of N, ii) the domain size of the significant digits η. From Fig. 3(g)-3(i), we can see that both computational and communication costs of all the protocols increase with N, as the protocols rely on the basic operations (modular multiplication and modular exponentiation). From Fig. 3(j)-3(l), we can see that only the computational cost and the communication overhead in **SFPA** increase with the significant digit η. This is due to more loops are executed with the increase of η which consumes more computation and communication resources in **SFPA**, while the other secure FPN calculations (**FPEQ**, **SFPS**, **SFPM**, and **SFPC**) are not affected.

B. Computational Analysis

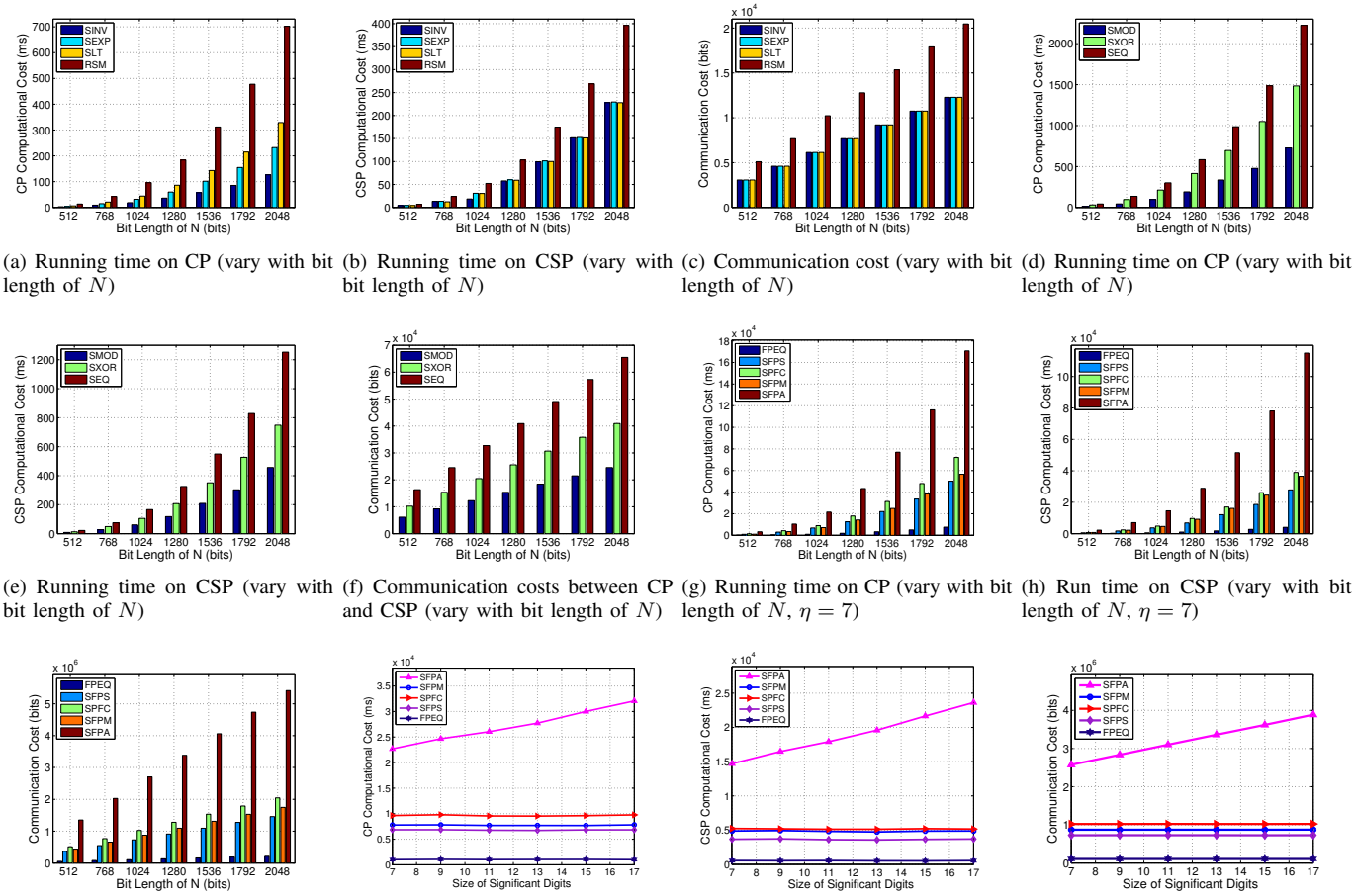
1) Computational Overhead: Let us assume that one regular exponentiation operation with an exponent of length $\|N\|$ requires $1.5\|N\|$ multiplications [31] (e.g. the length of r is $\|N\|$, and compute g^r requires $1.5\|N\|$ multiplications). As exponentiation operation is significantly more costly than the addition and multiplication operations, we ignore the fixed numbers of addition and multiplication operations in our analysis. For the PCPD scheme, **Enc** needs $1.5\|N\|$ multiplications to encrypt a message, **Dec** needs $1.5\|N\|$ multiplications to decrypt a ciphertext **PDec1** needs $3\|N\|$ multiplications to process, **PDec2** needs $3\|N\|$ multiplications (as both the length of λ_1 and λ_2 is $2\|N\|$, and compute g^{λ_1} and g^{λ_2} requires $3\|N\|$ multiplications), and **CR** needs $1.5\|N\|$ multiplications to refresh a ciphertext.

For the basic sub-protocols, it costs $13.5\|N\|$ multiplications for the CP and $7.5\|N\|$ multiplications for the CSP to run the **RSM**. For the **SXOR**, it costs $30\|N\|$ multiplications for the CP and $15\|N\|$ multiplications for the CSP to run. For the **SLT**, it costs $7.5\|N\|$ multiplications for the CP and $4.5\|N\|$ multiplications for the CSP to run. For the **SEQ**, it costs $45\|N\|$ multiplications for the CP and $24\|N\|$ multiplications for the CSP to run. For the **SEXP**, it costs $7.5\|N\|$ multiplications for the CP and $6\|N\|$ multiplications for the CSP to run. For the **SINV**, it costs $6\|N\|$ multiplications for the CP and $4.5\|N\|$ multiplications for the CSP to run. For the **SMOD**, it costs $13.5\|N\|$ multiplications for the CP and $9\|N\|$ multiplications for the CSP to run. For the secure floating point calculation, it takes $\mathcal{O}(\|N\|)$ multiplications for the **FPEQ**, **SFPS** and **SFPM**, while it takes $\mathcal{O}(\eta\|N\|)$ multiplications for the **SFPA**.

TABLE III
THE PERFORMANCE OF SUB-PROTOCOLS FOR INTEGER (1000-TIMES FOR AVERAGE, 80-BIT SECURITY LEVEL)

Protocol	CP compute.	CSP compute.	Commu.
RSM	95.635 ms	52.864 ms	1.248 KB
SLT	45.363 ms	31.255 ms	0.749 KB
SXOR	220.521 ms	113.203 ms	2.498 KB
SEQ	295.803 ms	161.788 ms	3.996 KB
SEXP	32.512 ms	31.802 ms	0.749 KB
SINV	18.993 ms	30.934 ms	0.749 KB
SMOD	96.641 ms	60.453 ms	1.499 KB

2) Communication Overhead: In the PCPD scheme, the ciphertext $[x]$ and $CT^{(1)}$ need to transmit $2\|N\|$ bits. For the



(a) Running time on CP (vary with bit length of N)
(b) Running time on CSP (vary with bit length of N)
(c) Communication cost (vary with bit length of N)
(d) Running time on CP (vary with bit length of N)
(e) Running time on CSP (vary with bit length of N)
(f) Communication costs between CP and CSP (vary with bit length of N)
(g) Running time on CP (vary with bit length of N, $\eta = 7$)
(h) Run time on CSP (vary with bit length of N, $\eta = 7$)
(i) Communication costs between CP and CSP (vary with bit length of N, $\eta = 7$)
(j) Communication costs on CP (vary with size of significant digit, $\|N\| = 1024$)
(k) Communication costs on CSP (vary with size of significant digit, $\|N\| = 1024$)
(l) Communication cost between CP and CSP (vary with size of significant digit, $\|N\| = 1024$)

Fig. 3. Simulation results

TABLE IV
THE PERFORMANCE OF SECURE CALCULATIONS OF FPNS (1000-TIME ON AVERAGE AVERAGE, 80-BIT SECURITY LEVEL, SINGLE PRECISION FPN)

Protocol	CP compute.	CSP compute.	Commu.
SFPA	21.602 s	14.567 s	330.095 KB
SFPM	7.788 s	4.980 s	106.694 KB
SFPC	9.797 s	5.234 s	124.928 KB
SFPS	6.845 s	3.732 s	88.949 KB
FPEQ	1.054 s	0.545 s	12.994 KB

basic sub-protocols, it takes $10\|N\|$ bits to run the **RSM**, $6\|N\|$ bits to run the **SLT**, **SEXP**, and **SINV**. Moreover, it takes $20\|N\|$ bits to run the **SXOR**, $32\|N\|$ bits to run the **SEQ**, and $12\|N\|$ bits to run the **SMOD**. For the secure floating point calculation, it takes $\mathcal{O}(\|N\|)$ bits for the **FPEQ**, **SFPS** and **SFPM**, while it takes $\mathcal{O}(\eta\|N\|)$ bits for the **SFPA**.

C. Analysis of secure FPNs computation

1) Rounding Errors: When approximating a nonzero real number x by RZ(x) (where RZ($\cdot$) is the *round toward zero* model defined in Section II-C), a relative error $\epsilon(x) = |(x - \text{RZ}(x))/x|$ happened. If x is in the normal range, the relative error $\epsilon(x)$ is less than or equal to $\beta^{1-\eta}$.

For any arithmetic operation $\divideontimes \in \{\text{FPN addition, FPN subtraction, FPN multiplication}\}$, for RZ rounding mode, and for all floating-point numbers a, b such that $a \divideontimes b$ does not underflow or overflow, we find that if z is the result of the correctly rounded operation $a \divideontimes b$ (that is, if $z = \circ(a \divideontimes b)$), then

$$\circ(a \divideontimes b) = (a \divideontimes b)(1 + \epsilon) + \epsilon',$$

with $|\epsilon| \leq \beta^{1-\eta}$ and $|\epsilon'| \leq \beta^{e_{min}+1-\eta}$, and ϵ and ϵ' cannot both be nonzero. For the detailed and comprehensive analysis, we refer the reader to [32].

2) The Correctness Guarantee : As the FPNs are stored as triplet $([s], [m], [t])$. If a RU only wants to securely store the FPNs without any calculations, the following restriction should be satisfied: $\|m\| < \|N\|$ and $\|t\| < \|N\|$. If a RU need to do securely FPNs storage & computation over $\langle x \rangle$ and $\langle y \rangle$, the following restriction should be satisfied: 1) $\|m_x\beta^\eta + m_y\| < \|N\|$, 2) $\|m_x \cdot m_y \cdot \beta^{2\eta}\| < \|N\|$, and 3) $\|t\| < \|N\|/4$.[11] Note that these restriction can be easily satisfied. For example, if we choose length of N as 1024 bits and $\beta = 10$, then $\|N\|/4$ is 256-bit. For single precision FPN, exponent t need

[11]Note that $s \in \{0, 1\}$ which satisfy the restriction $\|s\| < \|N\|$.

8-bit while significand m needs 23-bit to represent. For double precision FPN, exponent t need 11-bit while significand m needs 52-bit to represent. Both the situation can satisfy the above restriction. Here, we only use double precision FPN as an example to illustrate. For restriction 1, due to $|m_x \cdot 10^{\eta}| < 116$, so $|m_x \cdot 10^{\eta} + m_y| < 117$ which satisfies the restriction. For restriction 2, due to $\|m_x \cdot m_y\| \leq 105$, and $\|10^{2\eta}\| < 128$, thus $\|m_x \cdot m_y \cdot 10^{2\eta}\| < 256$ and satisfy the restriction). Notice that $\|N\| = 1024$ can achieve both the single precision FPN and double precision FPN calculation. If higher precision FPNs the user are needed (even the user-defined precision), such that, all above-mentioned conditions are not hold simultaneously. We can simply use the larger N to solve the problem.

3) Comparing with fix-point number strategy: Our secure FPNs computation has natural advantages compared with the existence secure *fix-point* number calculations (e.g., secure integer processing).

1) *Support arbitrary-time calculations*. The traditional secure fix-point calculations can only support limited times of calculations. For example, a $\|N\|/4$-bit length integer can only multiply himself 4 times by executing **RSM** protocol. Different from secure fix-point calculations, our secure FPNs computation can support arbitrary-time calculations. It is because of the 'refresh' property of our secure FPNs operations. For example, support two FPNs with s_t bit length exponent t and s_m bit length significand m. After executing **SFPA** and **SFPM**, s_t and s_m are not changed.

2) *Express more numbers*. For secure fix-point number storage, the bit-length of the number should be less than $\|N\|$. For storing FPNs, the bit-length of exponent can be represented up to $\|N\|$ length.

3) *Handle exceptional cases*. Different from traditional secure fix-point number processing, our secure FPNs processing can handle exceptional cases, such as, result overflow and underflow (see Section V-B).

D. Comprehensive Comparison

Here, we compare our POCF with the exiting secure computation on floating point numbers scheme, which can be categorized under two strategies: multiple parties storage strategy [10], [11], [12], [13], [14] and storage outsourcing strategy [15]. In the multiple parties storage strategy, n servers (In [12], [13], [14], three servers are needed) are used to storage data, i.e., each datum is randomly separated into n shares using the secret sharing technique, and the each share is distributed to a server for storage, respectively. However, it bring huge computational cost and communication overhead to the data owner for storing and managing the data, especially for updating and synchronizing the data. For example, once a datum need to be updated, the data owner needs to randomly separated the datum into n shares, encrypts the shares with the corresponding server's public key/session key, and distributes them to the different servers, respectively. Moreover, the data splitting methods in [10], [11] will increase the storage overhead with the number of the servers, while the ciphertext storage overhead will not be affected in [12], [13], [14]. Furthermore, each secret (symmetry) key is pre-shared between two servers ($n(n-1)/2$ are needed) in order to achieve secure communication, which increases the key management cost. Most importantly, every operating system must respond to floating-point exceptions [33], however, all the schemes about multiple parties storage strategy cannot completely handle the exceptional case, i.e., neither process the exceptional case of the input data [10], [11], [12], [14], [13], nor totally handle the overflow/underflow problem during the calculation [11], [14], [13].

Different from multiple parties storage strategy, the storage outsourcing strategy is more appropriated for the cloud computing environment, i.e., all the data are centralized stored in the cloud server after encryption, and it can solve the data managing problem in the multiple parties storage strategy. In [15], Ge and Zdonik try to use additive homomorphic scheme to solve the storage and computation problem about floating point numbers. Unfortunately, the [15] can only solve the secure floating point numbers addition problem, and the data expansion is huge. For example, using single precision FPN, each datum needs to expand 32 times before encryption and storage. If the double precision is needed, each FPN plaintext needs to expand 256 times than its original plaintext. Also, the floating-point exception problem is not considered in [15]. To sum up the above scheme, our POCF is designed for storage outsourcing strategy without extra plaintext expansion (stored in the cloud with constant length). Moreover, our POCF can process commonly used FPN operations and handle exceptional cases, such as process overflow and underflow during the whole calculation phases. Furthermore, we also list the performance of the above schemes. The performance can be measured in terms of two parameters: (i) the number of interactive operations (e.g., multiplications) necessary to perform the computation, and (ii) the number of sequential interactions, or rounds. A comprehensive comparison between the above schemes is shown in Table V.

VIII. Related Work

With the increasing development of IT industry, the demand for secure computation grows accordingly. One solution for secure computation across different parties is called Multi-party Computation (MPC) in which multiple parties jointly compute a function over their inputs while keeping those inputs private (for detailed definitions of secure computation see [34]). MPC is an active topic in cryptography which has been studied for more than twenty years since Yao's millionaire protocol [35]. One approach to achieve MPC is Garbled-Circuit based MPC: Bob creates a "garbled circuit", and sends the circuit to Alice. Alice evaluates the circuit with her inputs and returns the result to Bob. The term garbled circuit is from Beaver, Micali, and Rogaway [36], where the method is first based on a symmetric primitive. After that, enormous literatures use garbled-circuit to design protocol for real-world applications [37], [38], [39], [40], and several techniques are designed for improving the running time and memory requirements of the garbled-circuit technique [41], [42], [43]. However, these schemes still suffer from very high computation and

TABLE V
COMPARISON WITH SECURE FLOATING POINT CALCULATION(WITH CONSTANT SECURITY PARAMETER)

Function/Algorithm	[15]	[10]	[11]	[12]	[13]	[14]	Our
Add R	0	$\mathcal{O}(\log \eta)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log \eta)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Add I	0	$\mathcal{O}(\eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(\log \eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(\eta)$
Mul R	N.A.	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log \eta)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Mul I	N.A.	$\mathcal{O}(\eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(1)$	$\mathcal{O}(\log \eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(1)$
Cmp R	N.A.	$\mathcal{O}(1)$	N.A.	N.A.	$\mathcal{O}(\log \eta)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Cmp I	N.A.	$\mathcal{O}(\eta)$	N.A.	N.A.	$\mathcal{O}(\log \eta)$	$\mathcal{O}(\eta)$	$\mathcal{O}(1)$
Pre-shared Secret inf	0	$n \cdot (n-1)/2$	≥ 1	3	3	3	0
Data Storage	One Server	n Servers	≥ 2 Servers	Three Servers	Three Servers	Three Servers	One Server
Data Expansion	With Precision	With Server Num.	With Server Num.	Constant	Constant	Constant	Constant
Handle exceptional cases During calculation	×	×	×	×	×	×	✓
Process non-numeric data	×	×	×	×	×	×	✓

– In the table, 'Add' is short for 'floating point numbers addition', 'Mul' is short for 'floating point numbers multiplication', 'Cmp' is short for 'floating point numbers comparison'. 'R' is short for 'communication round', 'I' is short for 'the number of interactive operations'

communication complexities [44], [45]. Another approach to MPC is secret sharing based MPC protocols [46], [47], [48] which generate random data using secret sharing techniques, and distribute the shares to different servers. All the servers jointly compute some functions interactively. Although the secret sharing based MPC is promising, it requires multiple servers to store certain redundant data, and requires pairwise secure channels between servers. Very recently, fully homomorphic encryption (FHE) [49], [50] has been used to reduce the round complexity (e.g, reduced to two-round) in MPC [51], [52]. Unfortunately, one of the biggest drawbacks of fully homomorphic cryptosystems is the system complexity. The other approach is to use indistinguishability obfuscation (IO) [53], [54] to achieve two-round MPC protocols [55], [56]. Although IO has the power to dramatically broaden the scope of cryptography, how to construct practical IO is still an open research problem.

As more users choose to encrypt-and-outsource their data to cloud servers for storage with the constant evolution of cloud and related technologies, it is important to ensure that the outsourced encrypted data can be manipulated without compromising on the privacy of the data owner. Different from the huge computation and storage overhead of FHE, partial homomorphic encryptions (including additive, multiplicative and somewhat homomorphic encryptions) are often considered as the next best solution. Additive homomorphic encryption schemes, such as the Paillier cryptosystem [16] and the Bresson cryptosystem [57], allow a third party to perform some additive calculations over ciphertexts. Multiplicative homomorphic encryptions, such as the unpadded RSA cryptosystem [58] and the ElGamal cryptosystem [59], allow a third party to perform multiplication calculations over ciphertexts. Somewhat homomorphic encryption (SWHE) [60] is a crucial component of FHE which allows many additions and a small number of multiplications on ciphertexts, and it can be used to construct cloud applications [61], [62]. Due to the higher efficiency of partial homomorphic encryptions, many privacy-preserving protocols have been constructed, such as secure comparison protocols [63], secure set intersection protocols [64], and secure TOP-K protocols [21], [8]. Although homomorphic encryptions have been applied in a number of real-world scenarios [65], [28], most of the existing schemes can only process integer numbers and cannot securely handle FPNs. Although some works [10], [11], [12], [13], [14], [15] have constructed the SMC on floating point numbers, however, their framework has high interactive operation complexity (see Section VII-D for comparison).

IX. CONCLUSION

In this paper, we proposed POCF, a framework for privacy-preserving outsourced calculation on floating point numbers, which allows a user to outsource encrypted FPNs to a cloud service provider for storing and processing. We also proposed a new cryptographic primitive, Paillier cryptosystem with partial decryption (PCPD), to reduce both key management cost and private key exposure risk. We built toolkits to perform privacy preserving calculations to handle most commonly used integer operations, and to process outsourced FPNs numbers in a privacy-preserving way. The utility of our framework (and the underlying building blocks) was then demonstrated using simulations.

As a future research effort, we plan to apply our proposed POCF in a specific application domain, such as e-health cloud system and test it in a real-world setting. This will allow us to refine the framework to handle more complex real-world computations.

ACKNOWLEDGMENT

This research is supported in part by the Singapore National Research Foundation under NCR Award No. NRF2014NCR-NCR001-012. It is also supported in part by the National Natural Science Foundation of China under grant No. 61402109, No. 61502400, No. 61370078 and No. 61502248. 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," 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," http://www.comp.leeds.ac.uk/mscproj/reports/1011/wang.pdf.

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

[4] "U.s. federal cloud computing market forecast 2015-2020," http://www.marketresearchmedia.com/?p=145.

[5] S. Ried, "Sizing the cloud - a bt futures report," https://www.forrester.com/Sizing+The+Cloud/fulltext/-/E-RES58161.

[6] "The cloud computing and distributed systems (clouds) laboratory, university of melbourne," http://www.cloudbus.org/.

[7] "Mobile & cloud computing laboratory (mobile & cloud lab)," http://mc.cs.ut.ee/.

[8] X. Liu, R. Lu, J. Ma, L. Chen, and B. Qin, "Privacy-preserving patient-centric clinical decision support system on naive bayesian classification," in *IEEE Journal of Biomedical and Health Informatics [Preprint Online]*, 2015.

[9] A. Hidaka, S. Sasazuki, A. Goto, N. Sawada, T. Shimazu, T. Yamaji, M. Iwasaki, M. Inoue, M. Noda, H. Tajiri, and S. Tsugane, "Plasma insulin, c-peptide and blood glucose and the risk of gastric cancer: The japan public health center-based prospective study," *International Journal of Cancer*, vol. 136, no. 6, pp. 1402–1410, 2015.

[10] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, "Secure computation on floating point numbers," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013.

[11] Y. Liu, Y. Chiang, T. Hsu, C. Liau, and D. Wang, "Floating point arithmetic protocols for constructing secure data analysis application," in *17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, KES 2013, Kitakyushu, Japan, 9-11 September 2013*, 2013, pp. 152–161.

[12] L. Kamm and J. Willemson, "Secure floating point arithmetic and private satellite collision analysis," *Int. J. Inf. Sec.*, vol. 14, no. 6, pp. 531–548, 2015.

[13] T. Krips and J. Willemson, "Hybrid model of fixed and floating point numbers in secure multiparty computations," in *Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings*, 2014, pp. 179–197.

[14] P. Pullonen and S. Siim, "Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations," in *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*, 2015, pp. 172–183.

[15] T. Ge and S. B. Zdonik, "Answering aggregation queries in a secure system model," in *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, 2007, pp. 519–530.

[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, 1999, pp. 223–238.

[17] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the workshop on selected areas of cryptography*, 1994, pp. 120–128.

[18] I. S. Committee *et al.*, "754-2008 ieee standard for floating-point arithmetic," *IEEE Computer Society Std*, vol. 2008, 2008.

[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] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptologyEUROCRYPT99*. Springer, 1999, pp. 223–238.

[21] B. K. Samanthula, Y. Elmehdwi, and W. Jiang, "k-nearest neighbor classification over semantically secure encrypted relational data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 5, pp. 1261–1273, 2015.

[22] U. Consortium *et al.*, *The Unicode Standard, Version 2.0*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[23] B. Pinkas and T. Reinman, "Oblivious ram revisited," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 502–519.

[24] M. T. Goodrich and M. Mitzenmacher, "Privacy-preserving access of outsourced data via oblivious ram simulation," in *Automata, Languages and Programming*. Springer, 2011, pp. 576–587.

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

[26] C. Ding, *Chinese remainder theorem*. World Scientific, 1996.

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

[28] X. Liu, B. Qin, R. H. Deng, and Y. Li, "An efficient privacy-preserving outsourced computation over public data," in *IEEE Transactions on Service Computing [Preprint Online]*, 2015.

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

[30] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST special publication 800-57," *NIST Special Publication*, vol. 800, no. 57, pp. 1–142, 2007.

[31] D. E. Knuth, "Seminumerical algorithm (arithmetic) the art of computer programming vol. 2," 1981.

[32] J.-M. Muller, N. Brisebarre, F. De Dinechin, C.-P. Jeannerod, V. Lefevre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of floating-point arithmetic*. Springer Science & Business Media, 2009.

[33] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, 1991.

[34] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, 1998.

[35] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[36] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990, pp. 503–513.

[37] A. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," *IACR Cryptology ePrint Archive*, vol. 2009, p. 507, 2009.

[38] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[39] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A. Sadeghi, and T. Schneider, "Secure evaluation of private linear branching programs with medical applications," in *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, 2009, pp. 424–439.

[40] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, 2007, pp. 498–507.

[41] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004.

[42] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming*. Springer, 2008, pp. 486–498.

[43] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, 2011.

[44] J. Katz and R. Ostrovsky, "Round-optimal secure two-party computation," in *Advances in Cryptology–CRYPTO 2004*. Springer, 2004, pp. 335–354.

[45] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 784–796.

[46] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Computer Security-ESORICS 2008*. Springer, 2008, pp. 192–206.

[47] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 643–662.

[48] M. Keller, P. Scholl, and N. P. Smart, "An architecture for practical actively secure mpc with dishonest majority," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 549–560.

[49] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, 2009, pp. 169–178.

[50] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, 2013, pp. 75–92.

[51] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the 44th Symposium on Theory of Computing Confer-*

ence, STOC 2012, New York, NY, USA, May 19 - 22, 2012, 2012, pp. 1219–1234.
[52] P. Mukherjee and D. Wichs, "Two round MPC from LWE via multi-key FHE," *IACR Cryptology ePrint Archive*, vol. 2015, p. 345, 2015. [Online]. Available: http://eprint.iacr.org/2015/345
[53] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, 2013, pp. 40–49.
[54] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: deniable encryption, and more," in *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, 2014, pp. 475–484.
[55] S. Garg, C. Gentry, S. Halevi, and M. Raykova, "Two-round secure MPC from indistinguishability obfuscation," in *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, 2014, pp. 74–94.
[56] S. Garg and A. Polychroniadou, "Two-round adaptively secure MPC from indistinguishability obfuscation," in *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, 2015, pp. 614–637.
[57] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, 2003, pp. 37–54.
[58] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[59] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," vol. 31, no. 4, 1985, pp. 469–472.
[60] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.
[61] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Applied Cryptography and Network Security*. Springer, 2013, pp. 102–118.
[62] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 2012, pp. 1219–1234.
[63] I. Damgard, M. Geisler, and M. Kroigard, "Homomorphic encryption and secure comparison," *International Journal of Applied Cryptography*, vol. 1, no. 1, pp. 22–31, 2008.
[64] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*, 2012, pp. 85–86.
[65] M. Li, N. Cao, S. Yu, and W. Lou, "Findu: Privacy-preserving personal profile matching in mobile social networks," in *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, 10-15 April 2011, Shanghai, China*, 2011, pp. 2435–2443.

Ximeng Liu (S'13-M'16) received the B.Sc. degree in electronic engineering from Xidian University, Xi'an, China, in 2010 and Ph.D. degrees in Cryptography from Xidian University, China, in 2015. He was the research assistant at School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore from 2013 to 2014. Now, he is a research fellow at School of Information System, Singapore Management University, Singapore. His research interests include cloud security, applied cryptography and big data security.

Robert H. Deng has been a professor at the School of Information Systems, Singapore Management University since 2004. His research interests include data security and privacy, multimedia security, network and system security. He was the 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, an associate editor of 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 the 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. He is a fellow of the IEEE.

Wenxiu Ding received her B.Eng. degree in information security from Xidian University, Shaanxi, China in 2012. She is currently pursuing her Ph.D. degree in information security from the School of Telecommunications Engineering at Xidian University, and also a visiting research student at the School of Information Systems, Singapore Management University. Her research interests include privacy preservation, data security and trust management.

Rongxing Lu (S'09-M'11-SM'15) received the Ph.D degree in computer science from Shanghai Jiao Tong University, Shanghai, China in 2006 and the Ph.D. degree (awarded Canada Governor General Gold Medal) in electrical and computer engineering from the University of Waterloo, Waterloo, Ontario, Canada, in 2012. Since May 2013, he has been with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, as an Assistant Professor. His research interests include computer, network and communication security, applied cryptography.

Baodong Qin received his Ph.D degree (2015) from Shanghai Jiao Tong university, Shanghai, China. During May, 2014-July, 2015, he was a research assistant at School of Information System, Singapore Management University (SMU), Singapore. Currently, he is a lecturer of information security at Southwest University of Science and Technology (SWUST), Mianyang, China. His research interests include cryptography and information security.