5-2017

Encrypted data processing with Homomorphic Re-Encryption

Wenxiu DING
Xidian University

Zheng YAN
Aalto University

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

DOI: https://doi.org/10.1016/j.ins.2017.05.004

Published in Information Sciences, 2017 May, Volume 409-410, Pages 35-55
http://doi.org/10.1016/j.ins.2017.05.004

Encrypted data processing with Homomorphic Re-Encryption

Wenxiu Ding[a], Zheng Yan[a,b,*], Robert H. Deng[c]

[a] *State Key Lab on Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China*
[b] *Department of Communications and Networking, Aalto University, Espoo 02150, Finland*
[c] *School of Information Systems, Singapore Management University, 178902, Singapore*

ABSTRACT

Cloud computing offers various services to users by re-arranging storage and computing resources. In order to preserve data privacy, cloud users may choose to upload encrypted data rather than raw data to the cloud. However, processing and analyzing encrypted data are challenging problems, which have received increasing attention in recent years. Homomorphic Encryption (HE) was proposed to support computation on encrypted data and ensure data confidentiality simultaneously. However, a limitation of HE is it is a single user system, which means it only allows the party that owns a homomorphic decryption key to decrypt processed ciphertexts. Original HE cannot support multiple users to access the processed ciphertexts flexibly. In this paper, we propose a Privacy-Preserving Data Processing (PPDP) system with the support of a Homomorphic Re-Encryption Scheme (HRES). The HRES extends partial HE from a single-user system to a multi-user one by offering ciphertext re-encryption to allow multiple users to access processed ciphertexts. Through the cooperation of a Data Service Provider (DSP) and an Access Control Server (ACS), the PPDP system can support seven basic operations over ciphertexts, which include *Addition, Subtraction, Multiplication, Sign Acquisition, Comparison, Equivalent Test*, and *Variance*. To enhance the flexibility and security of our system, we further apply multiple ACSs to take in charge of the data from their own users and design computing operations over ciphertexts belonging to multiple ACSs. We then prove the security of PPDP, analyze its performance and advantages by comparing with some latest work, and demonstrate its efficiency and effectiveness through simulations with regard to big data process.

Homomorphic encryption
Privacy preservation
Data sharing
Proxy re-encryption
Access control
Cloud Computing

1. Introduction

Cloud computing provides various services based on user demands by rearranging resources over the Internet. It releases heavy burdens of users for managing information systems by themselves and breaks the bottlenecks of restricted local resources. In cloud computing, users outsource their data to a heterogeneous environment for storage and processing. Such an environment is often beyond the control of the users. This gives rise to a pressing need to safeguard the security and privacy of data in the cloud [34].

Meanwhile, with the rapid development of Internet of Things, huge amounts of data about environments, society, health and business [4,28] are produced and transmitted over the network. Data process, analytics and mining become significant

[*] Corresponding author.
E-mail addresses: wenxiuding_1989@126.com (W. Ding), zyan@xidian.edu.cn, zheng.yan@aalto.fi, zhengyan.pz@gmail.com (Z. Yan), robertdeng@smu.edu.sg (R.H. Deng).

to help a party gain profits not only from their own data, but also from the data provided by other parties. Due to security concerns, many parties may be reluctant to outsource their personal data to the cloud for analysis. However, some resource-constrained parties have to depend on the cloud to complete a complicated computation mission, especially for big data analytics and mining. Hence, it calls for a privacy-preserving data processing scheme in cloud computing, which can adapt to various scenarios. A promising solution to protect data privacy in the cloud is to store and process data in an encrypted form [17,31,37]. However, encryption greatly complicates data processing and brings many new challenges and problems.

First, encryption seriously restricts the access to processed ciphertexts. Homomorphic Encryption (HE) was proposed to support computation on encrypted data and ensure data confidentiality simultaneously. However, a limitation of HE is it is a single user system, which means it only allows a single party with a corresponding secret key to access processed ciphertexts [13,14] and cannot support multiparty access. However, in many situations data are observed and collected all the time for potential use without knowing a concrete aggregator or a data access requester. For example, medical and clinical research can benefit greatly from the statistics of patients and more than one party could be interested in requesting encrypted processing results after data collection and process. Original HE cannot support multiple users to access the processed ciphertexts flexibly. Second, the operations over the ciphertexts are limited. Fully Homomorphic Encryption (FHE) [15,29] is designed to support various computations over ciphertexts. But it introduces much computation overhead, thus is not yet practical to be employed in real applications [23,27]. Even though many constructions [1,12,14,33] were further designed to improve the efficiency of FHE, they still cannot be practically used due to large key size and high storage overhead. Compared with FHE, Partial Homomorphic Encryption (PHE) has lower computation overhead, but it merely supports limited computations [13,24]. Third, it incurs much computation overhead to complete data processing especially when there is a huge number of input data. The performance of existing schemes is hard to meet the demands of various application scenarios, especially for big data processing.

In order to achieve secure and privacy-preserving data processing at the cloud, many schemes have been designed with various techniques: FHE, PHE, and Secure Multiparty Computation (SMC). However, none of them can overcome the above three problems. FHE-based schemes [14,33] do not provide flexible access to ciphertext processing results and have high computation overhead. In addition, it has been proved that it is not sufficient to apply cryptography alone if data are shared among multiple clients [30] due to the difficulty of secret key transformation between users. Hence, some schemes [6,18,21,25] with multiple servers are proposed and designed. But they still suffer from some shortcomings. For example, a scheme [25] based on Paillier's cryptosystem [24] was proposed to achieve computations over ciphertexts outsourced by a number of users with their own keys. However, it can only support multiplication and addition. Another PHE-based scheme [21] achieves a number of computation operations, but it has high computation overhead, especially in multiplication based on a large number of input data. Sharemind [5], a popular SMC scheme, achieves secure addition and multiplication with the support of at least three servers. Some constructions based on Sharemind [6,18] further achieve various computations, but increase the difficulty of data management and the multiplication over a big number of data.

In this paper, we design a Privacy-Preserving Data Processing (PPDP) system aiming to overcome the above three problems. It contains a Homomorphic Re-Encryption Scheme (HRES), which extends PHE from a single-user system to a multi-user one by offering ciphertext re-encryption to allow multiple users to access processed ciphertexts. With this way, it achieves flexible and secure access control over encrypted data processing results. Through the cooperation of a Data Service Provider (DSP) and an Access Control Server (ACS), the PPDP system can support seven basic operations over ciphertexts, which include *Addition, Subtraction, Multiplication, Sign Acquisition, Comparison, Equivalent Test*, and *Variance*. To enhance the flexibility and security of our system, we further apply multiple ACSs to take in charge of the data from their own users and design computing operations over ciphertexts belonging to multiple ACSs. We then prove the security of PPDP, analyze its performance and advantages by comparing with some latest work, and demonstrate its efficiency and effectiveness through simulations with regard to big data processing.

The PPDP system is original and differs from the previous work with regard to the above three unsolved issues. It is based on PHE and supports most of basic computation operations with high efficiency in terms of big data processing. Specifically, the main contributions of this paper can be summarized as below:

- In order to support flexible access to ciphertext processing results, we design a new cryptographic primitive: Homomorphic Re-Encryption Scheme (HRES). It employs two service providers to manage encrypted data and flexibly support access control on ciphertext processing results with two-level decryption, thus successfully achieves re-encryption over the data encrypted with homomorphism. Only authorized users can access the ciphertext processing result in a secure way.
- We construct the PPDP system with the support of HRES to achieve efficient computations over ciphertexts, which supports seven basic operations: *Addition, Subtraction, Multiplication, Sign Acquisition, Comparison, Equivalent Test*, and *Variance*.
- We extend the PPDP system to support ciphertext computations across multiple ACSs in order to enhance the flexibility of data management and system security, as well as the feasibility of system deployment in practice.
- We prove the security of the PPDP system and evaluate its performance and advantages through analysis and comparison. The efficiency is demonstrated through simulations.
- We show that the PPDP system is suitable for big data processing. It can be applied in various scenarios with either a small or large number of data providers.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. Section 3 introduces the system and attack model of the PPDP, followed by its preliminaries and detailed design in Section 4. In Section 5, we further introduce the scheme that supports seven operations across multiple ACSs. Security analysis and performance evaluation are given in Section 6. Finally, we conclude the paper in the last section.

2. Related work

With the fast development of cloud computing, more and more users choose to outsource their data to the cloud for storage and further process. However, the risk of the data being revealed or disclosed makes it urgent to enhance the security and privacy of user data. In the literature, we can find many researches with regard to privacy-preserving data processing, which is briefly reviewed in this section by classifying them into three categorites.

2.1. Privacy-preserving aggregation

Literature has a number of studies on privacy-preserving data aggregation, mainly in the area of Wireless Sensor Networks (WSNs) and smart metering [3,10,11,16,19,20,26]. Some previous work [10,20] on data aggregation assumed a trusted aggregator, and hence cannot protect user privacy from a distrusted or semi-trusted aggregator. Castelluccia et al. proposed a simple and provably secure encryption scheme that allows efficient additive aggregation of encrypted data [10], in which an aggregator holds the sum of secret shares of all data providers for final decryption. Based on this work, Li et al. employed a novel key management scheme to obtain data sum [20]. Low aggregation error can further be achieved by leveraging a ring-based interleaved grouping technology [19]. Shi et al. [11,26] also studied encrypted data aggregation in the presence of a distrusted aggregator. The aggregator splits a secret key s_0 into additive shares among a set of users (say $s_0 = \sum_i s_i$), and s_i is issued to user i. Then each user applies its secret share to blind its private data. Hence, the aggregator can only obtain the sum of user data but nothing else through decryption with s_0. Joye and Libert [16] proposed a practical scheme that can accommodate large plaintext spaces. However, the above schemes have a major drawback that they are not tolerant of user absence or failure. Thus, they are not applicable for data aggregation where the number of data providers is not fixed or the provider is absent sometime. The PPDP system proposed in this paper overcomes this shortcoming and provides high flexibility.

2.2. Secure data processing based on SMC

SMC enables private data to be computed with a global function without leaking each individual input data. It also provides plausible solutions for such problems as privacy-preserving database query, privacy-preserving intrusion detection and privacy-preserving data mining [36]. For example, a method for financial analysis [6] based on SMC can obtain important data from collected data by deploying three servers. But this method introduces extra complexity and overhead. Kamm and Willemson proposed to realize various computations over the shared data, such as addition, multiplication and comparison [18]. But it directly employs the basic operations in Sharemind, which leads to the restriction to the number of multiplications over provided data. The product of N pieces of data needs 3^N multiplications of 32-bit numbers under the cooperation of three servers. In addition, deploying with three servers at least complicates the system. Compared with the above SMC-based schemes, the proposed PPDP system can support the computation over hundreds of data. It contains only two servers that can be provided by a public cloud, a private cloud or a key department of a company. Thus, it can satisfy with the requirements of many real applications in a better way.

2.3. Secure data processing based on homomorphic encryption

Some studies tried to improve the existing homomorphic encryption in order to support computations over encrypted data. Many FHE schemes [8,14,29] have been designed to support arbitrary computations over ciphertexts. But they have high computation overhead and are still impractical. Cheon et al. designed an extension of FHE [12] to support the encryption and processing over a vector of plaintext bits rather than a single plaintext bit to generate a ciphertext. Wang et al. [33] tried to improve the efficiency of FHE. However, the storage consumption overhead and computation cost of the above two studies are still not satisfactory, which makes them inapplicable to resource-constrained devices.

As one of the most popular PHE schemes, Paillier's cryptosystem has been widely employed and revised for various applications. Wang et al. [32] revised it to realize arithmetic functions over the ciphertexts of multiple users for addition and multiplication without learning the function inputs or intermediate results. But this scheme needs to solve the problem of discrete logarithm, which seriously restricts the length of input data. Therefore, it is not suitable to be applied into the scenarios where there are many data providers and the size of provided data is big. Ayday et al. [3] revised the Paillier's cryptosystem and achieved privacy-preserving data aggregation by dividing its decryption key into two parts and sharing them with a proxy and a medical center. But this scheme cannot support multiparty access to encrypted data processing results. The scheme in [35] can support multiparty access to evidence aggregation, but it is only applicable for addition and cannot support other computation operations. Peter et al. [25] proposed an efficient outsourcing multiparty computation framework under multiple keys based on additive homomorphic encryption [24]. However, this scheme can only support

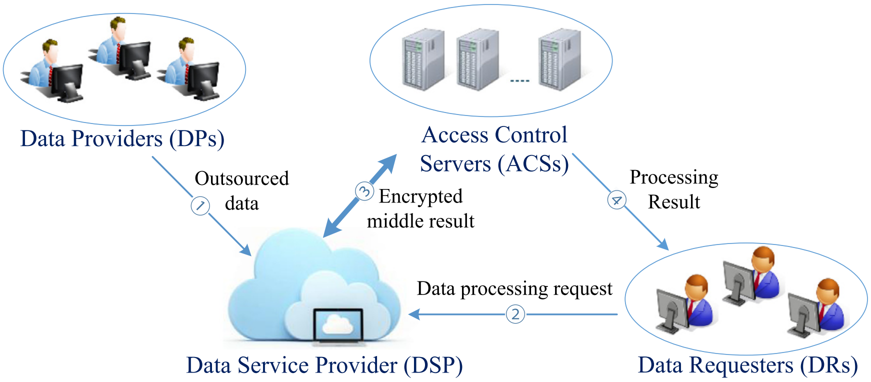

Fig. 1. System Model.

addition and multiplication, but no other operations. Moreover, it follows the idea proposed in [18], which results in the same weakness as [18] – unable to support the multiplication of a large number of data. Another PHE-based work [21] was constructed for efficient outsourced data calculation, which can deal with several types of operations, such as addition, multiplication, and division. Based on the same cryptosystem design as applied in [21] by splitting the secret keys of PHE into different shares, Liu et al. [22] proposed multiple calculations over outsourced data under different keys. However, similar to the work in [25], the schemes in [21,22] can only support the multiplication over a small number of input data. In addition, the single secret key of Paillier cryptosystem is splitted and distributed to specific servers, thus they do not support the computations over arbitrary servers flexibly. However, our scheme shows great superiority with regard to system flexibility, which can support the cooperation of DSP with arbitrary ACSs. The cloud users can choose an ACS optionally for data processing and get the processing result over the encrypted data managed by multiple ACSs.

More advanced than the above work related to the secure data processing based on HE, the PPDP system proposed in this paper supports seven basic computation operations and realizes flexible data access control over the encrypted data processing result for multiple authorized parties.

3. Problem statements

3.1. System Model

The PPDP system comprises four types of entities, as shown in Fig. 1:

(1) Data Service Provider (DSP) is served by a cloud service provider, which stores user data and provides some computation service.
(2) Access Control Server (ACS) is mainly in charge of secure data computation with data access control for its users, which can be served by a private cloud. In the PPDP system, there could exist several ACSs that are operated by different organizations, such as medical institutions and banks. Hence, a user can freely choose an ACS it trusts for service consumption. This could enhance user security but also bring challenges to the computations across ACSs.
(3) Data Providers (DPs) are the data collectors or producers that encrypt data for protecting data privacy and store the encrypted data in the DSP. The encrypted data can be used for further computation and analysis through the cooperation of the DSP and the ACSs.
(4) Data Requesters (DRs) are the data consumers that acquire the result of data processing and analyzing in a specific context. A DR can also be a DP. Since the provided data are encrypted, the data processing result is also in an encrypted form. This raises the issue of data access control with regard to encrypted data processing results or ciphertext processing results.

The DSP collects and/or stores the encrypted data outsourced by the DPs. After getting the request from the DR, the DSP cooperates with the ACS to process the stored data according to DR demands. Finally, the ciphertext processing results are returned to the eligible DR that can obtain the plain processing results.

3.2. Attack model

In the above system, all entities are assumed to be curious-but-honest. That is, they are curious about others' data but act honestly by following the system protocols strictly. In addition, the DSP and the ACSs would never collude with each other due to conflict of interests. Moreover, any collusion would decrease user trust in the ACS, which leads to the loss of its users. Therefore, we introduce an adversary $\mathcal{A}^*$ in our model, which aims to decrypt the ciphertext of a challenge sent to a cloud user (either a DR or a DP) with the following capabilities:

Table 1
Notations.

Symbols	Description
g	The system generator that is public;
n	The system parameter;
$(sk_{DSP},\ pk_{DSP})$	The key pair of DSP;
$(sk_{ACS},\ pk_{ACS})$	The key pair of ACS;
$PK = {pk_{DSP}}^{sk_{ACS}} = {pk_{ACS}}^{sk_{DSP}}$	The public parameter based on keys of DSP and ACS;
$(sk_j,\ pk_j)$	The key pair of DR j;
(sk_i, pk_i)	The key pair of DP i;
m_i	The raw data provided by DP i;
$[m]$	The ciphertext of m under PK;
$[m]^+$	The re-encryption result of m by DSP;
$[m]_{pk_i}$	The ciphertext of m under public key pk_i;
r	The random number;
CID	The public computation identifier, which specifies an operation type;
N	The number of provided data for processing;
$\mathcal{L}(*)$	The bit length of input data;
$H()$	The hash function;

(1) $\mathcal{A}^*$ may eavesdrop all communication channels to access any encrypted data;
(2) $\mathcal{A}^*$ may compromise the DSP to guess the raw data of all ciphertexts outsourced from the DPs, and the raw data of all ciphertexts sent to the ACS and the DR;
(3) $\mathcal{A}^*$ may compromise the ACS to guess the raw data of all ciphertexts received from the DSP;
(4) $\mathcal{A}^*$ may compromise the DSP or the ACS (but not concurrently) together with the DPs to guess the final data processing result;
(5) $\mathcal{A}^*$ may compromise the DSP or the ACS (but not concurrently) together with the DR to guess the raw data from the challenger DP.

The attack model has two restrictions: 1) the adversary $\mathcal{A}^*$ is unable to compromise the DSP and the ACS concurrently; 2) $\mathcal{A}^*$ cannot compromise the challenger DR or DP. The adversary would like to know the raw data of a provider (by attacking the DP) or the encrypted data processing result (by attacking the DR).

4. Privacy-preserving data processing with access control (PPDP)

4.1. Preliminaries and notations

4.1.1. Additive homomorphic encryption

Paillier's cryptosystem [24] is one of the most important additive homomorphic encryption. Suppose we have N encrypted data under same key pk, which can be presented as $[m_i]_{pk}$ $(i = 1,\ 2, \ldots, N)$. The additive homomorphic encryption satisfies the following equation:

$$D_{sk}\left(\prod_{i=1}^{N}[m_i]_{pk}\right) = \sum_{i=1}^{N} m_i \tag{1}$$

where $D_{sk}()$ is the corresponding homomorphic decryption algorithm with secret key sk.

4.1.2. Basic encryption and decryption algorithms

For easy presentation, we use EDD to represent the following mechanism proposed by Emmanuel, Dario and David [9].

Given two large primes p and q, then $n = p{*}q$. Let g and h be two elements of maximal order in $\mathbb{G}$, where $\mathbb{G}$ is the cyclic group of quadratic residues modulo n^2. Note that, if h is computed as $g^x \, mod \, n^2$, where $x \in_R [1,\ \lambda(n^2)]$ ($\lambda(*)$ is the Euler function), then x is coprime with ord($\mathbb{G}$) with high probability, and thus h is of maximal order.

Key Generation: The public parameters are n, g and $h = g^x$ mod n^2 by randomly choosing a secret value $x \in [1, ord(\mathbb{G})]$.

Encryption (***Enc***): Given a message $m \in \mathbb{Z}_n$, random number r is chosen in $\mathbb{Z}_n^*$. The ciphertext is computed as $[m]_h = (T,\ T') = \{h^r(1 + m{*}n), g^r\}$ $(mod\ n^2)$.

Decryption (***Dec***): Knowing x, m can be obtained as follows: $m = L(T/(T')^x\ mod\ n^2)$, where $L(u) = (u - 1)/n$.

4.1.3. Notations

For a better understanding, Table 1 summarizes the notations used throughout this paper.

4.2. Homomorphic Re-Encryption Scheme (HRES)

In order to support the PPDP, we revise the EDD and design the HRES that can realize proxy-invisible re-encryption and secure data processing. The HRES consists of the following algorithms:

KeyGen: Let k be a security parameter and p, q be two large primes, where $\mathcal{L}(p) = \mathcal{L}(q) = k$. Due to the property of safe primes, there exist two primes p' and q' which satisfy that $p = 2p' + 1$, $q = 2q' + 1$. We compute $n = p{*}q$ and choose a generator g with maximal order (Refer to [2]). The DSP and the ACS respectively generate key pairs: $(sk_{DSP} = a,\ pk_{DSP} = g^a)$ and $(sk_{ACS} = b,\ pk_{ACS} = g^b)$, and then negotiate their Diffie–Hellman key $PK = {pk_{DSP}}^{sk_{ACS}} = {pk_{ACS}}^{sk_{DSP}} = g^{a*b} mod\ n^2$. To support encrypted data processing, PK is public to all involved parties and can be published by ACS to its service users. Cloud user i generates its key pair $(sk_i, pk_i) = (k_i,\ g^{k_i})$. The public system parameters include $\{g,\ n,\ PK\}$.

The basic encryption (***Enc***) and decryption (***Dec***) algorithms are defined in Section 4.1.2 by applying the key pair $(sk_i, pk_i) = (k_i,\ g^{k_i})$. Next, we present a **Two-Level Decryption** scheme that can flexibly support encrypted data processing as below.

Encryption with Two Keys (*EncTK*): To flexibly support ciphertext processing, we propose encrypting original data under the keys of two servers (PK) rather than pk_i. Given message $m_i \in \mathbb{Z}_n$ provided by user i, we first select random number $r \in [1, \frac{n}{4}]$ and then encrypt it with PK. The ciphertext is generated as $[m_i] = [m_i]_{PK} = \{T_i, T_i^{'}\}$, where $T_i = (1 + m_i * n) * PK^r\ mod\ n^2$ and $T_i^{\prime} = g^r\ mod\ n^2$. Note: for ease of presentation, we use $[m_i]$ to denote the ciphertext of m_i encrypted with PK, which can only be decrypted under the cooperation of DSP and ACS.

Partial Decryption with *SK_{DSP}* (*PDec*1): Once $[m_i]$ is received by the DSP, the algorithm ***PDec*1** will be run to transfer it into another ciphertext that can be decrypted by the ACS as follows:

$$\begin{aligned}[m_i]_{pk_{ACS}} &= \left\{T_i^{(1)}, T_i^{\prime(1)}\right\} = \left\{T_i, \left(T_i^{\prime}\right)^{sk_{DSP}}\right\} = \{(1 + m_i * n)PK^r, g^{r*a}\}\ mod\ n^2 \\ &= \left\{(1 + m_i * n)\,{pk_{ACS}}^{a*r}, g^{r*a}\right\}\ mod\ n^2.\end{aligned} \tag{2}$$

Partial Decryption with *SK_{ACS}* (*PDec*2): Once the encrypted data $[m_i]_{pk_{ACS}}$ is received, the ACS can directly decrypt it with its own secret key as follows:

(1) $T_i^{'(2)} = (T_i^{'(1)})^{sk_{ACS}} = g^{r*a*b} = PK^r\ mod\ n^2$;

(2) $m_i = L(T_i^{(1)}/T_i^{'(2)}\ mod\ n^2)$, where $L(u) = (u - 1)/n$.

Note: the two-level decryption can change its decryption order.

To achieve the proxy-invisible re-encryption, we further propose a **Somewhat Re-Encryption** scheme:

Different from the scheme above, it aims to transfer the encrypted data to the ciphertext under the public key of an authorized requester. Here, we assume DR j with key pair $(sk_j, pk_j) = (k_j,\ g^{k_j}\ mod\ n^2)$ requires to obtain m_i through outsourced data $[m_i]$. In the **Somewhat Re-Encryption** scheme, the transformation needs the cooperation and recognition of both the DSP and the ACS. They together play a role of a proxy.

First Phase of Re-Encryption (*FPRE*): In order to prevent the *PDec*2 by the ACS, the DSP initiates the algorithm *FPRE* as follows:

(1) Select a public computation identifier *CID*; Compute $h_1 = H((pk_j)^{sk_{DSP}}||CID)$;

(2) $[m_i]^+ = \{\hat{T}, \widehat{T'}\} = \{T_i, (T_i^{'})^{sk_{DSP}} g^{h_1}\}$.

Second Phase of Re-Encryption (*SPRE*): Upon receiving the data packet $[m_i]^+$, the ACS launches the re-encryption algorithm *SPRE* as below:

(1) $h_2 = H((pk_j)^{sk_{ACS}}||CID)$;

(2) $[m_i]_{pk_{\mathrm{j}}} = \{\bar{T}, \overline{T'}\} = \{\hat{T}, (\widehat{T'})^{sk_{ACS}} {*} g^{h_2}\}$.

Decryption on Re-Encrypted Data (*DPRE*): DR j calls algorithm ***DPRE*** to decrypt $[m_i]_{pk_j}$:

(1) Based on *CID*, it computes: $h_1^{'} = H((pk_{DSP})^{sk_j}||CID) = H(g^{a*sk_j}||CID) = h_1$, and $h_2^{'} = H((pk_{ACS})^{sk_j}||CID) = H(g^{b*sk_j}$ $||CID) = h_2$;

(2) $m_i = L(\bar{T}{*}{pk_{ACS}}^{h_1^{'}}{*}g^{h_2^{'}}/\overline{T'} mod\ n^2)$.

Though a distribution of processing result can be achieved by a simple interaction between DSP and ACS via adding to and removing a mask from the ciphertext, our re-encryption scheme can avoid these interactions. In addition, the generation of two hash values can overcome impersonation attack and further resist the collusion between any servers with DR.

In addition, the encrypted data has the following properties:

(1) Additive homomorphism: $[m_1]_{pk_i}{*}[m_2]_{pk_i} = [m_1 + m_2]_{pk_i}$;

(2) $$\begin{aligned}([m]_{pk_i})^t &= \{\{(1 + m * n){pk_i}^r\}^t, (g^r)^t\}\ mod\ n^2 = \{(1 + m * n)^t {pk_i}^{r*t}, g^{r*t}\}\ mod\ n^2 \\ &= \{(1 + t * m * n){pk_i}^{r*t}, g^{r*t}\}\ mod\ n^2 = [t * m]_{pk_i};\end{aligned}$$

(3) $([m]_{pk_i})^{n-1} = \{\{(1+m*n)pk_i{}^{r}\}^{(n-1)}, (g^r)^{(n-1)}\} \ mod\ n^2$
$= \{(1+m*(n-1)*n)pk_i{}^{r(n-1)}, g^{r(n-1)}\} \ mod\ n^2$
$= \{(1-m*n+m*n^2)pk_i{}^{r(n-1)}, g^{r(n-1)}\} \ mod\ n^2$
$= \{(1-m*n)pk_i{}^{r(n-1)}, g^{r(n-1)}\} \ mod\ n^2 = [-m]_{pk_i}.$

Remark. If data outsourced by a user is extremely sensitive and not allowed to be processed or analyzed, the cloud user can choose to use the ***Enc*** to encrypt them with its own public key. If the data can be analyzed in a privacy-preserving way by authorized parties, the user need to store encrypted data by calling the ***EncTK***.

4.3. Encrypted data processing

Based on the HRES, we design an encrypted data processing scheme, which supports seven basic operations (indicated by different CID): 1) *Addition*; 2) *Subtraction*; 3) *Multiplication*; 4) *Sign Acquisition*; 5) *Comparison*; 6) *Equivalent Test*; and 7) *Variance*. System setup and data collection are the same for the seven operations. Hence, we first present the same process below and then introduce the detailed processing steps in sub-sections.

Step 1 (System Setup @ All Entities): The system calls the algorithm ***KeyGen*** in Section 4.2 to complete the process of setup. Note that if multiple ACSs are employed in the system, each ACS can negotiate a Diffie–Hellman key with the DSP and publish this key to its users. For simplifying presentation, we only present the detailed operations in the case that there is only one ACS interacting with the DSP as below; while the scheme that supports data computations across multiple ACSs will be described in Section 5.

Step 2 (Data Upload @ DP): DPs encrypt their personal data m_i with PK by calling the algorithm ***EncTK*** and then upload them to DSP: $[m_i] = (T_i, T_i') = \{(1+m_i*\mathrm{n})PK^{r_i}, g^{r_i}\} \ mod\ n^2$.

4.3.1. Addition

This operation aims to obtain the sum of all raw data: $m = \sum_{i=1}^{N} m_i$. Note that the number of the data in *Addition* affects the length of the provided data. If we want to get the sum result of N pieces of data, it should guarantee that $m_i < n/N$.

Step 3 (Data Preparation @ DSP): Due to additive homomorphism, the DSP can directly multiply encrypted data one by one as following:

$$[m] = \left(T, T'\right) = \prod_{i=1}^{N} [m_i] \ = \left(\prod_{i=1}^{N} T_i, \prod_{i=1}^{N} T_i^{'}\right). \tag{3}$$

To transfer it into the ciphertext under DR j's public key, the DSP further calls the algorithm ***FPRE*** to process the data with sk_{DSP} and DR j's public key pk_j:

$$[m]^{+} = \left(\hat{T}, \widehat{T'}\right) = \left\{T,\ \left(T'\right)^{a} * g^{H\left(\left(pk_j\right)^{a} || CID\right)}\right\}. \tag{4}$$

The DSP finally prepares a data packet $([m]^{+},\ CID)$ and sends it to the ACS.

Step 4 (Data Process @ ACS): The ACS calls the second re-encryption algorithm ***SPRE*** with sk_{ACS} to finally transfer the encrypted data to the ciphertext under DR j's public key:

$$[m]_{pk_j} = \ \left(\bar{T}, \overline{T'}\right) = \left\{\hat{T}, \left(\widehat{T'}\right)^{b} * g^{H\left(\left(pk_j\right)^{b} || CID\right)}\right\}. \tag{5}$$

Then the ACS sends $([m]_{pk_j},\ CID)$ to the DR.

Step 5 (Data Access @ DR): The DR can obtain the aggregated result by calling the algorithm ***DPRE***:

$$m = L\left(\bar{T} * PK_{ACS}{}^{H\left((pk_{DSP})^{sk_j} || CID\right)} * g^{H\left((pk_{ACS})^{sk_j} || CID\right)} / \overline{T'} \ mod\ n^2\right). \tag{6}$$

4.3.2. Subtraction

This operation aims to obtain the subtraction of some data $(m = \sum_{i=1}^{W} m_i - \sum_{i=W+1}^{N} m_i)$ with encrypted data $[m_i]$ $(i = 1, \ldots, N)$.

Step 3 (Data Preparation @ DSP): The DSP first computes $[\sum_{i=1}^{W} m_i] = \prod_{i=1}^{W} [m_i]$ and $[\sum_{i=W}^{N} m_i] = \prod_{i=W}^{N} [m_i]$. It further calculates $[-\sum_{i=\mathrm{W}}^{N} m_i] = ([\sum_{i=\mathrm{W}}^{N} m_i])^{n-1}$ and multiply them to obtain: $[m] = [(\sum_{i=1}^{W} m_i - \sum_{i=\mathrm{W}}^{N} m_i)] = [\sum_{i=1}^{W} m_i] * [-\sum_{i=W}^{N} m_i]$. Then the subsequent process is the same as that in *Addition*. Due to length and simplicity reasons, we skip its details.

4.3.3. Multiplication

This operation aims to obtain the product of all non-zero raw data ($m = \prod_{i=1}^{N} m_i$). For ease of presentation, we describe the details with two ciphertexts ($[m_1]$, $[m_2]$). The DR wants to get the multiplication result $m = m_1 * m_2$. Note that the available number of the data in *Multiplication* influences the length of raw data. If we need to get the product of N pieces of data, it must be guaranteed that the length of each raw data $\mathcal{L}(m_i) < \mathcal{L}(n)/(2N)$, which is different from *Addition*.

Step 3 (Data Preparation @ DSP): First, the DSP chooses two random numbers c_1 and c_2 and sets another one $c_3 = (c_1 * c_2)^{-1}$ *mod n*. The number of random numbers is equal to that of provided data.

To conceal each raw data from the ACS, the DSP first calls ***PDec*1** to decrypt ciphertexts and then encrypts c_3 with the ***Enc*** using pk_j of DR:

(1) $[c_i{*}m_i] = \{T_i^{c_i}, (T_i')^{c_i}\}$ for $i = 1,\ 2$;
(2) $[c_i{*}m_i]_{pk_{ACS}} = (T_i^{(1)}, T_i'^{(1)}) = \{T_i^{c_i}, (T_i')^{c_i*a}\} = \{(1 + c_i{*}m_i{*}n)PK^{r_i*c_i},\ g^{r_i*a*c_i}\}$ for $i = 1,\ 2$;
(3) $[c_3]_{pk_j} = (T_j, T_j') = \{(1 + c_3{*}n)pk_j^{r_3},\ g^{r_3}\}$.

The data packet sent to the ACS is $\{[c_1{*}m_1]_{pk_{ACS}}, [c_2{*}m_2]_{pk_{ACS}},\ [c_3]_{pk_j}\}$.

Step 4 (Data Process @ ACS): Upon receiving the data packet from the DSP, the ACS calls ***PDec2*** to obtain two values: $c_i * m_i = T_i^{(1)}/(T_i'^{(1)})^b$ $(i = 1,\ 2)$.

It further multiplies the two values and then calls the algorithm ***Enc*** to encrypt it as

$$\left(T, T'\right) = [c_1{*}c_2{*}m_1{*}m_2]_{pk_j}. \tag{7}$$

Finally, the ACS forwards (T, T') and $[c_3]_{pk_j}$ to the DR.'

Step 5 (Data Access @ DR): the DR can obtain the product by calling ***Dec*** to decrypt the two ciphertexts with its secret key:

$$m = m_1{*}m_2 = L\Big(T/\big(T'\big)^{sk_j} mod\ n^2\Big){*}\ L\Big(T_j/\big(T_j'\big)^{sk_j} mod\ n^2\Big) mod\ n. \tag{8}$$

4.3.4. Sign Acquisition

We assume that $\mathcal{L}(m) < \mathcal{L}(n)/4$ and that *BIG* is the largest raw data of m. Then the raw data is in the scope $[-BIG, BIG]$. DR j wants to know the sign of raw data m_1 from $[m_1]$.

Step 3 (Data Preparation @ DSP): The DSP chooses a random number c_1 where $\mathcal{L}(c_1) < \mathcal{L}(n)/4$. It first encrypts "1" and then computes as follows:

(1) $[1] = \{(1 + n)\ PK^{r'}, g^{r'}\}$;
(2) $[2{*}m_1 + 1] = (T, T') = [m_1]^2{*}[1] = \{(1 + (2{*}m_1 + 1){*}n)PK^{r'+2*r_1},\ g^{r'+2*r_1}\}$;
(3) Then it flips a coin s. If $s = 0$; it computes: $(T_1^{(1)}, T_1'^{(1)}) = \{T^{n-c_1}, (T')^{a*(n-c_1)}\} = [-c_1{*}(2{*}m_1 + 1)]$; otherwise, it calls the ***PDec*1** and computes: $(T_1^{(1)}, T_1'^{(1)}) = \{T^{c_1}, T'^{a*c_1}\} = [c_1{*}(2{*}m_1 + 1)]$;
(4) The DSP encrypts s with pk_j by calling the ***Enc***: $[s]_{pk_j} = (T_s, T_s') = \{(1 + s{*}n)pk_j^{r_s}, g^{r_s}\}$.

The data packet sent to the ACS is $\{(T_1^{(1)}, T_1'^{(1)}),\ [s]_{pk_j}\}$.

Step 4 (Data Process @ ACS): Upon receiving the data packet from the DSP, the ACS decrypts $(T_1^{(1)}, T_1'^{(1)})$ with the ***PDec2*** to obtain raw data $m' = (-1)^{s+1} * c_1 * (2 * m_1 + 1)$ *mod n*. The ACS compares $\mathcal{L}(m')$ with $\mathcal{L}(n)/2$. If $\mathcal{L}(m') < \mathcal{L}(n)/2$, it sets $u = 1$; otherwise, it sets $u = 0$. Then the ACS calls the ***Enc*** to encrypt u with pk_j as: $[u]_{pk_j} = (T_u, T_u')$. It further multiplies the two ciphertexts:

$[u + s]_{pk_j} = \left(\bar{T}, \overline{T'}\right) = \left\{T_s{*}T_u, T_u'{*}T_s'\right\}$. Finally, the ACS forwards $\left(\bar{T}, \overline{T'}\right)$ to DR j.

Step 5 (Data Access @ DR): DR j calls the ***Dec*** to obtain the final result: $u + s = L(\bar{T}/(\overline{T'})^{sk_j}\ mod\ n^2)$. Then DR j needs to check it and determine the sign of raw data: if $u + s = 1$, the original data is negative (i.e., $m_1 < 0$); otherwise, it is positive or zero (i.e., $m_1 \geq 0$).

Note: 1) $s = 1$, $m' = c_1{*}(2{*}m_1 + 1) mod\ n$: As $\mathcal{L}(c_1) < \mathcal{L}(n)/4$, $m_1 \in [0,\ BIG)$ if $\mathcal{L}(m') < \mathcal{L}(n)/2 (u = 1)$; $m_1 \in (-BIG,\ 0)$ if $\mathcal{L}(m') > \mathcal{L}(n)/2\ (u = 0)$. 2) $s = 0$, $m' = -c_1{*}(2{*}m_1 + 1)\ mod\ n$: As $c_1 \in [0,\ n/4]$, $m_1 \in [0,\ BIG)$ if $\mathcal{L}(m') > \mathcal{L}(n)/2$; $m_1 \in (-BIG,\ 0)$ if $\mathcal{L}(m') < \mathcal{L}(n)/2\ (u = 1)$. Hence, if $(s = 1,\ u = 0)$ or $(u = 1,\ s = 0)$, $m_1 < 0$; if $(s = 1,\ u = 1)$ or $(u = 0,\ s = 0)$, $m_1 \geq 0$.

4.3.5. Comparison

Similar to the operations above, DR j wants to compare the raw data $(m_1,\ m_2)$ based on their encrypted data. For ease of presentation, $m_1 - m_2$ is denoted as m_{1-2}.

Step 3 (Data Preparation @ DSP): DSP first computes to get the subtraction of encrypted data:

$$\left(T, T'\right) = \left\{T_1{*}(T_2)^{n-1}, T_1'{*}\big(T_2'\big)^{n-1}\right\} = [(m_1 - m_2)]. \tag{9}$$

The following steps are the same to that in *Sign Acquisition*, which is skipped for the reason of paper length limitation. Through the cooperation of the DSP and the ACS, the DR finally gets the sign of $m_{1-2} = m_1 - m_2$. In the end, the DR can obtain the comparison result. If $m_{1-2} \geq 0$, $m_1 \geq m_2$; otherwise, $m_1 < m_2$.

4.3.6. Equivalent test

DR j wants to know if m_1 is equal to m_2 with encrypted data ($[m_1]$, $[m_2]$). The DSP and the ACS directly interact with each other to perform two parallel computations of *Comparison*. They compare m_1 and m_2 in two forms: 1) $m_{1-2} = m_1 - m_2$; 2) $m_{2-1} = m_2 - m_1$. Through the operations in *Comparison*, DSP can get two computation results $[s_1 + u_1]_{pk_j}$ and $[s_2 + u_2]_{pk_j}$ respectively. To conceal the comparing result, $[s_1 + u_1]_{pk_j}$ and $[s_2 + u_2]_{pk_j}$ are sent to the DR in a random order. If both testing result are " $\geq$ ", we can know $m_1 = m_2$.

4.3.7. Variance

In some scenarios, DR j may want to get the variance of some data according to provided encrypted data. In this presentation, we set N be the number of provided data and $m = \sum_{i=1}^{N} m_i$. Variance operation can be presented as $M = \sum_{i=1}^{N} (m_i - \bar{m})^2/N = \sum_{i=1}^{N} (N * m_i - m)^2/N^3$, where $\bar{m}$ is the average of m_i $(i = 1, \ldots, N)$. For ease of presentation, we assume there are three pieces of encrypted data (i.e., $N = 3$, which is shared with DR j): $[m_1], [m_2]$ and $[m_3]$.

Step 3 (Data Preparation @ DSP): First, the DSP obtains $[N*m_i - \sum_{i=1}^{N} m_i]$ with following steps:

(1) $[m] = (T, T') = [m_1]*[m_2]*[m_3]$, $[-m] = (T^{n-1}, (T')^{n-1})$;
(2) $[N*m_i] = [m_i]^N$ for $i = 1, 2, 3$;
(3) $[N*m_i - m] = [m_i]^N*[-m]$ for $i = 1, 2, 3$;
(4) Partially decrypt the data by calling ***PDec*1** to obtain: $[N*m_i - m]_{pk_{ACS}}$ for $i = 1, 2, 3$.
(5) Choose three random numbers c_1, c_2, c_3, and compute to obtain: $[c_i(N*m_i - m)]_{pk_{ACS}} = ([N*m_i - m]_{pk_{ACS}})^{c_i}$ for $i = 1, 2, 3$.

Then the DSP sends the three ciphertexts to the ACS. In addition, the DSP stores ${c_1}^2$, ${c_2}^2$, and ${c_3}^2$.

Step 4 (Data Process @ ACS): Upon receiving the data from the DSP, the ACS directly decrypts to obtain raw data and then processes the data for DR j as follows:

(1) Decrypt to obtain: $C_i = c_i(N*m_i - m)$ for $i = 1, 2, 3$;
(2) Encrypt the processed data with pk_j: $[{C_i}^2]_{pk_j} = [{c_i}^2(N*m_i - m)^2]_{pk_j}$ for $i = 1, 2, 3$.

Then the ACS sends them back to the DSP.

Additional Operation @ DSP: The DSP first computes the reverse of ${c_i}^2 (i = 1, 2, 3)$ respectively: $c_i' = (c_i^2)^{-1} mod n^2$ for $i = 1, 2, 3$. Then the DSP can prepare the final result for DR j:

$$\begin{aligned}
[M']_{pk_j} &= \left([C_1^2]_{pk_j}\right)^{c_1'} * \left([{C_2}^2]_{pk_j}\right)^{c_2'} * \left([{C_3}^2]_{pk_j}\right)^{c_3'} \\
&= \prod_{i=1}^{3} \left[c_i' * {C_i}^2\right]_{pk_j} = \prod_{i=1}^{3} \left[c_i' * c_i^2 (3m_i - m)^2\right]_{pk_j} \\
&= \prod_{i=1}^{3} \left[(3m_i - m)^2\right]_{pk_j} = \left[(3m_1 - m)^2 + (3m_2 - m)^2 + (3m_3 - m)^2\right]_{pk_j} \\
&= \sum_{i=1}^{3} \left[(3m_i - m)^2\right]_{pk_j}.
\end{aligned} \tag{10}$$

Finally, $[M']_{pk_j}$ can be sent to DR j.

Step 5 (Data Access @ DR): DR j can obtain M' by calling ***Dec*** and then get the variance:

(1) $M' = \sum_{i=1}^{N} (N*m_i - m)^2$;
(2) Final variance is: $M = M'/N^3 = ((N*m_1 - m)^2 + (N*m_2 - m)^2 + (N*m_3 - m)^2)/N^3$.

5. Cross-ACS computations

Sometimes computations on data that belong to different ACSs are required. Hence, we propose a number of schemes to meet with this special requirement in this section.

5.1. Data processing over multiple ACSs

Due to paper length limitation, we only present such basic operations as addition, subtraction, multiplication, and comparison across the ACSs herein.

We set an example of two pieces of encrypted data belonging to two ACSs: ACS B and ACS V. Besides the settings above (e.g., the DSP with $(sk_{DSP}, pk_{DSP}) = (a, g^a \ mod \ n^2))$, we further set the key pairs of B and V as $(sk_B, pk_B) = (b, g^b \ mod \ n^2)$ and $(sk_V, pk_V) = (v, g^v \ mod \ n^2)$. Hence, we have $PK = {pk_B}^a = {pk_{DSP}}^b$ and $PK' = {pk_V}^a = {pk_{DSP}}^v$. Two messages are encrypted as:

$$[m_1]_{PK} = \left\{T_1 = (1 + m_1{*}n)PK^{r_1},\ T_1' = g^{r_1}\right\} \ mod \ n^2, \tag{11}$$

$$[m_2]_{PK'} = \left\{T_2 = (1 + m_1{*}n){PK'}^{r_2},\ T_2' = g^{r_2}\right\} \ mod \ n^2. \tag{12}$$

That is to say, the data provider of m_1 trusts ACS B; while the data provider of m_2 trusts ACS V. Hence, they encrypt their data with the corresponding Diffie–Hellman key (PK or PK'). DR j with key pair $(sk_j, pk_j) = (k_j, g^{k_j} \ mod \ n^2)$ wants to obtain the data processing result across ACSs. We assume DR j is a user of ACS B. The detailed procedure is introduced as follows.

5.1.1. Addition across ACSs

This computation wants to obtain the sum of data over two servers.

Setp 3 (Data Preparation @ DSP): DSP selects a random number o and then operates as follows:

(1) Encrypt o and $-o$: $[\mathrm{o}]_{PK}$ and $[-o]_{PK'}$;
(2) Compute $[m_1 + o]_{PK}$ and $[m_2 - o]_{PK'}$.

Then call the algorithm ***PDec*1** to decrypt the two data to obtain $[m_1 + o]_{pk_B}$ and $[m_2 - o]_{pk_V}$.

Step 4 (Data Process @ ACSs): Upon receiving $[m_1 + o]_{pk_B}$, ACS B first checks its CID and determines if the requester is allowed to access the data; if positive, ACS B calls the algorithm ***PDec*2** to obtain the fused raw data $m_1 + o$ and then encrypt it with DR j's public key as $[m_1 + o]_{pk_j}$. Similar to the operations of ACS B, ACS V also obtains $[m_2 - o]_{pk_j}$.

Additional Operation @ DSP: DSP multiplies the two ciphertexts to obtain $[m_1 + m_2]_{pk_j}$ and then forwards it to DR j.

Finally, DR j can directly get the sum of data $(m_1 + m_2)$ by calling the algorithm ***Dec***.

5.1.2. Subtraction across ACSs

The operation is similar to *Addition*, but it needs to do one more operation to obtain the negative of subtractor (e.g., m_2) first by doing exponentiation with the power of $(n - 1)$ to obtain $[-m_2]_{PK'}$ first.

5.1.3. Multiplication across ACSs

Different from *Multiplication* in Section 4.3.3, multiple ACSs are involved in the computation and leads to a little higher computation on the ACSs.

Setp 3 (Data Preparation @ DSP): The DSP selects two random numbers $(c_1,\ c_2)$ to conceal the raw data, and set $c_3 = (c_1{*}c_2)^{\ -1} \ mod \ n$. Then the DSP does the same operations about Multiplication as those described above and obtains:

(1) $[c_1{*}m_1]_{pk_B} = (\widehat{T_1}, \widehat{T_1'}) = \{{T_1}^{c_1}, (T_1')^{a*c_1}\} = \{(1 + c_1{*}m_1{*}n){*}PK^{r_1*c_1},\ g^{r_1*a*c_1}\}$;
(2) $[c_2{*}m_2]_{pk_V} = (\widehat{T_2}, \widehat{T_2'}) = \{{T_2}^{c_2}, (T_2')^{a*c_2}\} = \{(1 + c_2{*}m_2{*}n){*}PK'^{(r_2*c_2)},\ g^{r_2*a*c_2}\}$;
(3) $[c_3]_{pk_j} = (T_j, T_j') = \{(1 + c_3{*}n){*}{pk_j}^{r_3},\ g^{r_3}\}$.

The data packet sent to ACS B is $\{[c_1{*}m_1]_{pk_B},\ [c_3]_{pk_j}\}$; while the data $[c_2{*}m_2]_{pk_V}$ is sent to ACS V.

Step 4 (Data Process @ ACSs): Upon receiving the data package, the ACS first checks the legality and its access policy, and then calls the ***PDec*2** if it is positive. Concretely, ACS V obtains the value of $c_2 * m_2$, encrypts it with pk_B and then sends $[c_2 * m_2]_{pk_B}$ to ACS B. ACS B obtains the two plaintexts and multiplies them to get $c_1 * c_2 * m_1 * m_2$. Finally, ACS B encrypts $c_1 * c_2 * m_1 * m_2$ with the DR j's public key and sends it together with $[c_3]_{pk_j}$ to DR j.

Step 5 (Data Access @ DR): Upon obtaining the data form ACS B, the DR can directly calls the ***Dec*** to get $c_1 * c_2 * m_1 * m_2$ and c_3. Finally, it can get:

$$m = m_1{*}m_2 = c_1{*}c_2{*}m_1{*}m_2{*}c_3 \ mod \ n. \tag{13}$$

5.1.4. Comparison across ACSs

Different from the Comparison over one ACS, the initial operation is executed by the ACSs rather than the DSP. First, the DSP directly sends the data $[m_1]_{PK}$ and $[m_2]_{PK'}$ to ACS B and ACS V respectively.

Step 3 (Data Preparation @ ACSs): Owing to the ability of changing decryption order in Two-level, ACS V calls the ***PDec*1** to obtain $[m_2]_{PK_{DSP}}$ and then sends it to ACS B through a secure way.

ACS B first decrypts $[m_1]_{PK}$ to obtain $[m_1]_{pk_{DSP}}$ by calling the ***PDec*1** and then computes as follows:

(1) $[m_{1-2}]_{PK_{DSP}} = [m_1]_{pk_{DSP}}{*}([m_2]_{pk_{DSP}})^{n-1}$;
(2) $\{T,\ T'\} = [2{*}m_{1-2} + 1]_{pk_{DSP}} = \{[m_{1-2}]_{pk_{DSP}}\}^2{*}[1]_{pk_{DSP}}$;

(3) Then it flips a coin s. If $s = 0$; it computes $(\widehat{T_1}, \widehat{T_1'}) = \{T^{n-c_1}, (T')^{a*(n-c_1)}\}$; otherwise, it computes $(\widehat{T_1}, \widehat{T_1'}) = \{T^{c_1}, T'^{a*c_1}\}$.
(4) Finally, it encrypts s with pk_j: $[s]_{pk_j} = (T_s, T_s')$, and then sends $(\widehat{T_1}, \widehat{T_1'})$ and $[s]_{pk_j}$ to the DSP.

Step 4 (Data Process @ DSP): The DSP decrypts $(\widehat{\mathrm{T}_1}, \widehat{T_1'})$ to obtain raw data $m' = (-1)^{s+1} * c_1 * [2 * (m_1 - m_2) + 1] \bmod n^2$, and then compares its length with $\mathcal{L}(\mathrm{n})/2$. If $\mathcal{L}(\mathrm{m}') < \mathcal{L}(\mathrm{n})/2$, it sets $u = 1$; otherwise, it sets $u = 0$. Then it encrypts u with pk_j as $[u]_{pk_j} = (T_u, T_u')$, and further multiplies the two ciphertexts to obtain $[s+u]_{pk_j} = (\bar{T}, \overline{T'}) = \{T_s * T_u, T_s' * T_u'\}$.

Step 5 (Data Access @ DR): DR j can call the ***Dec*** to obtain the final result: $u + s = L(\bar{T}/(\overline{T'})^{sk_j}\ mod\ n^2)$. Then DR$j$ determines the sign of m_{1-2}. If $u + s = 1$, the original data is negative (i.e., $m_1 < m_2$); otherwise, it is positive or zero (i.e., $m_1 \geq m_2$).

Comparing with the scheme described in Section 4.3, we observe that the cross-ACS computation does not introduce too much overhead. For example, in *Addition*, the DSP needs to do some encryptions on random numbers, but it calls ***PDec*1** rather than ***SPRE***, which is more efficient. The ACS only needs to perform ***PDec*2** and ***Enc*** one more time. The computation cost of DR is low due to the high efficiency of ***Dec***.

6. Security analysis and performance evaluation

6.1. Security analysis

In this section, we analyze the security of the HRES and the PPDP as described in Section 4. We first introduce assumptions on the computational difficulty of several problems, based on which we prove the security of our proposed schemes.

6.1.1. Assumptions

Definition 5.1. Decisional Diffie–Hellman (DDH) [7] Problem over $\boldsymbol{Z}^*_{\boldsymbol{n}^2}$

For every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function *negl*() such that for sufficiently large ℓ:

$$\Pr\left[\mathcal{A}(n, X, Y, Z_b) = b \left| \begin{array}{l} p, q \leftarrow SP\left(\frac{\ell}{2}\right); n = pq; g \leftarrow \mathbb{G}; \\ x, y, z \leftarrow [1,\ ord(\mathbb{G})]; X = g^x mod\ n^2; \\ Y = g^y mod\ n^2; Z_0 = g^z mod\ n^2; \\ Z_1 = g^{xy} mod\ n^2; b \leftarrow \{0, 1\}; \end{array} \right.\right] - \frac{1}{2} = \mathrm{negl}(\ell). \tag{14}$$

6.1.2. Semantic security of HRES

In this section, we prove that the HRES is semantically secure against malicious DSP or ACS. We recall our assumption that the DSP would never collude with the ACS.

The basic encryption (***Enc***) and decryption (***Dec***) algorithms are directly obtained from [9], which has been proved to be semantically secure with the assumption of DDH problem. Hence, we omit its security proof. In the following section, we prove the security of the Two-Level Decryption and the Somewhat Re-Encryption in the HRES.

Theorem 1. *If the semantic security of EDD [9] holds, the proposed Two-Level Decryption in the HRES is semantically secure.*

Proof. We prove the theorem by contradiction. For this purpose, we assume that the HRES is not semantically secure. This means that there is a polynomial time distinguisher $\mathcal{A}$ that can break its semantic security. Our goal then is to use an adversary $\mathcal{A}$ to construct an algorithm $\mathcal{S}$ to break the semantic security of EDD.

Given the challenge public parameters $(n, g, h = pk_2 = g^{sk_2}\ mod\ n^2)$ of EDD, the adversary can construct public key in the HERS $pk_1 = h^{sk_1}$. Then once the adversary has chosen two messages of the same length m_0 and m_1, we flip the coin d and we encrypt m_d as follows: $E(m_d) = (T, T')$, where $T = h^r(1 + m_d n)\ mod\ n^2$ and $T' = g^r\ mod\ n^2$.

The adversary can simply compute $(\tilde{T}, \widetilde{T'})$ as follows:

(1) $\tilde{T} = (T)^{sk_1} = (h^{sk_1})^r(1 + sk_1 * m_d n)\ mod\ n^2 = (g^{sk_1 * sk_2})^r(1 + sk_1 * m_d n)\ mod\ n^2$,
(2) $\widetilde{T'} = T' = g^r$.

We can observe that $(\tilde{T}, \widetilde{T'})$ is one HRES ciphertext of $sk_1 * m_d$. The adversary can compute to obtain two raw data $m_0' = sk_1 * m_0$ and $m_1' = sk_1 * m_1$. We set $m_{d'} = sk_1 * m_d$. If the HRES is not semantically secure, the adversary can correctly guess the bit d'. Finally, it of course guesses the bit d and breaks the semantic security of EDD.

Theorem 2. *If the semantic security of EDD holds, then the proposed Somewhat Re-Encryption scheme in the HRES is semantically secure.*

Proof. Similar to the proof above, we assume an adversary $\mathcal{A}$ can break the semantic security of our scheme. For the sake of contradiction, our goal is to construct an algorithm $\mathcal{S}$ to break the semantic security of EDD.

Given challenge public parameters $(n, g, h = pk_2 = g^{sk_2} \ mod \ n^2)$ of EDD, the public key of authorized requester $pk_j = g^{k_j} \ mod \ n^2$ and CID, the adversary can construct public key $pk_1 = h^{sk_1}$. Then once the adversary has chosen two messages of the same length m_0 and m_1, we flip the coin d and we encrypt m_d as follows: $E(m_d) = (T, T')$, where $T = h^r(1 + m_d n) \ mod \ n^2$ and $T' = g^r \ mod \ n^2$.

Adversary $\mathcal{A}$ can simply compute $(\tilde{T}, \widetilde{T'})$ as follows:

(1) $\tilde{T} = (T)^{sk_1} = (h^{sk_1})^r(1 + sk_1{*}m_d n) \ mod \ n^2 = (g^{sk_1{*}sk_2})^r(1 + sk_1{*}m_d n) \ mod \ n^2$,
(2) $\widetilde{T'} = T' = g^r$.

Based on $(\tilde{T}, \widetilde{T'})$, $\mathcal{A}$ can further construct a re-encryption ciphertext as follows:

(1) $\bar{T} = \tilde{T} * g^{h_1} = (g^{sk_1 * sk_2})^r(1 + sk_1 * m_d n){*}g^{h_1} \ mod \ n^2$,
(2) $\overline{T'} = \widetilde{T'}$, where $h_1 = H((pk_j)^{sk_2}||CID)$.

We can observe that $(\bar{T}, \overline{T'})$ is one ciphertext of the HRES. Adversary $\mathcal{A}$ can compute to obtain two raw data $m_0^{'} = sk_1 * m_0$ and $m_1^{'} = sk_1{*}m_1$. We set $m_{d'} = sk_1{*}m_d$. If the HRES is not semantically secure, $\mathcal{A}$ can correctly guess the bit d'. Finally, it of course guesses the bit d and breaks the semantic security of EDD.

6.1.3. Security of PPDP

Here, we adopt the security model for securely realizing an ideal functionality in the presence of semi-honest (non-colluding) adversaries. For simplicity, we do it for the specific scenarios of our functionality, which involves four types of parties: DSP, ACS, DP and DR. First, we construct four simulators $Sim = (Sim_{DP}, Sim_{DSP}, Sim_{ACS}, Sim_{DR})$ to against four kinds of adversaries $(\mathcal{A}_{DP}, \mathcal{A}_{DSP}, \mathcal{A}_{ACS}, \ \mathcal{A}_{DR})$ that corrupt *DP, DSP, ACS* and *DR*, respectively.

Theorem 3. *The Addition scheme described in Section 4.3.1 can securely obtain the plaintext of addition via computations on ciphertexts in the presence of semi-honest (non-colluding) adversaries* $\mathcal{A} = (\mathcal{A}_{DR}, \mathcal{A}_{DSP}, \mathcal{A}_{ACS}, \mathcal{A}_{DP})$.

Proof. We present the construction of four independent simulators (Sim_{DP}, Sim_{DSP}, Sim_{ACS}, Sim_{DR}). Here, we prove the security of the case with two inputs (i.e., N = 2).

Sim_{DP} receives the input of m_1 and m_2, then it simulates $\mathcal{A}_{DP}$ as follows: it encrypts data m_1 as $[m_1] = EncTK(m_1)$, and encrypts data m_2 as $[m_2] = EncTK(m_2)$. Finally, it returns $[m_1]$ and $[m_2]$ to $\mathcal{A}_{DP}$, and outputs $\mathcal{A}_{DP}$'s entire view. The view of $\mathcal{A}_{DP}$ is the encrypted data. The views of $\mathcal{A}_{DP}$ in the real and the ideal executions are indistinguishable due to the security of the HRES mentioned above.

Sim_{DSP} simulates $\mathcal{A}_{DSP}$ as follows: it runs the ***EncTK*** on two randomly chosen numbers $\widetilde{m_1}$ and $\widetilde{m_2}$; then it multiplies $[\widetilde{m_1}]$ by $[\widetilde{m_2}]$; further it runs the ***FPRE*** to obtains $[\tilde{m}]^+$. Sim_{DSP} sends $[\tilde{m}]^+$ to $\mathcal{A}_{DSP}$. If $\mathcal{A}_{DSP}$ replies with $\perp$, Sim_{DSP} returns $\perp$. The view of $\mathcal{A}_{DSP}$ consists of the encrypted data it creates. Owing to the honest of challenge cloud users and the security of the HRES, $\mathcal{A}_{DSP}$ receives the same outputs both in real and ideal executions. Its views are distinguishable.

Sim_{ACS} simulates $\mathcal{A}_{ACS}$ as follows: it randomly chooses data $[m]^+$ and re-encrypts it with its secret key by calling the ***SPRE*** to obtain $[m]_{pk_j}$, and then sends it to $\mathcal{A}_{ACS}$. If $\mathcal{A}_{ACS}$ replies with $\perp$, Sim_{ACS} returns $\perp$. The view of $\mathcal{A}_{ACS}$ is also the encrypted data. In both the real and the ideal executions, $\mathcal{A}_{ACS}$ receives the output of encrypted data $[m]_{pk_j}$. The security in real world can be guaranteed by the security of the HRES. The views of $\mathcal{A}_{ACS}$ in the real and ideal executions are indistinguishable.

Sim_{DR} simulates $\mathcal{A}_{DR}$ as follows: (it cannot enquire for the challenge data) it randomly chooses data $[m']_{pk_j}$, and decrypts it to obtain m', and then sends it to $\mathcal{A}_{DP}$. If $\mathcal{A}_{DP}$ replies with $\perp$, Sim_{DR} returns $\perp$. The view of $\mathcal{A}_{DR}$ is the decrypted result. But in both the real and ideal executions, it is guaranteed by the semantic security of the HRES. The views are indistinguishable in both executions. No matter how many times the adversary accesses the simulator $\mathcal{A}_{DR}$, it is still difficult to obtain the original real data for the irrelevance between ciphertexts and the hardness of exhaustion attack.

The security proofs of *Multiplication, Sign Acquisition, Comparison*, and *Equivalent Test* are similar to that of the *Addition* under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{DR}, \mathcal{A}_{DSP}, \mathcal{A}_{ACS}, \mathcal{A}_{DP})$.

Theorem 4. *The Variance scheme described in Section 4.3.7 can securely obtains the variance of plaintext over outsourced encrypted data in the presence of semi-honest adversaries* $\mathcal{A} = (\mathcal{A}_{DR}, \mathcal{A}_{DSP}, \mathcal{A}_{ACS}, \mathcal{A}_{DP})$.

Proof. The four independent simulators Sim_{DP}, Sim_{DSP}, Sim_{ACS}, and Sim_{DR} are constructed as follows:

Sim_{DP} receives the input of m_1, m_2 and m_3, then it simulates $\mathcal{A}_{DP}$ as follows: it encrypts data m_1 as $[m_1] = EncTK(m_1)$, data m_2 as $[m_2] = EncTK(m_2)$ and data m_3 as $[m_3] = EncTK(m_3)$. Finally, it returns $[m_1]$, $[m_2]$ and $[m_3]$ to $\mathcal{A}_{DP}$, and outputs $\mathcal{A}_{DP}$'s entire view. The view of $\mathcal{A}_{DP}$ is the encrypted data. The views of $\mathcal{A}_{DP}$ in the real and the ideal executions are indistinguishable due to the security of the HRES mentioned above.

Sim_{DSP} simulates $\mathcal{A}_{DSP}$ as follows: it generates encryptions of the $\widetilde{m_1}$, $\widetilde{m_2}$ and $\widetilde{m_3}$ by running the ***EncTK*** on randomly chosen numbers $\widetilde{m_1}$, $\widetilde{m_2}$ and $\widetilde{m_3}$; then it computes to obtain $[-\tilde{m} \ ]$ where $\tilde{m} = \widetilde{m_1} + \widetilde{m_2} + \widetilde{m_3}$; further it computes $[N{*}\widetilde{m_i} - \tilde{m}]$ $(i = 1, \ 2, \ 3)$, and calls the ***PDec*1** to obtain $[c_i(N{*}\widetilde{m_i} - \tilde{m})]_{pk_{ACS}}(i = 1, \ 2, \ 3)$ by choosing three random numbers c_i $(i = 1, \ 2, \ 3)$. By accessing the simulator Sim_{ACS}, it receives the data $[{c_i}^2(N{*}\widetilde{m_i} - \tilde{m})^2]_{pk_j}(i = 1, \ 2, \ 3)$; finally, it obtains the encrypted data $[M]_{pk_j}$, where $M = \sum_{i=1}^{N} (N{*}m_i - m)^2$. The encrypted data $[c_i(N{*}\widetilde{m_i} - \tilde{m})]_{pk_{ACS}}(i = 1,\ 2,\ 3)$ and $[M]_{pk_j}$are sent to $\mathcal{A}_{DSP}$. If $\mathcal{A}_{DSP}$ replies with $\perp$, Sim_{DSP} returns $\perp$. The view of $\mathcal{A}_{DSP}$ consists of the encrypted data. Owing to the security of the HRES, $\mathcal{A}_{DSP}$ receives the same outputs both in real and ideal executions. Its views are distinguishable.

Sim_{CP} simulates $\mathcal{A}_{ACS}$ as follows: it generates encryptions $[{r_i}^2]_{PK_j}(i = 1,\ 2,\ 3)$ by running the ***Enc*** on randomly chosen r_i $(i = 1,\ 2,\ 3)$, and then sends the encrypted data and r_i $(i = 1,\ 2,\ 3)$ to $\mathcal{A}_{ACS}$. If $\mathcal{A}_{ACS}$ replies with $\perp$, Sim_{ACS} returns $\perp$. The view of $\mathcal{A}_{ACS}$ is the encrypted data. In both the real and the ideal executions, $\mathcal{A}_{ACS}$ receives the output of encrypted data $[{r_i}^2]_{pk_j}$. The security in real world can be guaranteed by the security of the HRES and the randomness of c_i $(i = 1,\ 2,\ 3)$ in ciphertexts. The views of $\mathcal{A}_{ACS}$ in the real and ideal executions are indistinguishable.

Sim_{DR} simulates $\mathcal{A}_{DP}$ as follows: (it cannot enquire for the challenging data) it randomly chooses data $[m']_{pk_j}$, decrypts it to obtain m', and then sends it to $\mathcal{A}_{DP}$. If $\mathcal{A}_{DP}$ replies with $\perp$, Sim_{DR} returns $\perp$. The view of $\mathcal{A}_{DP}$ is the decrypted result. But in both the real and the ideal executions, it is guaranteed by the semantic security of the HRES. The views are indistinguishable in both executions.

No matter how many times the adversary accesses the simulator $\mathcal{A}_{DR}$, it is still difficult to obtain the original real data due to the irrelevance between ciphertexts and the hardness of exhaustion attack.

6.2. Performance evaluations

In this section, we analyze the computation complexity and the communication overhead of our proposed schemes, including the HRES and the seven computing operation schemes. Further, we implemented them and tested their performances through simulations.

6.2.1. Computation complexity analysis

Herein, we analyze the computation complexity of our proposed HRES and the seven operations in the PPDP. As exponentiation operation is significantly more time-consuming than the addition and multiplication, we ignore the fixed numbers of additions and multiplications in our analysis. We regard the modular multiplication as the basic computing unit. As we choose $r \in [1,\ n/4]$, the modular exponentiation g^r needs at most n modular multiplications. Hence we set the computation complexity of modular exponentiation to be $\mathcal{O}(n)$ for a better presentation, where n is the basic public parameter for modular computations.

First, we concentrate on the computations of algorithms in the HRES, which lays the foundation of the proposed schemes.

The algorithm ***Enc*** involves two parts in its ciphertext. Each part needs a modular exponentiation. The ***EncTK*** has the same operation as the ***Enc***. The algorithm ***Dec*** simply does one exponentiation and one multiplication to obtain a plaintext. The algorithm***PDec***1 needs one exponentiation; while the ***PDec2*** needs one more multiplication. The algorithm ***FPRE*** needs three exponentiations and one hash computation; so does the ***SPRE***; while the ***DPRE*** has to do two hash functions and four exponentiations, two of which operate the power of a number with fixed length (because a hash output has only 160 bits).

Based on the analysis on the HRES, we further analyze the computation complexity of computing operation schemes. We hold the assumption that there are N pieces of provided data in *Addition, Subtraction, Multiplication*, and *Variance*.

6.2.1.1. Computation complexity of DSP. In *Addition*, the DSP needs to multiply a number of N ciphertexts one by one and then re-encrypts the above result with the algorithm ***FPRE***. Thus, its computation complexity is $\mathcal{O}(N)$ where N is the number of provided data. The operation in *Subtraction* is similar to that in *Addition*, but it has two more modular exponentiations to obtain the negative of subtractor with computation complexity $\mathcal{O}(n)$. Thus it has the computation complexity of $\mathcal{O}(N + n)$.

In *Multiplication*, the DSP needs to partially decrypt all data and do a corresponding number of exponentiations. Moreover, it encrypts one random number. Hence, it has to do $2N + 2$ modular exponentiations. Its computation complexity achieves $\mathcal{O}(N{*}n)$.

In *Sign Acquisition*, the DSP has to conceal the ciphertext with an exponentiation using a random number, and then partially decrypts it. It performs three modular exponentiations in total. In *Comparison*, the DSP first obtains the ciphertext of the difference of original data and then operates as *Sign Acquisition*, which needs altogether four modular exponentiations and one modular *Multiplication*. While in *Equivalent Test*, the DSP should execute *Comparison* twice. Hence, the computation complexities of the DSP are all $\mathcal{O}(n)$ in the above three schemes.

In *Variance*, it first multiplies the input data and then obtains its negative with two exponentiations. It totally needs $2N + 2$ exponentiations, and N partial decryptions in Step 3. In the last step, it should do additional N exponentiations. Its computation complexity results in $\mathcal{O}(N{*}n)$.

6.2.1.2. Computation complexity of ACS. In *Addition*, the ACS directly re-encrypts the data with the algorithm ***SPRE***. The operation in *Subtraction* is the same as that in *Addition*. Both *Addition* and *Subtraction* have the computation complexity of $\mathcal{O}(n)$ at the ACS.

In *Multiplication*, the ACS needs to partially decrypt all data to multiply the results and then encrypt it. Hence, it has to do $(N + 2)$ modular exponentiations. It results in the computation complexity of $\mathcal{O}(N{*}n)$.

In *Sign Acquisition*, the ACS has to obtain the plaintext with an exponentiation, and then encrypts it. It has three modular exponentiations in total. In *Comparison*, the ACS does the same operation. While in *Equivalent Test*, it should execute

Table 2
Computation complexity of entities in each scheme.

Roles	Schemes	Computations	Computation complexity
DSP	*Addition*	$3 * Exp + 1 * Hash + N * Mul$	$O(N + n)$
	Subtraction	$5 * Exp + 1 * Hash + N * Mul$	$O(N + n)$
	Multiplication	$(2N + 2) * Exp$	$O(N * n)$
	Sign Acquisition	$3 * Exp$	$O(n)$
	Comparison	$4 * Exp$	$O(n)$
	Equivalent Test	$8 * Exp$	$O(n)$
	Variance	$(4N + 2) * Exp$	$O(N * n)$
ACS	*Addition*	$3 * Exp + 1 * Hash$	$O(n)$
	Subtraction	$3 * Exp + 1 * Hash$	$O(n)$
	Multiplication	$(N + 2) * Exp$	$O(N * n)$
	Sign Acquisition	$3 * Exp$	$O(n)$
	Comparison	$3 * Exp$	$O(n)$
	Equivalent Test	$6 * Exp$	$O(n)$
	Variance	$2N * Exp$	$O(N * n)$
DR	*Addition*	$4 * Exp + 2 * Hash$	$O(n)$
	Subtraction	$4 * Exp + 2 * Hash$	$O(n)$
	Multiplication	$2 * Exp$	$O(n)$
	Sign Acquisition	$1 * Exp$	$O(n)$
	Comparison	$1 * Exp$	$O(n)$
	Equivalent Test	$2 * Exp$	$O(n)$
	Variance	$1 * Exp$	$O(n)$

Notes: N is the number of provided data; n is the public parameter; *Exp* stands for the modular exponentiation; *Mul* refers to the modular multiplication; *Hash* is the hash function.

Table 3
Communication overhead of each scheme.

Schemes	Communication overhead
Addition	$2 * 4\mathcal{L}(n)$
Subtraction	$2 * 4\mathcal{L}(n)$
Multiplication	$(N + 3) * 4\mathcal{L}(n)$
Sign Acquisition	$3 * 4\mathcal{L}(n)$
Comparison	$3 * 4\mathcal{L}(n)$
Equivalent Test	$6 * 4\mathcal{L}(n)$
Variance	$(2N + 1) * 4\mathcal{L}(n)$

Comparison twice. That is, it has six exponentiations. Hence, the computation complexity of the ACS is $\mathcal{O}(n)$ in all the above three schemes.

In *Variance*, the ACS needs to do N exponentiations to decrypt the data first and then encrypt them with the public key of DR. It totally has $2N$ exponentiations. Its computation complexity is $\mathcal{O}(N*n)$.

6.2.1.3. Computation complexity of DR. In both *Addition* and *Subtraction*, the DR can directly call the algorithm ***DPRE*** to obtain the final result, which needs four exponentiations. Its computation complexity is $\mathcal{O}(n)$. In other five schemes, the DR calls the ***Dec*** to decrypt the final result, which needs one exponentiation. A little difference is that the DR should do two decryptions in *Multiplication* and *Equivalent Test*. Its computation complexity is still $\mathcal{O}(n)$.

The computation complexity is summarized in Table 2.

6.2.2. Communication overhead

Each ciphertext is composed of two parts: $[m_i] = \{T_i, T_i^{'}\}$. It is highly related to the length of n^2, which has $2\mathcal{L}(n)$ bits. Hence, it has to transmit $4\mathcal{L}(n)$ bits for each ciphertext.

Further, we summarize the communication overhead of each scheme in Table 3 with the assumption of N pieces of provided data. Notably, our analysis excludes the communication cost of data provision by DPs as it cannot be circumvented in all schemes and can be amortized over various data analyses. We can observe that the communication overhead in *Addition* and *Subtraction* does not rely on the number of data providers. But it takes much higher communication overhead to run *Multiplication* and *Variance*, which is linearly related to the number of providers and will affect their applications in a scenario with big data. In addition, the communication overhead in other schemes does not vary much as it has only one or two data inputs. In general, the communication cost is reasonable and our schemes are suitable for various applications.

6.2.3. Experiment analysis

We further implemented the proposed seven computing operation schemes and tested their performances to check with our theoretic analysis. The evaluations are performed on a laptop with Intel Core i5-3337U CPU 1.8 GHz and 4GB RAM. To

Table 4
Parameter Settings.

Parameter	Value
$\mathcal{L}(m_i)$	100 bits
$\mathcal{L}(sk_i) = \mathcal{L}(sk_j)$	500 bits
$\mathcal{L}(sk_{DSP}) = \mathcal{L}(sk_{ACS})$	500 bits
$\mathcal{L}(CID)$	100 bits
$\mathcal{L}(n)$	1024 bits
Length of random numbers expressed with r	500 bits
Length of random numbers expressed with c	200 bits

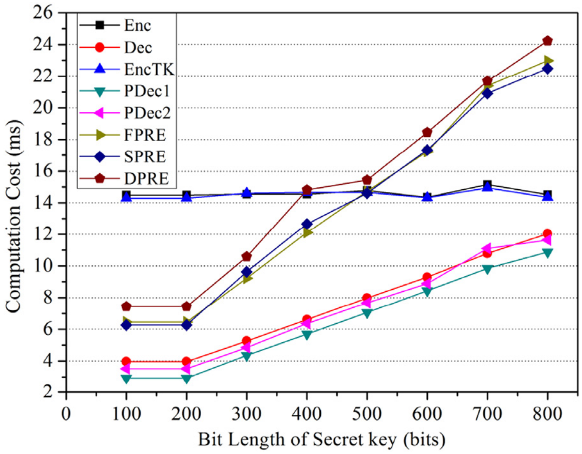

Fig. 2. Influence of the length of secret key on the HRES.

achieve better accuracy, we tested each algorithm 1000 times and reported the average value of all testing results. Unless particularly specified, the parameters in our tests are set as the default values listed in Table 4.

1) HRES Performance

Test 1: Influence of the length of secret key on the HRES

In this experiment, we tested the influence of the length of secret key on the algorithms of HRES. We set the bit length of all secret keys as 100 bits, 200 bits, 300 bits, 400 bits, 500 bits, 600 bits, 700 bits, and 800 bits. The performance of each algorithm in HRES is shown in Fig. 2.

We can observe that the secret key almost has no influence on the algorithms ***Enc*** and ***EncTK***. The computation costs of ***Enc*** and ***EncTK*** are similar, which conforms to the above computation analysis. Other algorithms need to do the modular exponentiations with the power of secret key. Therefore, their computation time grows up with the length of secret key size.

Moreover, we can get to know that the length of exponent affects efficiency. Hence, we fix the length of random numbers in the ***Enc*** and ***EncTK*** in the following tests in order to test the impact of other parameters on computation costs.

Test 2: Influence of the length of ***n on the HRES***

In this experiment, we tested the operation time of each algorithm in the HRES (including *Enc, Dec, EncTK, PDec*1, *PDec*2, *FPRE, SPRE* and *DPRE*) by adopting different module sizes ($\mathcal{L}(n)$: 256 bits, 512 bits, 768 bits, 1024 bits, 1280 bits, 1536 bits, 1792 bits, and 2048 bits). The simulation results are shown in Fig. 3.

We can observe that the length of n influences the efficiency seriously. The computation time of each algorithm increases as n increases, but on the other side bigger module guarantees higher security. We observe that the computation time of each algorithm is less than 20 milliseconds (ms) when n is 1024 bits that means the module is 2048 bits. In particular, the decryption is much more efficient than other operations, which is suitable for DRs with limited resources. It implies that the algorithm is efficient to be applied into a mobile domain where DRs are played by mobile devices with limited resources.

In addition, we can further prove the correctness of our implementation and analysis by comparing the simulation results with performance analysis. We can observe that the computation cost of the ***DPRE*** is the highest, which conforms to the analysis that it needs four exponentiations. Moreover, we can find that the ***DPRE*** does not consume much longer time than both the ***FPRE*** and the ***SPRE*** do. It is because the ***DPRE*** only needs to do one more exponentiation of 160-bit operator, which is very efficient. In addition, it can be observed that the two-level decryptions need one exponentiation and are the most efficient. In general, the simulation results conform to the theoretic analysis.

Test 3: Influence of the length of input data on the HRES

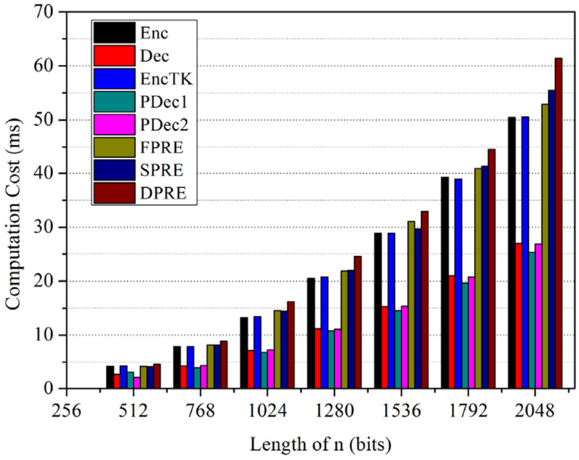

Fig. 3. Computation time of each algorithm in the HRES with various n.

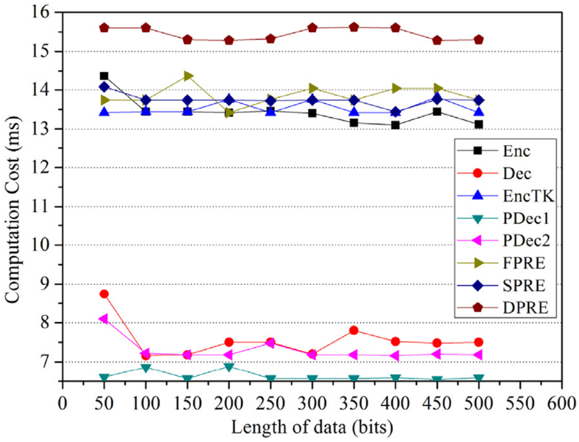

Fig. 4. Computation time of each algorithm in the HRES with various length of data.

We further tested the flexibility of the HRES with different length of original input data m_i, which is set to ten different values: 50 bits, 100 bits, 150 bits, 200 bits, 250 bits, 300 bits, 350 bits, 400 bits, 450 bits, and 500 bits.

We can observe from Fig. 4 that the computation time of each algorithm does not vary too much with the length of original input data. Therefore, the HRES can easily deal with various data.

The simulation results above present the practical potential of the HRES to support privacy-preserving data processing.

2) Efficiency

Test 4: Influence of length of ***n***

Besides the tests on the HRES, we further tested the efficiency of the following schemes: *Addition* (Add), *Subtraction* (Subt.), *Multiplication* (Multi.), *Sign Acquisition* (Sign), *Comparison* (Comp.), *Equivalent Test* (Equal), and *Variance* (Vari.).

As DPs only need to provide their data using PK, its computation efficiency is only affected by the length of n, shown in Fig. 3. Hence, our further analyses focus on the other three parties: DSP, ACS and DR.

The computation costs of the DSP in the seven computing operation schemes are shown in Fig. 5. As the DSP needs to process data twice in *Variance*, we mark it as Vari. (1) and Vari. (2). We can see that the *Equivalent Test* and *Variance* are time-consuming, while this is acceptable for a cloud server with sufficient resources. From Fig. 6 that shows the computation cost of the ACS, we can observe that the most time-consuming operations are also *Equivalent Test* and *Variance*.

By comparing Fig. 6 with Fig. 5, we find that the computation cost in the ACS is much lower than that in the DSP, which makes the ACS suitable to be run by a company or an institute. Further, Fig. 7 shows the computation time of the DR. Obviously, most computation overhead has been transferred to DSP and ACS, which fits the requirement of low computation cost for cloud users. It only takes less than 10 ms in the following schemes: *Sign Acquisition, Comparison*, and *Variance*.

In general, the above tests prove that our schemes have transferred the most time-consuming operations from the DR to the DSP and the ACS, which further proves the practical potential of the proposed schemes in mobile environments.

3) Flexibility of Addition, Subtraction and Multiplication

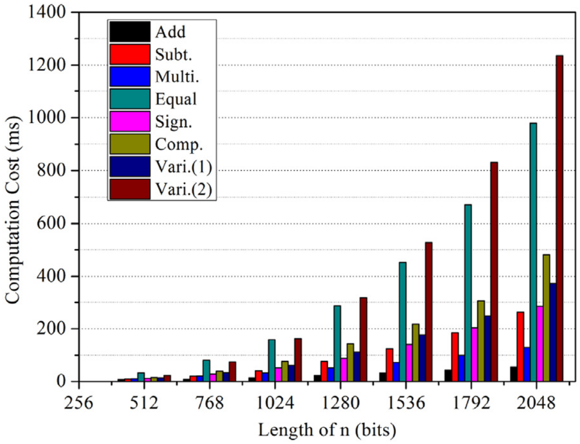

Fig. 5. Computation time of DSP in each algorithm with various *n*.

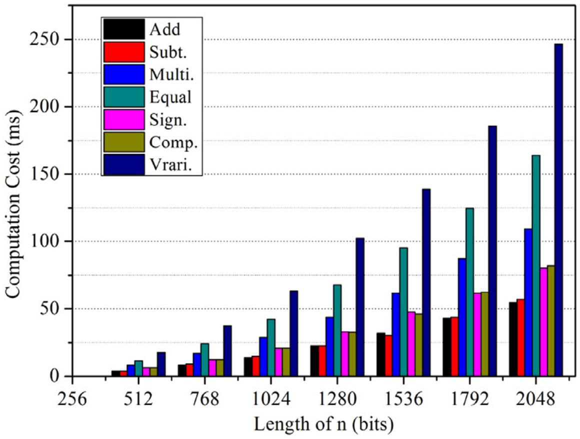

Fig. 6. Computation time of ACS in each algorithm with various *n*.

Test 5: Performance of Addition and Subtraction with a large number of provided data

In this test, we tested the performance of three kinds of system entities (i.e., DSP, ACS and DR) in *Addition* with different numbers of provided data ($N = 10,\ 10^2, 10^3,\ 10^4,\ 10^5$). The computation costs are shown in Fig. 8. Apparently, the computation cost of the DSP increases with the number of provided data, while it does not influence the DR and the ACS. Even though the number of provided data reaches 10^4, the DSP still takes less than 1000 ms for executing its operations. This fact implies that our schemes can deal with a great number of data for addition operation efficiently.

In the test of *Subtraction*, we assume the computation formula is $(\sum_{i=1}^{W} m_i - \sum_{i=W+1}^{N} m_i)$ with $= [N/2]$, that is, half of the provided data is subtracted from the sum of another half of data. The simulation result is shown in Fig. 9. The performance is similar to that of *Addition*.

Test 6: Performance of Multiplication with a large number of provided data

Multiplication is time-consuming. We only tested it with various numbers of provided data ($N = 100,\ 200,\ 300,\ 400,\ 500,\ 600,\ 700,\ 800$).

We can observe from Fig. 10 that it costs about 5 s to finish the computation in the DSP when the number of data is about 500. However, the computation time of the DR keeps unchanged. *Multiplication* does not introduce extra overhead to the DR. The above result implies that our scheme is practically suitable for cloud users that hold mobile devices with limited resources.

6.3. Comparison with existing work

We further compare the proposed PPDP system with some latest work based on different techniques: SMC [18], FHE [32], and PHE [21], in terms of the requested number of servers, their support on big data processing, flexible data access control, and flexible data management, and resistance to eversdropping. The comparison result is shown in Table 5.

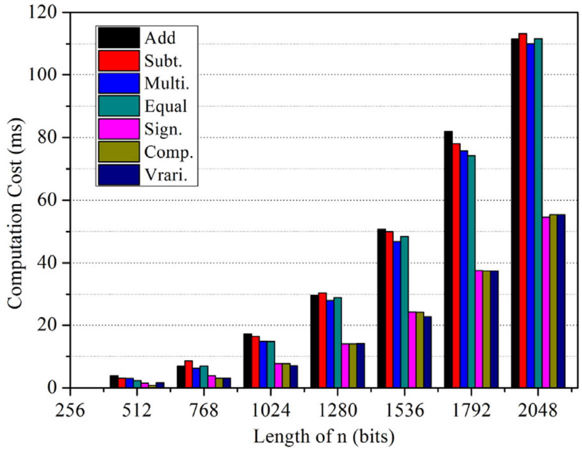

Fig. 7. Computation time of DR in each algorithm with various *n*.

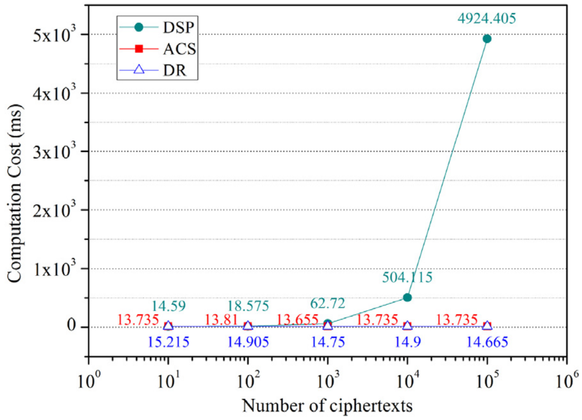

Fig. 8. Computation time of each entity in *Addition* with different number of DPs.

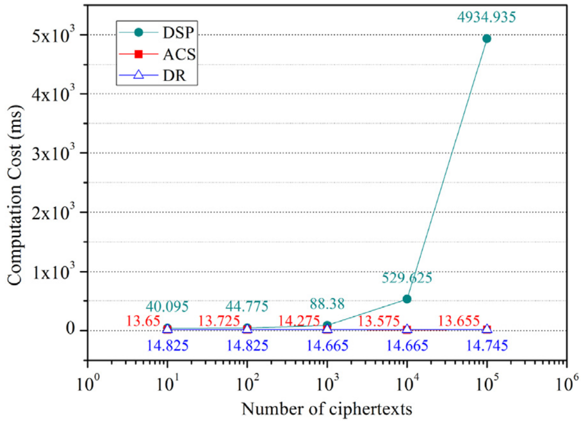

Fig. 9. Computation time of each entity in *Subtraction* with different number of DPs.

We can see that the SMC-based scheme [18] needs three servers to achieve secret sharing, while the PHE-based scheme [21] depends on two servers to realize ciphertext operation. The FHE-based scheme [33] can complete the arbitrary computations with only one server. With respect to the requested number of servers, the PPDP system that needs two servers sits in the middle level of system complexity. But the FHE-based scheme cannot support flexible data access control over multiple data requesters because all data are provided by encrypting them with an indicated public key, while other three schemes support this feature. The SMC-based scheme achieves flexible access control to the processing result by getting all

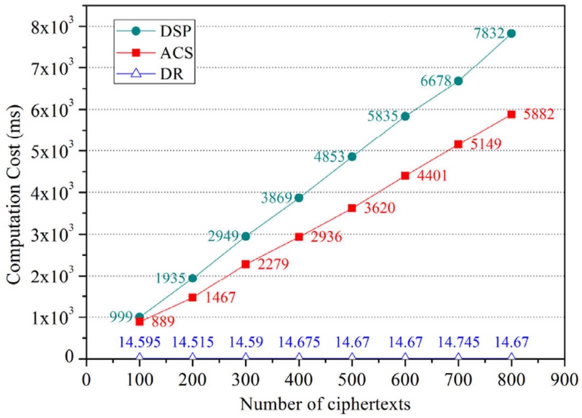

Fig. 10. Computation time of each entity in *Multiplication* with different number of DPs.

Table 5.
Comparison with existing work.

Schemes	SMC-based scheme [18]	FHE-based scheme [33]	PHE-based scheme [21]	PPDP
Number of servers	3	1	2	2
Flexible data access control	√	×	√	√
Flexible data management	×	×	×	√
Resistance to eavesdropping	×	√	√	√
Big data processing	×	×	×	√

Notes: ×- not supported; √ - supported.

shared and pre-processed values from its servers. But the SMC-based scheme weakens data management flexibility because it highly depends on the cooperation of three servers. The PHE-based scheme and the FHE-based scheme cannot support data management flexibility either because they cannot manage data that are controlled by different parties with different levels of trust evaluated by different users. The SMC-based scheme also suffers from eavesdropping since the processing result can be re-constructed from transmitted data. However, other schemes based on FHE and PHE can prevent unauthorized access. Although the PHE-based scheme [21] claimed to support multiparty access, it closely depends on the two claimed servers. Its multiplication operation is similar to Sharemind, which is achieved with 3^N multiplications of shared secrets. Thus, this scheme cannot support big data processing, which is not considered either in the SMC-based scheme. FHE-based schemes can theoretically support big data processing, but this cannot be practically implemented at present due to extremely high computation and storage costs.

Our PPDP system supports the cross-ACS computation over ciphertexts. Each user can choose a service according to its own willingness and decision. Thus, it can support flexible data management, meet the security demands of users and satisfy the design requirements of many real applications. Moreover, two non-colluding servers cooperate to support the flexible data access. The adoption of PHE guarantees the resistance to multiple attacks, such as eavesdropping. In particular, our design also satisfies the demand of big data processing, as demonstrated in performance evaluation. In general, the PPDP system has superiority that surpasses other existing work, thus it is more feasible to be applied in practice.

7. Conclusion

In this paper, we proposed the PPDP system to achieve seven basic operations over ciphertexts and flexibly control access to ciphertext processing results by employing the HRES. Moreover, we extended the system to support ciphertext computations across multiple ACSs. We implemented the proposed schemes and evaluated their security and performance. The results showed that the PPDP system is secure, efficient and effective with regard to big data processing.

In the future, we aim to make the PPDP system support more operations and realize group access control to processed ciphertexts.

Acknowledgments

This work is sponsored by the National Key Research and Development Program of China (grant 2016YFB0800704), the NSFC (grants 61672410 and U1536202), the Project Supported by Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2016ZDJC-06), the PhD grant of the Ministry of Education, China (grant 20130203110006), the China 111 project (grants B08038 and B16037), and Aalto University.

References

[1] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, R. Sirdey, Recent advances in homomorphic encryption: a possible future for signal processing in the encrypted domain, IEEE Signal Process Mag. 30 (2013) 108–117.

[2] G. Ateniese, K. Fu, M. Green, S. Hohenberger, Improved proxy re-encryption schemes with applications to secure distributed storage, ACM Trans. Inf. Syst. Secur. 9 (2006) 1–30.

[3] E. Ayday, J.L. Raisaro, J.-P. Hubaux, J. Rougemont, Protecting and evaluating genomic privacy in medical tests and personalized medicine, in: 12th ACM Workshop on Workshop on Privacy in the Electronic Society, ACM, 2013, pp. 95–106.

[4] A. Belle, R. Thiagarajan, S. Soroushmehr, F. Navidi, D.A. Beard, K. Najarian, Big data analytics in healthcare, BioMed Res. Int. 2015 (2015).

[5] D. Bogdanov, Sharemind: Programmable Secure Computations with Practical Applications, Doctoral dissertation, 2013.

[6] D. Bogdanov, R. Talviste, J. Willemson, Deploying secure multi-party computation for financial data analysis, in: Financial Cryptography and Data Security, Springer, 2012, pp. 57–64.

[7] D. Boneh, The decision Diffie-Hellman problem, in: Algorithmic Number Theory, Springer, 1998, pp. 48–63.

[8] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) fully homomorphic encryption without bootstrapping, in: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ACM, 2012, pp. 309–325.

[9] E. Bresson, D. Catalano, D. Pointcheval, A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications, in: Advances in Cryptology-ASIACRYPT 2003, Springer, 2003, pp. 37–54.

[10] C. Castelluccia, A.C. Chan, E. Mykletun, G. Tsudik, Efficient and provably secure aggregation of encrypted data in wireless sensor networks, ACM Trans. Sensor Netw. 5 (2009) 20.

[11] T.-H.H. Chan, E. Shi, D. Song, Privacy-preserving stream aggregation with fault tolerance, in: Financial Cryptography and Data Security, Springer, 2012, pp. 200–214.

[12] J.H. Cheon, J.-S. Coron, J. Kim, M.S. Lee, T. Lepoint, M. Tibouchi, A. Yun, Batch fully homomorphic encryption over the integers, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2013, pp. 315–335.

[13] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in: Advances in Cryptology, Springer, 1985, pp. 10–18.

[14] C. Gentry, Computing arbitrary functions of encrypted data, Commun. ACM 53 (2010) 97–105.

[15] C. Gentry, Fully homomorphic encryption using ideal lattices, in: 47th ACM Symposium on Theory of Computing (STOC), 2009, pp. 169–178.

[16] M. Joye, B. Libert, A scalable scheme for privacy-preserving aggregation of time-series data, in: Financial Cryptography and Data Security, Springer, 2013, pp. 111–125.

[17] S. Kamara, K. Lauter, Cryptographic cloud storage, in: Financial Cryptography and Data Security, Springer, 2010, pp. 136–149.

[18] L. Kamm, J. Willemson, Secure floating point arithmetic and private satellite collision analysis, Int. J. Inf. Secur. 14 (2015) 531–548.

[19] Q. Li, G. Cao, Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error, in: Privacy Enhancing Technologies, Springer, 2013, pp. 60–81.

[20] Q. Li, G. Cao, T. La Porta, Efficient and privacy-aware data aggregation in mobile sensing, IEEE Trans. Dependable Secure Comput. 11 (2014) 115–129.

[21] X. Liu, R. Choo, R. Deng, R. Lu, J. Weng, Efficient and privacy-preserving outsourced calculation of rational numbers, IEEE Trans. Dependable Secure Comput. (2016) 1-1.

[22] X. Liu, R.H. Deng, K.K.R. Choo, J. Weng, An efficient privacy-preserving outsourced calculation toolkit with multiple keys, IEEE Trans. Inf. Forensics Secur. 11 (2016) 2401–2414.

[23] L. Morris, Analysis of partially and fully homomorphic encryption, 2013. http://gauss.ececs.uc.edu/Courses/c6056/pdf/homo-outline.pdf.

[24] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Advances in Cryptology—EUROCRYPT'99, Springer, 1999, pp. 223–238.

[25] A. Peter, E. Tews, S. Katzenbeisser, Efficiently outsourcing multiparty computation under multiple keys, IEEE Trans. Inf. Forensics Secur. 8 (2013) 2046–2058.

[26] E. Shi, T.H. Chan, E. Rieffel, R. Chow, D. Song, Privacy-preserving aggregation of time-series data, in: 18th Annual Network and Distributed System Security Symposium (NDSS), 2011, pp. 1–17.

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

[28] J.J. Stephen, S. Savvides, R. Seidel, P. Eugster, Practical confidentiality preserving big data analysis, 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14), 2014.

[29] M. Van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully homomorphic encryption over the integers, in: Advances in Cryptology–EUROCRYPT 2010, Springer, 2010, pp. 24–43.

[30] M. Van Dijk, A. Juels, On the impossibility of cryptography alone for privacy-preserving cloud computing, HotSec 10 (2010) 1–8.

[31] Z. Wan, J.E. Liu, R.H. Deng, HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing, IEEE Trans. Inf. Forensics Secur. 7 (2012) 743–754.

[32] B. Wang, M. Li, S.S. Chow, H. Li, A tale of two clouds: computing on data encrypted under multiple keys, in: 2014 IEEE Conference on Communications and Network Security (CNS), IEEE, 2014, pp. 337–345.

[33] W. Wang, Y. Hu, L. Chen, X. Huang, B. Sunar, Exploring the feasibility of fully homomorphic encryption, IEEE Trans. Comput. 64 (2015) 698–706.

[34] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, A.V. Vasilakos, Security and privacy for storage and computation in cloud computing, Inf. Sci. 258 (2014) 371–386.

[35] Z. Yan, W. Ding, V. Niemi, A.V. Vasilakos, Two schemes of privacy-preserving trust evaluation, Future Gen. Comput. Syst. 62 (2015) 175–189.

[36] Z. Yan, P. Zhang, A.V. Vasilakos, A survey on trust management for Internet of Things, J. Netw. Comput. Appl. 42 (2014) 120–134.

[37] F. Zhang, X. Ma, S. Liu, Efficient computation outsourcing for inverting a class of homomorphic functions, Inf. Sci. 286 (2014) 19–28.

Wenxiu Ding received her B.Eng. degree in information security from Xidian University, Xi'an, China in 2012. She was the research assistant at the School of Information Systems, Singapore Management University from 2015 to 2016. Now, she is pursuing her Ph.D. degree in information security from the School of Telecommunications Engineering at Xidian University. Her research interests include RFID authentication, privacy preservation, data mining and trust management.

Zheng Yan is currently a professor at the Xidian University, Xi'an, China and a visiting professor at the Aalto University, Espoo, Finland. Her research interests are in trust, security and privacy. Prof. Yan serves as an associate editor of Information Sciences, Information Fusion, IEEE Internet of Things Journal, JNCA, IEEE Access, Security and Communication Networks etc. in total of nine journals, a steering committee, organization and program committee member of numerous international conferences. She is a senior member of the IEEE.

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 is a fellow of the IEEE.