A Tale of Two Clouds: Computing on Data Encrypted under Multiple Keys

Boyang Wang†, Ming Li†, Sherman S. M. Chow* and Hui Li§
† Department of Computer Science, Utah State University, Logan, UT, USA
* Department of Information Engineering, Chinese University of Hong Kong, Hong Kong
§ State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, Shaanxi, China
bywang.usu@gmail.com, ming.li@usu.edu, sherman@ie.cuhk.edu.hk, lihui@mail.xidian.edu.cn

***Abstract*—Cloud computing provides a convenient platform for big data computation such as machine learning and data mining. However, privacy conscious users often encrypt their data with their own keys before uploading them to the cloud. Existing techniques for computation on encrypted data are either in the single key setting or far from practical. In this paper, we show how two non-colluding servers can leverage proxy re-encryption to jointly compute arithmetic functions over the ciphertexts of multiple users without learning the inputs, intermediate or final results. Moreover, the computation is non-interactive to users and only requires minimal server-to-server interactions. Experimental results demonstrate that our schemes significantly improve the efficiency of outsourced computation when compared to the existing approach.**

I. Introduction

With the emergence of cloud computing, vast amounts of data are outsourced and stored in the cloud by users. Computing on these cloud data will benefit users in many ways. For instance, Immersion [1], a recent application developed by the MIT Media Lab, computes a social graph based on a user's data from Gmail, so that this user can have a better overview of her personal and professional email life. It is a new vision to perform computation not only on personal data, but also over cloud data contributed by multiple users. For example, utilizing machine learning techniques on e-health records from multiple users can improve the accuracy of disease diagnosis [2]; leveraging data mining methods on huge amounts of emails owned by multiple users helps spam detection [3].

Due to the security threats to the cloud, users still have a huge concern about the privacy leakage of their outsourced data and computation [2], [3]. As a result, users should encrypt their confidential data while storing and evaluating these data in the cloud. Generally speaking, users do not trust others easily, and they encrypt their data with different keys [4]. This is desirable in terms of privacy protection since each user can securely store and retrieve cloud data without revealing confidential data to other users or public cloud providers.

However, it brings new challenges to outsourced computation over cloud data contributed by multiple users. It is because most of the existing approaches of secure outsourced computation over cloud data only focus on the single key setting, such as Fully Homomorphic Encryption (FHE) [5]–[7] and Yao's Garbled Circuit [8], [9], which are not suitable for multiple keys. Different from these approaches, López et al. [4] recently proposed an FHE under multiple keys, where the inputs, intermediate and final results of outsourced computation can be preserved from the public cloud. Unfortunately, it requires heavy interactions among all the users during the decryption of the final result, and its efficiency is still similar as single-key FHE [5], which is far from practice.

Moving a step forward, Peter et al. [10] recently proposed a scheme in the *two non-colluding servers* setting (denoted by PTK in this paper) utilizing Bresson-Catalano-Pointcheval (BCP) encryption [11], an additively homomorphic encryption with double trapdoor decryption. In their scheme, the ciphertexts encrypted under different keys are first transformed into ciphertexts under a common key via server-to-server interactions before any computation. Then, the computation of *arithmetic functions* (i.e., functions can be represented with the combination of additions and multiplications) is performed by the two servers interactively. However, the complex interactions between the two non-colluding servers significantly hinder efficient computation. Therefore, it is fair to say that achieving efficient privacy-preserving outsourced computation over cloud data encrypted under multiple keys remains open.

In this paper, we tackle the above problem by leveraging the combination of two non-colluding servers and ElGamal-based Proxy Re-Encryption (PRE) [12]. Due to different versions of ElGamal encryption (i.e., additively homomorphic and multiplicatively homomorphic), we can build two schemes (named Banana^{+} and Banana*, respectively) with the same designing methodology. With our schemes, each user is able to encrypt data using her own public key and store encrypted data in the cloud. Then, she can freely retrieve and decrypt her outsourced data later without compromising data privacy. Meanwhile, arithmetic functions can be efficiently evaluated on multiple users' encrypted cloud data without revealing the inputs, intermediate or final results to the two non-colluding servers.

In addition, our schemes are scalable and efficient for users. Specifically, users do not need to share any secret; and they do not need to know the identities of other users or the total number of users involved in computation. Most importantly, the computation is *non-interactive* to users — users only need to outsource encrypted data initially and remain offline until retrieving encrypted outputs. Since it has been proven that the traditional single server model for secure outsourced computation cannot completely eliminate interactions between the user side and the server side (due to the impossibility of program obfuscation [13]), we believe the model of two non-colluding servers is a promising approach to totally free

978-1-4799-5890-0/14/$31.00 ©2014 IEEE

users from interactions in secure outsourced computation. This assumption of two non-colluding servers has also been recently leveraged in secure computation over encrypted data (under a single key) in many applications, such as searchable encryption [14] and biometric authentication [15], in order to achieve a better balance between efficiency and security.

The essential idea of our design is to utilize ElGamal-based PRE to transform ciphertexts encrypted under multiple keys into ciphertexts under the same key before computation. After the transformation, the two basic arithmetic operations (i.e., additions and multiplications) over ciphertexts are evaluated differently. Specifically, one of them is independently computed by one server, and another type of the two basic operations over ciphertexts is evaluated by server-to-server (*denoted as S-to-S in short*) interactions with extra blinding techniques (as described in Fig. 1).

Compared to PTK [10], which is also based on two non-colluding servers, our schemes do not require any server-to-server interactions during the transformation of ciphertexts (as shown in Table I), which can significantly improve the efficiency of outsourced computation over cloud data under multiple keys. Moreover, in contrast to PTK, both of our schemes can prevent collusion between any user and cloud server (i.e., $\mathcal{C}$), and one of our schemes (i.e., Banana^{+}) is even secure under the collusion between any user and evaluation server (i.e., $\mathcal{E}$). As a necessary trade-off, Banana^{+} is efficient only with small message size. Our recent approach [16] can reduce server-to-server interactions as well compared to PTK, but it fails to resist any type of user-server collusion.

TABLE I
COMPARISON AMONG OUR SCHEMES AND PTK [10]

	PTK	Banana^{+}	Banana*
S-to-S Interactions in ***TranEnc***	$O(n)$	*null*	*null*
S-to-S Interactions in ***Add***	*null*	*null*	$O(1)$
S-to-S Interactions in ***Mul***	$O(1)$	$O(1)$	*null*
S-to-S Interactions in ***TranDec***	$O(n)$	$O(n)$	$O(n)$
User-Server $\mathcal{C}$ Collusion Resistance	×	√	√
User-Server $\mathcal{E}$ Collusion Resistance	×	√	×
Efficient in Large Message Size	√	×	√

Another advantage of our design is that we can provide *flexible* choices for different types of arithmetic functions compared to PTK. Concretely, if a function requires a larger number of additions than multiplications, we can achieve a better performance with Banana^{+} by minimizing the total server-to-server interactions in the entire computation task. On the other hand, if the computation is dominated by multiplications, we can choose Banana* instead. By evaluating a machine learning application [17] experimentally, we will show that our schemes are more efficient than PTK, in terms of both communication and computation overhead.

II. Problem Statement

System Model. In our system model (Fig. 1), there are n users $\{U_1, \cdots, U_n\}$, Cloud Server $\mathcal{C}$, and Evaluation Server $\mathcal{E}$. Each user would like to outsource her data storage to the cloud, so that she can freely access her own outsourced data via the cloud. In addition, outsourced computation need to be performed over multiple users' cloud data to reveal the output of a function $f(m_1, \cdots, m_n)$ to each user, where m_i represents the individual data of user U_i. Cloud Server $\mathcal{C}$ is able to provide storage services to users. Cloud Server $\mathcal{C}$ and Evaluation Server $\mathcal{E}$ together are able to provide computation services to users.

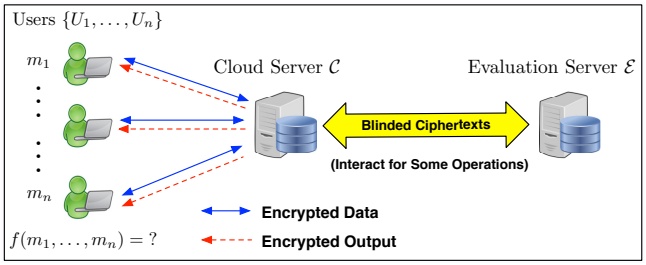


Fig. 1. System model: multiple users and two non-colluding servers

Threat Model. In this paper, we assume Cloud Server $\mathcal{C}$ and Evaluation Server $\mathcal{E}$ are both semi-honest (i.e., honest-but-curious). Meanwhile, we also assume these two servers are non-colluding. It means that neither of these two servers intends to corrupt users' data or computation process to prevent users from utilizing data correctly, but each server will try to learn the content of users' data (i.e., inputs), intermediate or final results of the computation without colluding with another server. A user may also be curious, and try to learn the content of other users' data stored in the cloud independently, or even collude with one of the servers sometimes.

Why Multiple Keys? In our model, each user encrypts her data with her own public key, so that she can easily retrieve and use her outsourced data from the cloud while protecting data privacy. Cloud data contributed by multiple users are encrypted under multiple keys, and the evaluation on those data is corporately operated by $\mathcal{C}$ and $\mathcal{E}$ together. Some recent approach [18], which also contains multiple users and two non-colluding servers, only needs all the users to encrypt data with a common public key (e.g., the public key of $\mathcal{E}$) for achieving secure outsourced computation.

This approach seems to be simpler than ours in terms of outsourced computation (no need of transformation on ciphertexts). Unfortunately, each data user in this approach [18] cannot retrieve and utilize her own data from the cloud after outsourcing autonomously. It is because none of the users in [18] is allowed to have the knowledge of the corresponding private key (e.g., the private key of $\mathcal{E}$) for decryption; otherwise, a user could easily reveal all other users' data. Moreover, the final result of outsourced computation in [18] is inevitably revealed as public information, while our schemes under multiple keys can still keep it private to each user.

Real-and-Ideal Model. In this paper, we analyze the security of our schemes under the Real-and-Ideal paradigm [19], which has been widely used in secure computation [4], [20] and searchable encryption [21], [22].

Specifically, in the *ideal world*, the computation of function f can be performed by an imaginary trusted party with input m_i from each user U_i, for $i \in [1, n]$. This trusted party then computes $y = f(m_1, \cdots, m_n)$ and returns y to all the n users $U_1, \cdots, U_n$. Therefore, each user only learns y without revealing other private information. While in the *real world*, this imaginary trusted party does not exist, and function f is evaluated by running a protocol Π (via the two non-colluding

servers $\mathcal{C}$ and $\mathcal{E}$ in this paper). A protocol Π is defined as *secure* under the semi-honest model if all the view of real-world adversary $\mathcal{A}^{\mathbb{SH}}$ can be simulated by ideal-world adversary $\mathcal{F}^{\mathbb{SH}}$, and the view of real-world adversary $\mathcal{A}^{\mathbb{SH}}$ is *computationally indistinguishable* (denoted as $\stackrel{c}{\equiv}$) from the view of ideal-world adversary $\mathcal{F}^{\mathbb{SH}}$:

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\vec{x})\} \stackrel{c}{\equiv} \{\text{REAL}_{\Pi,\mathcal{A}^{\mathbb{SH}}}(\vec{x})\}$$

where $\vec{x}$ denotes the input of protocol Π. Further descriptions of the Real-and-Ideal model can be found in [19].

III. Technical Preliminary

Proxy Re-Encryption (PRE) [23], [24] is a special type of public-key encryption, which allows a semi-honest proxy to convert ciphertexts for Alice into ciphertexts for Bob without revealing the corresponding plaintexts. We use an ElGamal-based PRE [12] (denoted as AFGH in this paper) to show how to build outsourced computation over cloud data encrypted under multiple keys with two non-colluding servers.

Fig. 2 briefly presents the details of AFGH. There are two levels of encryption in AFGH. We use $\mathbf{Enc}^*$ to denote the normal encryption algorithm that produces ciphertexts which can be re-encrypted and **Enc** to denote the one which will directly create re-encrypted ciphertexts. Further details and properties of AFGH can be found in [12]. Since AFGH is ElGamal-based, its ciphertexts (both before and after re-encryption) are multiplicatively homomorphic. Specifically,

$$\mathbf{Enc}_{pk}(m_1) \diamond \mathbf{Enc}_{pk}(m_2) = \mathbf{Enc}_{pk}(m_1 \cdot m_2),$$
$$\mathbf{Enc}_{pk}(m_1)^{\alpha} = \mathbf{Enc}_{pk}(m_1^{\alpha}),$$

where $\alpha \in \mathbb{Z}_p^*$ and $\diamond$ denotes a component-wise multiplicative operation over ElGamal-based ciphertexts. For example, given $C = (\alpha, \beta)$, $C' = (\alpha', \beta')$, $C \diamond C'$ is computed as $(\alpha \cdot \alpha', \beta \cdot \beta')$.

One may also build an additively homomorphic version of ElGamal and hence AFGH [25]. Concretely, a ciphertext in its additively homomorphic version can be encrypted as $C_a = (Z^m \cdot Z^r, Z^{ar})$ or $C_a^* = (Z^m \cdot Z^r, g^{ar})$. Then we have

$$\mathbf{Enc}_{pk}(m_1) \diamond \mathbf{Enc}_{pk}(m_2) = \mathbf{Enc}_{pk}(m_1 + m_2),$$
$$\mathbf{Enc}_{pk}(m_1)^{\alpha} = \mathbf{Enc}_{pk}(\alpha \cdot m_1).$$

Different from its multiplicatively homomorphic version, the decryption of ciphertexts in this additively homomorphic version requires computing a discrete logarithm, which is hard in general. However, if the size of messages is small, then the decryption can be done efficiently with Pollard's Kangaroo (a.k.a., Pollard's Lambda) algorithm [26] or a lookup table containing all the possible values. The well-known BGN encryption [27], which allows additions and only one multiplication over ciphertexts, also requires computing a discrete logarithm in decryption. In order to decrypt messages with a larger size in these schemes, Chinese Remainder Theorem (CRT) can be leveraged by a recent study [25]. As a necessary trade off, the size of ciphertexts will increase.

Another advantage of AFGH is that it is still secure if Alice colludes with the proxy (i.e., the collusion of Alice and the proxy does not reveal the private key of Bob) [12]. We will leverage this property to prevent collusion between any user and Cloud Server $\mathcal{C}$ in our later design.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p, g be a generator of $\mathbb{G}_1$, $\mathtt{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map, $Z = \mathtt{e}(g, g) \in \mathbb{G}_2$. The system parameters $\mathcal{SP} = (\mathtt{e}, p, \mathbb{G}_1, \mathbb{G}_2, g, Z)$.

KeyGen$(\mathcal{SP}) \to (sk_a, pk_a)$. Given $\mathcal{SP}$, output secret key $sk_a = a \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and public key $pk_a = g^a \in \mathbb{G}_1$.

ReKey$(sk_a, pk_b) \to (rk_{a\to b})$. Given pk_b and sk_a, output re-encryption key $rk_{a\to b} = (pk_b)^{1/a} = g^{b/a} \in \mathbb{G}_1$.

Enc$(pk_a, m) \to (C_a)$. Given pk_a and message $m \in \mathbb{G}_2$, compute a random $r \in \mathbb{Z}_p^*$, output $C_a = (m \cdot Z^r, Z^{ar})$.

Enc$^*(pk_a, m) \to (C_a^*)$. Given sk_a and $m \in \mathbb{G}_2$, generate a random $r \in \mathbb{Z}_p^*$, and output $C_a^* = (m \cdot Z^r, g^{ar})$.

ReEnc$(rk_{a\to b}, C_a^*) \to (C_b)$. Given $rk_{a\to b}$ and $C_a^* = (m \cdot Z^r, g^{ar})$, compute $Z^{br} = \mathtt{e}(g^{ar}, g^{b/a}) \in \mathbb{G}_2$ and output $C_b = (m \cdot Z^r, Z^{br})$.

Dec$(C_a, sk_a) \to (m)$. Given $C_a = (m \cdot Z^r, Z^{ar})$ and sk_a, decrypt as $m = \frac{m \cdot Z^r}{(Z^{ar})^{1/a}}$.

Dec$^*(C_a^*, sk_a) \to (m)$. Given $C_a^* = (m \cdot Z^r, g^{ar})$ and sk_a, decrypt as $m = \frac{m \cdot Z^r}{\mathtt{e}(g^{ar}, g^{1/a})}$.

Fig. 2. Details of AFGH [12]

IV. Outsourced Computation over Cloud Data Encrypted under Multiple Keys

Overview. To support outsourced computation over encrypted data under multiple keys, we first transform the ciphertexts into ones encrypted under the same key. Meanwhile, the original ones can still be kept in the cloud, so that each user can retrieve and decrypt her outsourced data with her own private key. Our first scheme Banana$^+$ is built based on the additively homomorphic version of AFGH while Banana* is based on the multiplicatively homomorphic version.

A. Banana$^+$

1) Scheme Details of Banana$^+$: Our first scheme Banana$^+$ contains seven algorithms, including **Setup**, **Upload**, **TranEnc**, **Add**, **Mul**, **TranDec**, and **Retrieve**. Specifically, each user and the two servers generate public keys, private keys and re-encryption keys in **Setup**. In **Upload**, each user encrypts data with her own public key with AFGH (i.e., **AFGH.Enc***), and then outsources encrypted data to Cloud Server $\mathcal{C}$. To compute cloud data encrypted under multiple keys, $\mathcal{C}$ first transforms the ciphertexts under multiple keys into ones under the public key of Evaluation Server $\mathcal{E}$ with corresponding re-encryption keys in **TranEnc**. After the transformation of ciphertexts, additions over ciphertexts can be computed by $\mathcal{C}$ independently in **Add** while multiplications over ciphertexts can be evaluated by $\mathcal{C}$ and $\mathcal{E}$ together via one round server-to-server interaction in **Mul**. All the computations are operated over re-encrypted ciphertexts. The final result of outsourced computation can be transformed back to the ciphertexts under each user's public key via server-to-server interactions in **TranDec**, so that each user is able to retrieve and reveal the final result of outsourced computation with her own private key in **Retrieve**. Each user can also download and decrypt her data m_i with her own private key with **Retrieve**. Details of each algorithm are described in Fig. 3, and the one round server-to-server interaction in **Mul** are presented in Fig. 4.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p, g be a generator of $\mathbb{G}_1$, $\mathtt{e} : \mathbb{G}_1 \times \mathbb{G}_1 \leftarrow \mathbb{G}_2$ be a bilinear map, $Z = \mathtt{e}(g, g) \in \mathbb{G}_2$. The system parameters $\mathcal{SP} = (\mathtt{e}, p, \mathbb{G}_1, \mathbb{G}_2, g, Z)$.
Setup. U_i outputs $(sk_i, pk_i) = (x_i, g^{x_i})$, $\mathcal{C}$ generates $(sk_\mathcal{C}, pk_\mathcal{C}) = (x_\mathcal{C}, g^{x_\mathcal{C}})$, $\mathcal{E}$ computes $(sk_\mathcal{E}, pk_\mathcal{E}) = (x_\mathcal{E}, g^{x_\mathcal{E}})$. U_i also generates a re-encryption key as

$$rk_{i\to\mathcal{E}} = \textbf{AFGH.ReKey}(sk_i, pk_\mathcal{E}) = g^{x_\mathcal{E}/x_i}.$$

and uploads it to $\mathcal{C}$.
Upload. U_i encrypts data m_i with the original encryption

$$C_i^*(m_i) = \textbf{AFGH.Enc}^*(pk_i, m_i) = (Z^{m_i} \cdot Z^{r_i}, g^{x_i r_i}),$$

and uploads $C_i^*(m_i)$ to $\mathcal{C}$.
TranEnc. $\mathcal{C}$ outputs re-encrypted ciphertext $C_\mathcal{E}(m_i)$ with $rk_{i\to\mathcal{E}}$

$$\begin{aligned} C_\mathcal{E}(m_i) &= \textbf{AFGH.ReEnc}(rk_{i\to\mathcal{E}}, C_i^*(m_i)) \\ &= (Z^{m_i} \cdot Z^{r_i}, Z^{x_\mathcal{E} \cdot r_i}). \end{aligned}$$

and keeps a copy of $C_i^*(m_i)$.
Add. Given $C_\mathcal{E}(m_1)$ and $C_\mathcal{E}(m_2)$, $\mathcal{C}$ outputs

$$\begin{aligned} C_\mathcal{E}(m_1 + m_2) &= C_\mathcal{E}(m_1) \diamond C_\mathcal{E}(m_2) \\ &= (Z^{m_1+m_2} \cdot Z^{r_1+r_2}, Z^{(r_1+r_2)\cdot x_\mathcal{E}}). \end{aligned}$$

Mul. Given $C_\mathcal{E}(m_1)$ and $C_\mathcal{E}(m_2)$, $\mathcal{C}$ computes

$$C_\mathcal{E}(\alpha_1 \cdot m_1) = C_\mathcal{E}(m_1)^{\alpha_1}, \quad C_\mathcal{E}(\alpha_2 \cdot m_2) = C_\mathcal{E}(m_2)^{\alpha_2}$$

where α_1 and α_2 are randomly chosen, and sends $C_\mathcal{E}(\alpha_1 \cdot m_1)$ and $C_\mathcal{E}(\alpha_2 \cdot m_2)$ to $\mathcal{E}$. Then, $\mathcal{E}$ computes as

$$\begin{aligned} \alpha_1 \cdot m_1 &= \textbf{AFGH.Dec}(sk_\mathcal{E}, C_\mathcal{E}(\alpha_1 \cdot m_1)), \\ \alpha_2 \cdot m_2 &= \textbf{AFGH.Dec}(sk_\mathcal{E}, C_\mathcal{E}(\alpha_2 \cdot m_2)), \\ m &= \alpha_1 \cdot m_1 \cdot \alpha_2 \cdot m_2, \\ C_\mathcal{E}(m) &= \textbf{AFGH.Enc}(pk_\mathcal{E}, m), \end{aligned}$$

and returns $C_\mathcal{E}(m)$ to $\mathcal{C}$. Finally, $\mathcal{C}$ outputs

$$C_\mathcal{E}(m_1 \cdot m_2) = C_\mathcal{E}(m)^{(\alpha_1\alpha_2)^{-1}}.$$

TranDec. Given $C_\mathcal{E}(y)$, where $y = f(m_1, ..., m_n)$, $\mathcal{C}$ generates a random α and outputs

$$C_\mathcal{E}(\alpha \cdot y) = C_\mathcal{E}(y)^\alpha$$

and sends $C_\mathcal{E}(\alpha \cdot y)$ to $\mathcal{E}$. Then, $\mathcal{E}$ decrypts $\alpha \cdot y$, and computes

$$C_i(\alpha \cdot y) = \textbf{AFGH.Enc}(pk_i, \alpha \cdot y), \quad 1 \le i \le n,$$

and returns $C_1(\alpha \cdot y), ..., C_n(\alpha \cdot y)$ to $\mathcal{C}$. Finally, $\mathcal{C}$ outputs

$$C_i(y) = C_i(\alpha \cdot y)^{\alpha^{-1}}, \quad 1 \le i \le n.$$

Retrieve. U_i retrieves $C_i(y)$ and $C_i^*(m_i)$ from $\mathcal{C}$, and outputs

$$\begin{aligned} y &= \textbf{AFGH.Dec}(sk_i, C_i(y)), \\ m_i &= \textbf{AFGH.Dec}^*(sk_i, C_i^*(m_i)) \end{aligned}$$

Fig. 3. Details of Each Step in Banana$^+$

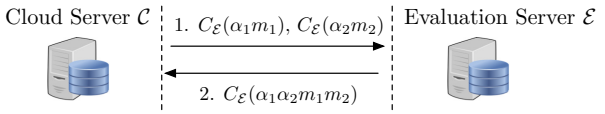


Fig. 4. The server-to-server interaction between $\mathcal{C}$ and $\mathcal{E}$ in **Mul**

2) Discussion: The recent work PTK [10] also supports outsourced computation under multiple keys. However, the two non-colluding servers in [10] transform the ciphertexts (from multiple keys into a single key) via server-to-server interactions linearly increasing with the number of data records (i.e., $O(n)$ for n records), which is not efficient. On the contrary, Banana$^+$ does not require any server-to-server interactions during the transformation of ciphertexts due to the utilization of PRE in **TranEnc**. Eliminating server-to-server interactions can save both communication and computation cost. Another advantage of Banana$^+$ is that it remains secure in face of collusion between any user and any server, while PTK assumes there is no collusion between any two parties [10].

3) Security Analysis of Banana$^+$: We now analyze the security of Banana$^+$ under the semi-honest model. Specifically, we use the Real-and-Ideal framework with a similar way as PTK by evaluating the security of each algorithm respectively based on Composition Theorem (under the semi-honest model) defined in [19].

***Theorem** 1: In Banana$^+$, it is computationally infeasible for Cloud Server $\mathcal{C}$ or Evaluation Server $\mathcal{E}$ to distinguish the inputs, intermediate results, and final results of outsourced computation over encrypted cloud data under multiple keys, as long as AFGH is semantically secure and the two servers are non-colluding.*

Proof: The security of **Setup**, **Upload**, **TranEnc**, and **Retrieve**, can be directly proved based on the security of AFGH [12], even considering collusion between any user and any server. We skip the detailed analyses of these algorithms here. In the following, we focus on how to analyze the security of **Add**, **Mul** and **TranDec** with the Real-and-Ideal framework. The essence of the following proof is to simulate the view of a semi-honest adversary in the real world with a simulator in the ideal world. If the view in the real world is computationally indistinguishable from the view in the ideal world, then an algorithm is believed to be a secure one.

Add: In this algorithm, since $\mathcal{C}$ is able to independently compute additions over ciphertexts, we prove that this algorithm is secure against a semi-honest adversary $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ corrupting $\mathcal{C}$ in the real world. More specifically, we build a simulator $\mathcal{F}^{\mathbb{SH}}$ in the ideal world to simulate the view of semi-honest adversary $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$. The view of $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ in this algorithm includes its input $\{C_\mathcal{E}(m_1), C_\mathcal{E}(m_2)\}$ and output $C_\mathcal{E}(m_1 + m_2)$.

Simulator $\mathcal{F}^{\mathbb{SH}}$ computes $C_\mathcal{E}(m_1') = \textbf{AFGH.Enc}(pk_\mathcal{E}, 1)$, $C_\mathcal{E}(m_2') = \textbf{AFGH.Enc}(pk_\mathcal{E}, 2)$, where $m_1' = 1$ and $m_2' = 2$ without loss of generality. Then, it evaluates $C_\mathcal{E}(m_1' + m_2') = C_\mathcal{E}(m_1') \diamond C_\mathcal{E}(m_2')$, and returns $\{C_\mathcal{E}(m_1'), C_\mathcal{E}(m_2'), C_\mathcal{E}(m_1' + m_2')\}$ to $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$.

Since the view of $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ are ciphertexts generated under AFGH and $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ has no knowledge of the private key of $\mathcal{E}$, if $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ could distinguish from the real world and the ideal world, then it indicates $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ could have an algorithm to distinguish ciphertexts generated by **AFGH.Enc**, which contradicts to the assumption that AFGH is semantically secure. Therefore, $\mathcal{A}_\mathcal{C}^{\mathbb{SH}}$ is computationally infeasible to distinguish from the real world and the ideal world. Therefore, we have

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\{C_\mathcal{E}(m_i)\})\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Add},\mathcal{A}_\mathcal{C}^{\mathbb{SH}}}(\{C_\mathcal{E}(m_i)\})\}$$

where $i \in \{1, 2\}$.

For the case of collusion between $\mathcal{C}$ and a user (e.g., user U_1 without loss of generality) in **Add**, the corresponding adversary $\mathcal{A}^{\mathbb{SH}}_{\{\mathcal{C},U_1\}}$ corrupting $\mathcal{C}$ and user U_1 at the same time will only bring a *negligible* advantage (*i.e., the advantage introduced by a message/ciphertext pair* $\{m_1, C_{\mathcal{E}}(m_1)\}$ *in the security game of AFGH*) to distinguish ciphertexts $\{C_{\mathcal{E}}(m_2), C_{\mathcal{E}}(m_1+m_2)\}$ between the real world and the ideal world under the input of $\{m_1, C_{\mathcal{E}}(m_2)\}$ in the above simulation. So, we can conclude

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\texttt{Input})\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Add},\mathcal{A}^{\mathbb{SH}}_{\{\mathcal{C},U_1\}}}(\texttt{Input})\}$$

where $\texttt{Input} = \{m_1, C_{\mathcal{E}}(m_2)\}$.

Mul: Since this algorithm is collaboratively participated by $\mathcal{C}$ and $\mathcal{E}$ together and we assume these two servers are non-colluding, we need to prove that **Mul** is secure not only against semi-honest adversary $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ corrupting cloud server $\mathcal{C}$ in the real world but also against semi-honest adversary $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ corrupting evaluation server $\mathcal{E}$ respectively.

Security Against Cloud Server $\mathcal{C}$. We first build a simulator $\mathcal{F}^{\mathbb{SH}}$ in the ideal world to simulate the view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ in the real world. The view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ in **Mul** includes input $\{C_{\mathcal{E}}(m_1), C_{\mathcal{E}}(m_2)\}$, randoms $\{\alpha_1, \alpha_2, \alpha_1\alpha_2, (\alpha_1\alpha_2)^{-1}\}$, ciphertexts $\{C_{\mathcal{E}}(\alpha_1 m_1), C_{\mathcal{E}}(\alpha_2 m_2), C_{\mathcal{E}}(\alpha_1\alpha_2 m_1 m_2)\}$ and output $C_{\mathcal{E}}(m_1 m_2)$.

Simulator $\mathcal{F}^{\mathbb{SH}}$ computes $C_{\mathcal{E}}(m'_1) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 1)$, $C_{\mathcal{E}}(m'_2) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 2)$, where $m'_1 = 1$ and $m'_2 = 2$ without loss of generality. Then, it generates randoms α'_1, α'_2, and computes $\alpha'_1\alpha'_2, (\alpha'_1\alpha'_2)^{-1}$. It evaluates $C_{\mathcal{E}}(\alpha'_1 m'_1) = C_{\mathcal{E}}(m'_1)^{\alpha'_1}$, $C_{\mathcal{E}}(\alpha'_2 m'_2) = C_{\mathcal{E}}(m'_2)^{\alpha'_2}$, $C_{\mathcal{E}}(\alpha'_1\alpha'_2 m'_1 m'_2) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, \alpha'_1\alpha'_2 \cdot 1 \cdot 2)$, $C_{\mathcal{E}}(m'_1 m'_2) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 1 \cdot 2)$. Finally, $\mathcal{F}^{\mathbb{SH}}$ returns $\{C_{\mathcal{E}}(m'_1)$, $C_{\mathcal{E}}(m'_2)$, α'_1, α'_2, $\alpha'_1\alpha'_2$, $(\alpha'_1\alpha'_2)^{-1}$, $C_{\mathcal{E}}(\alpha'_1 m'_1)$, $C_{\mathcal{E}}(\alpha'_2 m'_2)$, $C_{\mathcal{E}}(\alpha'_1\alpha'_2 m'_1 m'_2)$, $C_{\mathcal{E}}(m'_1 m'_2)\}$ to $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$.

The indistinguishability of ciphertexts in the view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ can also be proved by contradiction due to the semantic security of AFGH. In addition, the rest in the view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ are all randomly distributed. Therefore, $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ is computationally infeasible to distinguish from the real world and the ideal world,

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(C_{\mathcal{E}}(m_i))\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Mul},\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}}(C_{\mathcal{E}}(m_i))\}$$

where $i \in \{1, 2\}$. For the case of collusion between $\mathcal{C}$ and user U_1, we can also prove the security in a similar way as above with the input as $\{m_1, C_{\mathcal{E}}(m_2)\}$, and finally we obtain

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\texttt{Input})\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Mul},\mathcal{A}^{\mathbb{SH}}_{\{\mathcal{C},U_1\}}}(\texttt{Input})\}$$

where $\texttt{Input} = \{m_1, C_{\mathcal{E}}(m_2)\}$.

Security Against Evaluation Server $\mathcal{E}$. We build a simulator $\mathcal{F}^{\mathbb{SH}}$ in the ideal world to simulate the view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ in the real world. The view of $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ in **Mul** includes input $\{C_{\mathcal{E}}(\alpha_1 m_1), C_{\mathcal{E}}(\alpha_2 m_2)\}$, blinded messages $\alpha_1 m_1, \alpha_2 m_2, \alpha_1 m_1 \alpha_2 m_2$, and output $C_{\mathcal{E}}(\alpha_1 m_1 \alpha_2 m_2)$.

Simulator $\mathcal{F}^{\mathbb{SH}}$ computes $\alpha'_1 m'_1 = \alpha'_1 \cdot 1$, $\alpha'_2 m'_2 = \alpha'_2 \cdot 2$ and $\alpha'_1 m'_1 \alpha'_2 m'_2 = \alpha'_1 \cdot 1 \cdot \alpha'_2 \cdot 2$, where α'_1 and α'_2 are randomly selected and $m'_1 = 1$, $m'_2 = 2$ without loss of generality. Then, it computes $C_{\mathcal{E}}(\alpha'_1 m'_1) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, \alpha'_1)$, $C_{\mathcal{E}}(\alpha'_2 m'_2) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 2\alpha'_2)$, $C_{\mathcal{E}}(\alpha'_1 m'_1 \alpha'_2 m'_2) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 2\alpha'_1\alpha'_2)$. Finally, it returns $\{C_{\mathcal{E}}(\alpha'_1 m'_1)$, $C_{\mathcal{E}}(\alpha'_2 m'_2)$, $\alpha'_1 m'_1$, $\alpha'_2 m'_2$, $\alpha'_1 m'_1 \alpha'_2 m'_2$, $C_{\mathcal{E}}(\alpha'_1 m'_1 \alpha'_2 m'_2)\}$ to $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$.

Although $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ is able to decrypt the ciphertexts in its view with its private key, the (decrypted) messages are all blinded, which are randomly distributed due to the use of random α'_1 and α'_2. Therefore, $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ is not able to distinguish from the real world and the ideal world, and we have

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(C_{\mathcal{E}}(\alpha_i m_i))\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Mul},\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}}(C_{\mathcal{E}}(\alpha_i m_i))\}$$

where $i \in \{1, 2\}$. For the case of collusion between $\mathcal{E}$ and user U_1, we can also prove in a similar way as above with the input as $\{m_1, C_{\mathcal{E}}(\alpha_2 m_2)\}$, and have

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\texttt{Input})\} \stackrel{c}{\equiv} \{\text{REAL}_{\mathbf{Mul},\mathcal{A}^{\mathbb{SH}}_{\{\mathcal{E},U_1\}}}(\texttt{Input})\}$$

where $\texttt{Input} = \{m_1, C_{\mathcal{E}}(\alpha_2 m_2)\}$.

TranDec: This algorithm is also an interactive protocol between $\mathcal{C}$ and $\mathcal{E}$. Similar to the analyses in **Mul**, we can still discuss the security of this algorithm with a semi-honest adversary $\mathcal{A}^{\mathbb{SH}}_{\mathcal{C}}$ (in the real world) corrupting $\mathcal{C}$ with the view of $\{C_{\mathcal{E}}(y), \alpha, \alpha^{-1}, C_{\mathcal{E}}(\alpha y), C_i(\alpha y), C_i(y)\}$, and a semi-honest adversary $\mathcal{A}^{\mathbb{SH}}_{\mathcal{E}}$ corrupting $\mathcal{E}$ with the view of $\{C_{\mathcal{E}}(\alpha y), \alpha y, C_i(\alpha y)\}$ respectively. Due to the space limitation, we skip the details here.

Since an arithmetic function can be represented as a combination of additions and multiplications in general, while we have proved in the above that Banana$^+$ is able to securely compute one addition in **Add**, and evaluate one multiplication in **Mul**, therefore, the composition of multiple instances of **Add** and **Mul** is able to securely evaluate arithmetic functions over encrypted data (due to Composition Theorem [19]). ■

B. Banana*

1) Scheme Details: With the same design approach as our first scheme, we now describe the details of our second scheme, Banana*, which is based on the multiplicatively homomorphic version of AFGH. Banana* also has seven algorithms. The details in **Setup**, **Upload**, **TranEnc**, **TranDec** and **Retrieve** are almost the same as the ones in our first scheme, except the ciphertexts are in the form of multiplicatively homomorphic version. Specifically, the re-encrypted ciphertext is $C_i(m_i) = (m_i \cdot Z^{r_i}, Z^{x_i r_i})$, and the original ciphertext is $C^*_i(m_i) = (m_i \cdot Z^{r_i}, g^{x_i r_i})$. For Banana*, it is able to compute multiplications over ciphertexts independently at Cloud Server $\mathcal{C}$ in **Mul** while requires interactions between two servers on evaluating additions over ciphertexts in **Add**. Details of **Add** and **Mul** of Banana* are shown in Fig. 5.

2) Discussion: Similar to our first scheme, Banana* can also eliminate server-to-server interactions during the transformation of ciphertexts in **TranEnc**. Compared to Banana$^+$, Banana* does not require solving a discrete logarithm during the decryption of ciphertexts, which can improve the efficiency of outsourced computation. As a trade-off, Banana* needs an additional assumption to maintain the privacy.

More specifically, in order to correctly unblind the result of an addition over ciphertexts in **Add**, Cloud Server $\mathcal{C}$ uses a single blind factor α for the two ciphertexts $C_{\mathcal{E}}(m_1)$ and $C_{\mathcal{E}}(m_2)$ during one round server-to-server interaction with

Add. Given two ciphertexts $C_{\mathcal{E}}(m_1)$ and $C_{\mathcal{E}}(m_2)$, $\mathcal{C}$ first generates a random $\alpha \in \mathbb{Z}_p^*$, and computes

$$C_{\mathcal{E}}(\alpha) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, \alpha),$$
$$C_{\mathcal{E}}(\alpha^{-1}) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, \alpha^{-1}),$$
$$C_{\mathcal{E}}(\alpha \cdot m_1) = C_{\mathcal{E}}(\alpha) \diamond C_{\mathcal{E}}(m_1),$$
$$C_{\mathcal{E}}(\alpha \cdot m_2) = C_{\mathcal{E}}(\alpha) \diamond C_{\mathcal{E}}(m_2),$$

and sends $C_{\mathcal{E}}(\alpha \cdot m_1)$, $C_{\mathcal{E}}(\alpha \cdot m_2)$ to $\mathcal{E}$. Then, $\mathcal{E}$ decrypts

$$\alpha \cdot m_i = \textbf{AFGH.Dec}(sk_{\mathcal{E}}, C_{\mathcal{E}}(\alpha \cdot m_i)),$$

where $i \in \{1, 2\}$ and computes $m = \alpha(m_1 + m_2) = \alpha \cdot m_1 + \alpha \cdot m_2$. Then, $\mathcal{E}$ encrypts m as

$$C_{\mathcal{E}}(m) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, m),$$

and returns $C_{\mathcal{E}}(m)$ to $\mathcal{C}$. Finally, $\mathcal{C}$ outputs

$$C_{\mathcal{E}}(m_1 + m_2) = C_{\mathcal{E}}(m) \diamond C_{\mathcal{E}}(\alpha^{-1}).$$

Mul. Given two ciphertexts $C_{\mathcal{E}}(m_1)$ and $C_{\mathcal{E}}(m_2)$, $\mathcal{C}$ outputs

$$\begin{aligned} C_{\mathcal{E}}(m_1 \cdot m_2) &= C_{\mathcal{E}}(m_1) \diamond C_{\mathcal{E}}(m_2) \\ &= (m_1 m_2 Z^{r_1+r_2}, Z^{(r_1+r_2)\cdot x_{\mathcal{E}}}). \end{aligned}$$

Fig. 5. Details of **Add** and **Mul** in Banana*

Evaluation Server $\mathcal{E}$ (as presented in Fig. 5). As a result, although $\mathcal{E}$ is not able to directly reveal inputs (i.e., m_1, m_2) or intermediate result (i.e., $m_1 + m_2$), it can compute $m_1/m_2 = \alpha m_1/\alpha m_2$, and then distinguish blinded messages αm_1 and αm_2, if the knowledge of the distribution of users' data is available to $\mathcal{E}$ in advance.

Therefore, besides using different random values of α in different instances of **Add**, we also need to assume Evaluation Server $\mathcal{E}$ has no knowledge of the distribution of users' data in order to maintain privacy in Banana*. Note that the operations (i.e., multiplications) evaluated via server-to-server interactions in our first scheme do not have such kind of privacy issue, because the ciphertexts (i.e., $C_{\mathcal{E}}(m_1)$, $C_{\mathcal{E}}(m_2)$) are blinded with two different blind factors (i.e., α_1, α_2).

The above additional assumption indicates that Banana* is vulnerable under statistical attacks and it is more suitable for computation applications over unpublished data, where the distribution knowledge is unknown. For example, unpublished data are contributed by multiple users for some new or private research study, where the distribution knowledge is unknown to Evaluation Server $\mathcal{E}$.

In addition, due to this additional assumption, Banana* cannot securely support arithmetic functions with additions involving public constants. For instance, Banana* fails to provide security to the evaluation of an arithmetic function like

$$f(m_1, ..., m_n) = m_1 + \cdots + m_n + a,$$

where a is a public constant and is known to $\mathcal{E}$ in advance. Another limitation introduced by this additional assumption is that Banana* is not able to prevent collusion between $\mathcal{E}$ and any user (i.e., colluding one user helps $\mathcal{E}$ reveal other users' input and intermediate results).

3) Security Analysis of Banana*: The security of Banana* in the semi-honest model is presented as below. Compared to the first scheme, we have an additional assumption that $\mathcal{E}$ has no knowledge of the distribution of users' data.

***Theorem** 2: In Banana*, it is computationally infeasible for Cloud Server $\mathcal{C}$ or Evaluation Server $\mathcal{E}$ to distinguish the inputs, intermediate results, and final results of outsourced computation over encrypted cloud data under multiple keys, as long as AFGH is semantically secure, the two servers are non-colluding and $\mathcal{E}$ has no knowledge of the distribution of users' data.*

Proof: Compared to the analysis in Theorem 1, the security of **Setup**, **Upload**, **TranEnc**, **Retrieve**, and **TranDec** can be easily proved in the same way.

Different from our first scheme, we need to prove **Add** in Banana* is secure not only against adversary $\mathcal{A}_{\mathcal{C}}^{\mathbb{SH}}$ corrupting $\mathcal{C}$ but also against adversary $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$ corrupting $\mathcal{E}$; and we prove **Mul** is secure against adversary $\mathcal{A}_{\mathcal{C}}^{\mathbb{SH}}$ corrupting $\mathcal{C}$. The security against $\mathcal{A}_{\mathcal{C}}^{\mathbb{SH}}$ in **Add** and **Mul** can be proved in a similar way as in Theorem 1 based on the semantic security of AFGH. The major difference is the security against $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$ in **Add** due to the additional assumption that $\mathcal{E}$ has no knowledge of the distribution of users' data.

Specifically, we build a simulator $\mathcal{F}^{\mathbb{SH}}$ in the ideal world to simulate semi-honest adversary $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$ in the real world. The view of $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$ in **Add** includes input $\{C_{\mathcal{E}}(\alpha m_1), C_{\mathcal{E}}(\alpha m_2)\}$, messages αm_1, αm_2, $\alpha(m_1 + m_2)$, m_1/m_2, and output $C_{\mathcal{E}}(\alpha(m_1 + m_2))$.

Simulator $\mathcal{F}^{\mathbb{SH}}$ computes $m_1' \cdot \alpha' = r' \cdot \alpha'$, $m_2' \cdot \alpha' = 1 \cdot \alpha'$, $\alpha'(m_1' + m_2') = \alpha'(r' + 1)$, and $m_1'/m_2' = r'$, where $m_1' = r'$ is randomly selected and $m_2' = 1$ without loss of generality (*one of the two messages $\{m_1', m_2'\}$ is simulated as a random r' in order to capture the assumption that $\mathcal{E}$ has no knowledge of the distribution of users' data in the real world*). Then, it computes $C_{\mathcal{E}}(\alpha m_1') = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, r'\alpha')$, $C_{\mathcal{E}}(\alpha m_2') = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, 1 \cdot \alpha')$, $C_{\mathcal{E}}(\alpha'(m_1' + m_2')) = \textbf{AFGH.Enc}(pk_{\mathcal{E}}, \alpha'(r' + 1))$. Finally, it returns $\{C_{\mathcal{E}}(\alpha' m_1'), C_{\mathcal{E}}(\alpha' m_2'), \alpha' m_1', \alpha' m_2', \alpha'(m_1' + m_2'), m_1'/m_2', C_{\mathcal{E}}(\alpha'(m_1' + m_2'))\}$ to $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$.

Although $\mathcal{E}$ can decrypt the ciphertexts in its view with its private key, the (decrypted) messages in its view are randomly distributed due to the simulation of random α' and r'. Therefore, $\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}$ is not able to distinguish between the ideal world and the real world under our assumption, and we have

$$\{\text{IDEAL}_{f,\mathcal{F}^{\mathbb{SH}}}(\texttt{Input})\} \stackrel{c}{\equiv} \{\text{REAL}_{\textbf{Add},\mathcal{A}_{\mathcal{E}}^{\mathbb{SH}}}(\texttt{Input})\},$$

where $\texttt{Input} = \{C_{\mathcal{E}}(\alpha m_1), C_{\mathcal{E}}(\alpha m_2)\}$. ■

V. An Example

Our schemes can support many machine learning applications, and *classification* is one of them, which categorizes a new data instance based on a set of training (existing) data instances with known category. The large set of training data instances used in classification is generally contributed by multiple parties. Computing of classification includes two phases. The training phase generates a *classifier* based on a set of training data instances $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i$ is a data instance and $y_i \in \mathcal{Y}$ is the label indicating which category data instance $\mathbf{x}_i$ belongs. The classification phase outputs a label y as the category of a new data instance $\mathbf{x}$ based on the classifier from the training phase. In the following example, we

only consider a binary linear classifier, where data instances are classified into two categories (e.g., $\mathcal{Y} = \{+1, -1\}$).

Linear Mean Classifier. We first describe Linear Mean (LM) classifier [17] which can be presented as

$$f(\mathbf{x}; \mathbf{w}, c) = \mathbf{w}^T \mathbf{x} - c, \tag{1}$$

where $\mathbf{x}$ is a data instance, $\mathbf{w}$ is the weight vector and c is the threshold. An LM classifier splits d-dimensional inputs with a hyperplane (i.e., $f(\mathbf{x}; \mathbf{w}, c) = 0$), where all the points on one side of this hyperplane (i.e., above the threshold c) are labeled as $y = +1$ while the points on the other side are labeled as $y = -1$. Weight vector $\mathbf{w}$ and threshold c can be learned from a set of training data instances with known labels.

More specifically, given a set of training data instances $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$ and the index set of these training data instances $I_y = \{i \in \{1, \cdots, n\} | y_i = y\}$, class-conditional mean vector are computed as $\mathbf{m}_y = m_y^{-1} \mathbf{s}_y$, where $m_y = |I_y|$, $\mathbf{s}_y = \sum_{i \in I_y} \mathbf{x}_i$, for $y \in \{+1, -1\}$. Then, weight vector $\mathbf{w}$ and threshold c can be calculated as

$$\begin{aligned} \mathbf{w} &= \mathbf{m}_{+1} - \mathbf{m}_{-1}, \\ c &= \mathbf{w}^T (\mathbf{m}_{+1} + \mathbf{m}_{-1})/2. \end{aligned} \tag{2}$$

To evaluate LM classifier over ciphertexts, its Division-Free Integer (DFI) version [17] can be utilized to avoid divisions on integers, where the corresponding weight vector $\mathbf{w}^*$ and threshold c^* can be computed as

$$\begin{aligned} \mathbf{w}^* &= m_{+1} m_{-1} \mathbf{w} = m_{+1} m_{-1} (\mathbf{m}_{+1} - \mathbf{m}_{-1}), \\ c^* &= 2 m_{+1}^2 m_{-1}^2 c = m_{+1}^2 m_{-1}^2 \mathbf{w}^T (\mathbf{m}_{+1} + \mathbf{m}_{-1}). \end{aligned} \tag{3}$$

Further explanations of LM classifier and its DFI version can be found in [17]. Note that since the above computation does not involve any additions with public constants, our second scheme Banana* can still be used for this example.

With our schemes, we can evaluate the above classifier over data encrypted under multiple keys in the training phase, and compute an inner product based on a trained classifier and a new data instance over ciphertexts in the classification phase. Note that the comparison result of this inner product (i.e., $\mathbf{w}^{*T} \mathbf{x}$) and threshold (i.e., c^*) in the classification is evaluated locally by each user in plaintext domain since we do not use (relatively-expensive) solutions supporting comparison over encrypted data.

VI. Experimental Results

We now evaluate the performance of our schemes and compare with the performance of PTK [10] by running LM classifier we described in the previous section.

In the following experiments, the results of classification on plain data are independently evaluated with only one server, while the evaluations of Banana$^+$, Banana*, and PTK are operated with two non-colluding servers. Without loss of generality, we assume each server has the same computation ability, running Ubuntu 12.04 with Intel Core i5 3.30 GHz Processor and 2 GB Memory. In addition, we assume the delay of one server-to-server interaction is around 10 milliseconds on average, and the transmission rate between two servers is around 10 Mbps. As we introduced in Section III, since the decryption of ciphertexts in Banana$^+$ requires solving a discrete logarithm, we assume the size of messages in Banana$^+$ in the following experiments is always less than 20 bits. With the use of Chinese Reminder Theorem (CRT) [25], we can support a larger size of messages in Banana$^+$ with a factor of t (e.g., $20 \times t$ bits). As a trade-off, the computation and communication cost will both linearly increase with t.

The performance of LM classifier is evaluated in two aspects, including the training phase and classification phase. The number of training data instances is denoted by n, and the number of dimensions in each data instance is d. We assume each training data instance is contributed by one user, and all the n training data instances are contributed by n users (i.e., being encrypted with n multiple keys). Banana$^+$ and PTK are both based on additively homomorphic encryption, while Banana* uses multiplicatively homomorphic encryption. To achieve a similar security level of encryption, we choose $|p| = 160$ bits in our schemes and $|N| = 1024$ bits in PTK [10]. The computation time on different cryptographic operations in our schemes are evaluated based on Pairing Based Cryptography (PBC) library [28], and presented in Table VI.

TABLE VI
Computation time (in millisecond) on different operations

An exponentiation in $\mathbb{G}_2$	4.23
A multiplication in $\mathbb{G}_2$	0.019
A pairing operation	6.33

Training Phase. Since all the three schemes operate on data encrypted with multiple keys, they need to be transformed into ciphertexts under a common key in **TranEnc** before computation, such as the training phase of a linear classifier based on multiple users' data. The performance of **TranEnc** before the training phase is linearly increasing with the number of training data instances n and number of dimensions d. As we introduced in previous sections, our schemes do not require any server-to-server interactions during **TranEnc**, while PTK does. We can observe from Table II that our schemes have much better performance than PTK.

For the training phase of LM classifier, Table III shows that, Banana$^+$ has a better performance compared to PTK. Comparatively, Banana* is slower under the same parameters since the training phase of LM classifier requires a much larger number of additions (i.e., $O(dn)$) than multiplications (i.e., $O(d)$), while Banana* cannot perform additions over ciphertexts efficiently (due to the need of server-to-server interactions on additions over ciphertexts). It is clear that additively homomorphic-based approaches are more suitable than multiplicative ones for the training phase of LM classifier. Still, we remark that both of our schemes have better performance in the entire training process of LM classifier (over encrypted data under multiple keys) than PTK, which is illustrated later in Fig. 6.

After the generation of a classifier over ciphertexts, the two servers also need to perform transformation to get ciphertexts encrypted with users' keys in **TranDec**, so that each user can decrypt and access the result of outsourced computation later. As shown in Table IV, PTK requires more computation time and communication overhead than Banana$^+$ and Banana*.

According to the discussion above, our schemes are much more efficient than PTK for most of the steps except the results

TABLE II
PERFORMANCE OF TRANENC BEFORE THE TRAINING PHASE (TIME IN SECOND AND COMMUNICATION COST IN MB)

Data Instances n	Dimensions d	Banana$^+$ Time	Banana$^+$ Comm.	Banana* Time	Banana* Comm.	PTK [10] Time	PTK [10] Comm.
50	5	1.58	0	1.58	0	13.26	0.26
50	10	3.17	0	3.17	0	26.52	0.51
200	5	6.33	0	6.33	0	53.02	1.02
200	10	12.65	0	12.64	0	106.03	2.05

TABLE III
PERFORMANCE OF LM CLASSIFIER IN THE TRAINING PHASE (TIME IN SECOND AND COMMUNICATION COST IN MB)

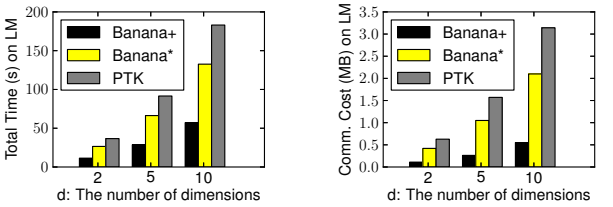
Data Instances n	Dimensions d	Plain Data Time	Banana$^+$ Time	Banana$^+$ Comm.	Banana* Time	Banana* Comm.	PTK [10] Time	PTK [10] Comm.
50	5	$1.13E-4$	0.33	0.004	10.36	0.20	0.54	0.008
50	10	$2.22E-4$	0.63	0.008	20.53	0.40	1.07	0.016
200	5	$4.43E-4$	1.22	0.015	40.96	0.79	2.13	0.031
200	10	$8.75E-4$	2.42	0.031	81.92	1.58	4.25	0.061

TABLE IV
PERFORMANCE OF TRANDEC AFTER THE TRAINING PHASE (TIME IN SECOND AND COMMUNICATION COST IN MB)

Data Instances n	Dimensions d	Banana$^+$ Time	Banana$^+$ Comm.	Banana* Time	Banana* Comm.	PTK [10] Time	PTK [10] Comm.
50	5	5.27	0.07	4.78	0.07	9.10	0.13
50	10	10.53	0.13	9.52	0.13	18.21	0.26
200	5	21.05	0.26	19.02	0.26	36.39	0.52
200	10	42.10	0.52	38.06	0.52	72.77	1.03

TABLE V
PERFORMANCE OF LINEAR CLASSIFIER IN THE CLASSIFICATION PHASE (TIME IN SECOND AND COMMUNICATION COST IN KB)

Dimensions d	Plain Data Time	Banana$^+$ Time	Banana$^+$ Comm.	Banana* Time	Banana* Comm.	PTK [10] Time	PTK [10] Comm.
5	$4.70E-6$	0.28	3.84	0.20	3.46	0.54	7.68
10	$9.42E-6$	0.58	7.68	0.39	6.92	1.06	15.36



(a) Computation Time (second) (b) Communication Cost (MB)

Fig. 6. Performance on LM classifier in total ($n = 200$)

of Banana* in Table III. As shown in Fig. 6, if we consider the entire training process of LM classifier over cloud data under multiple keys, including the transformation before the training phase (Table II), the training phase itself (Table III) and the transformation after the training phase (Table IV), both our schemes have a better performance than PTK.

Classification Phase. The performance of the classification phase is independent with the number of users. As shown in Table V, our schemes require less computation time and communication overhead than PTK.

VII. RELATED WORK

Our work is closely related to secure multiparty computation. Traditional secure multiparty computation approaches [8], [29]–[31] focus on how to compute privately for multiple users without any trusted party. The requirement of rather intensive interactions among users during computation is not affordable in the cloud data scenario we are considering. Besides, the one-time usage of garbled circuits significantly restricts its implementation since the algorithm for computing over multi-users' data must be processed in a bit-by-bit manner. While the recent advancement proposed by Goldwasser et al. [9] is able to reuse garbled circuits, it is still built based on a single key functional encryption, which is not suitable for the case of data encrypted under multiple keys in this paper.

Fully homomorphic encryption [5] enables arbitrary functions to be computed over the ciphertexts without revealing any private data. However, most existing schemes [6], [7], [32] are limited to the setting of a single key, which is not sufficient for our scenario. Although Gentry [5] proposed a solution for handling computation over multiple users where a single key is secretly shared among all the parties, it inevitably requires heavy interactions among users during the decryption of the final results. More importantly, the poor efficiency of FHE still limits its utilization in real applications.

A number of different approaches [18], [33]–[35] aim to improve the efficiency of secure multiparty computation by taking the advantages of cloud computing, where the cloud computes a lot while users only need to compute a little. For example, Halevi et al. [33] proposed a scheme to avoid user interactions during multiparty computation. Nikolaenko et al. [18] proposed a privacy-preserving ridge regression scheme with the help of two non-colluding servers. The two-server computation model has also been utilized for private set interaction by Chow, Lee, and Subramanian [36]. Unfortunately, these works cannot support outsourced computation over cloud data encrypted under multiple keys. Wang et al. [37], [38] also designed secure outsourced computation in the cloud for some practical problems, such as linear programming. However, they are not suitable for the case of multiple keys.

Moving a step forward, López et al. [4] first studied the case of FHE under multiple keys, named Multikey FHE, which can

be used to as a potential solution of the problem we want to address. Unfortunately, the efficiency of their scheme is still at a similar level compared to FHE. Moreover, it still needs interactions among users to decrypt final results of outsourced computation, which means users in their scheme are not totally free during computation.

The recent work of Peter, Tews, and Katzenbeisser [10] has the same assumption of two non-colluding servers. However, the heavy server-to-server interactions during the transformation of ciphertexts from multiple keys into a single key inevitably decrease the performance of this scheme. Our recent work [16] also face the same privacy leakage (i.e., m_1/m_2) in its multiplicatively homomorphic version (with two non-colluding servers) for handling outsourced computation on data encrypted by multiple keys, and it tried to fix this privacy leakage with three non-colluding servers and additional blind factors. Unfortunately, it still reveals similar information (i.e., $(m_1 + m_2)/m_1$). Therefore, how to design a multiplicatively homomorphic-based solution under two non-colluding servers without additional information leakage is still open.

VIII. Conclusion and Future Work

We propose two schemes for efficient privacy-preserving outsourced computation over cloud data encrypted under multiple keys with the help of two non-colluding servers. Our experimental results demonstrate that our schemes are more efficient than the existing approach due to the minimization of server-to-server interactions. Our future work will focus on how to overcome the current limitations (including the small message size in Banana$^+$ and the additional leakage of m_1/m_2 in Banana*) while still minimizing server-to-server interactions for computing data encrypted under multiple keys.

Acknowledgement

We would like to thank the reviewers for providing many helpful comments. The first two authors are supported in part by the U.S. NSF CNS-1218085. Sherman Chow is supported by the Early Career Scheme and the Early Career Award of the Research Grants Council, Hong Kong SAR (CUHK 439713), and Direct Grant (4055018) of the Chinese University of Hong Kong. Hui Li is supported by NSF of China 61272457, National Project 2012ZX03002003-002, 863 Project 2012AA013102, 111 Project B08038, IRT 1078, FRF K50511010001 and NSF of China 61170251.

References

[1] [Online]. Available: https://immersion.media.mit.edu/
[2] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud Data Protection for the Masses," *IEEE Computer*, vol. 45, no. 1, pp. 39–45, 2012.
[3] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.
[4] A. López, E. Tromer, and V. Vaikuntanathan, "On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," in *Proc. of STOC 2012*, 2012.
[5] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proc. of STOC'09*, 2009.
[6] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," in *Proc. of Eurocrypt*, 2010, pp. 24–43.
[7] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," in *Proc. of FOCS'11*, 2011.
[8] A. Yao, "Protocols for Secure Computations," in *Proc. of FOCS*, 1982, pp. 160–164.
[9] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable Garbled Circuits and Succinct Functional Encryption," in *Proc. of STOC'13*, 2013.
[10] 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.
[11] E. Bresson, D. Catalano, and D. Pointcheval, "A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications," in *Proc. of ASIACRYPT'03*, 2003, pp. 37–54.
[12] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage," in *Proc. of NDSS'05*, 2005.
[13] M. van Dijk and A. Juels, "On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing," in *Proc. of HotSec'10*. USENIX, 2010, pp. 1–8.
[14] Y. Elmehdwi, B. Samanthula, and W. Jiang, "Secure k-Nearest Neighbor Query over Encrypted Data in Outsourced Environments," in *Proc. of IEEE ICDE'14*, 2014.
[15] H. Chun, Y. Elmehdwi, F. Li, P. Bhattacharya, and W. Jiang, "Outsourceable Two-Party Privacy-Preserving Biometric Authentication," in *Proc. of ACM ASIACCS'14*, 2014.
[16] B. Wang, M. Li, S. S. M. Chow, and H. Li, "Computing Encrypted Cloud Data Efficiently under Multiple Keys," in *Proc. of CNS-SPCC*, 2013.
[17] T. Graepel, H. Lauter, and M. Naehrig, "ML Confidential: Machine Learning on Encrypted Data," in *International Conference on Information Security and Cryptography (SECRYPT)*, 2012.
[18] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-Preserving Ridge Regression on Hundred of Millions of Records," in *Proc. of IEEE S&P'13*, 2013.
[19] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2009.
[20] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *Proc. of FOCS'01*, 2001.
[21] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in *Proc. of ACM CCS'06*, 2006.
[22] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic Searchable Symmetric Encryption," in *Proc. of ACM CCS'12*, 2012, pp. 965–976.
[23] M. Blaze, G. Bleumer, and M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," in *Proc. of Eurocrypt*, 1998, pp. 127–144.
[24] S. S. M. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient Unidirectional Proxy Re-Encryption," in *Proc. of AfricaCrypt'10*, 2010.
[25] Y. Hu, W. J. Martin, and B. Sunar, "Enhanced Flexibility for Homomorphic Encryption Schemes via CRT," in *Proc. of ACNS'12*, 2012.
[26] J. M. Pollard, "Monte Carlo Methods for Index Computation mod p," *Mathematics of Computation*, vol. 32, no. 143, 918–924 1978.
[27] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. of TCC'05*, 2005.
[28] [Online]. Available: http://crypto.stanford.edu/pbc/
[29] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proc. of STOC*, 1987, pp. 218–229.
[30] D. Chaum, C. Crepeau, and I. Damgård, "Multiparty Unconditionally Secure Protocols," in *Proc. of STOC*, 1988, pp. 11–19.
[31] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic Encryption and Multiparty Computation," in *Proc. of Eurocrypt*, 2011, pp. 169–188.
[32] N. P. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," in *Proc. of PKC*, 2010, pp. 420–443.
[33] S. Halevi, Y. Lindell, and B. Pinkas, "Secure Computation on the Web: Computing without Simultaneous Interaction," in *Proc. of CRYPTO'11*, 2011, pp. 132–150.
[34] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing Multi-Party Computation," Available at http://eprint.iacr.org/2011/272.
[35] S. Kamara, P. Mohassel, and B. Riva, "Salus: A System for Server-Aided Secure Function Evaluation," in *Proc. of ACM CCS'12*, 2012, pp. 797–808.
[36] S. S. M. Chow, J.-H. Lee, and L. Subramanian, "Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases," in *Proc. of NDSS'09*, 2009.
[37] C. Wang, K. Ren, and J. Wang, "Secure and Practical Outsourcing of Linear Programming in Cloud Computing," in *Proc. of INFOCOM*, 2011, pp. 820–828.
[38] C. Wang, K. Ren, J. Wang, and K. M. R. Urs, "Harnessing the Cloud for Securely Solving Large-scale Systems of Linear Equations," in *Proc. of ICDCS*, 2011.