10-2020

Catch you if you deceive me: Verifiable and privacy-aware truth discovery in crowdsensing systems

Guowen XU
University of Electronic Science and Technology of China

Hongwei LI
University of Electronic Science and Technology of China

Shengmin XU
Singapore University of Technology and Design

Hao REN
University of Electronic Science and Technology of China

Yonghui ZHANG
Singapore Management University, yhzhang@smu.edu.sg

See next page for additional authors

Catch You If You Deceive Me: Verifiable and Privacy-Aware Truth Discovery in Crowdsensing Systems

Guowen Xu
guowen.xu@foxmail.com
University of Electronic Science and Technology of China

Hongwei Li*
hongweili@uestc.edu.cn
University of Electronic Science and Technology of China

Shengmin Xu
shengmin_xu@stud.edu.sg
Singapore University of Technology and Design

Hao Ren
renhao.uestc@gmail.com
University of Electronic Science and Technology of China

Yinghui Zhang
yhzhaang@163.com
Singapore Management University

Jianfei Sun
sjf215.uestc@gmail.com
University of Electronic Science and Technology of China

Robert H. Deng
robertdeng@smu.edu.sg
Singapore Management University

ABSTRACT

Truth Discovery (TD) is to infer truthful information by estimating the reliability of users in crowdsensing systems. To protect data privacy, many Privacy-Preserving Truth Discovery (PPTD) approaches have been proposed. However, all existing PPTD solutions do not consider a fundamental issue of trust. That is, if the data aggregator (e.g., the cloud server) is not trustworthy, how can an entity be convinced that the data aggregator has correctly performed the PPTD? A "lazy" cloud server may partially follow the deployed protocols to save its computing and communication resources, or worse, maliciously forge the results for some shady deals. In this paper, we propose V-PATD, the first Verifiable and Privacy-Aware Truth Discovery protocol in crowdsensing systems. In V-PATD, a publicly verifiable approach is designed enabling any entity to verify the correctness of aggregated results returned from the server. Since most of the computation burdens are carried by the cloud server, our verification approach is efficient and scalable. Moreover, users' data is perturbed with the principles of local differential privacy. Security analysis shows that the proposed perturbation mechanism guarantees a high aggregation accuracy even if large noises are added. Compared to existing solutions, extensive experiments conducted on real crowdsensing systems demonstrate the superior performance of V-PATD in terms of accuracy, computation and communication overheads.

*Corresponding author

ASIA CCS '20, October 5–9, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-6750-9/20/10…$15.00
https://doi.org/10.1145/3320269.3384720

CCS CONCEPTS

• **Security and privacy** → **Public key (asymmetric) techniques**; **Digital signatures**; **Security protocols**.

KEYWORDS

Truth Discovery; Verifiable Computation; Privacy Protection; Crowdsensing Systems

ACM Reference Format:
Guowen Xu, Hongwei Li, Shengmin Xu, Hao Ren, Yinghui Zhang, Jianfei Sun, and Robert H. Deng. 2020. Catch You If You Deceive Me: Verifiable and Privacy-Aware Truth Discovery in Crowdsensing Systems. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20), October 5–9, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3320269.3384720

1 INTRODUCTION

The rapid growth in the number of mobile devices has made crowdsensing system a popular data collection paradigm [10, 32], where the cloud server can pay to assign data collection tasks to a group of mobile users, and ask them to upload the sensory data (such as road conditions and air indicators) that they have recorded using the various sensors embedded in mobile devices. Such crowdsensing systems have been widely applied in healthcare, smart transportation, and many others.

Although users' sensory data serves a wide range of data-driven applications, there still exists an unresolved challenge of extracting truthful data from noise-filled, heterogeneous, large-scale data generated from various mobile devices [33]. In reality, the uncontrollable crowdsensing environments always result in the uneven quality of the collected data. Some users will provide high-quality data while others may upload incorrect data due to hardware issues or environmental noise. In this context, traditional data truth inference methods, such as averaging and voting, may fail to work as they typically assume that all users are equal in reliability (i.e., weight). Truth Discovery (TD) [13, 17, 21] overcomes this problem,

which aims to find truthful information among conflicting data based on each user's weight. In general, existing TD algorithms obey the following two principles: (1) If a data item provided by a user is closer to the aggregated result, this user will be assigned a higher weight. (2) If a user holds a higher weight, the data of this user will be counted more in the execution. Based on this, TD algorithms take as input the users' sensory data, and then iteratively estimate each user's weight to eventually find the optimal aggregated result (i.e., ground truth).

Undoubtedly, the TD method provides a powerful channel to find real data in complex crowdsensing environments. However, it neglects two fundamental problems in real-world scenarios. i.e., *data privacy protection* and *verifiability of aggregated results*. First of all, users' data privacy may be compromised once they upload sensory data to the untrusted cloud server. For example, sharing personal real-time GPS information can help with traffic monitoring, navigation and transportation control, but also increases the risk of exposing user's location privacy to the public. Aggregating the healthcare data submitted from various wearable devices can improve the analysis of the therapeutic effects of new drugs. However, a malicious server may abuse healthcare data because it has full access to individual users' data [30, 35]. On the other hand, how to verify the correctness of the aggregated data is another critical issue. In a typical crowdsensing system, a task requester pays for the server to collect data and find truthful values through a truth discovery algorithm. However, a dishonest server may deviate from the aggregation protocol in order to reduce its computation overhead, or worse, maliciously forge the results for some shady deals.

To protect data privacy, many PPTD approaches [13, 17, 21] have been proposed and widely applied in diverse fields. *Miao et al.* [21] proposed the first PPTD framework in crowdsensing systems. It can execute TD procedures in the ciphertext domain while guaranteeing the confidentiality of users' data. *Xu et al.* [31] designed EPTD, an efficient PPTD mechanism to solve the problem of privacy leakage during the truth discovery process. Moreover, EPTD can ensure that the entire process is smoothly executed even if there is a certain number of users offline. Recently, *Tang et al.* [28] presented a non-interactive PPTD scheme by utilizing the Yao's Garbled Circuit (GC) under the two non-colluding servers assumption. However, these cryptographic primitive-based techniques always involve expensive cost among mobile device users, which makes it difficult to achieve low latency for large-scale data or user sets. Other technologies, such as data perturbation based on differential privacy [17], is a feasible approach to protect users' privacy while preserving the efficiency. The fly in the ointment is that traditional centralized differential privacy mechanisms usually require a trusted server to perform the data perturbation [7, 20], that is, relying on the server to add pre-calculated noise to the aggregated results. This contradicts the assumption of untrusted server setting in many real-life scenarios. Moreover, to the best of our knowledge, existing solutions do not consider the verifiability of aggregated results returned from the untrusted cloud server. Therefore, it is critical to propose a PPTD approach, which can efficiently verify the correctness of results returned from the server, while protecting each user's data privacy.

However, it is challenging to design a satisfactory solution that meets the above requirements. First, local differential privacy based technology [18] has been demonstrated as a potential way to alleviate the above problems. However, compared with centralized differential privacy, it requires each user adding noise to their local data, which inevitably incurs larger errors in the aggregated results. Therefore, to reduce the impact of large noises on the aggregated results, one challenge that needs to be conquered is to carefully design the PPTD protocol, so as to achieve the elegant balance between security and aggregation accuracy. On the other hand, in TD process, the sensory data collected by the server is often sourced from multiple users. Besides, the outsourced function performed by the server changes as the number of users/data set size changes. As a consequence, the intrinsic nature of the truth discovery requires that a qualified verifiable algorithm should satisfy the following characteristics: (1) *Verifiability*: the cloud server can provide evidence (proof message) to the task requester for verifying the correctness of the returned results. (2) *Efficiency*: the cost of the task requester to verify the correctness of the result R is significantly less than the cost of computing R locally. (3) *Scalability*: the verifying time is independent of the size of the outsourced function's inputs. (4) *No prefixed functions*: the outsourced function does not require to be fixed, and each user does not need to know the function before outsourcing data. (5) *Public verification*: any entity can verify the correctness of the server's calculation results. (6) *Support of multiple data sources*: the inputs of the outsourced function can be contributed by multiple independent users.

To the best of our knowledge, most existing verifiable computation schemes [9, 14, 24, 25, 34] only support verification with a single data source or fixed outsourced functions, and hard to keep a small overhead when dealing with large-scale data and users. Recently, *Song et al.* [27] proposed the first publicly verifiable computation of polynomials framework over outsourced data. The authors claimed that their verifiable approach satisfied the above characteristics. Regrettably, *Wang et al.* [29] showed that a core building block of their protocol allows an adversary to forge the signatures on the outsourced data, so as to invalidate the security of the protocol. Therefore, it is also challenging to propose a light-weight verifiable approach which is highly supportive for verifying the correctness of results generated in TD process.

To address the above challenges, in this paper, we propose V-PATD, the first verifiable and privacy-aware truth discovery protocol in crowdsensing systems. We first perturb each user's sensory data independently under the definition of local differential privacy. Then, we design a publicly verifiable approach to enable any entity to verify the correctness of aggregated results returned from the server. In summary, the contributions of our V-PATD can be summarized as follows:

- We propose a novel perturbation scheme to protect the user's data privacy based on local differential privacy. By fully considering the characteristics of truth discovery, we demonstrate the ability of the proposed mechanism guaranteeing high aggregation accuracy even if large noise is added.
- We propose a publicly verifiable approach in the privacy-preserving truth discovery process. Our solution is the first approach meeting the requirements of *publicly verifiable*, *low cost*, *support for non-fixed outsourced functions and multiple data contributors*.
- We conduct extensive analysis to show that our V-PATD has high security and data utility. Besides, extensive experiments conducted on real-world data also demonstrate the high performance of V-PATD in terms of aggregated accuracy, computation and communication overheads.

The remainder of this paper is organized as follows. In Section 2, we describe the background and problem statement. In Section 3, we review some important concepts used in this paper. Then we present the technical details of our V-PATD in Section 4. Security analysis and performance evaluation are shown in Section 5 and Section 6. Finally, we introduce the related works in Section 7 and conclude the paper in Section 8.

2 PROBLEM STATEMENT

2.1 System Model

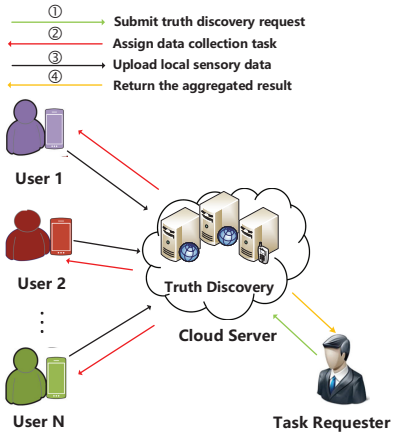

Figure 1: System Model

As shown in Figure 1, the system model in our scenario consists of three generic entities: *users*, *a task requester* and *a cloud server*. Specifically, the task requester first submits its truth discovery request (such as air quality monitoring and healthcare data collection) to the cloud server. Then, the cloud server assigns data collection tasks to a group of users with mobile devices, and asks them to upload the sensory data recorded through the various sensors embedded in mobile devices. In order to find the truthful data, the server performs the truth discovery algorithm on the cloud and returns the final aggregated result to the task requester.

2.2 Threat Model and Security and Privacy Requirements

Based on the system model, there are two potential security and privacy threats in the truth discovery process. One is that by exploiting the weaknesses in the truth discovery process, the server and certain users may violate the data privacy of honest users. The other is that the server may return incorrect aggregated results to the task requester for some improper benefits. In our V-PATD, we assume that the task requester is trustworthy and will not collude with any entity. All users are considered to be honest-but-curious [16, 26], which means that every user will honestly upload its local sensory data required by the server, however, it may also try to compromise other users' data privacy by utilizing the mastered prior knowledge. The cloud server is considered to be dishonest and may compromise our V-PATD model. Specifically, we consider the following attacks launched by the server: ①The cloud server can exploit the potential vulnerabilities in the network and the service interface to infer the privacy of sensory data submitted by users. ②The cloud server may corrupt the sensory data uploaded by each user, thereby breaking the correctness of aggregated results and gaining additional user's data privacy. ③The cloud server may launch data integrity attacks by compromising the agreed computing protocol or maliciously forging the aggregated results.

Under the above threat model, we formulate the security and privacy requirements as follows.

- *Protect the integrity of aggregated result*: As explained above, the cloud server is fully capable of modifying the aggregated results to deceive the task requester. The goal of our proposed model is to verify the integrity of aggregated results in an efficient manner.
- *Privacy protection of user's sensory data*: User-uploaded sensory data may contain certain sensitive information such as healthcare, location and credit. Consider the user's personal privacy, all user's sensory data should be protected from being disclosed to other parties (such as the server and malicious users).

3 PRELIMINARIES

3.1 Truth Discovery

The goal of truth discovery is to find truthful information among conflicting data based on each user's weight. Various truth discovery approaches [13, 17, 21] have been proposed for different scenarios. In this paper, we adopt the truth discovery algorithm designed by *Li et al.* [15] due to its superior aggregated accuracy and efficiency in crowdsensing systems. In general, a truth discovery algorithm can be divided into two interactive parts: *Weight Update* and *Truth Update*.

Specifically, assume that there are a total of $\mathcal{M}$ distinctive objects observed by $\mathcal{N}$ users (each user is denoted as

$n, n = (1, 2, \cdots \mathcal{N})$). We use the symbol x_n^m to represent the recorded sensory data of the n-th user for the m-th object. Similarly, the aggregated result (i.e., *ground truth*) of object m is denoted as x_*^m.

Weight Update: Given x_*^m, each user's weight w_n can be updated as below.

$$w_n = f\left(\sum_{m=1}^{\mathcal{M}} d_{ist}\left(x_n^m, x_*^m\right)\right) \tag{1}$$

f is a monotonically decreasing function, and $d_{ist}(x_n^m, x_*^m)$ is the distance function for measuring the distance between x_n^m and x_*^m. Usually, $d_{ist}(x_n^m, x_*^m)$ is computed as $d_{ist}(x_n^m, x_*^m) = (x_n^m - x_*^m)^2$. Based on the definition of f in [15], the above formula can be expanded as follows.

$$w_n = log\left(\frac{\sum_{n'=1}^{\mathcal{N}}\sum_{m=1}^{\mathcal{M}} d_{ist}(x_{n'}^m, x_*^m)}{\sum_{m=1}^{\mathcal{M}} d_{ist}(x_n^m, x_*^m)}\right) \tag{2}$$

From the above formula, we can see that a user n will be assigned a higher weight w_n if the data provided by this user is closer (i.e., $d_{ist}\left(x_n^m, x_*^m\right)$ is smaller) to the aggregated result.

Truth Update: Similarly, given the weight w_n of each user, the ground truth x_*^m of each object can be updated as below.

$$x_*^m = \frac{\sum_{n=1}^{\mathcal{N}} w_n \cdot x_n^m}{\sum_{n=1}^{\mathcal{N}} w_n} \tag{3}$$

Obviously, those users holding higher weights will be counted more in the truth update process. In truth discovery, the cloud server and all users iteratively perform the above two phases until the final results satisfy the agreed-upon convergence conditions.

3.2 Arithmetic Circuit

Arithmetic circuit [2, 27] is a basic architecture that can be exploited to compute arbitrary polynomials. In our V-PATD, the arithmetic circuit is an important aid to verify the integrity of the results returned by the server. We describe the definition of the arithmetic circuit as follows.

Definition 1: Given a finite field $\mathcal{F}$ and a set of variables $\mathcal{X} = (x_1, x_2, \cdots, x_{\mathcal{N}})$, the arithmetic circuit Ψ can be expressed as a directed acyclic graph (shown in Figure 2). The vertices of Ψ are called "gates", where gates with in-degree 0 are input gates, and gates with out-degree 0 are output gates. Other gates are labeled by either '$\times$' or '$+$' with in-degree 2. Specifically, a gate labeled by '$\times$' is called a product gate, and a gate labeled by '$+$' is called a sum gate.

3.3 Bilinear Map

A bilinear map e [11] is denoted as e: $G_1 \times G_1 \rightarrow G_2$, where both G_1 and G_2 are groups with prime order p. Suppose that g and h are the generator of the group G_1, the bilinear map e has following properties.

(1) **Bilinearity**: For any integer $a, b \in Z_p$, and $g^a, h^b \in G_1$, we have $e(g^a, h^b) = e(g, h)^{ab}$.
(2) **Computability**: $e(g^a, h^b)$ is computed efficiently for any $\{g^a, h^b\} \in G_1$.

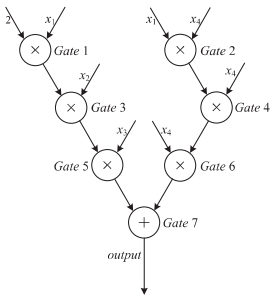

Figure 2: A example of arithmetic circuit to express a polynomial, i.e., $y = 2x_1x_2x_3 + x_1x_4^3$.

(3) **Non-degeneracy**: $e(g, h) \neq 1$, where g and h are the generator of G_1.

4 PROPOSED SCHEME

In this section, we describe the technical details of our V-PATD.

4.1 Overview

At a high level, before submitting raw sensory data to the cloud, each user first perturbs its sensory data independently by adding noises obeying a pre-set Gaussian distribution, where the perturbation mechanism strictly satisfies the definition of local differential privacy. In addition, each user is required to sign its data to facilitate subsequent generation of *proof* by the server. Then, the server executes the privacy-preserving truth discovery algorithm to find the optimal aggregated results. In the end, the server returns the aggregated results along with corresponding *proof*s to the task requester. The task requester can choose to accept or reject the results by only checking the *proof*s. In general, our V-PATD can be divided into two interactive parts: *Perturbation Mechanism* and *Verification Mechanism*. In the following subsections, we will present the technical details of these two parts.

4.2 Perturbation Mechanism

Assume that there are a total of $\mathcal{M}$ distinctive objects observed by $\mathcal{N}$ users (denoted as $n, n = (1, 2, \cdots, \mathcal{N})$). We use the symbol x_n^m to represent the recorded sensory data of the n-th user for the m-th object. Before uploading x_n^m to the cloud, each user n first adds noise ζ_n^m selected from $N(0, \gamma_n^2)$ to x_n^m as follows,

$$\widehat{x}_n^m = x_n^m + \zeta_n^m \tag{4}$$

where γ_n^2 is the variance of Gaussian distribution selected by the n-th user for uploading all its data items. Obviously, the amount of noise added to raw data is closely related to the level of data privacy protection and the accuracy of the aggregation. In general, adding larger noise helps provide a higher level of privacy protection, but bounds to impair the accuracy of the aggregated results. For the controllability of

accuracy, we ask all users to select the variance independently from an exponential distribution with hyper parameter η_2. Then, each user submits the perturbed data $\widehat{x}_n^m$ to the server.

Clearly, our perturbation mechanism is very simple. However, it has the following outstanding advantages that make it ideal for practical privacy-preserving truth discovery.

- We support each user to randomly select the distribution of noise's variance independently. It ensures that the server and other users are unable to know the distribution of target users' noise. To control the accuracy of aggregated results, the variance of all noises is chosen from an exponential distribution with hyper parameter η_2. This is feasible and we will prove that our perturbation mechanism meets the requirements of local differential privacy in Section 5.
- By combining the principles of truth discovery, the sensory data added with large noise will be counted less in the truth discovery process. This will benefit our perturbation mechanism to achieve better aggregation accuracy.
- Our perturbation mechanism is simple, which only requires each user to generate noise and adds it to the raw sensory data. Compared with techniques based on cryptographic primitives, the proposed mechanism is very efficient and easy to implement in real-world applications.

4.3 Verification Mechanism

In the perturbation process, each user has distorted its data before outsourcing to the server. To support verifiable computation, we also require each user to upload additional information to the server. Based on this, the server can generate *proof*s of the aggregated results returned to the task requester. Referring to Eqn.(2) and Eqn.(3), we know that the server needs to perform polynomial and logarithmic operations during the truth discovery process. However, it usually involves expensive cost to generate *proof*s for exponential results [2, 27]. In our V-PATD, we only require the server to compute the polynomials in Eqn.(2) and Eqn.(3) (i.e., $\sum_{n'=1}^{\mathcal{N}}\sum_{m=1}^{\mathcal{M}} d_{ist}(x_{n'}^m, x_*^m)$, $\sum_{m=1}^{\mathcal{M}} d_{ist}(x_n^m, x_*^m)$, $\sum_{n=1}^{\mathcal{N}} w_n \cdot x_n^m$ and $\sum_{n=1}^{\mathcal{N}} w_n$) and the corresponding *proof*s. The logarithmic operations (i.e., $log(\cdot)$) can be computed by each user after verifying the integrity of the above computations. This is feasible and requires only a small amount of communication and computation overheads for each user. For the convenience of description, here we assume that each user $n, n = (1, 2, \cdots \mathcal{N})$ has only one data item x_n, where x_n represents the sensory data of a target object that all users need to record. Besides, our verification mechanism works on integer domains due to the inherent requirements of embedded cryptographic primitives. To handle each floating point number $\widehat{x}_n$, we use a large rounding factor $\mathcal{L}$ to scale each $\widehat{x}_n$ to an integer $\mathcal{L} \cdot \widehat{x}_n$ whenever needed. $\widehat{x}_n$ can be recovered by simply removing $\mathcal{L}$ in the final verification process. This is a common trick widely adopted in works such as [21, 22, 31, 32, 35, 36]. For simplicity, we omit the above operation and default to all $\widehat{x}_n$ being integers in the following description.

In general, our verification mechanism uses as a basis the publicly verifiable computation from [27], that we modify significantly to improve the unforgeability of signatures in [27], thus resisting the attacks proposed in work [29]. Specifically, it consists of six algorithms, i.e., **Setup**, **Gen_key**, **Gen_sign**, **Gate_eval**, **Gen_proof** and **Proof_verify**.

Setup$(1^\lambda) \rightarrow (pp)$. Given a security parameter λ, the algorithm **Setup**(1^λ) outputs the global security parameters $pp = (e, p, G_1, G_2, g, h, H, H')$ used in our verification mechanism. G_1 and G_2 are two groups with prime order p, e is a bilinear map denoted as e: $G_1 \times G_1 \rightarrow G_2$, where g and h are the generator of the group G_1. $H : \{0,1\}^* \rightarrow Z_p$ and $H' : \{0,1\}^* \rightarrow Z_p$ are two different collision-resistant hash functions which map arbitrary strings to the elements in Z_p, respectively.

Gen_key$(pp) \rightarrow (sk, pk)$. Given the public parameter pp, the algorithm **Gen_key**(pp) outputs the secret key sk and public key pk for each user. The secret key $sk_n = (\beta_n, \beta_n', \beta_n'')$ are randomly selected by each user n. The public key is pk_n $= \{(g^{\beta_n}, h^{\beta_n}, h^{\frac{1}{\beta_n}}), (g^{\beta_n'}, h^{\beta_n'}, h^{\frac{1}{\beta_n'}}), (g^{\beta_n''}, h^{\beta_n''}, h^{\frac{1}{\beta_n''}})\}$. Then, each user n keeps the secret key locally and submits the public key to the cloud.

Gen_sign$(\widehat{x}_n, sk_n) \rightarrow \rho_{x_n}$. Take the perturbed sensory data $\widehat{x}_n$ (with label τ, denote the description of $\widehat{x}_n$) and the secret key sk_n as the inputs, the algorithm **Gen_sign**$(\widehat{x}_n, sk_n)$ outputs the signature ρ_{x_n}. Specifically, the user n first computes $\varphi_\tau = H(\tau)$ and $\digamma_\tau = H'(\tau)$. Then, it randomly chooses an integer $t_n \in Z_p$ and sets $\mu_n = h^{t_n}$, $\nu_n^{(1)} = \beta_n(\varphi_\tau + t_n)$ `mod` p, $\nu_n^{(2)} = \beta_n'(\digamma_\tau + t_n)$ `mod` p, and $\nu_n^{(3)} = \beta_n''(\widehat{x}_n + t_n)$ `mod` p. Then the signature ρ_{x_n} is denoted as follows.

$$\rho_{x_n} = (\mu_n, \nu_n^{(1)}, \nu_n^{(2)}, \nu_n^{(3)}). \tag{5}$$

In the end, each user n sends ρ_{x_n} as well as label τ to the cloud.

After receiving the signature ρ_{x_n} and τ from each user n, the server verifies the authenticity of ρ_{x_n} as follows.

$$\begin{aligned} e(g, h^{\nu_n^{(1)}}) &\stackrel{?}{=} e(g^{\beta_n}, \mu_n \times h^{\varphi_\tau}) = e(pk_n^{(1)}, \mu_n \times h^{\varphi_\tau}) \\ e(g, h^{\nu_n^{(2)}}) &\stackrel{?}{=} e(g^{\beta_n'}, \mu_n \times h^{\digamma_\tau}) = e(pk_n^{(2)}, \mu_n \times h^{\digamma_\tau}) \\ e(g, h^{\nu_n^{(3)}}) &\stackrel{?}{=} e(g^{\beta_n''}, \mu_n \times h^{\widehat{x}_n}) = e(pk_n^{(3)}, \mu_n \times h^{\widehat{x}_n}) \end{aligned} \tag{6}$$

where $pk_n^{(1)} = g^{\beta_n}$, $pk_n^{(2)} = g^{\beta_n'}$ and $pk_n^{(3)} = g^{\beta_n''}$. If verified, the server stores $\widehat{x}_n$ and ρ_{x_n}. Otherwise, outputs $\perp$.

Gate_eval$(\widehat{s}_1, \widehat{s}_2, \sigma_1, \sigma_2) \rightarrow \sigma'$. **Gate_eval** is a bit complicated but the core of our verification mechanism. Here we first introduce some preliminary knowledge before introducing it. Concretely, as described before, we only require the server to compute the polynomial in Eqn.(2) and Eqn.(3). Here we use the symbol $f(\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_\mathcal{N})$ to represent the polynomials executed by the server, where $\widehat{x}_n$ is the perturbed data of user n. $f(\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_\mathcal{N})$ can be denoted as follows.

$$f(\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_\mathcal{N}) = \sum (c_i \times \prod_{n \in [1,\mathcal{N}]} \widehat{x}_n^{e_n}) \tag{7}$$

where c_i is the constant-coefficient and e_n denotes the exponent of $\widehat{x}_n$. Then, we adopt the arithmetic circuit to express the $f(\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_{\mathcal{N}})$ (See Figure 2 for an example).

Therefore, the algorithm **Gate_eval** is run by the server to compute the verification tags σ' for the outputs of gates in the arithmetic circuit, where the tag σ' is used to verify the correctness of gates' output. For example, given a product or a sum gate $\mathtt{G}$, the input wires $\widehat{s}_1$ and $\widehat{s}_2$ of $\mathtt{G}$, and the corresponding verification tags σ_1 and σ_2, the algorithm **Gate_eval**$(\widehat{s}_1, \widehat{s}_2, \sigma_1, \sigma_2)$ outputs the verification tag σ' of $\mathtt{G}$'s output.

Based on the structure of the arithmetic circuit, we divide the verification tags into two categories: i.e., *category-1 verification tag* and *category-2 verification tag*. We describe the concepts of these two types of verification tags as below.

Definition 2: The category-1 verification tag includes the following tags: ①the verification tag of each user's perturbed data. ②the verification tag of the product gate's output. ③the verification tag of the sum gate $\mathtt{G}$'s output, where the input wires of $\mathtt{G}$ hold category-1 verification tag signed by the same public key. Taking $\widehat{s}_1$ (with label τ_1) as an example, the category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ of $\widehat{s}_1$ can be denoted as below.

$$pk_{\widehat{s}_1} = \{(g^{\beta_{\widehat{s}_1}}, h^{\beta_{\widehat{s}_1}}, h^{\frac{1}{\beta_{\widehat{s}_1}}}), (g^{\beta'_{\widehat{s}_1}}, h^{\beta'_{\widehat{s}_1}}, h^{\frac{1}{\beta'_{\widehat{s}_1}}}), (g^{\beta''_{\widehat{s}_1}}, h^{\beta''_{\widehat{s}_1}}, h^{\frac{1}{\beta''_{\widehat{s}_1}}})\}$$
$$\rho_{\widehat{s}_1} = (\mu_{\widehat{s}_1}, \nu^{(1)}_{\widehat{s}_1}, \nu^{(2)}_{\widehat{s}_1}, \nu^{(3)}_{\widehat{s}_1}) \tag{8}$$

where the secret key of variable $\widehat{s}_1$ is $(\beta_{\widehat{s}_1}, \beta'_{\widehat{s}_1}, \beta''_{\widehat{s}_1})$. $\mu_{\widehat{s}_1} = h^{t_{\widehat{s}_1}}$, $\nu^{(1)}_{\widehat{s}_1} = \beta_{\widehat{s}_1}(\varphi_{\tau_1} + t_{\widehat{s}_1}) \bmod p$, $\nu^{(2)}_{\widehat{s}_1} = \beta'_{\widehat{s}_1}(F_{\tau_1} + t_{\widehat{s}_1}) \bmod p$, and $\nu^{(3)}_{\widehat{s}_1} = \beta''_{\widehat{s}_1}(\widehat{s}_1 + t_{\widehat{s}_1}) \bmod p$, $t_{\widehat{s}_1}$ is a random integer selected from Z_p.

Definition 3: The category-2 verification tag includes the verification tag of the sum gate $\mathtt{G}$'s output, except the input wires of $\mathtt{G}$ hold category-1 verification tag signed by the same public key. In our verification mechanism, the category-1 verification tag can be transformed into the category-2 verification tag. However, the category-2 verification tag cannot be transformed into the category-1 verification tag. Similarly, taking the $\widehat{s}_1$ as an example, the category-2 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ of $\widehat{s}_1$ can be denoted as below.

$$pk_{\widehat{s}_1} = \{(g^{\beta_{\widehat{s}_1}}, h^{\beta_{\widehat{s}_1}}, h^{\frac{1}{\beta_{\widehat{s}_1}}}), (g^{\beta'_{\widehat{s}_1}}, h^{\beta'_{\widehat{s}_1}}, h^{\frac{1}{\beta'_{\widehat{s}_1}}}), (g^{\beta''_{\widehat{s}_1}}, h^{\beta''_{\widehat{s}_1}}, h^{\frac{1}{\beta''_{\widehat{s}_1}}})\}$$
$$\rho_{\widehat{s}_1} = (\mu_{\widehat{s}_1}, \nu^{(1)}_{\widehat{s}_1}, \nu^{(2)}_{\widehat{s}_1}, \nu^{(3)}_{\widehat{s}_1}) \tag{9}$$

where the secret key of variable $\widehat{s}_1$ is $(\beta_{\widehat{s}_1}, \beta'_{\widehat{s}_1}, \beta''_{\widehat{s}_1})$. $\mu_{\widehat{s}_1} = h^{t_{\widehat{s}_1}}$, $\nu^{(1)}_{\widehat{s}_1} = h^{\beta_{\widehat{s}_1}(\varphi_\tau + t_{\widehat{s}_1})}$, $\nu^{(2)}_{\widehat{s}_1} = h^{\beta'_{\widehat{s}_1}(F_\tau + t_{\widehat{s}_1})}$, and $\nu^{(3)}_{\widehat{s}_1} = h^{\beta''_{\widehat{s}_1}(\widehat{s}_1 + t_{\widehat{s}_1})}$, $t_{\widehat{s}_1}$ is a random integer selected from Z_p.

Therefore, given the input wires $\widehat{s}_1$ and $\widehat{s}_2$ of a gate $\mathtt{G}$, and the corresponding verification tags σ_1 and σ_2, **Gate_eval** is to generate the verification tag σ' of $\mathtt{G}$'s output. Please note that σ' will be in turn transmitted as an input to calculate the verification tag of the next gate's output.

Specifically, if $\mathtt{G}$ is a sum gate, then

Case 1: The inputs of $\mathtt{G}$ are a variable $\widehat{s}_1$ and a constant $\widehat{s}_2 = c$, where $\widehat{s}_1$ possesses the category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$. The verification tag $\sigma' = (pk', \rho')$ of $\mathtt{G}$'s output (i.e., $y = c + \widehat{s}_1$) can be computed as follows.

$$pk' = pk_{\widehat{s}_1}, \varphi'_\tau = c + \varphi_{\tau_1}, F'_\tau = c + F_{\tau_1}$$
$$\rho' = (\mu', \nu') = (\mu_{\widehat{s}_1}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \tag{10}$$

where $\nu'^{(1)} = h^{\beta_{\widehat{s}_1} \times c} \times h^{\nu^{(1)}_{\widehat{s}_1}}$, $\nu'^{(2)} = h^{\beta'_{\widehat{s}_1} \times c} \times h^{\nu^{(2)}_{\widehat{s}_1}}$ and $\nu'^{(3)} = h^{\beta''_{\widehat{s}_1} \times c} \times h^{\nu^{(3)}_{\widehat{s}_1}}$.

Case 2: The inputs of $\mathtt{G}$ are a variable $\widehat{s}_1$ and a constant $\widehat{s}_2 = c$, where $\widehat{s}_1$ possesses the category-2 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$. The verification tag $\sigma' = (pk', \rho')$ of the $\mathtt{G}$'s output (i.e., $y = c + \widehat{s}_1$) can be computed as follows.

$$pk' = pk_{\widehat{s}_1}, \varphi'_\tau = c + \varphi_{\tau_1}, F'_\tau = c + F_{\tau_1}$$
$$\rho' = (\mu', \nu') = (\mu_{\widehat{s}_1}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \tag{11}$$

where $\nu'^{(1)} = h^{\beta_{\widehat{s}_1} \times c} \times \nu^{(1)}_{\widehat{s}_1}$, $\nu'^{(2)} = h^{\beta'_{\widehat{s}_1} \times c} \times \nu^{(2)}_{\widehat{s}_1}$ and $\nu'^{(3)} = h^{\beta''_{\widehat{s}_1} \times c} \times \nu^{(3)}_{\widehat{s}_1}$.

Case 3: The inputs of $\mathtt{G}$ (e.g., the gate 7 in Figure 2) are two variables $\widehat{s}_1$ and $\widehat{s}_2$, both of which possess category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2}, \rho_{\widehat{s}_2})$, respectively. The verification tag $\sigma' = (pk', \rho')$ of the $\mathtt{G}$'s output (i.e., $y = \widehat{s}_1 + \widehat{s}_2$) can be computed as follows.

If $pk_{\widehat{s}_1} = pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the same user, the server can compute $\sigma' = (pk', \rho')$ as below.

$$pk' = pk_{\widehat{s}_1}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2}$$
$$\rho' = (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \tag{12}$$

where $\nu'^{(1)} = \nu^{(1)}_{\widehat{s}_1} + \nu^{(1)}_{\widehat{s}_2}$, $\nu'^{(2)} = \nu^{(2)}_{\widehat{s}_1} + \nu^{(2)}_{\widehat{s}_2}$, and $\nu'^{(3)} = \nu^{(3)}_{\widehat{s}_1} + \nu^{(3)}_{\widehat{s}_2}$.

If $pk_{\widehat{s}_1} \neq pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the different secret keys, the server can compute $\sigma' = (pk', \rho')$ as below.

(i) The server sends $(h^{\frac{1}{\beta_{\widehat{s}_1}}}, h^{\frac{1}{\beta'_{\widehat{s}_1}}}, h^{\frac{1}{\beta''_{\widehat{s}_1}}}, h^{\frac{1}{\beta_{\widehat{s}_2}}}, h^{\frac{1}{\beta'_{\widehat{s}_2}}}, h^{\frac{1}{\beta''_{\widehat{s}_2}}})$ to the task requester.

(ii) The task requester randomly selects three integers ξ, ξ' and $\xi'' \in Z_p$ and then submits $(g^{\xi}, h^{\xi}, h^{\frac{1}{\xi}}, h^{\frac{\xi}{\beta_{\widehat{s}_1}}}, h^{\frac{\xi}{\beta_{\widehat{s}_2}}})$, $(g^{\xi'}, h^{\xi'}, h^{\frac{1}{\xi'}}, h^{\frac{\xi'}{\beta'_{\widehat{s}_1}}}, h^{\frac{\xi'}{\beta'_{\widehat{s}_2}}})$, and $(g^{\xi''}, h^{\xi''}, h^{\frac{1}{\xi''}}, h^{\frac{\xi''}{\beta''_{\widehat{s}_1}}}, h^{\frac{\xi''}{\beta''_{\widehat{s}_2}}})$ to the cloud.

(iii) The server computes the verification tag $\sigma' = (pk', \rho')$ of $\mathtt{G}$'s output as below.

$$pk' = \{(g^{\xi}, h^{\xi}, h^{\frac{1}{\xi}}), (g^{\xi'}, h^{\xi'}, h^{\frac{1}{\xi'}}), (g^{\xi''}, h^{\xi''}, h^{\frac{1}{\xi''}})\}$$
$$\varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2}$$
$$\rho' = (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \tag{13}$$

where $\nu'^{(1)} = h^{\frac{\xi}{\beta_{\widehat{s}_1}} \times \nu^{(1)}_{\widehat{s}_1}} \times h^{\frac{\xi}{\beta_{\widehat{s}_2}} \times \nu^{(1)}_{\widehat{s}_2}}$, $\nu'^{(2)} = h^{\frac{\xi'}{\beta'_{\widehat{s}_1}} \times \nu^{(2)}_{\widehat{s}_1}} \times h^{\frac{\xi'}{\beta'_{\widehat{s}_2}} \times \nu^{(2)}_{\widehat{s}_2}}$, and $\nu'^{(3)} = h^{\frac{\xi''}{\beta''_{\widehat{s}_1}} \times \nu^{(3)}_{\widehat{s}_1}} \times h^{\frac{\xi''}{\beta''_{\widehat{s}_2}} \times \nu^{(3)}_{\widehat{s}_2}}$. In this case, the new verification tag $\sigma' = (pk', \rho')$ is changed to the category-2 verification tag.

Case 4: The inputs of $\mathtt{G}$ are two variables $\widehat{s}_1$ and $\widehat{s}_2$, where $\widehat{s}_1$ possesses category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ while $\widehat{s}_2$ possesses category-2 verification tag $\sigma_2 = (pk_{\widehat{s}_2}, \rho_{\widehat{s}_2})$. The verification tag $\sigma' = (pk', \rho')$ of $\mathtt{G}$'s output (i.e., $y = \widehat{s}_1 + \widehat{s}_2$) can be computed as follows.

If $pk_{\widehat{s}_1} = pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the same user, the server can compute $\sigma' = (pk', \rho')$ as below.

$$\begin{aligned} pk' &= pk_{\widehat{s}_1}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2} \\ \rho' &= (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \end{aligned} \tag{14}$$

where $\nu'^{(1)} = h^{\nu^{(1)}_{\widehat{s}_1}} \times \nu^{(1)}_{\widehat{s}_2}$, $\nu'^{(2)} = h^{\nu^{(2)}_{\widehat{s}_1}} \times \nu^{(2)}_{\widehat{s}_2}$, and $\nu'^{(3)} = h^{\nu^{(3)}_{\widehat{s}_1}} \times \nu^{(3)}_{\widehat{s}_2}$.

If $pk_{\widehat{s}_1} \neq pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the different secret keys, the server can compute $\sigma' = (pk', \rho')$ as below.

(i) The server sends $(h^{\frac{1}{\beta_{\widehat{s}_1}}}, h^{\frac{1}{\beta'_{\widehat{s}_1}}}, h^{\frac{1}{\beta''_{\widehat{s}_1}}})$ to the task requester.

(ii) Since all the keys of category-2 verification tags are generated by the task requester, it computes $(h^{\frac{\beta_{\widehat{s}_2}}{\beta_{\widehat{s}_1}}}, h^{\frac{\beta'_{\widehat{s}_2}}{\beta'_{\widehat{s}_1}}}, h^{\frac{\beta''_{\widehat{s}_2}}{\beta''_{\widehat{s}_1}}})$ and sends it to the cloud.

(iii) The server computes the verification tag $\sigma' = (pk', \rho')$ of the gate G's output as below.

$$\begin{aligned} pk' &= pk_{\widehat{s}_2}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2} \\ \rho' &= (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \end{aligned} \tag{15}$$

where $\nu'^{(1)} = \nu^{(1)}_{\widehat{s}_2} \times h^{\frac{\beta_{\widehat{s}_2}}{\beta_{\widehat{s}_1}} \times \nu^{(1)}_{\widehat{s}_1}}$, $\nu'^{(2)} = \nu^{(2)}_{\widehat{s}_2} \times h^{\frac{\beta'_{\widehat{s}_2}}{\beta'_{\widehat{s}_1}} \times \nu^{(2)}_{\widehat{s}_1}}$, and $\nu'^{(3)} = \nu^{(3)}_{\widehat{s}_2} \times h^{\frac{\beta''_{\widehat{s}_2}}{\beta''_{\widehat{s}_1}} \times \nu^{(3)}_{\widehat{s}_1}}$.

Case 5: The inputs of gate G are two variables $\widehat{s}_1$ and $\widehat{s}_2$, both of which possess category-2 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2}, \rho_{\widehat{s}_2})$, respectively. The verification tag $\sigma' = (pk', \rho')$ of the gate G's output (i.e., $y = \widehat{s}_1 + \widehat{s}_2$) can be computed as follows.

If $pk_{\widehat{s}_1} = pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the same user, the server can compute $\sigma' = (pk', \rho')$ as below.

$$\begin{aligned} pk' &= pk_{\widehat{s}_1}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2} \\ \rho' &= (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \end{aligned} \tag{16}$$

where $\nu'^{(1)} = \nu^{(1)}_{\widehat{s}_1} \times \nu^{(1)}_{\widehat{s}_2}$, $\nu'^{(2)} = \nu^{(2)}_{\widehat{s}_1} \times \nu^{(2)}_{\widehat{s}_2}$, and $\nu'^{(3)} = \nu^{(2)}_{\widehat{s}_1} \times \nu^{(3)}_{\widehat{s}_2}$.

If $pk_{\widehat{s}_1} \neq pk_{\widehat{s}_2}$, which means that σ_1 and σ_2 are signed by the different secret keys, the server can compute $\sigma' = (pk', \rho')$ as below.

(i) The server first selects a variable (e.g., $\widehat{s}_1$) as the primary input.

(ii) Since all the keys of category-2 verification tags are generated by the task requester, it can compute $(\frac{\beta_{\widehat{s}_1}}{\beta_{\widehat{s}_2}}, \frac{\beta'_{\widehat{s}_1}}{\beta'_{\widehat{s}_2}}, \frac{\beta''_{\widehat{s}_1}}{\beta''_{\widehat{s}_2}})$ and sends it to the cloud.

(iii) The server computes the verification tag $\sigma' = (pk', \rho')$ of the gate G's output as below.

$$\begin{aligned} pk' &= pk_{\widehat{s}_1}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2} \\ \rho' &= (\mu', \nu') = (\mu_{\widehat{s}_1} \times \mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \end{aligned} \tag{17}$$

where $\nu'^{(1)} = \nu^{(1)}_{\widehat{s}_1} \times (\nu^{(1)}_{\widehat{s}_2})^{\frac{\beta_{\widehat{s}_1}}{\beta_{\widehat{s}_2}}}$, $\nu'^{(2)} = \nu^{(2)}_{\widehat{s}_1} \times (\nu^{(2)}_{\widehat{s}_2})^{\frac{\beta'_{\widehat{s}_1}}{\beta'_{\widehat{s}_2}}}$, and $\nu'^{(3)} = \nu^{(3)}_{\widehat{s}_1} \times (\nu^{(3)}_{\widehat{s}_2})^{\frac{\beta''_{\widehat{s}_1}}{\beta''_{\widehat{s}_2}}}$.

If G is a product gate, then

Case 1: The inputs of gate G are a variable $\widehat{s}_1$ and a constant $\widehat{s}_2 = c$, where the $\widehat{s}_1$ can be a perturbed sensory data $\widehat{x}_n, n = (1, 2, \cdots \mathcal{N})$ or a output of other product gate. Based on the Eqn.(7), we can see that all the product gates are performed before the sum gates. Therefore, the verification tag σ_1 of $\widehat{s}_1$ must be the category-1 verification tag. Assume $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$, the verification tag $\sigma' = (pk', \rho')$ of the gate G's (such as the gate 1 in Figure 2) output (i.e., $y = c \times \widehat{s}_1$) can be computed as follows.

$$\begin{aligned} pk' &= pk_{\widehat{s}_1}, \varphi'_\tau = c \times \varphi_{\tau_1}, F'_\tau = c \times F_{\tau_1} \\ \rho' &= (\mu', \nu') = (\mu^c_{\widehat{s}_1}, c \times \nu^{(1)}_{\widehat{s}_1}, c \times \nu^{(2)}_{\widehat{s}_1}, c \times \nu^{(3)}_{\widehat{s}_1}) \end{aligned} \tag{18}$$

Case 2: The inputs of gate G (e.g., the gate 2 and gate 3 in Figure 2) are two variables $\widehat{s}_1$ and $\widehat{s}_2$, both of which possess category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2}, \rho_{\widehat{s}_2})$, respectively. In this case, the server first selects a variable (e.g., $\widehat{s}_1$) as the primary input. Then, the verification tag $\sigma' = (pk', \rho')$ of the gate G's output (i.e., $y = \widehat{s}_1 \times \widehat{s}_2$) can be computed as follows.

(i) The server sends $\sigma_1 = (pk_{\widehat{s}_1}, \rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2}, \rho_{\widehat{s}_2})$ to the task requester.

(ii) The task requester verifies the authenticity of σ_1 and σ_2 by the Eqn.(6). If fails, the task requester outputs $\perp$. Otherwise, it randomly selects new secret keys as ξ, ξ' and $\xi'' \in Z_p$ to compute $pk' = \{(g^\xi, h^\xi, h^{\frac{1}{\xi}}), (g^{\xi'}, h^{\xi'}, h^{\frac{1}{\xi'}}), (g^{\xi''}, h^{\xi''}, h^{\frac{1}{\xi''}})\}$.

(iii) The task requester computes the verification tag $\sigma' = (pk', \rho')$ of the gate G's output as below.

$$\begin{aligned} pk' &= \{(g^\xi, h^\xi, h^{\frac{1}{\xi}}), (g^{\xi'}, h^{\xi'}, h^{\frac{1}{\xi'}}), (g^{\xi''}, h^{\xi''}, h^{\frac{1}{\xi''}})\} \\ \varphi'_\tau &= \varphi_{\tau_1} \times \varphi_{\tau_2}, F'_\tau = F_{\tau_1} \times F_{\tau_2} \\ \rho' &= (\mu', \nu') = (\mu', \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)}) \end{aligned} \tag{19}$$

where $\mu' = h^{t'}$, $\nu'^{(1)} = \xi(\varphi'_\tau + t')$ mod p, $\nu'^{(2)} = \xi'(F'_\tau + t')$ mod p, $\nu'^{(3)} = \xi''(\widehat{s}_1 \times \widehat{s}_2 + t')$ mod p, and t' is a random integer selected from Z_p.

(iv) The task requester submits $\{\sigma', \varphi'_\tau, F'_\tau\}$ to the cloud.

As the polynomial shown Eqn.(7), all the product gates are performed before the sum gates. Therefore, according to the *Definition* 2, all the data with category-2 verification tags will not be the inputs of product gates. Hence, we have discussed how the cloud server generates the verification tag under various types of gates. For some gates, we need the cooperation of the task requester to successfully generate the verification tag. In real-world applications, the task requester can do these things in bulk or delegating a trusted third party to assist with this process.

Gen_proof$(\sigma_R) \rightarrow P$. When the computation reaches the last gate of the arithmetic circuit, the cloud server computes the verification tag $\sigma_R = (pk_R, \rho_R)$ of the last gate G's output, and sets the *proof* message $P = \sigma_R$. Then, the server returns the aggregated results $R = f(\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_\mathcal{N})$ and *proof* message P to the task requester.

Proof_verify$(P) \rightarrow (True, False)$. After receiving the aggregated results R and *proof* message P, the task requester verifies the correctness of R by checking the *proof* message $P = \sigma_R$. Specifically, for each input $\widehat{s}_i$ with label τ_i, the task requester first computes $\varphi_{\tau_i} = H(\tau_i)$ and $F_{\tau_i} = H'(\tau_i)$, and then sets $\varphi = f(\varphi_{\tau_1}, \varphi_{\tau_2}, \cdots, \varphi_{\tau_N})$ as well as $F = f(F_{\tau_1}, F_{\tau_2}, \cdots, F_{\tau_N})$. If the verification tag $\sigma_R = (pk_R, \rho_R)$ is the category-1 verification tag, the task requester verifies P by Eqn.(20). Otherwise, the task requester verifies P by Eqn.(21). If verified, the task requester accepts the result R; otherwise, the task requester rejects the result R.

$$\begin{aligned} e(g, h^{\nu_R^{(1)}}) &\stackrel{?}{=} e(g^{\beta_R}, \mu_R \times h^{\varphi}) = e(pk_R^{(1)}, \mu_R \times h^{\varphi}) \\ e(g, h^{\nu_R^{(2)}}) &\stackrel{?}{=} e(g^{\beta'_R}, \mu_R \times h^{F}) = e(pk_R^{(2)}, \mu_R \times h^{F}) \\ e(g, h^{\nu_R^{(3)}}) &\stackrel{?}{=} e(g^{\beta''_R}, \mu_R \times h^{R}) = e(pk_R^{(3)}, \mu_R \times h^{R}) \end{aligned} \tag{20}$$

where $\rho_R = (\mu_R, \nu_R^{(1)}, \nu_R^{(2)}, \nu_R^{(3)})$. Specifically, $\mu_R = h^{t_R}$, $\nu_R^{(1)} = \beta_R(\varphi + t_R)$ `mod` p, $\nu_R^{(2)} = \beta'_R(F + t_R)$ `mod` p, and $\nu_R^{(3)} = \beta''_R(R + t_R)$ `mod` p, $t_R \in Z_p$ is the random integer calculated from the last gate.

$$\begin{aligned} e(g, \nu_R^{(1)}) &\stackrel{?}{=} e(g^{\beta_R}, \mu_R \times h^{\varphi}) = e(pk_R^{(1)}, \mu_R \times h^{\varphi}) \\ e(g, \nu_R^{(2)}) &\stackrel{?}{=} e(g^{\beta'_R}, \mu_R \times h^{F}) = e(pk_R^{(2)}, \mu_R \times h^{F}) \\ e(g, \nu_R^{(3)}) &\stackrel{?}{=} e(g^{\beta''_R}, \mu_R \times h^{R}) = e(pk_R^{(3)}, \mu_R \times h^{R}) \end{aligned} \tag{21}$$

where $\rho_R = (\mu_R, \nu_R^{(1)}, \nu_R^{(2)}, \nu_R^{(3)})$. Specifically, $\mu_R = h^{t_R}$, $\nu_R^{(1)} = h^{\beta_R(\varphi + t_R)}$, $\nu_R^{(2)} = h^{\beta'_R(F + t_R)}$, and $\nu_R^{(3)} = h^{\beta''_R(R + t_R)}$, $t_R \in Z_p$ is the random integer calculated from the last gate.

The correctness of the above verification comes from the homomorphism of the verification tags, and we will give a complete analysis in the next section. Based on the algorithm **Proof_verify**, we can see that the task requester verifies the correctness of aggregated result R by only requiring to check the verification tag $\sigma_R = (pk_R, \rho_R)$ of the last gate `G`'s output. This feature will help the task requester to verify the results without the user's inputs, i.e., supporting public verification. Moreover, it also significantly reduces the communication cost in our verification mechanism.

5 SECURITY ANALYSIS

In this section, we first prove that the proposed perturbation mechanism satisfies the definition of local differential privacy, and then we analyze the security of our verification mechanism.

We first introduce some symbols used in the following analysis. Reviewing the scenario of V-PATD, there are a total of $\mathcal{M}$ distinctive objects to be observed by each user (denoted as $n, n = (1, 2, \cdots \mathcal{N})$), where x_n^m is the recorded sensory data of n-th user for the m-th object. Similarly, the aggregated result (i.e., *ground truth*) of object m is denoted as x_*^m. In addition, we use symbols $\mathcal{W}$ and $\mathcal{A}$ to denote the perturbation mechanism and original truth discovery algorithm, respectively. Therefore, the original data set of all users can be denoted as $D = \{x_n^m\}_{n=1,m=1}^{n=\mathcal{N},m=\mathcal{M}}$. Correspondingly, the perturbed data set is $\mathcal{W}(D) = \{\widehat{x}_n^m\}_{n=1,m=1}^{n=\mathcal{N},m=\mathcal{M}}$. Besides, the final aggregated results(i.e., the ground truths) on data set D and $\mathcal{W}(D)$ are indicated as $\{x_*^m\}_{m=1}^{m=\mathcal{M}} = \mathcal{A}(D)$ and $\{\widehat{x}_*^m\}_{m=1}^{m=\mathcal{M}} = \mathcal{A}(\mathcal{W}(D))$.

As discussed before, for each x_n^m, user n adds noise ζ_n^m selected from $N(0, \gamma_n^2)$ to x_n^m, where γ_n^2 is the variance of Gaussian noise selected from the n-th user. To ensure the controllability of accuracy, we ask all users to select the variance independently from an exponential distribution with hyper parameter η_2. This means that for each user n, the p.d.f of the γ_n^2's variance is $g(\gamma_n^2) = \eta_2 e^{\eta_2(\gamma_n^2)}$. On the other hand, in the original truth discovery process, we assume that the error of user n (i.e., the different between the user n's raw sensory data and the final aggregated results) follows another Gaussian distribution $N(0, \gamma_{n'}^2)$. Similarly, $\gamma_{n'}^2$ is selected from an exponential distribution with hyper parameter η_1. By the nature of the exponential distribution, the expectation of γ_n^2 and $\gamma_{n'}^2$ are $1/\eta_2$ and $1/\eta_1$. Let $1/\eta_2 = r/\eta_1$, where r is the ratio between the expectation of γ_n^2's variance and the error's variance. Obviously, a large r means that large noises added in the raw sensory data, and thus r can be regarded as the noise level compared with the raw sensory data.

5.1 Analysis for Perturbation Mechanism

In this subsection, we describe the privacy-preserving level of our perturbation mechanism under the definition of local differentia privacy.

Definition 4: ((ϵ, δ)-Local Differential Privacy): Assume that $\mathcal{R}$ is the universal set of perturbation mechanism $\mathcal{W}$'s outputs. Given any subset $\mathcal{S} \subseteq \mathcal{R}$, and two different sensory data $x^{(1)}$ and $x^{(2)}$, the perturbation mechanism $\mathcal{W}$ satisfies the (ϵ, δ)-local differential privacy as long as

$$Pr\{\mathcal{W}(x^{(1)}) \in \mathcal{S}\} \leq e^{\epsilon} Pr\{\mathcal{W}(x^{(2)}) \in \mathcal{S}\} + \delta \tag{22}$$

Based on the definition, for any two different sensory data $x^{(1)}$ and $x^{(2)}$, a smaller tuple (ϵ, δ) means a higher probability of the two $\mathcal{W}$'s outputs in the same range. In other words, it makes the outputs obtained under different inputs more indistinguishable, and provides a higher level of data privacy protection.

Next, we define the concept of *sensitive information* for each user n, which is also an important term in local differential privacy. Specifically, the *sensitive information* $\triangle_n$ of each user n can be denoted as $\triangle_n = \max_{(x_n^{(1)}, x_n^{(2)} \in D)} |x_n^{(1)} - x_n^{(2)}|$, where $x_n^{(1)}$ and $x_n^{(2)}$ are two data collected by user n for the same object. $\triangle_n$ is used to describe the range of sensory data claimed by user n. Intuitively, $\triangle_n$ is closely related to the parameter η_1 since η_1 controls the variance (i.e., $\gamma_{n'}^2$) size of users' error. We formally give the following lemma to define the relationship between the $\triangle_n$ and η_1.

Lemma 1: For user n, the p.d.f of the error's variance is $g(\gamma_{n'}^2) = \eta_1 e^{\eta_1(\gamma_{n'}^2)}$, and the sensitive information $\triangle_n$ satisfies that $\triangle_n = \max_{(x_n^{(1)}, x_n^{(2)} \in D)} |x_n^{(1)} - x_n^{(2)}| \leq \frac{\alpha_n}{\eta_1}$ with probability

at least $q(1-\frac{2e^{\frac{-b^2}{2}}}{b})$, *where* $\alpha_n = b\sqrt{2\ln\frac{1}{1-q}}$, q *and* b *are constants.*

Proof: See the proof in **Appendix 1.1**.

Based on the above lemma, we can infer that the size of sensitive information of each user is inversely proportional to η_1. This means that the larger the η_1 is, the smaller error' variance and sensitive information will be obtained. In the following analysis, we set $\triangle_n = \frac{b\sqrt{2\ln\frac{1}{1-q}}}{\eta_1}$ and demonstrate the proposed perturbation mechanism satisfying the definition of local differential privacy.

Theorem 1: Given the parameter η_1 *and* η_2, *where* $1/\eta_2 = r/\eta_1$, *the perturbation mechanism* $\mathcal{W}$ *of user* n *satisfies* (ϵ, δ)*-Local Differential Privacy with* $r \geq \frac{\alpha_n^2}{2\eta_1\epsilon\ln(\frac{1}{1-\delta})}$, *where* $\alpha_n = b\sqrt{2\ln\frac{1}{1-q}}$.

Proof: See the proof in **Appendix 1.2**.

5.2 Analysis for Verification Mechanism

In this subsection, we analyze the security of our verification mechanism. First of all, we give two lemmas to prove that our verification tag has the property of homomorphism with addition and multiplication.

Lemma 2: The verification tag in our verification mechanism is additive homomorphic.

Proof: The additive homomorphic means that given a sum gate G with the incoming wires $\widehat{s}_1$ and $\widehat{s}_2$, anyone can generate a new verification tag σ' for G's output (i.e., $\widehat{s}_1+\widehat{s}_2$) based on the verification tag σ_1 and σ_2, and verify the correctness of $\widehat{s}_1+\widehat{s}_2$ without knowing $\widehat{s}_1$ and $\widehat{s}_2$. For more details, please refer to **Appendix 1.3**.

Lemma 3: The verification tag in our verification mechanism is multiplicative homomorphic.

Proof: The multiplicative homomorphic means that given a product gate G with the incoming wires $\widehat{s}_1$ and $\widehat{s}_2$, anyone can generate a new verification tag σ' for G's output (i.e., $\widehat{s}_1 \times \widehat{s}_2$) based on the verification tag σ_1 and σ_2, and verify the correctness of $\widehat{s}_1 \times \widehat{s}_2$ without knowing $\widehat{s}_1$ and $\widehat{s}_2$. For more details, please refer to **Appendix 1.4**.

Based on the properties of our verification tag, we give the following theorem to prove the correctness of our verification mechanism.

Theorem 2: We say that the cloud server returns the correct result as long as its corresponding proof information is verified.

Proof: Based on Lemma 2 and Lemma 3, we know that the task requester can verify the result of a polynomial without knowing the inputs. Therefore, the correctness of our verification mechanism can be reduced to the correctness of algorithm **Proof_verify**. Based on the properties of bilinear maps, the correctness of Eqn.(20) and Eqn.(21) can be easily proved. Here we omit the detailed proofs for simplicity.

Next, we analyze the soundness (i.e., the *proof* information corresponding to any false computation answer will be detected and cannot pass the result integrity check) of our verification mechanism. Since the soundness of the verification mechanism stems from the unforgeability of the signature method designed in this paper, we give a theorem as follows.

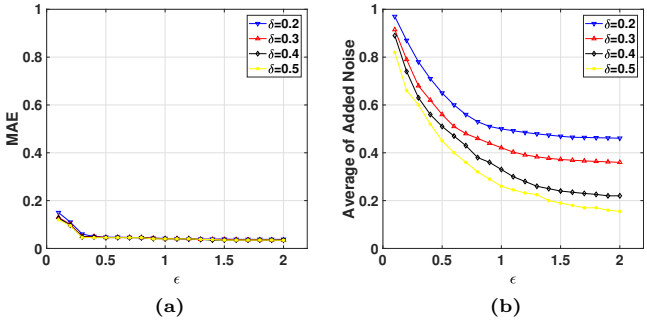

Figure 3: Effect of (ϵ, δ)**. (a) MAE. (b) Noise.**

Theorem 3: We say that our verification mechanism achieves soundness as long as our proposed digital signature scheme is EUF-CMA secure.

Proof: See the proof in **Appendix 1.5**.

6 PERFORMANCE EVALUATION

To evaluate the performance of our V-PATD, we simulate 150 mobile users to observe 50 pre-set objects in the application of floorplan construction [8]. Floorplan construction has gained a lot of attention because it can use mobile users to observe objects (such as the height, length, layout) and reconstruct the structure of the target building. We utilize Java Pairing-Based Cryptography Library (jPBC) [6] to implement the pairing computation in the proposed scheme, where we set the size of security parameter as 256 bits. In our experiments, each user is a smartphone equipped with 4GB RAM, Android 6.0 system. The " Cloud " is simulated with a Lenovo server which has Intel(R) Xeon(R)E5-2620 2.10GHZ CPU, 16GB RAM, 256SSD, 1TB mechanical hard disk and runs on the Ubuntu 18.04 operating system.

6.1 Accuracy

6.1.1 Effect of (ϵ, δ). We first analyze the relationship between the accuracy and privacy of our proposed scheme. In this part, we first perturbed the raw sensory data by the perturbation mechanism described in Section 4.2. Then, to measure the accuracy of aggregated results, we adopt the mean of absolute error (MAE) to compare the difference in aggregation results between raw data and perturbed data. For this measure, the lower value indicates higher accuracy.

As discussed before, a smaller tuple (ϵ, δ) provides a higher level of data privacy protection. In other words, it requires adding more noise to each sensory data. Figure 3(a) shows the MAE with the different values of ϵ. We can see that the MAE decreases and changes slowly as the value of ϵ increases. Moreover, from the Figure 3(b), we can also observe that the added average noise incurred in perturbation is small with the increase of (ϵ, δ). This is mainly due to the fact that

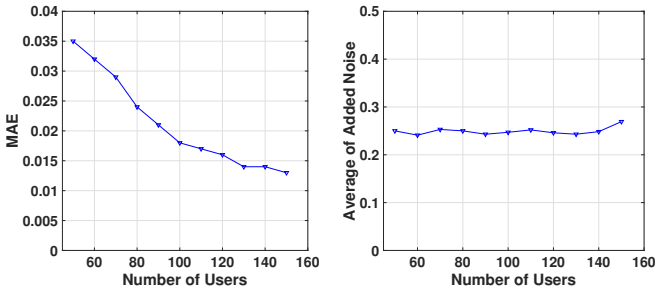

Figure 4: Effect of $\mathcal{N}$. (a) MAE. (b) Noise.

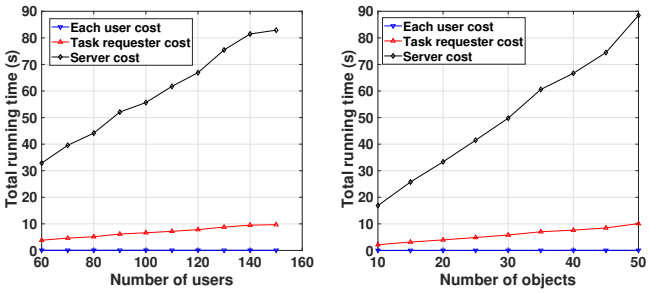

Figure 5: Computation Overhead. (a)$\mathcal{M}$=25, with the different number of users. (b)$\mathcal{N}$=75, with the different number of objects.

the truth discovery algorithm can automatically adjust the weight of each user, which helps lower the weight of users holding sensory data with large noise added, thereby reducing the impact of these users' data on the final result.

6.1.2 Effect of $\mathcal{N}$. Next, we analyze the effect of $\mathcal{N}$ on the results of the aggregation. Since each user perturbs its sensory data independently by adding noises obeying a pre-set Gaussian distribution, the change in $\mathcal{N}$ will not affect the average noise added in raw data, which is also shown in Figure 5(b). On the other hand, from the Figure 5(a), we can see that the increase in the number of users helps improve the accuracy of the aggregation results. The reason for this is that the truth discovery algorithm can collect more information when more users upload data, which helps to improve the quality of the aggregated results.

6.2 Computation Overhead

In this section, we analyze the computation overhead of our V-PATD under the different number of users/objects. As discussed before, our V-PATD excels in performance compared to existing solutions based on Homomorphic Encryption (HE) or SMC. To be more convincing, the latest solutions (based on SMC and HE)[22, 35] are also evaluated in our experiments to demonstrate the efficiency of our proposed scheme.

In our V-PATD, the overall computation overhead comes from three entities, i.e., the users, the cloud server and the task requester. Figure 6(a) and Figure 6(b) show that the computation overhead of each entity under the different number of users/objects. We can see that the cost of each user keeps almost constant as the number of users/objects increases. This is because in our scenario, each user is only required to perturb its data and sign it to the server, which requires only a small amount of computation overhead. The overhead of the server and the task requester are proportional to the number of users and objects. This is mainly because the variation of the number of users/objects affects the complexity of the arithmetic circuit of the outsourced function. In general, the increase in the number of users/objects results in more complex arithmetic circuits, which makes the server/the task requester (in some gates, the server needs to interact with the task requester to generate verification tags) require more overhead to generate verification tags for all gates. However, the overhead of the other two entities is not significant compared to the overhead of the server. In particular, the user's overhead grows very slowly and can be considered constant with the different number of users/objects. Therefore, our verifiable approach is efficient and scalable. Figure 7(a) and Figure 7(b) compare the computation overhead of V-PATD with those of the latest model CATD[35] and F-PPTD[22]. Obviously, compared to [22, 35], V-PATD significantly reduces the computation overhead in the privacy-preserving truth discovery process. This is mainly due to the shortcomings of the underlying structures of CATD and F-PPTD. Specifically, CATD[35] and F-PPTD[22] are constructed utilizing SMC and homomorphic encryption, respectively, and both of these require participants executing PPTD with multiple complex interactions. Therefore, these techniques always involve time-consuming computation or expensive communication among mobile device users, which makes it difficult to achieve low latency for large-scale data or user sets. Conversely, our perturbation mechanism is very simple, which just requires each user to perturb sensory by adding noises obeying a pre-set Gaussian distribution. Hence, compared with the latest model based on SMC or homomorphic encryption, our V-PATD is very efficient and can be easily applied to real-life scenarios.

6.3 Communication Overhead

Next, we analyze the communication overhead of our V-PATD under the different number of users/objects. Similarly, as shown in Figure 8(a), the communication cost of each user almost constant as the number of users/objects increases. This is also because the user only needs to sign his perturbed data and send them to the server. On the other hand, the overhead of the server and the task requester are proportional to the number of users and objects. In V-PATD, since the server primarily interacts with the task requester to generate tags for some gates and return the final aggregated result, which makes the communication overhead of these entities

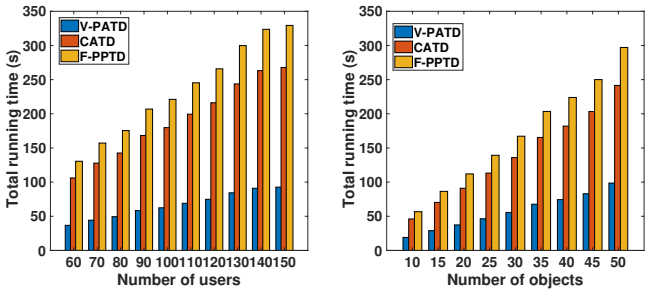

Figure 6: Computation overhead compared with existing models. (a)$\mathcal{M}$=25, with the different number of users. (b)$\mathcal{N}$=75, with the different number of objects.

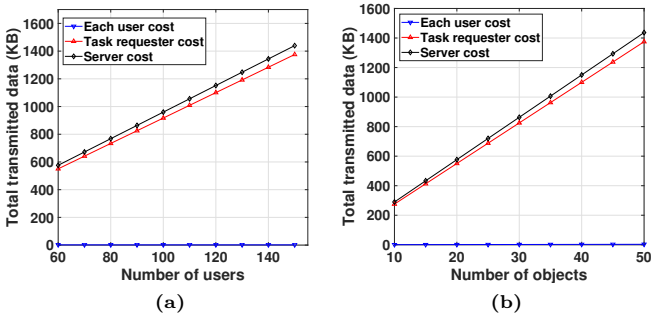

Figure 7: Communication Overhead. (a)$\mathcal{M}$=25, with the different number of users. (b)$\mathcal{N}$=75, with the different number of objects.

comparable. Figure 9(a) and Figure 9(b) shows that the comunication overhead of V-PATD compared with the latest model CATD[35] and F-PPTD[22]. As described before, F-PPTD is constructed under the SMC protocol, which requires the server to interact with each participant multiple times (using garbled circuits) to complete a ciphertext aggregation operation. This inevitably leads to high communication overhead. Also, to ensure the privacy of the aggregated results, CATD uses the threshold variant of the Paillier scheme [5] in their framework, which requires the cooperation of multiple users to complete the aggregation operation under ciphertext. However, compared to [22, 35], the performance of our V-PATD is excellent in terms of communication overhead, which grows slowly as the number of users/objects increases.

7 RELATED WORK

To protect the users' privacy in the truth discovery process, many PPTD approaches[13, 17, 21] have been proposed and widely applied in diverse fields. *Miao et al.*[21] proposed the

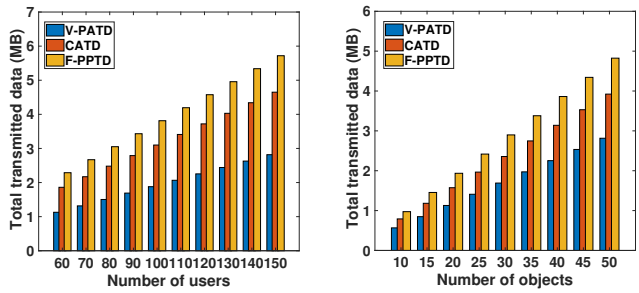

Figure 8: Communication overhead compared with existing models. (a)$\mathcal{M}$=25, with the different number of users. (b)$\mathcal{N}$=75, with the different number of objects.

first privacy-preserving truth discovery framework in crowdsensing systems. Based on Threshold Paillier Cryptosystem[5], it can execute TD procedures in the ciphertext domain while guaranteeing the confidentiality of users' privacy. To address the problem of users dropping out in the truth discovery process, *Xu et al.*[31] designed EPTD. It exploited the Secure Multiparty Computing (SMC) based technologies to ensure smooth execution even if there is a certain amount of users offline. *Tang et al.*[28] also presented a non-interactive PPTD, which utilized two non-colluding servers assumption to remove the online requirement for each user. On the other hand, to improve efficiency, several approaches[4, 22, 23, 36] have also been proposed. However, these techniques such as homomorphic encryption and SMC always involve time-consuming computation or expensive communication among mobile device users, which makes it difficult to achieve low latency for large-scale data or user sets. Recently, *Li et al.*[17] designed a two-layer data perturbation mechanism under the centralized differential privacy. However, their data perturbation process relies on a trusted server to add noise to the aggregation results, which contradicts the assumption of untrusted server setting in many real-life scenarios. To the best of our knowledge, all existing solutions do not consider the verifiability of aggregated results returned from the cloud server. As introduced above, a dishonest server may compress the agreed computing protocol or maliciously forge the aggregated results for some shady deals. Hence, it is urgent to proposed a verifiable truth discovery approach, which can efficiently verify the correctness of results returned from the server while protecting user's data privacy.

8 CONCLUSION

In this paper, we first designed a perturbation mechanism to protect users' sensory data before submitting them to the cloud. Based on this, a publicly verifiable technique is designed to enable any entity to verify the correctness of aggregated results returned from the server. We gave a formal

analysis to prove the security of our V-PATD. Moreover, extensive experiments demonstrated the superior performance of our V-PATD in terms of aggregated accuracy, computation and communication overhead. As part of future research effort, we will focus on improving the aggregation accuracy of our V-PATD.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YFB0802000, the National Natural Science Foundation of China under Grants 61802051, 61772121, 61728102, 61972094 and 61472065, the Peng Cheng Laboratory Project of Guangdong Province PCL2018KP004, the Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201804. In particular, I sincerely thank Ms. Tian Yan for her understanding and support in the past few years. Will you marry me?

REFERENCES

[1] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. 2011. Optimal structure-preserving signatures in asymmetric bilinear groups. In *Annual Cryptology Conference*. Springer, 649–666.

[2] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segevand V. Vaikuntanathan, and D. Vinayagamurthy. 2014. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 533–556.

[3] D. Boneh and M. Zhandry. 2013. Secure signatures and chosen ciphertext security in a quantum computing world. In *Annual Cryptology Conference*. Springer, 361–379.

[4] C. Cai, Y. Zheng, and C. Wang. 2018. Leveraging crowdsensed data streams to discover and sell knowledge: A secure and efficient realization. In *Proceedings of IEEE ICDCS*. 589–599.

[5] Ivan Damgård and Mads Jurik. 2001. A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*. Springer, 119–136.

[6] Angelo De Caro and Vincenzo Iovino. 2011. jPBC: Java pairing based cryptography. In *2011 IEEE symposium on computers and communications (ISCC)*. IEEE, 850–855.

[7] Cynthia Dwork. 2011. Differential privacy. *Encyclopedia of Cryptography and Security* (2011), 338–340.

[8] M. Elhamshary, M. Alzantot, and M. Youssef. 2018. JustWalk: A Crowdsourcing Approach for the Automatic Construction of Indoor Floorplans. *IEEE Transactions on Mobile Computing* (2018).

[9] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno. 2016. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1304–1316.

[10] O. Galinina, K. Mikhaylov, K. Huang, S. Andreev, and Y. Koucheryavy. 2018. Wirelessly powered urban crowd sensing over wearables: Trading energy for data. *IEEE Wireless Communications* 25, 2 (2018), 140–149.

[11] J. Groth and A. Sahai. 2008. Efficient non-interactive proof systems for bilinear groups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 415–432.

[12] D. Hsu, S. Kakade, and T. Zhang. 2012. A tail inequality for quadratic forms of subgaussian random vectors. *Electronic Communications in Probability* 17 (2012).

[13] H. Jin, L. Su, H. Xiao, and K. Nahrstedt. 2018. Incentive mechanism for privacy-aware data aggregation in mobile crowd sensing systems. *IEEE/ACM Transactions on Networking* 26, 5 (2018), 2019–2032.

[14] J. Keuffer, R. Molva, and H. Chabanne. 2018. Efficient Proof Composition for Verifiable Computation. In *Proceedings of the ESORICS*. Springer, 152–171.

[15] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han. 2014. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *Proceedings of ACM SIGMOD*. 1187–1198.

[16] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. 2016. A survey on truth discovery. *ACM Sigkdd Explorations Newsletter* 17, 2 (2016), 1–16.

[17] Y. Li, C. Miao, L. Su, J. Gao, Q. Li, B. Ding, Z. Qin, and K. Ren. 2018. An efficient two-layer mechanism for privacy-preserving truth discovery. In *Proceedings of ACM SIGKDD*. ACM, 1705–1714.

[18] Y. Li, H. Xiao, Z. Qin, C. Miao, L. Su, J. Gao, K. Ren, and B. Ding. 2018. Towards Differentially Private Truth Discovery for Crowd Sensing Systems. *arXiv preprint arXiv:1810.04760* (2018).

[19] A. Marshall and I. Olkin. 1967. A multivariate exponential distribution. *J. Amer. Statist. Assoc.* 62, 317 (1967), 30–44.

[20] F. McSherry and K. Talwar. 2007. Mechanism Design via Differential Privacy. In *Proceedings of the FOCS*, Vol. 7. 94–103.

[21] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren. 2015. Cloud-Enabled Privacy-Preserving Truth Discovery in Crowd Sensing Systems. In *Proceedings of the ACM SenSys*. ACM, 183–196.

[22] C. Miao, W. Jiang, L. Su, Y. Li, S. Guo, Z. Qin, H. Xiao, J. Gao, and K. Ren. 2019. Privacy-Preserving Truth Discovery in Crowd Sensing Systems. *ACM Transactions on Sensor Networks* 15, 1 (2019), 1–32.

[23] C. Miao, L. Su, W. Jiang, Y. Li, and M. Tian. 2017. A lightweight privacy-preserving truth discovery framework for mobile crowd sensing systems. In *Proceedings of IEEE INFOCOM*. IEEE, 1–9.

[24] B. Parno, J. Howell, C. Gentry, and M. Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Security and Privacy*. 238–252.

[25] B. Parno, M. Raykova, and V. Vaikuntanathan. 2012. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*. Springer, 422–439.

[26] H. Ren, H. Li, Y. Dai, K. Yang, and X. Lin. 2018. Querying in Internet of Things with Privacy Preserving: Challenges, Solutions and Opportunities. *IEEE Network* 32, 6 (2018), 144–151.

[27] W. Song, B. Wang, Q. Wang, C. Shi, W. Lou, and Z. Peng. 2017. Publicly verifiable computation of polynomials over outsourced data with multiple sources. *IEEE Transactions on Information Forensics and Security* 12, 10 (2017), 2334–2347.

[28] X. Tang, C. Wang, X. Yuan, and Q. Wang. 2018. Non-interactive privacy-preserving truth discovery in crowd sensing applications. In *Proceedings of the IEEE INFOCOM*. 1988–1996.

[29] Xu An Wang, Kim-Kwang Raymond Choo, Jian Weng, and Jianfeng Ma. 2019. Comment on "Publicly Verifiable Computation of Polynomials over Outsourced Data with Multiple Sources". *IEEE Transactions on Information Forensics and Security* (2019).

[30] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin. 2019. Enabling efficient and geometric range query with access control over encrypted spatial data. *IEEE Transactions on Information Forensics and Security* 14, 4 (2019), 870–885.

[31] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu. 2019. Efficient and Privacy-Preserving Truth Discovery in Mobile Crowd Sensing Systems. *IEEE Transactions on Vehicular Technology* 68, 4 (2019), 3854–3865.

[32] G. Xu, H. Li, and R. Lu. 2018. Practical and Privacy-Aware Truth Discovery in Mobile Crowd Sensing Systems. In *Proceedings of ACM CCS*. 2312–2314.

[33] G. Xu, H. Li, C. Tan, D. Liu, Y. Dai, and K. Yang. 2017. Achieving efficient and privacy-preserving truth discovery in crowd sensing systems. *Computers & Security* 69 (2017), 114–126.

[34] L. Zhao and L. Chen. 2018. Sparse Matrix Masking-based Non-Interactive Verifiable (Outsourced) Computation, Revisited. *IEEE Transactions on Dependable and Secure Computing* (2018).

[35] Y. Zheng, H. Duan, and C. Wang. 2018. Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing. *IEEE Transactions on Information Forensics and Security* 13, 10 (2018), 2475–2489.

[36] Y. Zheng, H. Duan, X. Yuan, and C. Wang. 2017. Privacy-aware and efficient mobile crowdsensing with truth discovery. *IEEE Transactions on Dependable and Secure Computing* (2017).

APPENDIX

1.1 Proof of Lemma 1

Proof. Since user n's error follows the Gaussian distribution $N(0,\gamma_{n'}^2)$, the sensory data x_n follows $x_n \sim N(x_*,\gamma_{n'}^2)$, where x_* represents the true value, and $\gamma_{n'}^2$ is selected from the exponential distribution with hyper parameter η_1. Given a large number B, we can conclude that $Pr\{\gamma_{n'} \le B\} = 1-e^{-\eta_1 B^2} = q$ by the nature of light tail of the exponential distribution [19], which implies $B = \frac{\sqrt{\ln\frac{1}{1-q}}}{\sqrt{\eta_1}}$. Since we assume that most of users are reliable, the η_1 should be larger than 1. Therefore, we have $B \le \frac{\sqrt{\ln\frac{1}{1-q}}}{\eta_1}$.

Next, we try to bound $\triangle_n$. Given two data $x_n^{(1)}$ and $x_n^{(2)}$, as the sensory data $x_n \sim N(x_*,\gamma_{n'}^2)$, we know that $x_n^{(1)} - x_n^{(2)} \sim N(0,2\gamma_{n'}^2)$. Therefore, we have $Pr\{|x_n^{(1)} - x_n^{(2)}| > b\sqrt{2}\gamma_{n'}\} \le \frac{2e^{\frac{-b^2}{2}}}{b}$ based on the Gaussian Tail Inequality[12], which implies $\triangle_n = \max_{(x_n^{(1)},x_n^{(2)}\in D)} |x_n^{(1)} - x_n^{(2)}| \le b\sqrt{2}\gamma_{n'}$ with probability at least $1-\frac{2e^{\frac{-b^2}{2}}}{b}$. Let $\alpha_n = b\sqrt{2\ln\frac{1}{1-q}}$, we have $\triangle_n = \max_{(x_n^{(1)},x_n^{(2)}\in D)} |x_n^{(1)} - x_n^{(2)}| \le b\sqrt{2}\gamma_{n'} \le \frac{b\sqrt{2\ln\frac{1}{1-q}}}{\eta_1} \le \frac{\alpha_n}{\eta_1}$ with probability at least $\eta_1(1-\frac{2e^{\frac{-b^2}{2}}}{b})$. ∎

1.2 Proof of Theorem 1

Proof. As discussed before, user n adds noise selected from $N(0,\gamma_n^2)$ to its sensory data, where γ_n^2 is chosen from an exponential distribution with hyper parameter η_2. Then, we have

$$\begin{aligned}
&Pr\{\mathcal{W}(x_n^{(1)}) = x\} = \frac{1}{\sqrt{2\pi\gamma_n^2}}\exp(-\frac{(x-x_n^{(1)})^2}{2\gamma_n^2})\\
&\le \frac{1}{\sqrt{2\pi\gamma_n^2}}\exp(-\frac{(x-x_n^{(2)})^2-(x_n^{(2)}-x_n^{(1)})^2}{2\gamma_n^2})\\
&= \exp(\frac{(x_n^{(2)}-x_n^{(1)})^2}{2\gamma_n^2})\frac{1}{\sqrt{2\pi\gamma_n^2}}\exp(-\frac{(x-x_n^{(2)})^2}{2\gamma_n^2})\\
&\le \exp(\frac{\triangle_n^2}{2\gamma_n^2})Pr\{\mathcal{W}(x_n^{(2)}) = x\}\\
&\le e^{\epsilon}Pr\{\mathcal{W}(x_n^{(2)}) = x\}.
\end{aligned} \tag{23}$$

From the above formula, we know that $Pr\{\mathcal{W}(x_n^{(1)}) = x\} \le e^{\epsilon}Pr\{\mathcal{W}(x_n^{(2)}) = x\}$ holds if and only if $\gamma_n^2 \ge \frac{\triangle_n^2}{2\epsilon}$. As γ_n^2 is chosen from an exponential distribution with hyper parameter η_2, we limit the probability of event $Pr\{\gamma_n^2 : \gamma_n^2 \ge \frac{\triangle_n^2}{2\epsilon}\}$ to at least $1-\delta$, where $\delta \in [0,1]$. Therefore, we have $Pr\{\gamma_n^2 : \gamma_n^2 \ge \frac{\triangle_n^2}{2\epsilon}\} = \exp(-\frac{\eta_2\triangle_n^2}{2\epsilon}) \ge 1-\delta$, which implies $-\frac{\eta_2\triangle_n^2}{2\epsilon} \le \ln(\frac{1}{1-\delta})$. Because $1/\eta_2 = r/\eta_1$, we have $r \ge \frac{\eta_1\triangle_n^2}{2\epsilon\ln(\frac{1}{1-\delta})}$. Based on the Lemma 1, we have $r \ge \frac{\alpha_n^2}{2\eta_1\epsilon\ln(\frac{1}{1-\delta})}$, where $\alpha_n = b\sqrt{2\ln\frac{1}{1-q}}$.

Since the domain of user n's noise variance γ_n^2 is $\mathtt{R}^+$, we divide $\mathtt{R}^+$ as $\mathtt{R}^+ = \mathtt{R}_1 \cup \mathcal{R}_2$, where $\mathtt{R}_1 = \{\gamma_n^2 \in \mathtt{R}^+ : \gamma_n^2 \ge \frac{\triangle_n^2}{2\epsilon}\}$ and $\mathtt{R}_2 = \{\gamma_n^2 \in \mathtt{R}^+ : \gamma_n^2 < \frac{\triangle_n^2}{2\epsilon}\}$. Then, we use $\mathcal{W}(x_n,\gamma_n^2)$ to denote the perturbation mechanism of user n. Given a subset $\mathcal{S} \subseteq \mathcal{R}$, we also divide $\mathcal{S}$ as $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, where we claim $\mathcal{W}(x_n,\gamma_n^2) \in \mathcal{S}_1$ if $\gamma_n^2 \in \mathtt{R}_1$ or $\mathtt{W}(x_n,\gamma_n^2) \in \mathtt{S}_2$ if $\gamma_n^2 \in \mathtt{R}_2$. Thus, we have

$$\begin{aligned}
\Pr_{\gamma_n^2\in\mathtt{R}^+}\{\mathcal{W}(x_n^{(1)},\gamma_n^2)\in\mathcal{S}\} &\le (\Pr_{\gamma_n^2\in\mathtt{R}_1} + \Pr_{\gamma_n^2\in\mathtt{R}_2})\{\mathcal{W}(x_n^{(1)},\gamma_n^2)\in\mathcal{S}\}\\
&\le \Pr_{\gamma_n^2\in\mathtt{R}_1}\{\mathcal{W}(x_n^{(1)},\gamma_n^2)\in\mathcal{S}_1\} + \delta\\
&\le e^{\epsilon}\Pr_{\gamma_n^2\in\mathtt{R}^+}\{\mathcal{W}(x_n^{(2)},\gamma_n^2)\in\mathcal{S}\} + \delta
\end{aligned}$$

∎

1.3 Proof of Lemma 2

Proof: Given a sum gate $\mathtt{G}$ and two inputs $\widehat{s}_1$ and $\widehat{s}_2$, the process of generating σ' is divided into five cases (shown in algorithm **Gate_eval**). For ease of description, we take **Case** 3 in the algorithm **Gate_eval** as an example, where $pk_{\widehat{s}_1} = pk_{\widehat{s}_2}$. Given two variables $\widehat{s}_1$ and $\widehat{s}_2$, both of which possess category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1},\rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2},\rho_{\widehat{s}_2})$, respectively. The verification tag $\sigma' = (pk',\rho')$ of the gate $\mathtt{G}$'s output (i.e., $y = \widehat{s}_1 + \widehat{s}_2$) is computed as follows.

$$\begin{aligned}
pk' = pk_{\widehat{s}_1}, \varphi'_\tau = \varphi_{\tau_1} + \varphi_{\tau_2}, F'_\tau = F_{\tau_1} + F_{\tau_2}\\
\rho' = (\mu',\nu') = (\mu_{\widehat{s}_1}\times\mu_{\widehat{s}_2}, \nu'^{(1)}, \nu'^{(2)}, \nu'^{(3)})
\end{aligned} \tag{24}$$

where $\nu'^{(1)} = \nu_{\widehat{s}_1}^{(1)} + \nu_{\widehat{s}_2}^{(1)}$, $\nu'^{(2)} = \nu_{\widehat{s}_1}^{(2)} + \nu_{\widehat{s}_2}^{(2)}$, and $\nu'^{(3)} = \nu_{\widehat{s}_1}^{(3)} + \nu_{\widehat{s}_2}^{(3)}$. With verification tag $\sigma' = (pk',\rho')$, anyone can verify the correctness of $\widehat{s}_1 + \widehat{s}_1$ without knowing $\widehat{s}_1$ and $\widehat{s}_1$ as below.

$$\begin{aligned}
e(g,h^{\nu'^{(1)}}) &\stackrel{?}{=} e(g^{\beta_{\widehat{s}_1}}, h^{(\varphi_{\tau_1}+\varphi_{\tau_2})+(t_{\widehat{s}_1}+t_{\widehat{s}_2})})\\
&= e(pk'^{(1)}, \mu'\times h^{(\varphi_{\tau_1}+\varphi_{\tau_2})})\\
e(g,h^{\nu'^{(2)}}) &\stackrel{?}{=} e(g^{\beta'_{\widehat{s}_1}}, h^{(F_{\tau_1}+F_{\tau_2})+(t_{\widehat{s}_1}+t_{\widehat{s}_2})})\\
&= e(pk'^{(2)}, \mu'\times h^{(F_{\tau_1}+F_{\tau_2})})\\
e(g,h^{\nu'^{(3)}}) &\stackrel{?}{=} e(g^{\beta''_{\widehat{s}_1}}, h^{(\widehat{s}_1+\widehat{s}_2)+(t_{\widehat{s}_1}+t_{\widehat{s}_2})})\\
&= e(pk'^{(3)}, \mu'\times h^{(\widehat{s}_1+\widehat{s}_2)})
\end{aligned} \tag{25}$$

Clearly, the verification tag in our verification mechanism is additive homomorphic. ∎

1.4 Proof of Lemma 3

Proof. Specifically, given a product gate $\mathtt{G}$ and two inputs $\widehat{s}_1$ and $\widehat{s}_2$, the process of generating σ' is divided into two cases (shown in algorithm **Gate_eval**). Similarly, we take **Case** 2 in the algorithm **Gate_eval** as an example. Specifically, given two variables $\widehat{s}_1$ and $\widehat{s}_2$, both of which possess category-1 verification tag $\sigma_1 = (pk_{\widehat{s}_1},\rho_{\widehat{s}_1})$ and $\sigma_2 = (pk_{\widehat{s}_2},\rho_{\widehat{s}_2})$, respectively. The process of generating verification tag $\sigma' = (pk',\rho')$ of the gate $\mathtt{G}$'s output is shown in Eqn.(19). Then, based on the verification tag $\sigma' = (pk',\rho')$ (refer to Eqn.(19)), anyone

can verify the correctness of $\widehat{s}_1 \times \widehat{s}_2$ without knowing $\widehat{s}_1$ and $\widehat{s}_2$ as below.

$$\begin{aligned} e(g, h^{\nu'^{(1)}}) &\stackrel{?}{=} e(g^{\xi}, h^{(\varphi_{\tau_1}+\varphi_{\tau_2})+(t_{\widehat{s}_1}+t_{\widehat{s}_2})}) \\ &= e(pk'^{(1)}, \mu' \times h^{(\varphi_{\tau_1}+\varphi_{\tau_2})}) \\ e(g, h^{\nu'^{(2)}}) &\stackrel{?}{=} e(g^{\xi'}, h^{(\digamma_{\tau_1}+\digamma_{\tau_2})+(t_{\widehat{s}_1}+t_{\widehat{s}_2})}) \\ &= e(pk'^{(2)}, \mu' \times h^{(\digamma_{\tau_1}+\digamma_{\tau_2})}) \\ e(g, h^{\nu'^{(3)}}) &\stackrel{?}{=} e(g^{\xi''}, h^{(\widehat{s}_1+\widehat{s}_2)+(t_{\widehat{s}_1}+t_{\widehat{s}_2})}) \\ &= e(pk'^{(3)}, \mu' \times h^{(\widehat{s}_1+\widehat{s}_2)}) \end{aligned} \tag{26}$$

Clearly, the verification tag in our verification mechanism is multiplicative homomorphic. ■

1.5 Proof of Theorem 3

We first review the definition of the proposed signature, and then give a formal security model. Finally, we demonstrate the soundness of our verification mechanism under this security model.

1.5.1 Signature method. Our proposed signature is a tuple of probabilistic polynomial time algorithms, i.e., **Setup**, **Gen_key**, **Gen_sign**, **Sign_verify**, as follows:

Setup$(1^\lambda) \to pp$. The setup algorithm generates the description of a pairing (e, p, G_1, G_2, g, h) by running the bilinear group generator $\mathcal{G}(1^\lambda)$, where $g, h \in G_1$. The algorithm randomly chooses two collision-resistant hash functions $H : \{0,1\}^* \to Z_p$ and $H' : \{0,1\}^* \to Z_p$. It returns the public parameter $pp = (e, p, G_1, G_2, g, h, H, H')$.

Gen_key$(pp) \to (sk, pk)$. The key generation algorithm randomly chooses $\beta, \beta', \beta'' \in \mathbb{Z}_p$ and returns the public key pk and the secret key sk as:

$$\begin{aligned} pk &= \{(g^\beta, h^\beta, h^{\frac{1}{\beta}}), (g^{\beta'}, h^{\beta'}, h^{\frac{1}{\beta'}}), (g^{\beta''}, h^{\beta''}, h^{\frac{1}{\beta''}})\} \\ sk &= (\beta, \beta', \beta'') \end{aligned}$$

Gen_sign$(m, sk) \to \rho$. Take a message m (with label τ, denote the description of m) and the secret key sk as the inputs, the sign algorithm first computes $\varphi_\tau = H(\tau)$ and $\digamma_\tau = H'(\tau)$. Then, it randomly chooses a integer $t \in Z_p$ and sets $\mu = h^t$, $\nu^{(1)} = \beta(\varphi_\tau + t) \bmod p$, $\nu^{(2)} = \beta'(\digamma_\tau + t) \bmod p$, and $\nu^{(3)} = \beta''(m + t) \bmod p$. It returns the signature $\rho = (\mu, \tau, \nu^{(1)}, \nu^{(2)}, \nu^{(3)})$.

Sign_verify$(pk, m, \rho) \to b$: The algorithm returns 1 indicating a valid message and signature pair if the following equations are correct; otherwise, it returns 0.

$$\begin{aligned} &e(g, h^{\nu^{(1)}}) \stackrel{?}{=} e(g^\beta, \mu \times h^{\varphi_\tau}) = e(pk^{(1)}, \mu \times h^{\varphi_\tau}) \\ &e(g, h^{\nu^{(2)}}) \stackrel{?}{=} e(g^{\beta'}, \mu \times h^{\digamma_\tau}) = e(pk^{(2)}, \mu \times h^{\digamma_\tau}) \\ &e(g, h^{\nu^{(3)}}) \stackrel{?}{=} e(g^{\beta''}, \mu \times h^{m}) = e(pk^{(3)}, \mu \times h^{m}) \end{aligned} \tag{27}$$

1.5.2 Security Model. Next, we introduce the security model of existential unforgeability under chosen message attack (*EUF-CMA*)[1, 3] for a digital signature scheme, which allows an adversary to query for signatures on messages of his choice deceptively and then adversary outputs a valid signature for a message which has not been queried before.

Specifically, given the digital signature scheme consists of four algorithms given above and an adversary $\mathcal{A}$ who is allowed to access a series of oracles, i.e., $\mathcal{O}_{\mathsf{Gen\_key}}(\cdot), \mathcal{O}_{\mathsf{Corrupt}}(\cdot)$ and $\mathcal{O}_{\mathsf{Gen\_sign}}(\cdot,\cdot)$, we define the following experiment under *EUF-CMA*:

Experiment $\mathbf{Exp}^{EUF\text{-}CMA}_{Signature,\mathcal{A}}(1^\lambda)$ $pp \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathcal{D}_u := \emptyset; \mathcal{D}_c := \emptyset; \mathcal{D}_\rho := \emptyset$; $(pk^*, m^*, \rho^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$; *return* 1 iff $pk^* \notin \mathcal{D}_c$, $(pk^*, m^*) \notin \mathcal{D}_\rho$, *and* $\mathsf{Sign\_verify}(pk^*, m^*, \rho^*) = 1$.	Oracle $\mathcal{O}_{\mathsf{Gen\_key}}(i)$ $(pk_i, sk_i) \leftarrow \mathsf{Gen\_key}(pp)$ $\mathcal{D}_u \leftarrow \mathcal{D}_u \cup \{i, pk_i, sk_i\}$ *return* pk_i.
Oracle $\mathcal{O}_{\mathsf{Gen\_sign}}(pk_i, m)$ $\rho \leftarrow \mathsf{Gen\_sign}(sk_i, m)$ $\mathcal{D}_\rho \leftarrow \mathcal{D}_\rho \cup \{pk_i, m, \rho\}$ *return* ρ	Oracle $\mathcal{O}_{\mathsf{Corrupt}}(i)$ $(pk_i, sk_i) \leftarrow \mathsf{Gen\_key}(pp)$ $\mathcal{D}_u \leftarrow \mathcal{D}_u \cup \{i, pk_i, sk_i\}$ $\mathcal{D}_c \leftarrow \mathcal{D}_c \cup \{pk_i\}$ *return* sk_i.

We say that our signature method is *EUF-CMA* secure if for any probabilistic polynomial time adversary $\mathcal{A}$, the following advantage is negligible:

$$\mathbf{Adv}^{EUF\text{-}CMA}_{Signature,\mathcal{A}}(1^\lambda) = \Pr[\mathbf{Exp}^{EUF\text{-}CMA}_{Signature,\mathcal{A}}(1^\lambda) = 1].$$

Proof: Suppose there exists a polynomial-time adversary $\mathcal{A}$ that can break our digital signature scheme in *EUF-CMA* model with non-negligible advantage. Then, we can build a simulator $\mathcal{B}$ that can break either the discrete logarithm (DL) assumption [1, 3] or collision-resistant hash functions with non-negligible advantage. Specially, $\mathcal{B}$ receives the tuple $(g, A = g^a)$ to output a, and two hash functions H, H' to find τ, τ' such that $\tau' \neq \tau$ and they are either $H(\tau') = H(\tau)$ or $H'(\tau') = H'(\tau)$.

Setup: $\mathcal{B}$ generates the description of a pairing (e, p, G_1, G_2, g) by running the bilinear group generator $\mathcal{G}(1^\lambda)$ and derives public key $pp = (e, p, G_1, G_2, g, h, H, H')$ by simulating

$$g = g^a, \quad h = g$$

$\mathcal{B}$ returns the public key $pp = (e, p, G_1, G_2, g^a, g, H, H')$ to $\mathcal{A}$, where g^a and g are uniform in the view of $\mathcal{A}$.

Querying Phase: At the beginning of querying phase, $\mathcal{B}$ randomly picks $i^* \in \{1, 2, ..., q_{\mathsf{Gen\_key}}\}$ then queries the following oracles adaptively.

$\mathcal{O}_{\mathsf{Gen\_key}}(i)$: $\mathcal{A}$ queries the key generation oracle on the index i at most $q_{\mathsf{Gen\_key}}$ times. For each query, if i has been queried before, $\mathcal{B}$ returns the public key pk_i from the database $\mathcal{D}_u$; otherwise, $\mathcal{B}$ simulates the public key as following:

- If $i \neq i^*$, $\mathcal{B}$ randomly chooses $\beta, \beta', \beta'' \in \mathbb{Z}_p$ and generates the public key pk_i and the secret key sk_i as:

$$\begin{aligned} pk_i &= \{(g^\beta, h^\beta, h^{\frac{1}{\beta}}), (g^{\beta'}, h^{\beta'}, h^{\frac{1}{\beta'}}), (g^{\beta''}, h^{\beta''}, h^{\frac{1}{\beta''}})\} \\ sk_i &= (\beta, \beta', \beta'') \end{aligned}$$

  $\mathcal{B}$ updates the database $\mathcal{D}_u$ as $\mathcal{D}_u \leftarrow \mathcal{D}_u \cup \{i, pk_i, sk_i\}$.
- If $i = i^*$, $\mathcal{B}$ randomly picks $r_1, r_2, r_3 \in \mathbb{Z}_p$ and generates the public key pk as:

$$\begin{aligned} pk_i &= \{(A^{r_1}, g^{r_1}, g^{\frac{1}{r_1}}), (A^{r_2}, g^{r_2}, g^{\frac{1}{r_2}}), (A^{r_3}, g^{r_3}, g^{\frac{1}{r_3}})\} \\ &= \{(g^{ar_1}, g^{r_1}, g^{\frac{1}{r_1}}), (g^{ar_2}, g^{r_2}, g^{\frac{1}{r_2}}), (g^{ar_3}, g^{r_3}, g^{\frac{1}{r_3}})\} \end{aligned}$$

  $\mathcal{B}$ updates $\mathcal{D}_u$ as $\mathcal{D}_u \leftarrow \mathcal{D}_u \cup \{i, pk_i, (r_1, r_2, r_3)\}$,

where r_1, r_2 and r_3 are uniform in the domain $\mathcal{Z}_p$ and hence the public key pk_i is uniform from the view of $\mathcal{A}$. $\mathcal{B}$ returns the public key pk_i to $\mathcal{A}$.

$\mathcal{O}_{\mathsf{Corrupt}}(i)$: $\mathcal{A}$ queries the corrupt oracle on the index i at most q_{Corrupt} times. For each query, if i has been queried before, $\mathcal{B}$ returns the secret key sk_i from the database $\mathcal{D}_u$; otherwise, $\mathcal{B}$ simulates the secret key as follows:

- If $i \neq i^*$, $\mathcal{B}$ randomly chooses $\beta, \beta', \beta'' \in \mathbb{Z}_p$ and generates the public key pk_i and the secret key sk_i as:

$$pk_i = \{(g^{\beta}, h^{\beta}, h^{\frac{1}{\beta}}), (g^{\beta'}, h^{\beta'}, h^{\frac{1}{\beta'}}), (g^{\beta''}, h^{\beta''}, h^{\frac{1}{\beta''}})\}$$
$$sk_i = (\beta, \beta', \beta'')$$

  $\mathcal{B}$ updates the database $\mathcal{D}_u$ as $\mathcal{D}_u \leftarrow \mathcal{D}_u \cup \{i, pk_i, sk_i\}$ and $\mathcal{D}_c \leftarrow \mathcal{D}_c \cup \{pk_i\}$.
- If $i = i^*$, $\mathcal{B}$ aborts since the secret key of challenging index cannot be simulated.

$\mathcal{B}$ returns the secret key $sk_i = (\beta, \beta', \beta'')$ to $\mathcal{A}$.

$\mathcal{O}_{\mathsf{Gen\_sign}}(pk_i, m)$: $\mathcal{A}$ queries the signing oracle on the public key pk_i and the message m at most $q_{\mathsf{Gen\_sign}}$ times. For each query, if (pk_i, m) has been queried before, $\mathcal{B}$ returns the signature ρ from the database $\mathcal{D}_\rho$; otherwise, $\mathcal{B}$ simulates the signature as follows:

- If $pk_i \neq pk_{i^*}$, $\mathcal{B}$ computes the signature based on the proposed scheme. Specifically, $\mathcal{B}$ randomly chooses a term $\tau, t \in \mathbb{Z}_p$ and computes

$$\begin{aligned}\mu &= h^t = g^t \\ v^{(1)} &= \beta(H(\tau) + t) \bmod p \\ v^{(2)} &= \beta'(H'(\tau) + t) \bmod p \\ v^{(3)} &= \beta''(m + t) \bmod p\end{aligned}$$

- If $pk_i = pk_{i^*}$, $\mathcal{B}$ randomly chooses a term $\tau, t \in \mathbb{Z}_p$ and retrieve the randomly r_1, r_2 and r_3 to compute

$$\begin{aligned}\mu &= h^t = g^t \\ v^{(1)} &= r_1(H(\tau) + t) \bmod p \\ v^{(2)} &= r_2(H'(\tau) + t) \bmod p \\ v^{(3)} &= r_3(m + t) \bmod p\end{aligned}$$

$\mathcal{B}$ returns the signature $\rho = (\mu, \tau, \nu_n^{(1)}, \nu_n^{(2)}, \nu_n^{(3)})$. Note that the simulation for $pk_i \neq pk_{i^*}$ is identical to our proposed scheme. Hence, it passes the verification algorithm and $\mathcal{A}$ cannot find $\mathcal{B}$ is processing the real scheme or the scheme simulation. For $pk_i \neq pk_{i^*}$, we have

$$e(g, h^{v^{(1)}}) = e(g^a, g^{r_1(H(\tau)+t)}) = e(g^{ar_1}, g^{H(\tau)+t}) = e(g^{\beta}, h^{H(\tau)}h^t)$$
$$e(g, h^{v^{(2)}}) = e(g^a, g^{r_2(H'(\tau)+t)}) = e(g^{ar_2}, g^{H'(\tau)+t}) = e(g^{\beta'}, h^{H'(\tau)}h^t)$$
$$e(g, h^{v^{(3)}}) = e(g^a, g^{r_3(m+t)}) = e(g^{ar_3}, g^{m+t}) = e(g^{\beta''}, h^m h^t)$$

Therefore, the signature for pk_{i^*} is correct and uniform from the view of $\mathcal{A}$.

Output: $\mathcal{A}$ outputs (pk^*, m^*, σ^*), then we consider following two cases.

- Case 1: $pk^* = pk_i$ then $\mathcal{A}$ breaks the DL problem. We have the following equations from the signature $\rho^* = (\mu, \tau, \nu^{(1)}, \nu^{(2)}, \nu^{(3)})$ such that

$$\begin{cases}\nu_n^{(1)} = \beta(H(\tau) + t) \bmod p = ar_1(H(\tau) + t) \bmod p \\ \nu_n^{(2)} = \beta'(H'(\tau) + t) \bmod p = ar_2(H'(\tau) + t) \bmod p \\ \nu_n^{(3)} = \beta''(m^* + t) \bmod p = ar_3(m^* + t) \bmod p.\end{cases}$$

  We can calculate a and t from the above equations since there are three equations with two unknown elements. Therefore, we break the BL by returning a.
- Case 2: $pk^* \neq pk_{i^*}$. We have the following equations from the signature $\rho^* = (\mu, \tau, \nu^{(1)}, \nu^{(2)}, \nu^{(3)})$ such that

$$\begin{cases}\nu_n^{(1)} = \beta(H(\tau) + t) \bmod p \\ \nu_n^{(2)} = \beta'(H'(\tau) + t) \bmod p \\ \nu_n^{(3)} = \beta''(m^* + t) \bmod p.\end{cases}$$

  We can calculate $H(\tau)$ and $H'(\tau)$ from the above equations since there are three equations with three unknown elements. Therefore, we break the collusion-resistant hash function by returning τ and τ', where τ' has the same hash result to τ and is queried before.

Probability analysis: The simulation aborts if $\mathcal{A}$ guesses i^* and j^* incorrect. Hence, we have the following advantage:

$$\mathbf{Adv}^{EUF\text{-}CMA}_{Signature,\mathcal{A}}(1^\lambda) = 1/q_{\mathsf{Gen\_sign}} \cdot \epsilon_{\mathsf{DL}} + 1/q_{\mathsf{Sign\_verify}} \cdot \epsilon_{\mathsf{H}},$$

where ϵ_{DL} and ϵ_{H} are the advantages to break the DL problem and collision-resistant hash function. ■