Publicly Verifiable Computation of Polynomials over Outsourced Data with Multiple Sources

Wei Song, Bing Wang, Qian Wang, *Member, IEEE,* Chengliang Shi, Wenjing Lou, *Fellow, IEEE,* and Zhiyong Peng, *Member, IEEE*

***Abstract*—Among all types of computations, the polynomial function evaluation is a fundamental, yet an important one due to its wide usage in the engineering and scientific problems. In this paper, we investigate publicly verifiable outsourced computation for polynomial evaluation with the support of multiple data sources. Our proposed verification scheme is universally applicable to all types of polynomial computations and allows the clients to outsource new data at any time. While the existing solutions only support the verification for polynomial evaluation over a single data source, i.e., all the inputs of the polynomial function are outsourced and signed by a single entity, our solution supports polynomial evaluations over multiple different data sources, which are more common and have wider applications, e.g., to assess the city air pollution, one needs to evaluate the environmental data uploaded from the multiple environmental monitor sites. In our proposed scheme, the verification cost for the client is independent with either the input size or the polynomial size so that it scales well in practice. We formally prove the correctness and soundness of our scheme and conduct numerical analysis and evaluation study to validate its high efficiency and scalability. The experimental results show that the data contributor signing 1,000 new data only takes 2.1 seconds, and the verification of the delegated polynomial function takes only 22 milliseconds, which is practically efficient for real-world applications.**

***Index Terms*—Verifiable polynomial, arithmetic circuit, homomorphic verifiable tags.**

I. Introduction

As cloud computing provides affordable and scalable computation and storage resources, outsourcing computation to the cloud becomes an unavoidable trend nowadays. Among different types of computations, the polynomial evaluation is considered to be a fundamental, yet an important one, and it has been widely used in the engineering and scientific problems. For example, the company utilizes the financial software to analyze its economic performance by executing polynomial functions over the past sales data. In cloud computing, the cloud server executes polynomials over the personal health data uploaded from various wearable devices to evaluate the personal health and make suggestion for people to keep healthy. Also, every year the government needs to execute polynomials over the statistical data to evaluate the social development situation and make the plan for the next year. However, while enjoying all the benefits of the cloud, the users also lose the control of their computation in the cloud. Due to the cloud's possible misbehaviors motivated by the monetary reasons (e.g., to save the computing resources) or caused by the hacking and the system failures, the client would like to verify the correctness of the computation result output by the cloud. According to [1], a verifiable computation scheme for the delegated polynomials should satisfy the following requirements: (1) *security*, meaning that the cloud server can prove the correctness of the delegated computation for the polynomial function f, and the client can correctly verify the result of the function f by checking the proof message; (2) *efficiency*, meaning that the client should be able to verify the result with communication and computation costs significantly lower than the cost of computing f locally; (3) *input-independent efficiency*, meaning that the verifying time is independent of the size of the inputs of the delegated polynomial function; (4) *unbound storage*, meaning that the client is able to outsource new input data to the cloud for polynomial evaluations; (5) *not fixed polynomial functions*, meaning that neither the delegated functions should be fixed nor the client is required to know the functions before outsourcing the data.

To the best of our knowledge, the authors in [1] proposed the first protocol which satisfies the above requirements. From the practical perspective, however, it still has some limitations which make it not practical in real-world application scenarios. First of all, the scheme of [1] does not support public verification. It has an implicit assumption that the data contributor and the computation requestor are the same entity (or the data owner has to share the private key with the computation requestor). But, it is not always true in practice, where the computation requestor is most likely to be a third-party entity. The second limitation is that the inputs of the polynomial must come from one single data source. In real-world applications, however, the input data of the polynomial may come from multiple contributors. Finally, the communication cost of the solution in [1] depends on the polynomial size, which raises the scalability concern in practice. Take the computation of the air pollution level as an example, there are many air quality monitor sites collecting the environmental pollution data and uploading these data to the cloud in the fixed time interval. Thus, the computation is expected to be conducted by a third-

This work is partially supported by National Natural Science Foundation of China (Grant Nos. 61202034, 61373167, U1636219, 61232002, and 61572378), National Basic Research Program of China (Grant No. 2014CB340600), and National High Technology Research and Development Program of China (Grant No. 2015AA016004). *Corresponding author: Qian Wang(qianwang@whu.edu.cn).*

W. Song, Q. Wang, C. Shi, and Z. Peng are with The State Key Laboratory of Software Engineering, School of Computer Science, Wuhan University, Wuhan, China. Email: {songwei, qianwang, chengliangshi, peng}@whu.edu.cn

B. Wang and W. Lou are with the Department of Computer Science, Virginia Polytechnic Institute and State University, Falls Church, Virginia, USA. Email: {bingwang, wjlou}@vt.edu

party agency over the environmental pollution data uploaded from multiple monitors (different data sources). At last, the air pollution information will be released to the public by the agency.

To verify the correctness of the result of the delegated computation over the outsourced data from multiple contributors, a publicly verifiable computation scheme for polynomial functions is highly required. To address this problem, we define three key requirements for a *practically verifiable computation* design, which are lacking in existing solutions: 1) *Public verification*. It means that any entity can verify the correctness of the result of the delegated polynomial function even if the inputs are not uploaded or signed by itself; 2) *The Support of multiple data sources*. It means that the inputs of the polynomial can be contributed by multiple independent data sources; 3) *Function-independent bandwidth efficiency*. It means that the communication cost should be independent of the complexity of the delegated function, i.e., the overhead of communication for a verification task is constant.

In this paper, we propose a novel and efficient publicly verifiable computation scheme for the delegated polynomial evaluation. The key characteristic of our design is that the proposed scheme is homomorphically verifiable for any general polynomial function. We formally prove the security of our scheme based on the Computational Diffie-Hellman (CDH) assumption and evaluate its efficiency in terms of the computation and the communication costs through extensive experiments. The main contributions in this paper can be summarized as follows.

- We, for the first time, model the problem of publicly verifiable delegated computation for the polynomial functions and identify three new practical requirements: *public verification*, *the support of multiple data contributors*, and *function-independent efficiency*. We propose an efficient and publicly verifiable outsourcing computation scheme to meet the above key requirements.
- To achieve the high bandwidth efficiency, we design a homomorphic verifiable computation tag structure for the delegated polynomial function, by which our verifiable computation scheme can achieve the constant communication cost and scales well in practice.
- We formally prove the correctness and the soundness of our scheme under the well-defined CDH assumption. We also implement a prototype and carry out a series of evaluation studies. The evaluation results further validate the effectiveness and efficiency of our scheme.

The rest of our paper is organized as follows. In the next section, we discuss the related work. Then, we introduce several cryptographic primitives used in this paper in Section III. We setup up the system model and formulate the problem in Section IV. Section V details our publicly verifiable computation scheme for the polynomial functions, which is followed by the performance analysis in Section VII. Section VIII shows the results of our simulation study. Finally, we conclude our paper in Section IX.

II. Related Work

The idea of designing *authenticated data structures* [2] can be considered as the first research stream related to our work. The follow-ups [3], [4], [5], [6], [7] extended the concept of authenticated data structure by enabling the clients to securely delegate special operations, such as the range search queries [3], [4], the authenticated dictionaries [5], [6], and the set operations [7] over the outsourced data to the cloud. However, none of the authenticated data structures can support the secure outsourcing of arbitrary polynomial computation over the remotely outsourced data.

Gennaro et al. proposed the verifiable computation model [8]. In this verifiable computation model, the client outsources data and a function evaluation to the remote server. The client computes a one-time encoding of the function and stores it at the server. This enables the server to not only evaluate the function, but also provide a proof that the evaluation has been correctly done. However, this model requires the client to know the delegated polynomial function in advance. To provide verifiable computation for not fixed polynomial functions, Parno et al. proposed the concept of '*multi-function verifiable computation*' [9], by which a client can delegate the computation of many functions on the same input D while being able to efficiently verify the results. However, in the multi-function verifiable computation, the client has to store additional information for every input D. Furthermore, it is impossible to update the local meta data without locally storing the previous data. As a result, all the outsourced data D has to be sent out at once. Thus, the scheme cannot apply to scenarios where the input data is continually growing. Benabbas et al. [10] proposed a practical verifiable computation model for high degree polynomial functions. In [10], the client stores the polynomial in the clear with the server as a vector of the coefficients. It also has the problem of only supporting fixed polynomial evaluations and high computation overheads.

Other researchers have proposed the verifiable computation schemes based on the homomorphic signature. Boneh et al. [11] proposed a realization of homomorphic signatures for the bounded constant-degree polynomials. Gennaro et al. [12] introduced homomorphic MACs to design a delegation scheme for arbitrary computation which is proven secure in a weaker model. Catalano et al. [13] proposed the practical homomorphic MACs for the arithmetic circuits to support the verifiable polynomial functions with the bounded degree. But their scheme does not support the public verification due to the use of a symmetric encryption algorithm. Backes et al. [1] reduced the verification communication overhead, however, it shares the same limitation with [13]. Fiore et al. [14] presented protocols for publicly verifiable secure outsourcing of evaluation of polynomials and matrix multiplication. However, this solution cannot support the scenario of multiple data contributors. Zhang et al. [15] constructed the batch verifiable computation model to efficiently compute the same function over multiple data sets. Backes et al. [16] proposed ADSNARK, a practical system for proving arbitrary computations over authenticated data, and it can be extended to support the scenario of multiple data contributors.

Chung et al. [17] proposed a solution for memory delegation, in which the client uploads its memory to the server, and he can later ask the server to update the outsourced memory and compute a function on the entire memory. However, to prove the correctness of the delegated computation, the client needs to store approximate $\log(n)$ data at local, where n is the scale of entirely outsourced data. Chen et al. [18] proposed the concept of the verifiable database with incremental updates, which is highly efficient for the database with frequent and small modifications. Zhang et al. [19] measured the trade-off between the storage and the verification time and reduce the storage overhead by slightly increasing the verification time. Lai et al. [20] proposed a verifiable homomorphic encryption (VHE), which enables verifiable computation on the outsourced encrypted data. Other solutions [21], [22], [23], [24] were proposed under a different model where the client needs to know the input of the computation.

Comparisons between the verification of polynomial computation and the verification of SQL range queries: In this paper, we design a new and novel verification scheme for the delegated computation of polynomial evaluation over the outsourced data, while our previous work [25] was designed to achieve the integrity verification for SQL range query. We list the main differences and challenges from three aspects as follows.

First, for the integrity verification of SQL range queries, the client has obtained the result before verification. Unlike this type of verification, the verification for polynomial evaluation requires the verifier to verify the result without knowing the inputs. Besides, for the range query, the tuples in a range are continuous. Based on this property, in [25], we designed the signature chain to verify the integrity of result, and the chain is built with the tuple identifiers of the previous tuple and the successive tuple (Readers can refer to Section 5 of [25] for the details). However, the delegated polynomial does not have this property. Therefore, in this paper we tackle the challenge of expressing any type of delegated polynomials and enable the server to generate proof messages to prove the correctness of the result without knowing the inputs. Because theoretically any polynomial can be computed by an arithmetic circuit, we utilize the arithmetic circuits to express the delegated polynomials and design the verification scheme (Please refer to Section III-A for the design details). On top of the arithmetic circuits, we let the server process the polynomial in a gate-by-gate mannner (Readers may refer to GateEval algorithm in Section V-B for the details). In comparison, the scheme in [25] does not have this construction design.

Second, in this paper we have to tackle the challenge of designing a verifying data structure to enable the verification based on the arithmetic circuits. Specifically, we design the Homomorphic Verifiable Computation Tags (HVCTs) as the verifying tags while we proposed Homomorphic Verifying Tags (HVTs) in [25]. However, these two tags are quite different. Actually, for the range query verification, the client only needs to verify the tuples in the result which are continuous outsourced data. Thus, we designed the verifying data structure in [25] based on HVTs, which is only additive homomorphic. Note that the polynomials not only have the addition operations but also the multiplication operations, so the functionality of HVTs are not sufficient to support the polynomial computation. Therefore, we designed HVCTs which is both additive and multiplicative homomorphic. In addition, our design of HVCTs allows the client to verify the result without knowing the inputs. That is, the client can use some public information ρ to verify the proof message. In comparison, the scheme in [25] does not have these nice properties and powerful functionalities, which are the key difference between the two designs. Readers may refer to GateEval algorithm in Section V-B of this paper and Section 6 of [25] for the details.

Third, we need to address the challenge of supporting multiple data sources. In this paper, the server processes the polynomial gate by gate based on the arithmetic circuits. However, it is difficult to have a unified verification data structure to support both the addition and multiplication operations when two inputs are signed by different keys (from different data contributors). To address this problem, our general idea is to put all the sum gates behind the product gates to express the delegated polynomial function. Then, based on this structure, we further design 1-level verifying tags and 2-level verifying tags. By leveraging these designs, the server is able to output the homomorphic verifying tag for every gate even if the verifying tags of two inputs are signed by different keys (Readers may refer to GateEval algorithm in Section V-B for the details). Although the scheme in [25] also supports multiple data contributors, the case is much simpler and the resulting solution is quite different. In [25], the server only needs to group the signatures from different signers, and then the client can verify the data integrity in a batch. However, in this paper, we have to carefully design the 1-level verifying tags and the 2-level verifying tags to address the multi-sourced data verification challenge so as to achieve the same functionality. Readers may refer to GateEval algorithm in Section V-B of this paper and Section 6 of [25] for the details.

To the best of our knowledge, there is no practical solution of publicly verifiable computation over polynomial functions on a large scale of outsourced data so far, which meets all of our previously-defined three requirements.

III. Preliminaries

A. Arithmetic Circuits

In this paper, we focus on the problem of verifiable delegated computation for the polynomial functions. Note that, we investigate the problem from the perspective of ensuring the integrity of computation results, instead of protecting the input privacy. This is also the key difference of our work from the existing privacy-preserving outsourcing computation protocols, most of which neglect the result verification and assume that the cloud will honestly follow the protocols to execute the delegated computations.

Because any polynomial $f \in \mathbb{F}[X]$ can be computed by an arithmetic circuit [26], we first give a brief overview of the arithmetic circuits.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2705628, IEEE Transactions on Information Forensics and Security

Definition 1. *An arithmetic circuit Φ over the field $\mathbb{F}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph. The vertices of Φ are called 'gates'. Every gate in Φ with in-degree 0 is called an input gate, which is labeled by either a variable from X or a field element from $\mathbb{F}$. Every other gate in Φ is labeled by either '$\times$' or '$+$' and has in-degree 2. Every gate labeled by '$\times$' is called a product gate and every gate labeled by '$+$' is called a sum gate. Every gate in Φ with out-degree 0 is called an output gate.*

An arithmetic circuit computes a polynomial function in a nature way. The gates compute the polynomial defined by their labels. A product gate computes the product of the polynomials on its incoming wires. A sum gate computes the sum of the polynomial on its incoming wires. The output of the circuit is the value contained on the outgoing wire of the output gate.

B. Bilinear Maps

We say a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a *bilinear map* [27] if:

1) $\mathbb{G}_1$ and $\mathbb{G}_2$ are the groups of the same prime order p;
2) g, h are generators of $\mathbb{G}_1$;
3) for all $a, b \in Z_p$(Z_p stands for the set of integer numbers from 0 to $p-1$), $g^a, g^b, h^a, h^b \in \mathbb{G}_1$, then $e(g^a, h^b) = e(g, h)^{ab}$ is efficiently computable;
4) the map is non-degenerate, i.e., $e(g, g) \neq 1$.

The bilinear maps have the following security property, which is also the security basis of our scheme.

Assumption 1 (Computational Diffie-Hellman (CDH) assumption [27])**.** *For $x, y \in Z_p$, given g, g^x, $g^y \in \mathbb{G}_1$, it is hard to compute g^{xy}.*

C. Homomorphic Verifiable Computation Tags

Ateniese et al. [28] proposed the concept of Homomorphic Verifiable Tags (HVTs), which allows a public verifier to verify the integrity of the data stored on a remote untrusted server. We extend this concept to Homomorphic Verifiable Computation Tags (HVCTs) to verify the delegated polynomial computation. First of all, whether the value x is an original input or an intermediate computational result of a polynomial, the HVCTs δ of the value x must be unforgeable. Then the HVCTs also should have the following important properties: 1) ***Verification without knowing the inputs***. The cloud server can use the HVCTs to generate a proof message which allows a verifier to verify result correctness of the delegated computation for the polynomial function without knowing the inputs; 2) ***Additive homomorphism***. Given a sum gate with the incoming wires x_1 and x_2, anyone can combine their homomorphic verifiable computation tags δ_1 and δ_2 into a new HVCT $\delta_{x_1+x_2}$ for the output $(x_1 + x_2)$ of the sum gate; 3) ***Multiplicative homomorphism***. Given a product gate with the incoming wires x_1 and x_2, anyone can combine their homomorphic verifiable computation tags δ_1 and δ_2 into a new HVCT $\delta_{x_1 \times x_2}$ for the output $(x_1 \times x_2)$ of the product gate.

IV. Problem Statement

A. System Model

In this work, we consider a cloud-based data service system composed of three entities as shown in Fig. 1, i.e., the cloud server, the clients, and the agency. The cloud server offers the storage service to the clients and executes the polynomial computations over the outsourced data for the agency. It is not a fully trusted entity. In our system model, the client and the agency are trusted entities. All the clients are able to create and upload data to the cloud server. To enable verifiable polynomial function over the outsourced data, the client signs the outsourced data. The agency asks the cloud server to execute the polynomial function f over the outsourced data from multiple clients. Besides executing the delegated function f, the cloud server generates the proof message P with which it proves that f has been correctly executed. To guarantee the correct execution of delegated polynomial function, the agency can also play as a verifier to verify the result R of f returned from the cloud server by checking the proof message P.

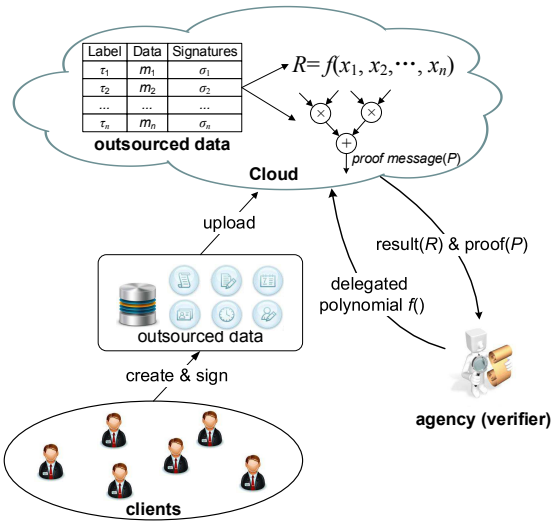


Fig. 1: System model

B. Threat Model and Security Definitions

In our system model, the cloud server stores and manages the outsourced data and executes the polynomial functions over the outsourced data for the agency. However, it is not fully trusted. It might misbehave due to the monetary reasons (e.g., for computation resource savings), the hacking or the system failures. For example, assuming there are 10,000 clients (air quality monitor sites) collecting the environmental pollution data all over the city. The client uploads the pollution data to the cloud every day. The agency asks the cloud to analyze the air pollution data by executing some polynomial functions such as average function and deviation function over the pollution data. But, the cloud may execute the polynomial function over a subset of daily pollution data rather than the whole data set for saving its computation resources. Even worse, the cloud server returns a random result or the history result to the agency. The main motivation of this work is to provide a verifiable polynomial evaluation scheme by which

the agency is able to validate whether the cloud server has correctly executed the delegated polynomial function. In this work, we consider the following potential security threatens in this work.

Data corruption. In this type of security threat, the outsourced data is corrupted. The corrupted data used as the input of delegated computation can lead to a wrong result. The adversaries could be the outside attackers or the cloud server.

Incorrect results. The cloud server may not fully perform the delegated computations over the entire inputs or randomly output a result to save the computation resources for the monetary reasons.

Forgery attack. Following the above two types of attacks, the adversary (e.g., the cloud server or the external attackers) may deliberately forge the signature or the proof message to cheat the agency to pass the verification.

The security of the result verification scheme is twofold: the *correctness* and the *soundness*. In our scheme, the *correctness* means that as long as the cloud server correctly performs the computation, the corresponding proof message will always pass the result integrity check, i.e., there is no false negative. The *soundness* means that the proof message corresponding to any false computation answer will be detected and cannot pass the integrity check, i.e., there is no false positive. Note that in the game between the adversary and the agency (verifier), the adversary has full access to the information stored in the cloud server, and the objective of the adversary is to pass the verification with the forged proof message.

C. Design Objectives

The main motivation of this work is to develop a practically verifiable computation scheme for the delegated polynomial functions over the outsourced data from multiple data sources. To this end, besides five requirements for a practical verifiable computation scheme mentioned in [1], this work proposes to achieve the following additional goals: (1) ***Correctness***: for three types of security threats, the agency is able to verify the correctness of the delegated polynomial executed at the remote cloud server; (2) ***Supporting public verification***: any entity should be able to act as the agency to verify the correctness of the result of the delegated computation over the outsourced data as long as the entity can access the public keys; (3) ***Supporting multiple data contributors***: the verifier is able to verify the result from the cloud, even if the inputs of the delegated polynomial function are uploaded and signed by the multiple independent clients; (4) ***Efficiency***: the verification overheads of the computation and the communication are significantly lower than the costs of the agency to download the data and execute the delegated function f locally. The agency should be able to verify the correctness of the result without knowing the inputs, storing any meta data, or sharing the information with the clients. Moreover, the verification efficiency meets the requirements of '*input-independent efficiency*' and '*function-independent efficiency*'.

D. An Example of Application Scenarios

In this paper, we use an air pollution data collection and analysis system as the example to illustrate the problem and to motivate our design. There are many environment monitor sites in this system. These sites act like the clients in the system model have an unbound collection of the air pollution data $D = \{D_1, D_2, \ldots, D_n\}$. Then, every site uploads the data to the cloud server at the fixed intervals (e.g., every hour or every day). A data item $D_i(m, \tau, \sigma)$ uploaded by the site includes the data m, the label τ, and the signature σ. The label τ of a data m is a binary string τ to describe the physical meaning of m such as 'air pollution in Washington DC on 2016-07-15 at 9 am'. The environment ministry, which acts like the agency in our system model, analyzes the air pollution data and releases the result to the public. It requests the cloud server to execute the delegated computation over the environment pollution data even the data are uploaded and signed by the multiple independent monitor sites. After getting the computation result from the cloud server, the agency verifies the correctness of the result. If the result passes the verification, the agency releases the air pollution information (e.g., air pollution level, PM2.5 level) to the public.

V. Publicly Verifiable Computation for Polynomial Evaluations

A. Overview

In the cloud-based data service system, there are n clients, i.e., $u_i (1 \leq i \leq n)$. Every client u_i has a key pair (a public key and a private key). To enable the publicly verifiable computation, u_i signs the data using its private key before uploading it. While the cloud server performs the delegated computation of the polynomial function f, it also outputs a proof message to prove that f has been correctly executed. The agency is able to verify the correctness of the result by checking the proof message. Specifically, the publicly verifiable computation scheme in this paper consists of six polynomial-time algorithms, i.e., SetUp, KeyGen, Sign, GateEval, ProofGen and VerifyProof. We introduce the verifiable computation scheme through detailing these algorithms. Table I summarizes the notations used in this paper.

TABLE I: Notations used in this paper

Symbols	Descriptions
$\mathbb{G}_1, \mathbb{G}_2$	two groups of the same prime order p
e	a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$
g, h	the generators of $\mathbb{G}_1$
H	$H : \{0,1\}^* \rightarrow Z_p$ is a one way hash function which maps the arbitrary strings to the elements in Z_p
sk, pk	$sk = \alpha$ is the user's private key.
	$pk = (g^\alpha, h^\alpha, h^{1/\alpha})$ is user's public key.
τ	the label for the outsourced data
t_τ	$t_\tau = H(\tau)$
m	the outsourced data
$\delta(pk, \sigma)$	the verifying tag for the polynomial
σ_m	$\sigma_m = (r, s)$ is the signature of m.
P	$P = \delta_R(pk, \sigma)$ is the proof message for the result R of the delegated polynomial function f.

B. Publicly Verifiable Computation

In this subsection, we detail the algorithms of the publicly verifiable computation for the polynomial functions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2705628, IEEE Transactions on Information Forensics and Security

$\mathsf{SetUp}(1^\lambda) \rightarrow (e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, H)$ is run by the cloud server to initialize the system. It takes a security parameter λ as input and outputs the global security parameters for the system. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order p, and g, h be the generators of $\mathbb{G}_1$, and $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map. The global parameters of our scheme are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, H)$, where $H : \{0,1\}^* \rightarrow Z_p$ is the one-way hash function which maps arbitrary strings to the elements in Z_p.

$\mathsf{KeyGen}(1^\kappa) \rightarrow (pk, sk)$ is run by a new client. It takes the security parameter κ as input and returns the public key pk and private key sk for the new client. When a new client u joins the cloud data service system, u selects a random $\alpha \in Z_p$ as its private key, and outputs its public key as $pk = (g^\alpha, h^\alpha, h^{1/\alpha})$. Then, u stores its private key locally and sends its public key pk to the cloud server. The cloud server stores u's identification and its public key in the table as shown in table II.

No.	*UserID*	*PK*
1	u_1	$(g^{\alpha_1}, h^{\alpha_1}, h^{1/\alpha_1})$
2	u_2	$(g^{\alpha_2}, h^{\alpha_2}, h^{1/\alpha_2})$
3	u_3	$(g^{\alpha_3}, h^{\alpha_3}, h^{1/\alpha_3})$
...	...	...

TABLE II: The client's identification is unique in the system. The private key is used to sign the data, and the public key is used to verify the result.

$\mathsf{Sign}(m, sk) \rightarrow \sigma_m$ is run by the client to generate the signature for the new outsourced data m, while the client creates or modifies m. The Sign algorithm takes m and the client's private key sk as inputs and outputs the signature σ_m. To support the publicly verifiable computation for the polynomials, the client u signs the data $m \in Z_p$ which has the label τ when u adds or modifies m. u first computes $t_\tau = H(\tau)$ and selects a random $k \in Z_p$, and then sets $r = h^k$, $s = \alpha(t_\tau + m + k) \mathsf{mod}\ p$, in which α is u's private key. The Sign algorithm generates the signature σ_m as Eq. (1). u uploads m, m's label τ and the signature σ_m to the cloud.

$$\sigma_m = (r, s) = (h^k, \alpha(t_\tau + m + k) \mathsf{mod}\ p). \quad (1)$$

The signature in our scheme is an El Gamal-type algorithm over bilinear map, it is also a type of unidirectional proxy re-signature scheme [29]. It is correct and secure under the CDH and 2-DL assumptions in bilinear map [29].

After the cloud server receives m and the corresponding signature $\sigma_m(r, s)$, it verifies the signature by Eq. (2), where $pk^{(1)} = g^\alpha$ is the first component of u's public key. If the verification fails, the cloud server outputs $\perp$; otherwise, the cloud server stores m, the label τ, and the signature σ_m.

$$e(g, h^s) \stackrel{?}{=} e(g^\alpha, r \times h^{H(\tau)+m}) = e(pk^{(1)}, r h^{t_\tau + m}). \quad (2)$$

$\mathsf{GateEval}(x_1, x_2, \delta_1, \delta_2) \rightarrow \delta'$ is run by the cloud server to generate the verifying tag δ' for the output of a gate in the arithmetic circuits. For a gate G, whether G is a sum gate or a product gate, the GateEval algorithm takes the input wires x_1 and x_2, and their verifying tags δ_1 and δ_2 as inputs, and returns the verifying tag δ' for G's output. Let $f(x_1, x_2, \ldots, x_n)$ be the delegated polynomial function, where $x_i (1 \leq i \leq n)$ denotes an outsourced data item used as one of f's inputs. Generally, any polynomial function f can be expressed as Eq. (3), where c_i is the constant coefficient and e_j is the exponent of x_j, e.g., $f = x_1(2x_2x_3 + x_4^3) = 2x_1x_2x_3 + x_1x_4^3$. Then, the cloud server uses the arithmetic circuit to express the delegated polynomial function in Fig. 2. When the cloud server executes a polynomial function f, it calls the GateEval algorithm to process f's arithmetic circuit gate-by-gate.

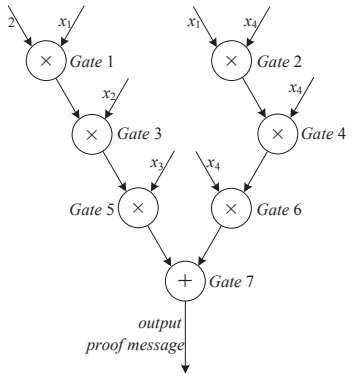


Fig. 2: The polynomial function is expressed by an arithmetic circuit, e.g., $x_1(2x_2x_3 + x_4^3) = 2x_1x_2x_3 + x_1x_4^3$. In our scheme, the sum gate is always after the product gate.

$$f(x_1, x_2, \ldots, x_n) = \sum (c_i \times \prod_{j \in [1,n]} x_j^{e_j}). \quad (3)$$

By the GateEval algorithm, the cloud server generates the verifying tag for the output of each gate. In our scheme, the inputs of a gate can be the original outsourced data, a constant $c \in Z_p$, or the output of other gate. For a polynomial function expressed by Eq. (3), the two inputs of a gate will not both be the constants.

In our scheme, we utilize the verifying tag to verify the result of the delegated computation. As the polynomial expressed in Eq. (3), all the product gates are executed before the sum gates. Based on the structure of the arithmetic circuit for polynomials, we divide the verifying tags into two categories: the 1-level verifying tag and the 2-level verifying tag. The verifying tag $\delta(pk, \sigma)$ in this paper consists of two parts: the user's public key pk and σ.

Definition 2. *The 1-level verifying tag includes: the raw data's verifying tag, the verifying tag for the output of product gate, and the verifying tag for the output of a sum gate G if two inputs of G have the 1-level verifying tags signed by the same private key. The structure of 1-level verifying tag $\delta(pk, \sigma)$ can be defined as*

$$\begin{aligned} pk &= (g^\alpha, h^\alpha, h^{\frac{1}{\alpha}}), \\ \sigma &= (r, s) = (h^k, \alpha(t_\tau + m + k) \mathsf{mod}\ p). \end{aligned} \quad (4)$$

Definition 3. *The 2-level verifying tag refers to the verifying tag for the output of sum gate, except two inputs of the sum gate both have the 1-level verifying tags and they are signed by the same private key. In our scheme, a 1-level verifying tag can be transformed to a 2-level verifying tag but a 2-level verifying tag will no longer be transformed back to a 1-level*

verifying tag. The structure of 2-level verifying tag $\delta(pk, \sigma)$ can be defined in Eq. (5).

$$pk = (g^{\alpha}, h^{\alpha}, h^{\frac{1}{\alpha}}),\ \sigma = (r, s) = (h^{k}, h^{\alpha(t_{\tau}+m+k)}). \quad (5)$$

For an original outsourced data m, its verifying tag $\delta(pk, \sigma(r, s))$ is the 1-level verifying tag, in which pk is the public key and $\sigma(r, s)$ is the signature σ_m. At each gate G, given two inputs x_1, x_2 having the verifying tags δ_1, δ_2, the cloud server runs the algorithm $\delta' \leftarrow$GateEval$(x_1, x_2, \delta_1, \delta_2)$ described below to generate a new verifying tag δ' for the output of G, which is in turn passed on as an input to the next gate in the circuit.

If G is a product gate, then

- Two inputs of the gate G are a constant $c \in Z_p$ and a variable x. The input x has the verifying tag $\delta(pk, \sigma(r, s))$ and the label t_{τ}. x could be the original outsourced data or the output of other product gate. Because the product gate is executed before the sum gates, the verifying tag δ must be a 1-level verifying tag. The GateEval algorithm outputs the verifying tag $\delta'(pk', \sigma')$ and the new label t'_{τ} for the output $y = c \times x$ of G (e.g., the gate 1 in Fig. 2) as

$$\begin{aligned} pk' = pk,\ t'_{\tau} &= c \times t_{\tau}, \\ \sigma' = (r', s') &= (r^{c}, c \times s) \\ &= (h^{ck}, \alpha(ct_{\tau} + cx + ck)). \end{aligned} \quad (6)$$

- Two inputs of the gate G are x_1, x_2, both of which have the 1-level verifying tags δ_1, δ_2 respectively. The cloud server randomly chooses an input (e.g., x_1) as the main input and outputs the verifying tag $\delta'(pk', \sigma')$ and the new label t'_{τ} for the output $y = x_1 \times x_2$ of G, e.g., the gate 2 and 3 in Fig. 2, as below.
  1) The cloud server sends r_1, pk_2, $\sigma_2(r_2, s_2)$ and x_2 to the agency u_a.
  2) u_a first verifies the signature $\sigma_2(r_2, s_2)$ by Eq. (2). If fails, u_a outputs $\perp$. Otherwise, u_a chooses two random numbers β and $k' \in Z_p$ and computes $\hat{s} = \beta(t_{\tau_2} + x_2 + k')$, $\hat{r} = r_1^{(t_{\tau_2}+x_2+k')}$, $h^{k'}$ and $pk' = (g^{\beta\alpha_1}, h^{\beta\alpha_1}, h^{1/\beta\alpha_1})$.
  3) u_a uploads $\{\hat{s}, \hat{r}, h^{k'}, pk'\}$ to the cloud.
  4) The cloud server outputs the verifying tag $\delta'(pk', \sigma')$ as

$$\begin{aligned} pk' &= (g^{\beta\alpha_1}, h^{\beta\alpha_1}, h^{1/\beta\alpha_1}),\ t'_{\tau} = t_{\tau_1} \times t_{\tau_2}, \\ \sigma' = (r', s') &= (\hat{r} \times h^{t_{\tau_1}x_2 + t_{\tau_2}x_1} \times h^{k' \times (t_{\tau_1}+x_1)}, \hat{s} \times s_1) \\ &= (r_1^{(t_{\tau_2}+x_2+k')} \times h^{t_{\tau_1}x_2 + t_{\tau_2}x_1} \times h^{k't_{\tau_1} + k'x_1}, \\ &\quad \beta(t_{\tau_2} + x_2 + k') \times \alpha_1(t_{\tau_1} + x_1 + k_1)) \\ &= (h^{k_1(t_{\tau_2}+x_2+k') + t_{\tau_1}x_2 + t_{\tau_2}x_1 + k't_{\tau_1} + k'x_1}, \\ &\quad \beta\alpha_1(t_{\tau_1}t_{\tau_2} + x_1x_2 + k_1(t_{\tau_2} + x_2 + k') \\ &\quad + t_{\tau_1}x_2 + t_{\tau_2}x_1 + k't_{\tau_1} + k'x_1)). \end{aligned} \quad (7)$$

As the polynomial expressed in Eq. (3), all the product gates are executed before the sum gates, so the data with 2-level verifying tag will not be the input of a '*product gate*'.

If G is a sum gate, then

- Two inputs of gate G are a constant $c \in Z_p$ and a variable x with the 1-level verifying tag $\delta(pk, \sigma)$. The GateEval algorithm outputs the verifying tag δ' and the new label t'_{τ} for the output $y = c + x$ of G as

$$\begin{aligned} pk' = pk = (g^{\alpha}, h^{\alpha}, h^{1/\alpha}),\ t'_{\tau} &= t_{\tau} + H(c), \\ \sigma' = (r', s') = (r, h^{s} \times h^{\alpha(H(c)+c)}) & \\ = (h^{k}, h^{\alpha((t_{\tau}+H(c))+(x+c)+k)}). & \end{aligned} \quad (8)$$

- Two inputs of G are a constant $c \in Z_p$ and a variable x. The input x has the 2-level verifying tag $\delta(pk, \sigma(r, s))$. The GateEval algorithm outputs the verifying tag δ' and the new label t'_{τ} for the output $y = c + x$ of G as

$$\begin{aligned} pk' = pk = (g^{\alpha}, h^{\alpha}, h^{1/\alpha}),\ t'_{\tau} &= t_{\tau} + H(c), \\ \sigma' = (r', s') = (r, s \times h^{\alpha \times (H(c)+c)}) & \\ = (h^{k}, h^{\alpha((t_{\tau}+H(c))+(x+c)+k)}). & \end{aligned} \quad (9)$$

- Two inputs of G are the variables x_1, x_2 with the verifying tags $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$, both of which are 1-level verifying tags. The GateEval algorithm outputs the verifying tag δ' and the label t'_{τ} for the output $y = x_1 + x_2$ of G, e.g., the gate 7 in Fig. 2 as
  - If $pk_1 = pk_2$, it means $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$ are signed by the same private key.

$$\begin{aligned} pk' = pk_1,\ t'_{\tau} = t_{\tau_1} + t_{\tau_2}, & \\ \sigma' = (r', s') = (r_1 \times r_2, s_1 + s_2) = (h^{k_1+k_2}, & \\ \alpha_1((t_{\tau_1} + t_{\tau_2}) + (x_1 + x_2) + (k_1 + k_2)). & \end{aligned} \quad (10)$$

    In this case, the cloud server is able to output the verifying tag without communicating with the agency u_a. The new verifying tag is still a 1-level verifying tag.
  - If $pk_1 \neq pk_2$, it means $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$ are signed by different private keys.
    1) The cloud server sends the third components $(h^{1/\alpha_1}, h^{1/\alpha_2})$ of pk_1 and pk_2 to the agency u_a.
    2) u_a chooses a random $\beta \in Z_p$, and then returns $(g^{\beta}, h^{\beta}, h^{1/\beta}, h^{\beta/\alpha_1}, h^{\beta/\alpha_2})$ to the cloud.
    3) The cloud server outputs the verifying tag $\delta'(pk', \sigma')$ and t'_{τ} as

$$\begin{aligned} pk' = (g^{\beta}, h^{\beta}, h^{1/\beta}),\ t'_{\tau} &= t_{\tau_1} + t_{\tau_2}, \\ \sigma' = (r', s') = (r_1 \times r_2, h^{\frac{\beta}{\alpha_1} \times s_1} \times h^{\frac{\beta}{\alpha_2} \times s_2}) & \\ = (h^{k_1+k_2}, h^{\beta((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k_1+k_2)}). & \end{aligned} \quad (11)$$

    In this case, the new verifying tag is changed to the 2-level verifying tag.
- Two inputs of G are the variables x_1 and x_2. The input x_1 has the 1-level verifying tag $\delta_1(pk_1, \sigma_1)$ and the input x_2 has the 2-level verifying tag $\delta_2(pk_2, \sigma_2)$. The GateEval algorithm outputs the verifying tag δ' and t'_{τ} for the output $y = x_1 + x_2$ of G as
  - if $pk_1 = pk_2$, it means $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$ are signed by the same private key.

$$\begin{aligned} pk' = pk_1 = pk_2,\ t'_{\tau} &= t_{\tau_1} + t_{\tau_2}, \\ \sigma' = (r', s') = (r_1 \times r_2, s_2 \times h^{s_1}) & \\ = (h^{k_1+k_2}, h^{\alpha_1((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k_1+k_2)}). & \end{aligned} \quad (12)$$

– if $pk_1 \neq pk_2$, it means $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$ are signed by the different private keys.
  1) The cloud server sends the third component h^{1/α_1} of the 1-level verifying tag δ_1's public key to the agency u_a.
  2) Because all the keys of the 2-level verifying tags are generated by u_a, it is able to output h^{α_2/α_1} and sends it to the cloud.
  3) The cloud server outputs the verifying tag $\delta'(pk', \sigma')$ and t'_τ as

$$\begin{aligned} pk' &= pk_2 = (g^{\alpha_2}, h^{\alpha_2}, h^{1/\alpha_2}), \\ t'_\tau &= t_{\tau_1} + t_{\tau_2}, \\ \sigma' &= (r', s') = (r_1 \times r_2, s_2 \times h^{\frac{\alpha_2}{\alpha_1} \times s_1}) \\ &= (h^{k_1+k_2}, h^{\alpha_2((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k_1+k_2)}). \end{aligned} \tag{13}$$

- Two inputs of G are the variables x_1 and x_2, both of which have the 2-level verifying tags $\delta_1(pk_1, \sigma_1)$ and $\delta_2(pk_2, \sigma_2)$. The GateEval algorithm outputs the verifying tag δ' and the new label t'_τ for the output $y = x_1 + x_2$ of G as below:
  – if $pk_1 = pk_2$,

$$\begin{aligned} pk' &= pk_1 = pk_2, \ t'_\tau = t_{\tau_1} + t_{\tau_2}, \\ \sigma' &= (r', s') = (r_1 \times r_2, s_1 \times s_2) \\ &= (h^{k_1+k_2}, h^{\alpha_1((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k_1+k_2)}). \end{aligned} \tag{14}$$

  – if $pk_1 \neq pk_2$,
    1) The cloud server randomly chooses an input x_1 as the main input.
    2) The agency u_a outputs α_1/α_2 and sends it to the cloud. Because all the secret parameters in the 2-level verifying tags are selected by the agency, u_a grasps the parameters α_1 and α_2.
    3) The cloud server outputs the verifying tag $\delta'(pk', \sigma')$ and the label t'_τ as

$$\begin{aligned} pk' &= pk_1 = (g^{\alpha_1}, h^{\alpha_1}, h^{1/\alpha_1}), \ t'_\tau = t_{\tau_1} + t_{\tau_2}, \\ \sigma' &= (r', s') = (r_1 \times r_2, s_1 \times s_2^{\frac{\alpha_1}{\alpha_2}}) \\ &= (h^{k_1+k_2}, h^{\alpha_1((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k_1+k_2)}). \end{aligned} \tag{15}$$

It is worth noting that although the agency needs to select the secret parameters and generate the messages for some gates, the agency can select the secret parameters in a batch for these gates and utilize a trusted verifying entity to generate the messages for a verification task in the real world cloud applications.

ProofGen$(\delta) \rightarrow P$ is run by the cloud server. When the computation reaches the last gate of the circuit, the cloud server outputs the proof message P based on the verifying tag δ output by running the GateEval algorithm on the last gate. By the GateEval algorithm, the cloud server processes the arithmetic circuit of the delegated polynomial function f gate by gate. When the computation reaches the last output gate of the circuit f, the cloud server outputs the verifying tag $\delta_R(pk, \sigma)$ generated by the GateEval algorithm from the last gate as the proof message P.

VerifyProof$(P) \rightarrow (True, False)$ is executed by the agency who is also the querier and can act as the verifier to verify the result of delegated polynomial function. After the agency u_a receives the result $R = f(x_1, x_2, \ldots, x_n)$ and the proof message $P = \delta_R(pk, \sigma)$ for the polynomial function f, it verifies the correctness of R by checking P. It outputs $True$ if the checking passes the verification; Otherwise, it returns $False$.

For every input x_i of f with the label τ_i[1], the agency u_a computes $t_{\tau_i} = H(\tau_i)$. Next, u_a evaluates the circuit on the labels $(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$, i.e., computing $\rho \leftarrow f(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$.

If δ_R is a 1-level verifying tag, i.e., the last gate of the circuit of the function f is a product gate, u_a verifies P by Eq. (16). Otherwise, u_a verifies P by Eq. (17). If the verification passes, u_a accepts the result R; otherwise, u_a believes that R is incorrect.

$$e(g, h^s) \stackrel{?}{=} e(g^\alpha, h^{\rho+R+k}) = e(pk^{(1)}, r \times h^\rho \times h^R), \tag{16}$$

$$e(g, s) \stackrel{?}{=} e(g^\alpha, h^{\rho+R+k}) = e(pk^{(1)}, r \times h^\rho \times h^R). \tag{17}$$

The verifying tags in our paper are the homomorphic verifiable computation tags, which will be proven in the next section. This feature enables the agency to verify the delegated computation without knowing the inputs. Moreover, it allows the cloud server to generate a proof message which is independent of the size of the inputs and the complexity of the polynomial function to save the communication cost.

Note that the labels of the outsourced data are chosen by the clients to describe the physical meaning of them, so they are all public information. In practice, computing $\rho = f(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$ can be executed in advance to improve the efficiency of the real-time applications. For example, the agency can compute ρ for the whole year's pollution data statistics in advance, and this is reasonable for most related applications.

VI. Security Analysis

A. Correctness

Before proving the correctness of the proposed verification scheme, we first prove that the verifying tags in our scheme are additive homomorphic and multiplicative homomorphic.

Lemma 1. *The verifying tag in our scheme is additive homomorphic.*

Proof. For a sum gate G, the input wires are x_1 and x_2 with the labels τ_1 and τ_2 (Note that for a constant c, it's label is c), and $t_{\tau_1} = H(\tau_1)$ and $t_{\tau_2} = H(\tau_2)$. By the GateEval algorithm in our scheme, the cloud server generates the verifying tag for the output $y = x_1 + x_2$ of G as $\sigma' = (r', s') = (h^k, h^{\beta((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k)})$, where β is the security parameter selected by the agency for the sum gate G. Therefore, the agency can verify the result $x_1 + x_2$ of G

[1] u_a can decide the labels of inputs based on its needs. For example, the labels of inputs for calculating the average PM2.5 value in a day at Washington DC should be '*PM2.5 in Washington DC on 2015-07-15 at 9 am*', '*PM2.5 in Washington DC on 2015-07-15 at 10 am*', and so on.

by Eq. (18) based on the bilinear maps without knowing the inputs x_1 and x_2.

$$\begin{aligned} e(g, s') &= e(g, h^{\beta((t_{\tau_1}+t_{\tau_2})+(x_1+x_2)+k)}) \\ &= e(pk^{(1)}, r' \times h^{t_{\tau_1}+t_{\tau_2}} \times h^{x_1+x_2}). \end{aligned} \tag{18}$$

It is clear that the verifying tags are additive homomorphic. □

Lemma 2. *The verifying tag in our scheme is multiplicative homomorphic.*

Proof. For a product gate G, the input wires are x_1 and x_2 with the labels τ_1 and τ_2, and $t_{\tau_1} = H(\tau_1)$ and $t_{\tau_2} = H(\tau_2)$. By the GateEval algorithm, the cloud server generates the verifying tag for the output $y = x_1 \times x_2$ of G as $\sigma' = (r', s') = (h^k, \alpha(t_{\tau_1} \times t_{\tau_2} + x_1 \times x_2 + k))$. Therefore, the agency can verify the output $x_1 \times x_2$ of G by Eq. (19) based on the bilinear maps without knowing the inputs x_1, x_2.

$$\begin{aligned} e(g, h^{s'}) &= e(g, h^{\alpha(t_{\tau_1} \times t_{\tau_2} + x_1 \times x_2 + k)}) \\ &= e(pk^{(1)}, r' \times h^{t_{\tau_1} \times t_{\tau_2}} \times h^{x_1 \times x_2}). \end{aligned} \tag{19}$$

It is clear that the verifying tags are multiplicative homomorphic. □

Theorem 1. *The verification scheme in our scheme achieve correctness. That is, as long as the cloud server correctly performs the computation, the corresponding proof message will always pass the result integrity check.*

Proof. According to Lemmas 1 and 2, the verifying tags in our scheme are homomorphic verifiable computation tags (HVCTs). Due to the property of HVCTs, the agency is able to verify the correctness of the result of a polynomial function without knowing the inputs. The correctness of the verification of our scheme is equivalent to the correctness of the VerifyProof(P) algorithm. Based on the properties of bilinear maps, the correctness of Eq.s (16) and (17) can be proved as

$$\begin{aligned} e(g, h^s) &= e(g, h^{\alpha(\rho+R+k)}) = e(g^\alpha, h^{\rho+R+k}) \\ &= e(pk^{(1)}, r \times h^\rho \times h^R), \end{aligned} \tag{20}$$

$$\begin{aligned} e(g, s) &= e(g, h^{\beta(\rho+R+k)}) = e(g^\beta, h^{\rho+R+k}) \\ &= e(pk^{(1)}, r \times h^\rho \times h^R). \end{aligned} \tag{21}$$

□

B. Soundness

Theorem 2. *The verification scheme in our scheme achieves soundness, i.e., the proof message corresponding to any false computation answer will be detected and cannot pass the result integrity check.*

Proof. We prove the soundness of the verification scheme by showing that once the cloud server or the external attackers can forge the proof message corresponding to any false computation answer to pass the verification, they would be able to build an adversary $\mathcal{A}$ with non-negligible probability ϵ that solves the CDH problem in the bilinear maps.

Assume that the system global parameters are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, h, H)$, where $h = g^x, x \in Z_p$. Given a proof message $P = \delta_R(pk, \sigma(r, s))$ where $pk^{(1)} = g^\alpha$, set $\alpha = x \times y$ $(y \in Z_p)$. In this attack scenario, the adversary $\mathcal{A}$ outputs the CDH challenge (g^x, g^y) as $(g^{xy}, g^x) = (pk^{(1)}, h)$. Let q_{H_i} be the total number of the queries on H for computing $t_{\tau_i} = H(\tau_i)$. Thus, the collision in H for computing t_{τ_i} occurs with the probability at most $q_{H_i}/2^l$ where l is the length of H's output. Applying the Reset Lemma [30], $\mathcal{A}$ can produce two valid verifying tags $\delta_1(pk, \sigma_1(r, s_1))$ and $\delta_2(pk, \sigma_2(r, s_2))$ for the delegated polynomial function $f(x_1, x_2, \ldots, x_n)$ with probability at least $\prod_{i=1}^{n}(\epsilon - (\epsilon \times q_{H_i} + 1)/2^l)^2$.

If δ_R is a 1-level verifying tag, $r = h^k$, $s_1 = \alpha(c_1 + R + k) \bmod p$ and $s_2 = \alpha(c_2 + R + k) \bmod p$, where c_1 and c_2 are two different random responses for $\rho \leftarrow f(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$. Based on these assumptions, $\mathcal{A}$ can solve the CDH problem by computing and outputting:

$$\begin{aligned} (\frac{h^{s_1}}{h^{s_2}})^{1/(c_1-c_2)} &= (\frac{h^{\alpha(c_1+R+k)}}{h^{\alpha(c_2+R+k)}})^{1/(c_1-c_2)} \\ &= h^\alpha = (g^x)^{xy} = g^{x^2 y}. \end{aligned} \tag{22}$$

If δ_R is a 2-level verifying tag, $r = h^k$, $s_1 = h^{\alpha(c_1+R+k)}$ and $s_2 = h^{\alpha(c_2+R+k)}$, where c_1 and c_2 are two different random responses for $\rho \leftarrow f(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$. Based on these assumptions, $\mathcal{A}$ can solve the CDH problem by computing:

$$\begin{aligned} (\frac{s_1}{s_2})^{1/(c_1-c_2)} &= (\frac{h^{\alpha(c_1+R+k)}}{h^{\alpha(c_2+R+k)}})^{1/(c_1-c_2)} \\ &= h^\alpha = (g^x)^{xy} = g^{x^2 y}. \end{aligned} \tag{23}$$

It is clear that if the adversary can build a forgery of the verifying tag then we can solve the CDH problem in the bilinear map $\mathbb{G}_1$, which is computationally infeasible. Therefore, the proposed verification scheme is sound, i.e., the adversary cannot generate a forgery of proof message for any false computation answer. □

VII. Performance Analysis

A. Communication Cost

For a verification task of the delegated polynomial function f, the cloud server needs to return the proof message to the agency u_a and communicate with u_a to generate the security parameter for some gates in the corresponding arithmetic circuit. The size of proof message $\delta(pk, \sigma)$ is $(|S_{pk}| + |S_\sigma|)$ bits, where $|S_{pk}|$ is the the size of first component of public key, and $|S_\sigma|$ is the size of signature. For a sum gate in the arithmetic circuit, the message for generating the security parameter is $c \times |S_G|$, where c is a number from 0 to 7 according to the different inputs of the sum gate, and $|S_G|$ is the size of an element of $\mathbb{G}_1$. Thus, the total communication cost for a verifying task is $(|S_{pk}| + |S_\sigma|) + (c \times sum)|S_G| = (3 + c \times sum)|S_G|$, where sum is the total number of sum gates in the arithmetic circuit for the delegated function f.

B. Computation Cost

The Sign$(m, sk) \rightarrow \sigma_m$ procedure takes $T_{exp} + T_{hash} + T_{mul} + 2T_{add} + T_{mod}$ computations, where T_{exp}, T_{hash}, T_{mul},

T_{add}, and T_{mod} represent one exponentiation operation, one hash operation, one multiplication operation, one addition operation, and one modular operation in $\mathbb{G}_1$, respectively.

For the product gate, it takes $T_{exp} + T_{mul}$ for Eq. (6) and $2T_{hash} + 2T_{add} + 4T_{mul} + 2T_{exp}$ for Eq. (7) in GateEval algorithm. For the sum gate, the computation cost is $2T_{exp} + T_{mul} + T_{hash} + T_{add}$ for Eq. (8), $2T_{mul} + 2T_{exp}$ for Eq. (11), $T_{mul} + T_{exp} + T_{hash} + T_{add}$ for Eq. (9), $2T_{mul} + T_{exp}$ for Eq. (13), and $2T_{mul} + T_{exp}$ for Eq. (15) in GateEval algorithm.

The computation cost to generate a proof message by the ProofGen algorithm is $\sum_{g \in |f|} T_g$, where $|f|$ is the set of gates in the arithmetic circuit f, and T_g is the computation cost for gate g by GateEval algorithm.

In the VerifyProof algorithm, the agency verifies the proof message P from the cloud. If the verifying tag in P is a 1-level verifying tag, the computation cost is $3T_{exp} + 2T_{mul} + T_{pair} + T_\rho$ where T_{pair} denotes one pairing operation as $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, and T_ρ denotes the computation cost for computing $\rho = f(t_{\tau_1}, t_{\tau_2}, \ldots, t_{\tau_n})$. Otherwise, the computation cost is $2T_{exp} + 2T_{mul} + T_{pair} + T_\rho$. It is worth noting that ρ can be calculated in advance (i.e., one-time cost), then the agency does not have to compute ρ during the verification to improve the computation efficiency.

VIII. Experimental Results

A. Experiment Setup

We carry out the experiments to evaluate the performance of our scheme. We utilize Java Pairing-Based Cryptography Library (jPBC)[2] to implement the pairing computation in the proposed scheme in this paper. All the experiments are conducted under Ubuntu with an Intel 2.6GHz processor and 4GB memory. We set the size of security parameter p to be 128 bits.

We use the representative real-world data set of city air pollution in China[3] as our experimental data set. The experimental data set collects the daily Air Quality Index (AQI) of 367 major cities in China for one month. The total size of experimental data is $11,010$. In the experiments, we assume that the pollution monitor site in every city is a client, which generates the original air quality data and uploads the data to the cloud. And the environment ministry of China acts as the agency to analyze the air pollution data from all the cities and release the analysis result to the public. In our experiments, two delegated polynomial functions are executed over the experimental data, including the average function which calculates the average AQI value in a month for every city and the standard deviation function which measures the standard deviation of the AQI value for every city. We compare the performance of our scheme with the work homomorphic MACs in [1] and ADSNARK in [16]. Because the verification scheme in [1] does not support the scenario of the multiple data contributors, we assume all the air pollution data in the experiments for the reference [1] are outsourced by one client.

[2]jPBC, http://gas.dia.unisa.it/projects/jpbc/

[3]http://datacenter.mep.gov.cn/

B. Performance of Signature Generation

In our scheme, the client signs the data before outsourcing. For a practically verifiable computation scheme, it should ensure that the signing process is highly efficient and will not burden the client. We measure the time for signing data with respect to the scale of data, which ranges from $1,000$ to $10,000$. We compare the time cost of signing data by the Sign algorithm in our scheme with those in homomorphic MACs and ADSNARK.

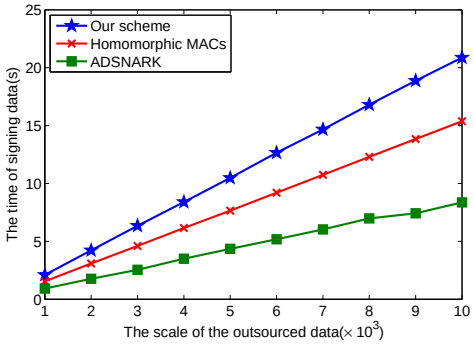


Fig. 3: Time cost for signing the outsourced data

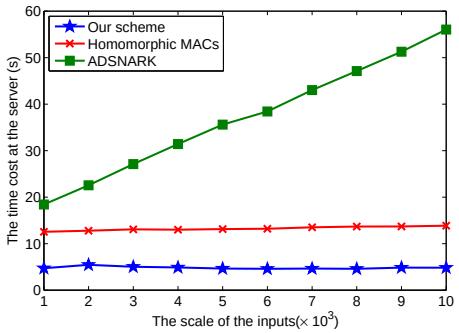
As shown in Fig. 3, the computation cost for signing data in our scheme increases linearly with respect to the scale of outsourced data. The time cost of signing the outsourced data is from 2.105 to 20.876 seconds. The efficiency of signing the outsourced data in our scheme is comparable with those in homomorphic MACs and ADSNARK. That clearly shows the client in our scheme is able to efficiently outsource the new data. Moreover, it is worth noting that the solutions of homomorphic MACs and ADSNARK cannot support the multiple data contributors directly. But, our scheme is able to support the public verification with multiple data contributors efficiently. This scenario is usually common in the real applications.

C. Computation Cost of Verification at Server Side

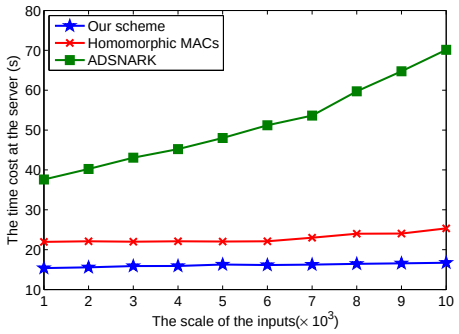
In our scheme, the computation cost of a verification task is divided into two parts: the computation cost at the cloud server and the computation cost at the agency. The computation cost at the server side is to generate the proof message for the delegated polynomial by the ProofGen algorithm. We carry out the experiments to evaluate the computation cost at the cloud server for the verification task with the scale of inputs from $1,000$ to $10,000$ and the different number of data contributors from 1 to 300. First, to evaluate the server performance for the polynomial function with the different scale of inputs, we assume all the air pollution data are uploaded and signed by one client. We compare the server cost of our scheme with those in the verification schemes homomorphic MACs [1] and ADSNARK [16].

The experimental results of the time cost at the server are shown in the Fig. 4. The cloud server is able to generate the proof message by the ProofGen algorithm in 6 seconds for the delegated average function and in 17 seconds for the delegated deviation function, no matter what scale the inputs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2705628, IEEE Transactions on Information Forensics and Security

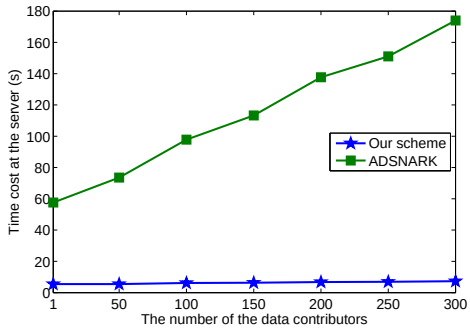


(a) The delegated polynomial is the average function

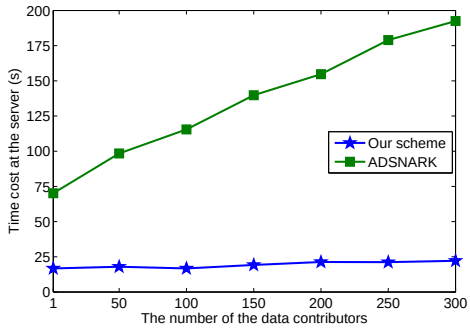


(b) The delegated polynomial is the standard deviation function

Fig. 4: Computation cost at the cloud server for the delegated polynomial verification with the different scales of the inputs



(a) The delegated polynomial is the average function



(b) The delegated polynomial is the standard deviation function

Fig. 5: Computation cost at the cloud server for the delegated polynomial verification with the different number of the data contributors

are. And the experimental results also show that our scheme can effectively reduce the server overhead for the delegated polynomial verification compared with homomorphic MACs and ADSNARK.

Second, one of the main research motivations is to support the integrity verification of the delegated polynomial over the outsourced data from the multiple data contributors. So we carry out the experiments to evaluate the computation overheads for the server while the air pollution data are contributed by the different clients (from 1 to 300). We fixed the scale of the inputs as $10,000$. Because the scheme of homomorphic MACs cannot support the multiple data contributors, we compare the performance of our scheme with the extended ADSNARK [16].

From the experimental results in the Fig. 5, we can find that our scheme can provide the efficient verification services for the delegated polynomials even the inputs are uploaded and signed by the different users. As shown, while the outsourced data are uploaded by the multiple clients (from 1 to 300), the cloud server is able to generate the proof message by the ProofGen algorithm in 5.4 seconds to 7.3 seconds for the delegated average function and in 16.7 seconds to 22.1 seconds for the delegated deviation function. Moreover, while the data contributors increase from 1 to 300, the server overhead increases slightly by our scheme. However, the server overhead by the extended ADSNARK increases significantly from 57.6 seconds to 174 seconds for the delegated average function and from 70 seconds to 193 seconds. It indicates our scheme is an efficient verification scheme for the delegated polynomials over the outsourced data from multiple data contributors.

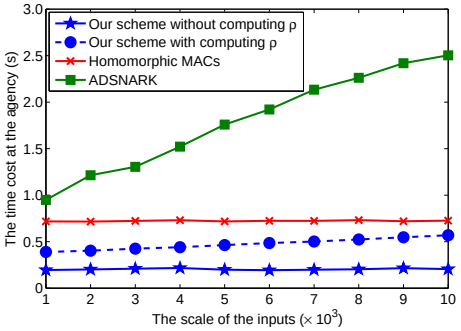
D. Computation Cost of Verification at the Agency

The computation cost at the agency is the agency (verifier) verifies the proof message returned from the cloud server by the VerifyProof algorithm. We carry out the experiments to evaluate the agency performance with the scale of inputs from $1,000$ to $10,000$ and the different number of data contributors from 1 to 300. First, we assume all the air pollution data are uploaded and signed by one client. We measure the computation cost of verification at the agency for two cases: the verification with computing ρ and the verification without computing ρ (the parameter ρ has been computed in advance). We compare the verification cost of our scheme with those in the verification schemes homomorphic MACs [1] and ADSNARK [16].

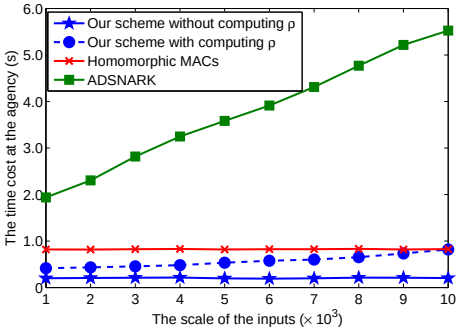
From the experiments, we can find that our scheme enables the agency to verify the integrity of the delegated polynomials more efficiently than those in the homomorphic MACs and the ADSNARK. As shown in the Fig. 6, while the agency computes the parameter ρ during the verification, it is able to finish the verification from 0.39 seconds to 0.57 seconds for the average function and from 0.42 seconds to 0.82 seconds for the standard deviation function. And if the agency computes the parameter ρ in advance, it can complete the verification more efficiently. The agency is able to complete the verification in 0.22 seconds for the average function and in 0.21 seconds for the standard deviation function. It is practical for the real applications.

Second, we also carry out the experiments to evaluate the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2705628, IEEE Transactions on Information Forensics and Security

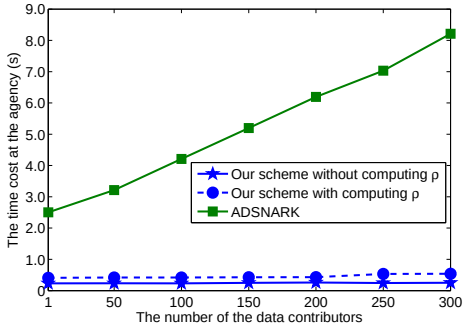


(a) The delegated polynomial is the average function

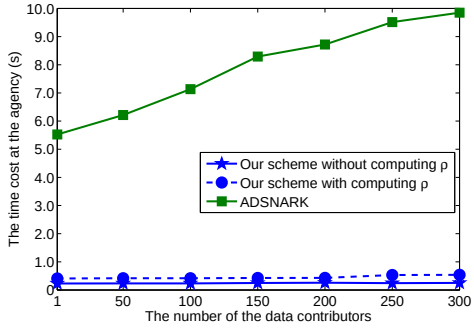


(b) The delegated polynomial is the standard deviation function

Fig. 6: Computation cost at the agency (verifier) for the delegated polynomial verification with the different scales of the inputs



(a) The delegated polynomial is the average function



(b) The delegated polynomial is the standard deviation function

Fig. 7: Computation cost at the agency (verifier) for the delegated polynomial verification with the different numbers of the data contributors

agency performance while the outsourced data are uploaded and signed by the multiple data contributors. We fixed the scale of the inputs as $10,000$. Because the scheme of homomorphic MACs cannot support the multiple data contributors, we compare the performance of agency under the average function and the standard deviation function with the extended ADSNARK. The experimental results are shown in Fig. 7. From the experimental results, we find that our scheme enables the agency to verify the integrity of the delegated polynomial function f more efficiently especially when the inputs of f are contributed by the multiple entities. When f is the average function, the agency of our scheme is able to verify the result in 0.25 seconds computing the parameter ρ in advance and in 0.54 seconds computing ρ during verification. When f is the standard deviation function, the agency also can complete the verification in 0.26 seconds computing the parameter ρ in advance and in 0.54 seconds computing ρ during verification. Moreover, the efficiency of the verification in our scheme will not remarkably declined with the data contributors increase. It indicates that our scheme is an efficient verification solution for the delegated polynomials with the multiple data contributors.

E. Communication Cost of Verification

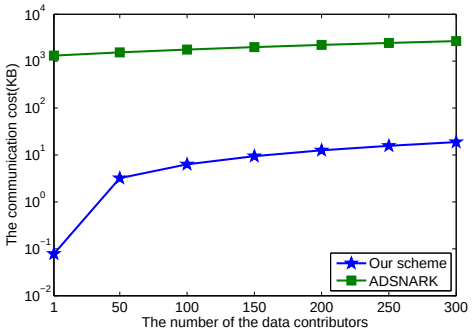
We carry out the experiments to evaluate the communication costs of the verification for the delegated polynomial functions. First, we evaluate the communication overhead with the different scale of the inputs from $1,000$ to $10,000$. We assume all the air pollution data are uploaded by one client. The experimental results are shown in the Table III. While all the data are from one source, the cloud server is able to generate the proof message independently without communicating with the agency. Then, the only message for the verification task is the proof message returned from the cloud. It is 0.078 KBytes, which is irrelevant to the scale of the inputs or the delegated polynomials. And the communication overhead of our scheme is more lower than those in homomorphic MACs and ADSNARK.

Inputs	Communication cost (KBytes)		
	our scheme	homomorphic MACs	ADSNARK
1000	0.078	0.59	136.7
2000	0.078	0.59	258.7
3000	0.078	0.59	380.7
4000	0.078	0.59	502.7
5000	0.078	0.59	623.7
6000	0.078	0.59	743.7
7000	0.078	0.59	865.7
8000	0.078	0.59	987.7
9000	0.078	0.59	1109.7
10000	0.078	0.59	1311.7

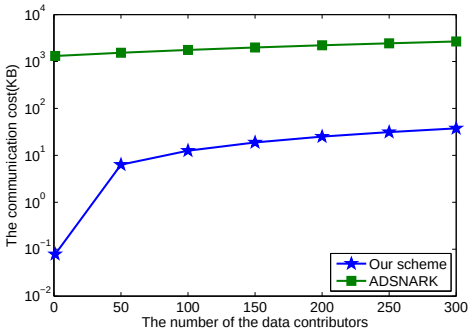
TABLE III: Communication costs for verification (KB) while all the inputs are from one source

We also carry out the experiments to evaluate the communication overheads with the different numbers of the data sources. We fixed the scale of the input as $10,000$. We compare the communication cost of our scheme with the extended ADSNARK. The experimental results are shown in the Fig. 8.

Because the cloud server only needs to communicate with the agency at the sum gates the inputs of which are signed by the different public keys, the communication cost of our scheme is small and more lower than that in ADSNARK. The communication costs for the average function are from 0.078KB to 18.83KB, and the communication costs for the standard deviation function are from 0.078 KB to 37.58KB. The experimental results demonstrate that the proposed scheme is able to efficiently support the outsourcing computation services for polynomials over the outsourced data in the cloud nowadays.



(a) The delegated polynomial is the average function



(b) The delegated polynomial is the standard deviation function

Fig. 8: Communication overhead for the delegated polynomial verification with the different numbers of the data contributors

IX. Conclusion

In this paper, we examined the practical problem of outsourcing polynomial evaluation over the outsourced cloud data. Due to the possible misbehaviors of the cloud server, result verification for the outsourced computation is a must. We further identified three new practical requirements for the verifiable polynomial evaluation outsourcing scheme. To fulfill the requirements, we proposed a public key-based verification scheme utilizing the bilinear maps. Our scheme allows the agency to verify the result of the delegated polynomial function over the outsourced data without knowing the inputs even if they are uploaded and signed by multiple different clients. In addition, our scheme also meets the requirements of '*input-independent efficiency*', '*unbound storage*', '*not fixed function*', '*function-independent efficiency*'. Based on the thorough security analysis, we proved that our scheme is correct and sound. We conducted several experiments to show that the verification processes are efficient and our scheme is appropriate for using in cloud-based outsourcing computation systems.

References

[1] M. Backes, D. Fiore, and R. M. Reischuk, "Veriable delegation of computation on outsourced data," in *Proc. of CCS'13*. ACM, 2013, pp. 863–874.

[2] M. Naor and K. Nissim, "Certificate revocation and certificate update," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 561–570, 2000.

[3] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *Proc. of RSA Conference on the Cryptographers' Track*. Springer, 2003, pp. 295–313.

[4] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data strucutures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, 2004.

[5] G. D. Battista and B. Palazzi, "Authenticated relational tables and authenticated skip lists," in *Proc. of DBSec*. Springer, 2007, pp. 31–46.

[6] C. Papamanthou and R. Tamassia, "Time and space efficient algorithms for two-party authenticated data structures," in *Proc. of ICICS*. Springer, 2007, pp. 1–15.

[7] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operatios on dynamic sets," in *Proc. of CRYPTO*. Springer, 2011, pp. 91–110.

[8] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. of CRYPTO*. Springer, 2010, pp. 465–482.

[9] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Proc. of TCC*. Springer, 2012, pp. 422–439.

[10] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proc. of CRYPTO*. Springer, 2011, pp. 111–131.

[11] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Proc. of EUROCRYPT'11*. Springer, 2011, pp. 149–168.

[12] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *Proc. of ASIACRYPT*. Springer, 2013, pp. 301–320.

[13] D. Catalano and D. Fiore, "Practical homomorphic MACs for arithmetic circuits," in *Proc. of EUROCRYPT'13*. Springer, 2013, pp. 336–352.

[14] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proc. of CCS'12*. ACM, 2012, pp. 501–512.

[15] L. Zhang and R. Safavi-Naini, "Batch verifiable computation of polynomials on outsourced data," in *Proc. of ESORICS*. Springer, 2015, pp. 167–185.

[16] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data," in *Proc. of S&P'15*. IEEE, 2015, pp. 271–286.

[17] K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz, "Memory delegation," in *Proc. of CRYPTO*. Springer, 2011, pp. 151–168.

[18] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," in *Proc. of ESORICS*. Springer, 2014, pp. 148–162.

[19] L. Zhang and R. Safavi-Naini, "Verifiable delegation of computations with storage-verification trade-off," in *Proc. of ESORICS*. Springer, 2014, pp. 112–129.

[20] J. Lai, R. H. Deng, H. Pang, and J. Weng, "Verifiable computation on outsourced encrypted data," in *Proc. of ESORICS*. Springer, 2014, pp. 273–291.

[21] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," in *Proc. of Eurosys'13*. ACM, 2013, pp. 71–84.

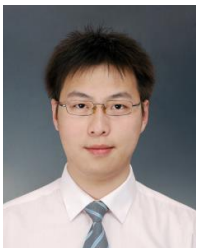
[22] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *Proc. of NDSS*. IEEE, 2012.

[23] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *Proc. of USENIX Security Symposium*. ACM, 2012, pp. 253–268.

[24] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish, "A hybrid architecture for interactive verifiable computation," in *Proc. of S&P'13*. IEEE, 2013, pp. 223–237.

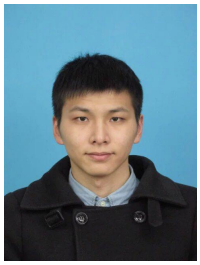
[25] W. Song, B. Wang, Q. Wang, Z. Peng, and W. Lou, "Tell me the truth: Practically public authentication for outsourced databases with multi-user modification," *Information Sciences*, vol. 387, pp. 221–237, 2017.
[26] A. Shpilka and A. Yehudayoff, "Arithmetic circuits: a survey of recent results and open questions," *Foundations & Trends in Theoretical Computer Science*, vol. 5, no. 3-4, pp. 207–388, 2010.
[27] N. Bourbaki, *Elements of Mathematics. Algebra: Algebraic Structures. Linear Algebra*. Addison-Wesley, 1974.
[28] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of CCS'07*. ACM, 2007, pp. 598–610.
[29] G. Ateniese and S. Hohenberger, "Proxy re-signatures: New definitions, algorithms, and applications," in *Proc. of CCS'05*. ACM, 2005, pp. 310–319.
[30] M. Bellare and A. Palacio, "GQ and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks," in *Proc. of CRYPTO*. Springer, 2002, pp. 162–177.

Wei Song is an Associate Professor with the School of Computer Science, Wuhan University, China. He received his B.S. and Ph.D. degree from Huazhong University of Science and Technology, China, in 2001 and 2008, respectively. His current research interests are in the areas of applied cryptography and network security, with focus on secure data service in cloud computing.



Bing Wang received the B.S. degree from Fudan University, China, in 2008, the M.S. degree from Shanghai Jiao-Tong University, China, in 2011, and the Ph.D. degree from Illinois Institute of Technology, USA, in 2016. His current research interests are in the areas of applied cryptography and network security, with current focus on secure data service in cloud computing and next generation Internet. He is a student member of the IEEE.

Qian Wang is a Professor with the School of Computer Science, Wuhan University. He received the B.S. degree from Wuhan University, China, in 2003, the M.S. degree from Shanghai Institute of Microsystem and Information Technology (SIMIT), Chinese Academy of Sciences, China, in 2006, and the Ph.D. degree from Illinois Institute of Technology, USA, in 2012, all in Electrical Engineering. His research interests include wireless network security and privacy, cloud computing security, and applied cryptography. Qian is an expert under National "1000 Young Talents Program" of China. He is a co-recipient of the Best Paper Award from IEEE ICNP 2011 and a recipient of IEEE Asia-Pacific Outstanding Young Researcher Award 2016. He is a Member of the IEEE and a Member of the ACM.



Chengliang Shi received his the B.S. degree in Computer Science from Wuhan University, China. He is currently working towards the M.S. degree in Computer Science in Wuhan University, China. His current research interests are in the areas of applied cryptography and network security, with focus on secure data service in cloud computing and encrypted data search.

Wenjing Lou is a Professor at Virginia Polytechnic Institute and State University. Prior to joining Virginia Tech in 2011, she was a faculty member at Worcester Poly-technic Institute from 2003 to 2011. She received her Ph.D. in Electrical and Computer Engineering at the University of Florida in 2003. Her current research interests are in cyber security, with emphases on wireless network security and data security and privacy in cloud computing. She was a recipient of the U.S. National Science Foundation CAREER award in 2008. She is a fellow of the IEEE.

Zhiyong Peng received the B.S. and M.S. degree in Computer Science from Wuhan University and Changsha Institute of Technology of China, respectively. He received Ph.D. degree from Kyoto University of Japan in 1995. He is a professor of at Wuhan University. Prior to join Wuhan University in 2000, he worked as a researcher at the Advanced Software Technology and Mechatronics Research Institute of Kyoto from 1995 to 1997 and was a member of the technical staff at Hewlett-Packard Laboratories, Japan from 1997 to 2000. His current research interests are in the database, trusted data management, and complex data management.